

WEB TECHNOLOGIES

SUB CODE: ACS006

IARE-R16

**Prepared by: Dr. K Srinivasa Reddy, Professor,
Mr. B. Venkateswara Rao, Associate Professor
Mr. A. Krishna Chaitanya, Associate Professor**

UNIT-I

HyperText Markup Language
(HTML)

Agenda

Introduction to HTML

- Hyper Text Markup Language
- HTML Example
- The structure of an HTML document

Introduction to HTML

- **The World Wide Web**

 - The set of computers on internet that supports HTTP

- **Web Server**

 - The special software that allows the website management

 - Accepts client request for information

 - Responds to the client request by providing the page with the
n i

- **Web Client**

 - A special software also known as browser that allows

 - Connecting to the appropriate server

 - Query the server for the information to be read

 - Provide the interface to read the information returned by the server

- **HTTP**

 - The Hypertext Transfer Protocol

 - u b t e s Internet
Explore, Firefox etc.) to request the documents from
e

Hyper Text Markup Language

- TextMixed with markup tags
 - Tags enclosed in angledbrackets
(`<h1> Heading</h1>`)
- The language interpreted by the browser
- Web Pages are also know as theHTML documents
- What does markup describe
 - Appearance
 - Layout
 - Content

HTML Example

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN"
"http://www.w3.org/TR/html4/loose.dtd">
<html>
<head>
<title>WebMail</title>
<LINK rel="stylesheet" type="text/css"
href="/twig/styles/classic/basic.css"></head>
<body>
<div class="twigheader">
<div align="center"></div></div><hr><form action="/twig/index.php" method="POST"
name="login_form">

<div><center><table
border=1><tr><td><table border=0
cellspacing=0>
<tr><td colspan=2 class="tabletitle"><b>Please enter your user
information</b></td></tr><tr><td align="right"> Username: </td><td><input
name="login_username"></td></tr>
<tr><td align="right">Password:</td><td><input name="login_password" type="password"></td></tr>
<tr><td colspan=2 align="center"><HR><input type="submit" name="set_twig_authenticated"
value="Log In"></td></tr></table></td></tr></table></center></div></form>
<hr><div
class="twigfooter"></div></body>
</html>
```

The structure of an HTML document

- DOCTYPE
- Document
- Head • Document
- Body • Titles and
- Footers • Text
- Formatting • Line
- Breaks
- Text Styles

DOCTYPE

- Doctype declaration is the very first thing in the HTML document before <html> tag.
- This tag tells the browser which HTML or XHTML specification the document uses

example-

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML  
4.01//EN"  
"http://www.w3.org/TR/html4/strict.dtd">
```


Document Head

- Information placed in this section is essential for the inner content of the document

- `<title></title>`, all other information will not be visible in the document

- **example**

```
<html>
  <head>
    <title>First
app</title></head>
  <body>
    ....
  </body>
</html>
```

Document Body

- The tags used `<body>` of the textual information `</body>` end of

`<body>`

.....

`<body>`

page default like background color, text color,
the attribute of the `<body>` tag

Titles and Footers

- **Title**

- Describes what page is all about without being too wordy
- The text included in the `<title></title>` shows up in the title bar of the browser

```
<title>....TheMessage ....</title>
```

- **Footer**

- Information commonly placed in the foot of the webpage. Generally hold copyright

```
<address>the text</address>
```

Example

```
<head><title>This is the  
title</title></head><body>  
  <address> This is the  
  footer</address></body>
```

Text Formatting

- tag that provide this functionality is `<p>`. hsintextualinformation.The

Example

```
<html><head
  >
  <title>First
  app</title></head>
  <body>
  first paragraph 1.....this is the sample text <p>this
  is the second paragraph...this is the sample text
  </body>
</html>
```

Line Breaks

- Use when the text need to be started from the newline and not from
l t

- **Example**

```
<html><head  
>  
  <title>First  
  app</title></head>  
  <body>  
    firestparagraph 1.....thisisthasampletext<br>thisisthesecond  
pa t  
  </body>  
</html>
```

Text Styles

- **Bold**
Displays the text in the BOLDFACE style. The tag used are `....`
- **Italics**
Displays text in ITALICS. The tags used in this case are `<i>.....</i>`
- **Underline**
Displays the text as underlined. The tags used in this case are `<u>...</u>`
- **Example**

```
<html><head
  >
  <title>First
  app</title></head>
  <body>
  <b>first paragraph 1.....this is the sample
  text</b><br><i>this is the second paragraph...this is the
  sample text</i><br><u>third line...underlined</u>
  </body>
</html>
```

Lists

- Types of list

- Unordered List

- ```

```

- Ordered List

- ```
<ol> </ol>
```

Type - Specifies the type of bullet

Type=**fillround** gives solid round black bullet

Type=**square** gives solid square blackbullet

Type="1" will give counting numbers (1,2...)

Type="A" will give uppercase letters (A,B...)

Type="a" will give lowercase letters (a,b....)

Type="I" will give uppercase Roman Numbers(I,II,III....)

Type="i" will give lowercase Roman Numbers(i,ii,iii....)

Start used to alter the numbering sequence.

Each list item startswith

```
<li>
```

OL Ordered (Numbered)List & Unordered List

- **Element**

Example

```
-<ol  
  type="I"> <li>First  
  Element <li>Second  
  Element <li>Third  
  Element </ol>
```

Attributes -type, Start, value

- **ListEntries:LI**

... End Tag Optional

- **Element**

Example

```
-<ul  
  type="square"> <li>F  
  irst Element  
  <li>Second Element  
  <li>Third Element  
  </ul>
```


DefinitionList

- Is not list of items
- List of terms and explanation of terms
- Example
 - `<dl>`
 - `<dt>Coffee</dt>`
 - `<dd>Black hot
drink</dd>`
 - `<dt>Milk</dt>`
 - `<dd>White cold
drink</dd>`
 - `</dl>`

OUTPUT

- Definition list starts with `<dl>` tag
Definition list term starts with `<dt>` tag
Definition list definition starts with `<dd>` tag

Displaying Images

- Images are defined with `` tag
- `` tag is empty

`<img`

`src="url">` **src**

stands for source

url points to the location where image is stored

``

alt attributes are used to define alternate text for the image

Table

- Defined with `<table>` tag
table is divided into rows(with the `<tr>` tag)
and each row is divided into data cells(with `<td>` tags. Td
stands for table data.

Example

Table Attributes

border

cellspacing

cellpadding

etc

```
<table border=1>  
<tr><td>First row first cell</td><td>First row second  
cell</td></tr><tr><td>Second row first cell</td><td>Second row second  
cell</td></tr></table>
```

HTML Links

- Uses `<a>` (anchor) tag and `href` attribute to link to the another
- Can point to any resource on the web

Example `India Development Gateway`

Target Attribute

defines where the link will be opened

`India Development`

`Gateway` Here `target="_blank"` specifies that document will open in new window.

Anchor Tag and nameAttribute

- Name attribute is used to create named anchor used to jump to specific location in the webpage

Example

```
<a name="anchor1">
```

```
<a href="#anchor1">go to anchor1</a>
```

HTML frames

- Used to display more than one HTML in the same browser window document
- Uses `<frameset>` tag to define how to divide window into frames

Java script

Introduction

JavaScript is the most popular scripting language on the internet, and works in all major browsers, such as Internet Explorer, Firefox, Chrome, Opera, and Safari.

What is JavaScript?

- JavaScript is a scripting language
- JavaScript is usually embedded directly into HTML pages
- JavaScript was designed to add interactivity to HTML pages
- JavaScript is an interpreted language (means that scripts execute without preliminary compilation)
- Everyone can use JavaScript without purchasing a license

Introduction

What can a JavaScript do?

- **JavaScript gives HTML designers a programming tool** - HTML authors are normally not programmers, but JavaScript is a scripting language with a very simple syntax! Almost anyone can put small "snippets" of code into their HTML pages
- **JavaScript can put dynamic text into an HTML page** - A JavaScript statement like this: `document.write("<h1>" + name + "</h1>")` can write a variable text into an HTML page
- **JavaScript can react to events** - A JavaScript can be set to execute when something happens, like when a page has finished loading or when a user clicks on an HTML element

Introduction

JavaScript can be used to validate data - A JavaScript can be used to validate form data before it is submitted to a server. This saves the server from extra processing

JavaScript can be used to create cookies - A JavaScript can be used to store and retrieve information on the visitor's computer

JavaScript can read and write HTML elements - A JavaScript can read and change the content of an HTML element

Introduction

Put a JavaScript into an HTML page

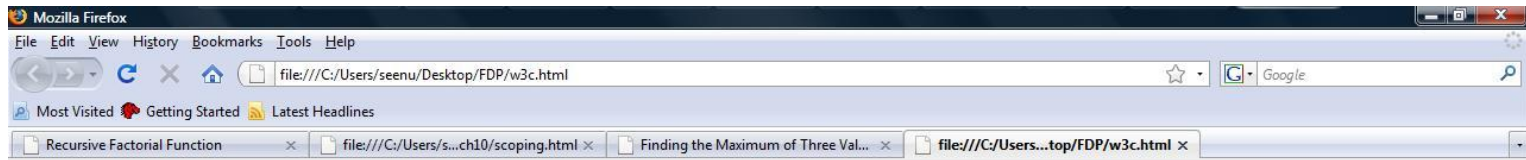
The example below shows how to use JavaScript to write text on a web page:

The HTML `<script>` tag is used to insert a JavaScript into an HTML page.

Example

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

Introduction



Hello World!



Introduction

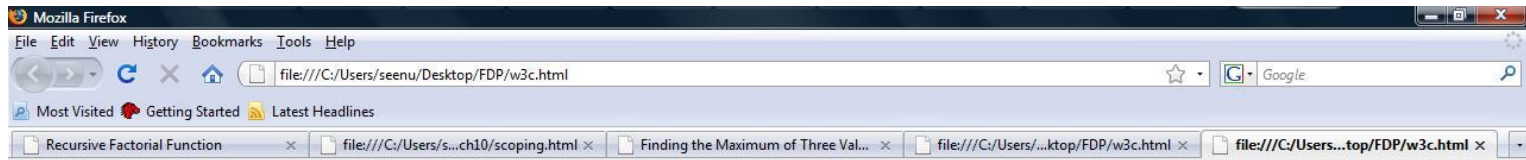
The example below shows how to add HTML tags to the JavaScript:

Example

```
<html>
<body>
<script type="text/javascript">
document.write("<h1>Hello World!</h1>");
</script>
</body>
</html>
```

- The **document.write** command is a standard JavaScript command for writing output to a page.
- JavaScripts in the body section will be executed WHILE the page loads.
- JavaScripts in the head section will be executed when CALLED.

Introduction



Hello World!



Introduction

Where to Put the JavaScript

JavaScripts in a page will be executed immediately while the page loads into the browser. This is not always what we want. Sometimes we want to execute a script when a page loads, other times when a user triggers an event.

Introduction

Scripts in <head>

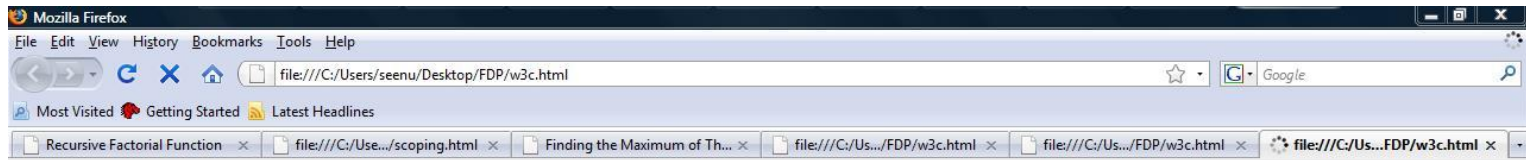
- Scripts to be executed when they are called, or when an event is triggered, go in the head section.
- If you place a script in the head section, you will ensure that the script is loaded before anyone uses it.

Example

```
<html>
<head>
<script type="text/javascript">
function message()
{
alert("This alert box was called with the onload event");
}
</script>
</head>

<body onload="message()">
</body>
</html>
```


Introduction



Introduction

Scripts in <body>

Scripts to be executed when the page loads go in the body section.

If you place a script in the body section, it generates the content of a page.

Example

```
<html>
```

```
<head>
```

```
</head>
```

```
<body>
```

```
Introduction<script type="text/javascript">
```

```
document.write("This message is written by  
JavaScript"); </script>
```

```
</body>
```

```
</html>
```

Introduction



Introduction

Scripts in <head> and <body>

You can place an unlimited number of scripts in your document, so you can have scripts in both the body and the head section.

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

Introduction

JavaScript Statements

JavaScript is a sequence of statements to be executed by the browser.

JavaScript is Case Sensitive

Unlike HTML, JavaScript is case sensitive - therefore watch your capitalization closely when you write JavaScript statements, create or call variables, objects and functions

The semicolon is optional (according to the JavaScript standard), and the browser is supposed to interpret the end of the line as the end of the statement. Because of this you will often see examples without the semicolon at the end.

Note: Using semicolons makes it possible to write multiple statements on one line.

Introduction

JavaScript Code

JavaScript code (or just JavaScript) is a sequence of JavaScript statements. Each statement is executed by the browser in the sequence they are written.

This example will write a heading and two paragraphs to a web page:

Example

```
<script type="text/javascript">
document.write("<h1>This is a heading</h1>");
document.write("<p>This is a paragraph.</p>");
document.write("<p>This is another
paragraph.</p>"); </script>
```

Introduction

JavaScript Blocks

JavaScript statements can be grouped together in blocks.

Blocks start with a left curly bracket {, and ends with a right curly bracket }.

The purpose of a block is to make the sequence of statements execute together.

This example will write a heading and two paragraphs to a web page:

Example

```
<script type="text/javascript">
{
document.write("<h1>This is a heading</h1>");
document.write("<p>This is a paragraph.</p>");
document.write("<p>This is another
paragraph.</p>"); }
</script>
```

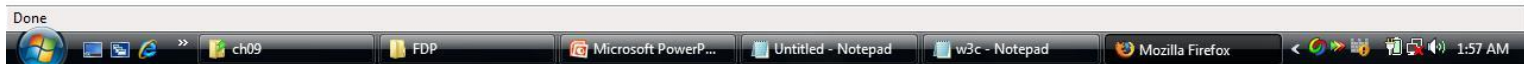
Introduction



This is a heading

This is a paragraph.

This is another paragraph.



Introduction

JavaScript Comments

Comments can be added to explain the JavaScript, or to make the code more readable.

Single line comments start with `//`.

The following example uses single line comments to explain the code:

Example

```
<script type="text/javascript">  
// Write a heading  
document.write("<h1>This is a heading</h1>");  
    Write two paragraphs: document.write("<p>This is  
a paragraph.</p>"); document.write("<p>This is  
another paragraph.</p>"); </script>
```

Introduction

JavaScript Multi-Line Comments

Multi line comments start with `/*` and end with `*/`.

The following example uses a multi line comment to explain the code:

Example

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

```
/*
```

```
The code below will write  
one heading and two paragraphs
```

```
*/
```

```
document.write("<h1>This is a heading</h1>");
```

```
document.write("<p>This is a paragraph.</p>");
```

```
document.write("<p>This is another  
paragraph.</p>"); </script>
```

Introduction

Using Comments at the End of a Line

In the following example the comment is placed at the end of a code line:

Example

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

Introduction

Additional problems

```
<html>
  <head><title> Printing Multiple Lines</title>

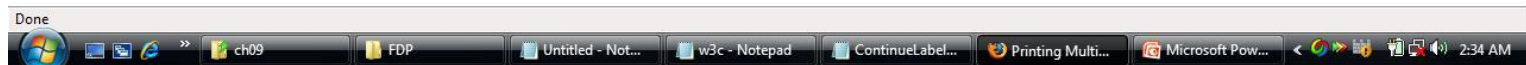
  <script type = "text/javascript">
    <!--
      document.writeln( "<h1>Welcome to<br/>JavaScript"
      + />Programming!</h1>" );
    </script>

  </head><body></body>
</html>
```

Introduction



**Welcome to
JavaScript
Programming!**



Introduction

JavaScript Variables

- As with algebra, JavaScript variables are used to hold values or expressions.
- A variable can have a short name, like `x`, or a more descriptive name, like `carname`.

Rules for JavaScript variable names:

- Variable names are case sensitive (`y` and `Y` are two different variables)
- Variable names must begin with a letter or the underscore character

Introduction

Declaring (Creating) JavaScript Variables

Creating variables in JavaScript is most often referred to as "declaring" variables.

You can declare JavaScript variables with the var statement:

```
var x;
```

```
var carname;
```

After the declaration shown above, the variables are empty (they have no values yet).

However, you can also assign values to the variables when you declare them:

```
var x=5;
```

```
var carname="Volvo";
```

Introduction

Assigning Values to Undeclared JavaScript Variables

If you assign values to variables that have not yet been declared, the variables will automatically be declared.

These statements:

```
x=5;
```

```
carname="Volvo";
```

Redeclaring JavaScript Variables

If you redeclare a JavaScript variable, it will not lose its original value.

```
var x=5;
```

```
var x;
```

After the execution of the statements above, the variable x will still have the value of 5. The value of x is not reset (or cleared) when you redeclare it.

Operators

JavaScript Arithmetic Operators

Arithmetic operators are used to perform arithmetic between variables and/or values.

JavaScript Assignment Operators

Assignment operators are used to assign values to JavaScript variables.

The + Operator Used on Strings

The + operator can also be used to add string variables or text values together.

Operators

Comparison Operators

Comparison operators are used in logical statements to determine equality or difference between variables or values.

Logical Operators

Logical operators are used to determine the logic between variables or values.

Arithmetic operators

```
<html>
  <head>
    <title>An Addition Program</title>

    <script type = "text/javascript">
      var firstNumber, // first string entered by user
          secondNumber, // second string entered by user
          number1,
          number2,
          sum;          // sum of number1 and number2

      read in first number from user as a
      string firstNumber =
        window.prompt( "Enter first integer", "0" );

      read in second number from user as a string
      secondNumber =
        window.prompt( "Enter second integer", "0" );
```

Operators

convert numbers from strings to integers

```
number1 = parseInt( firstNumber );  
number2 = parseInt( secondNumber );
```

```
// add the numbers  
sum = number1 + number2;
```

```
// display the results  
document.writeln( "<h1>The sum is " + sum + "</h1>" );  
</script>
```

```
</head>
```

```
<body>
```

```
<p>Click Refresh (or Reload) to run the script  
again</p> </body>
```

```
</html>
```

Operators



Operators



Operators



The sum is 30

Click Refresh (or Reload) to run the script again



Introduction-JavaScript Popup Boxes

Alert Box

An alert box is often used if you want to make sure information comes through to the user.

When an alert box pops up, the user will have to click "OK" to proceed.

```
<html>
```

```
<head><title>Printing Multiple Lines in a Dialog Box</title>
```

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

```
    <!--
```

```
      window.alert( "Welcome to\nJavaScript\nProgramming!" );
```

```
    -->
```

```
  </script>
```

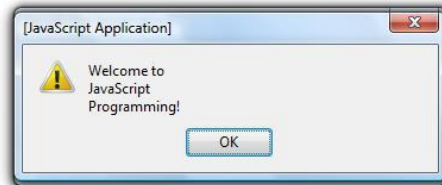
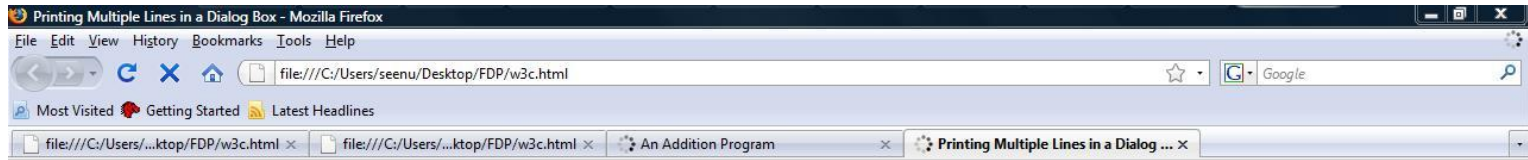
```
</head>
```

```
<body>
```

```
  <p>Click Refresh (or Reload) to run this script  
  again.</p> </body>
```

```
</html>
```


Introduction-JavaScript Popup Boxes



Introduction-JavaScript Popup Boxes

Confirm Box

A confirm box is often used if you want the user to verify or accept something.

When a confirm box pops up, the user will have to click either "OK" or "Cancel" to proceed.

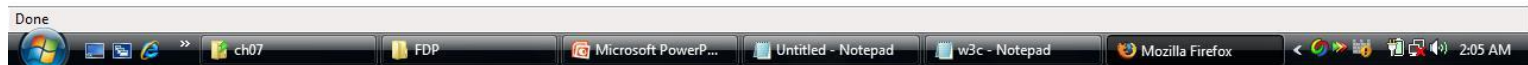
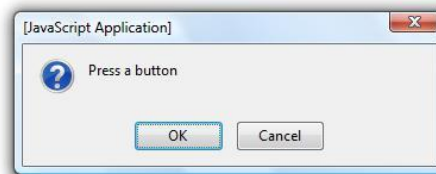
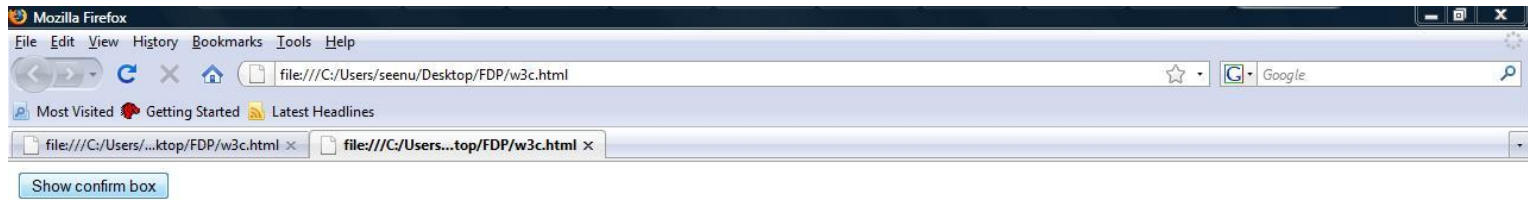
If the user clicks "OK", the box returns true. If the user clicks "Cancel", the box returns false.

Introduction-JavaScript Popup Boxes

Example

```
<html>
<head>
<script type="text/javascript">
function show_confirm()
{
var r=confirm("Press a button");
if (r==true)
{
document.write("You pressed OK!");
}
else
{
document.write("You pressed Cancel!");
}
}
</script>
</head>
<body>
<input type="button" onclick="show_confirm()" value="Show confirm box" />
</body>
</html>
```

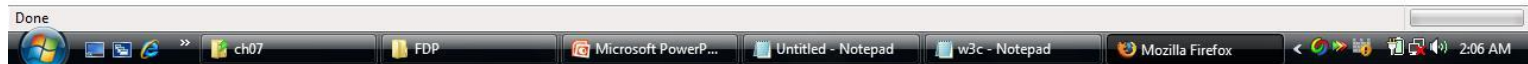
Introduction-JavaScript Popup Boxes



Introduction-JavaScript Popup Boxes



You pressed Cancel!



Introduction-JavaScript Popup Boxes

Prompt Box

A prompt box is often used if you want the user to input a value before entering a page.

When a prompt box pops up, the user will have to click either "OK" or "Cancel" to proceed after entering an input value.

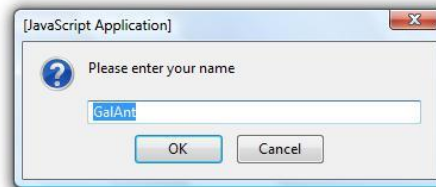
If the user clicks "OK" the box returns the input value. If the user clicks "Cancel" the box returns null.

```
<html >
  <head>
    <title>Using Prompt and Alert Boxes</title>
    <script type = "text/javascript">
      var name; // string entered by the user
      // read the name from the prompt box as a string
      name = window.prompt( "Please enter your name", "GalAnt"
    ); document.writeln( "<h1>Hello " + name +
      ", welcome to JavaScript programming!</h1>"
    ); </script>

  </head>

  <body>
    <p>Click Refresh (or Reload) to run this script
    again.</p> </body>
</html>
```

Introduction-JavaScript Popup Boxes



Introduction-JavaScript Popup Boxes



Hello GalAnt, welcome to JavaScript programming!

Click Refresh (or Reload) to run this script again.



Control structures-if

```
<html>
```

```
  <head>
```

```
    <title>Using Relational Operators</title>
```

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

```
      var name;
```

```
      now = new Date(); // current date and time
```

```
      hour = now.getHours(); // current hour (0-23)
```

```
      // read the name from the prompt box as a string
```

```
      name = window.prompt( "Please enter your name", "GalAnt" );
```

Control structures-if

```
if ( hour < 12 )
    document.write( "<h1>Good Morning, " );

    determine whether the time is
    PM if ( hour >= 12 )
    {
        hour = hour - 12;

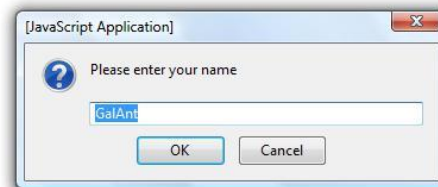
        if ( hour < 6 )
            document.write( "<h1>Good Afternoon, " );

        if ( hour >= 6 )
            document.write( "<h1>Good Evening, " );
    }

    document.writeln( name +
        ", welcome to JavaScript programming!</h1>"
    ); </script> </head>

<body> <p>Click Refresh (or Reload) to run this script again.</p> </body>
</html>
```

Control structures-if

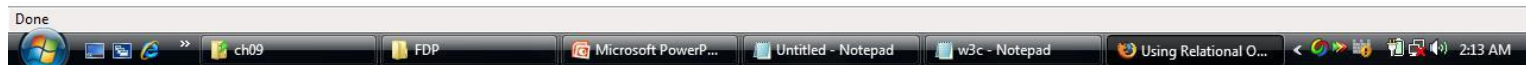


Control structures-if



Good Morning, GalAnt, welcome to JavaScript programming!

Click Refresh (or Reload) to run this script again.



Control structures-Switch

```
<script type="text/javascript">Statement //You will
  receive a different greeting based
  //on what day it is. Note that Sunday=0,
  //Monday=1, Tuesday=2, etc.

  var d=new Date();
  theDay=d.getDay();
  switch (theDay)
  {
  case 5:
    document.write("Finally Friday");
    break;
  case 6:
    document.write("Super Saturday");
    break;
  case 0:
    document.write("Sleepy Sunday");
    break;
  default:
    document.write("I'm looking forward to this weekend!");
  }
</script>
```

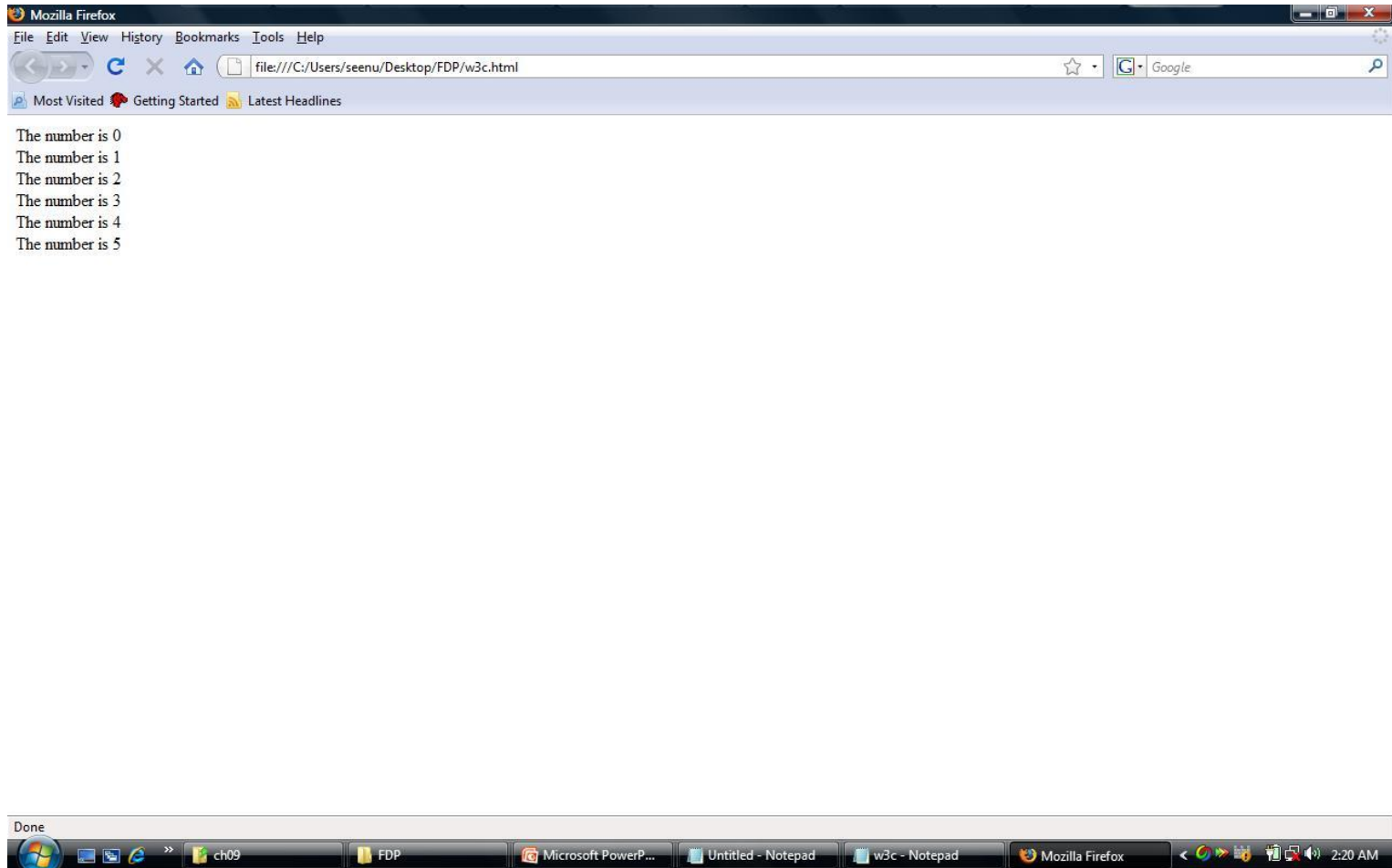
JavaScript Loops

for Loop

The for loop is used when you know in advance how many times the script should run.

```
<html>
<body>
<script type="text/javascript">
var i=0;
for (i=0;i<=5;i++)
{
document.write("The number is " +
i); document.write("<br />");
}
</script>
</body>
</html>
```

JavaScript Loops



JavaScript Loops

JavaScript While Loop

oops execute a block of code a specified number of times, or while a specified condition is true.

Example

```
<html>
<body>
<script type="text/javascript">
var i=0;
while (i<=5)
{
  document.write("The number is " +
  i); document.write("<br />"); i++;
}
</script>
</body>
</html>
```

JavaScript Loops

The do...while Loop

This loop will execute the block of code ONCE, and then it will repeat the loop as long as the specified condition is true.

Example

```
<html>
<body>
<script type="text/javascript">
var i=0;
do
{
  document.write("The number is " + i);
  document.write("<br />");
  i++;
}
while (i<=5);
</script>
</body>
</html>
```

JavaScript Break

The break Statement

The break statement will break the loop and continue executing the code that follows after the loop (if any).

```
<html>
<head>

<script type = "text/javascript">
  <!--
  for ( var count = 1; count <= 10; ++count ) {
    if ( count == 5 )
      break; // break loop only if count == 5

    document.writeln( "Count is: " + count + "<br />" );
  }

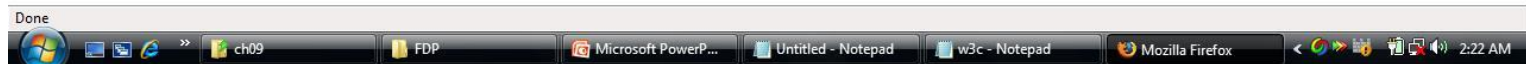
  document.writeln(
    "Broke out of loop at count = " + count );
  -->
</script>

</head><body></body>
</html>
```

JavaScript Break



Count is: 1
Count is: 2
Count is: 3
Count is: 4
Broke out of loop at count = 5



Continue Statements

The continue Statement

The continue statement will break the current loop and continue with the next value

```
<html>
<head>
  <script type = "text/javascript">
    for ( var count = 1; count <= 10; ++count )
    { if ( count == 5 )
      continue; // skip remaining code in loop
        // only if count == 5

      document.writeln( "Count is: " + count + "<br />" );
    }
    document.writeln( "Used continue to skip printing 5"
); </script>
  </head><body></body>
</html>
```

Continue Statements



Count is: 1
Count is: 2
Count is: 3
Count is: 4
Count is: 6
Count is: 7
Count is: 8
Count is: 9
Count is: 10
Used continue to skip printing 5

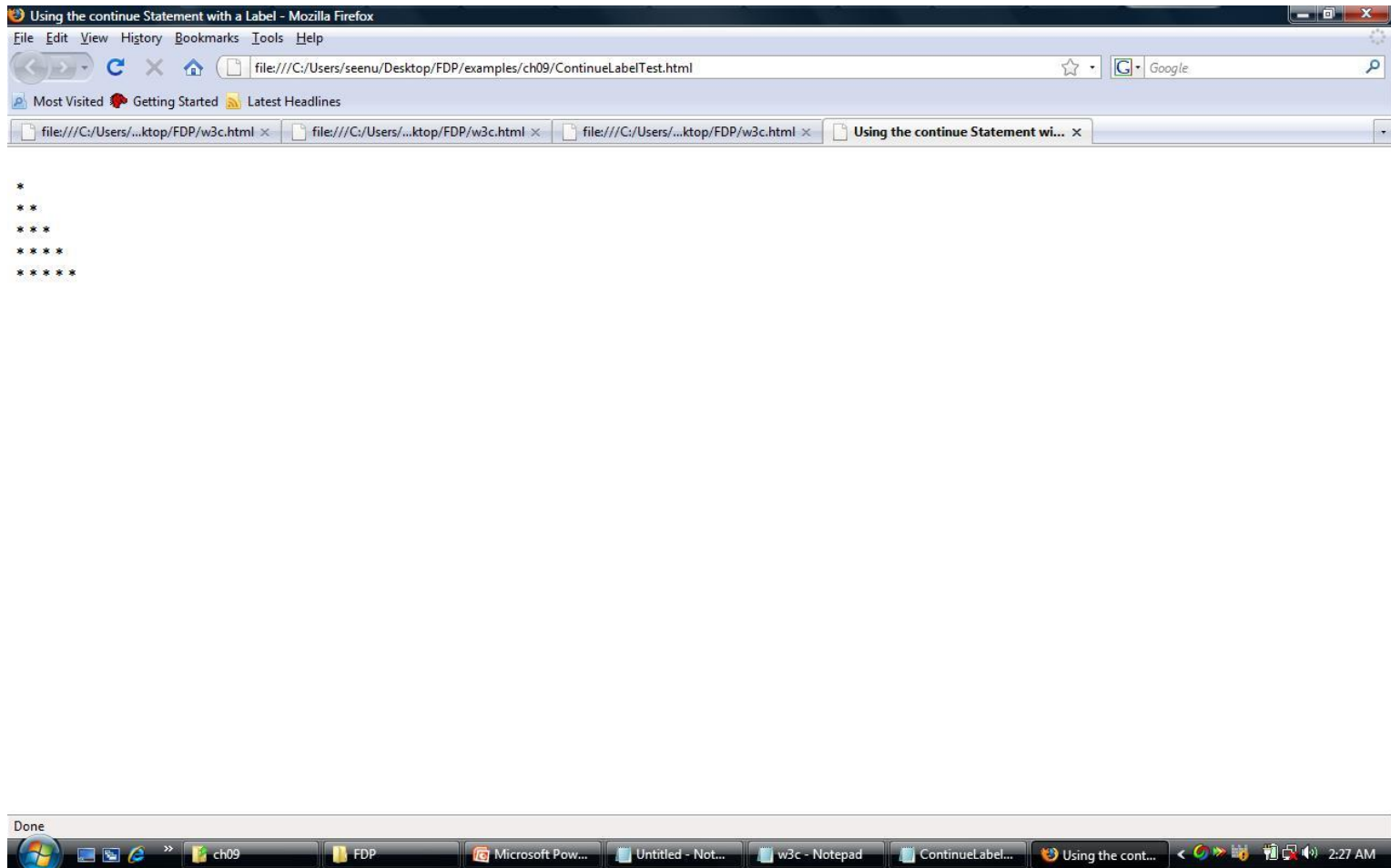


The labeled continue Statement

```
<html >
  <head>
    <title>Using the continue Statement with a
    Label</title> <script type = "text/javascript">
      nextRow: // target label of continue statement
      for ( var row = 1; row <= 5; ++row ) {
        document.writeln( "<br />" );
        for ( var column = 1; column <= 10; ++column ) {

          if ( column > row )
            continue nextRow; // next iteration of labeled
            loop document.write( "*" );
        }
      }
    </script>
  </head><body></body>
</html>
```

The labeled continue Statement



The labeled break Statement

```
<html>
  <head>
    <script type = "text/javascript">
      stop: { // labeled block
        for ( var row = 1; row <= 10; ++row ) {
          for ( var column = 1; column <= 5 ; ++column ) {

            if ( row == 5 )
              break stop; // jump to end of stop block

            document.write( "* " );
          }

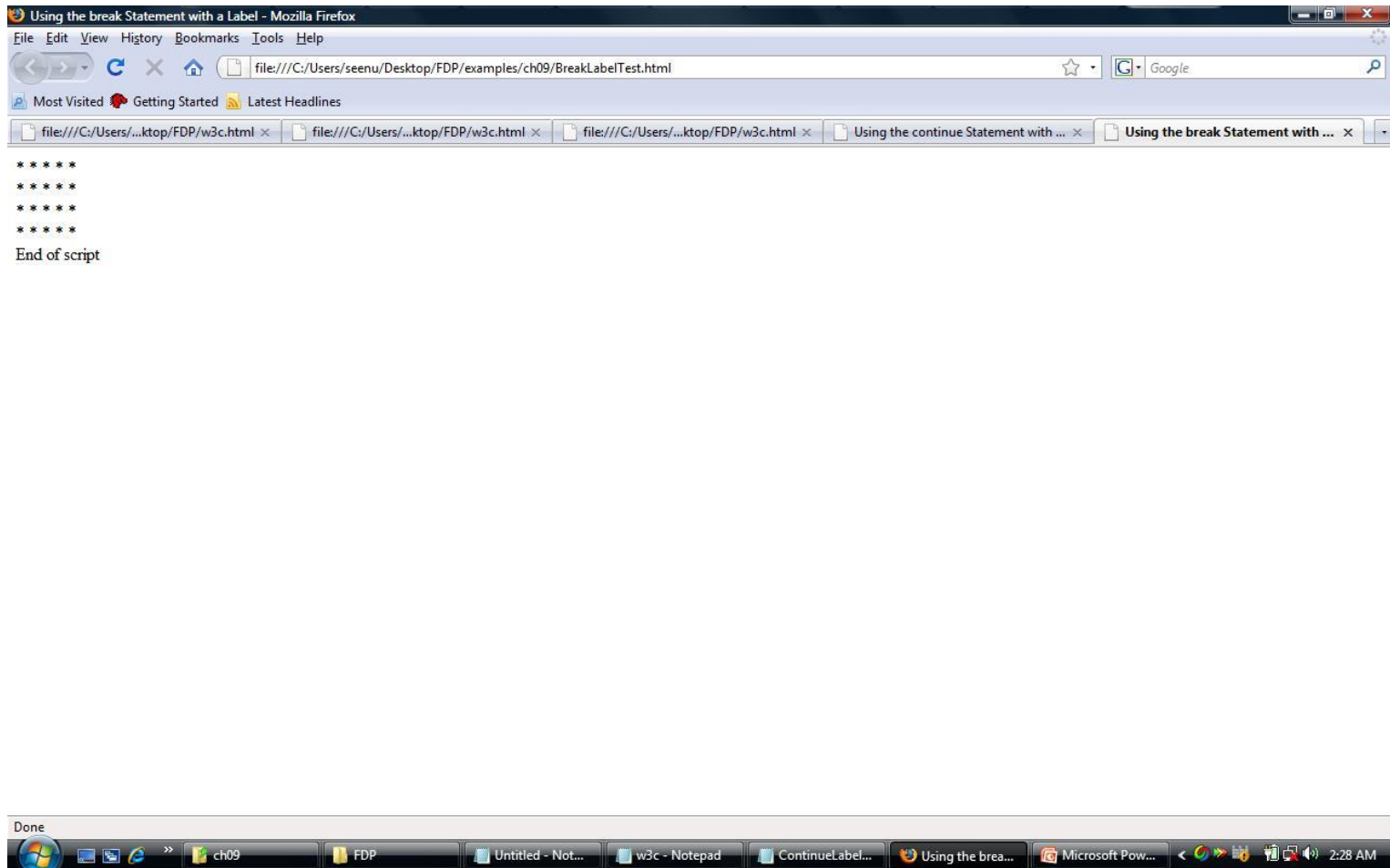
          document.writeln( "<br />" );
        }

        the following line is skipped document.writeln(
        "This line should not print" );
      }

      document.writeln( "End of script" );
    </script>

    </head><body></body>
  </html>
```

The labeled break Statement



JavaScript Try...Catch Statement

The try...catch statement allows you to test a block of code for errors.

JavaScript - Catching Errors

When browsing Web pages on the internet, we all have seen a JavaScript alert box telling us there is a runtime error and asking "Do you wish to debug?". Error message like this may be useful for developers but not for users. When users see errors, they often leave the Web page.

This chapter will teach you how to catch and handle JavaScript error messages, so you don't lose your audience.

JavaScript Try...Catch Statement

Example

```
<html>
  <head>
    <script type="text/javascript">
      var txt="";
      function message()
      {
        try
        {
          addlert("Welcome guest!");
        }
        catch(err)
        {
          txt="There was an error on this page.\n\n";
          txt+="Error description: " + err.description +
            "\n\n"; txt+="Click OK to continue.\n\n";
          alert(txt);
        }
      }
    </script></head>

    <body>
      <input type="button" value="View message" onclick="message()"
      /> </body></html>
```

The Throw Statement

The throw statement allows you to create an exception. If you use this statement together with the try...catch statement, you can control program flow and generate accurate error messages.

The Throw Statement

Example

```
<html>
<body>
<script type="text/javascript">
var x=prompt("Enter a number between 0 and 10:", "");
try
{
  if(x>10)
  {
    throw "Err1";
  }
  else if(x<0)
  {
    throw "Err2";
  }
}
catch(er)
{
  if(er=="Err1 ")
  {
    alert("Error! The value is too high");
  }
  if(er=="Err2")
  {
    alert("Error! The value is too low");
  }
}
</script>
</body>
</html>
```

JavaScript Functions

A function will be executed by an event or by a call to the function.

Functions can be defined both in the <head> and in the <body> section of a document.

However, to assure that a function is read/loaded by the browser before it is called, it could be wise to put functions in the <head> section.

JavaScript Functions

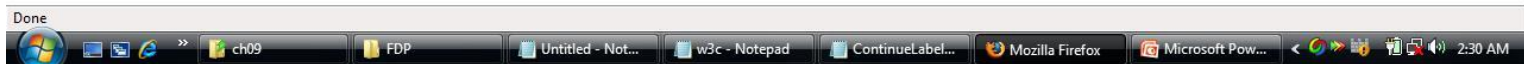
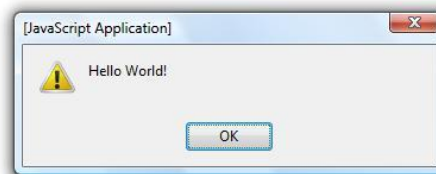
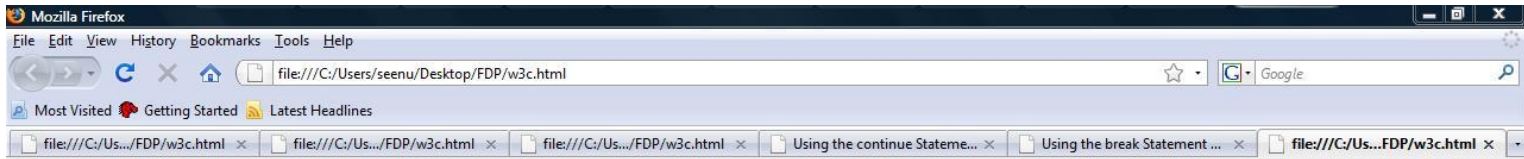
JavaScript Function Example

Example

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

<body>
<form>
<input type="button" value="Click me!" onclick="displaymessage()" />
</form>
</body>
</html>
```


JavaScript Functions



JavaScript Functions

The return Statement

The return statement is used to specify the value that is returned from the function.

Example

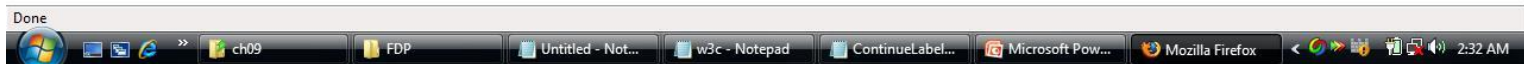
```
<html>
<head>
<script type="text/javascript">
function product(a,b)
{
return a*b;
}
</script>
</head>

<body>
<script type="text/javascript">
document.write(product(4,3));
</script>
</body>
</html>
```

JavaScript Functions



12



JavaScript Functions- another example

```
<head>
  <script type = "text/javascript">
    var input1 = window.prompt( "Enter first number", "0" );
    var input2 = window.prompt( "Enter second number", "0" );
    var input3 = window.prompt( "Enter third number", "0" );

    var value1 = parseFloat( input1 );
    var value2 = parseFloat( input2 );
    var value3 = parseFloat( input3 );

    var maxValue = maximum( value1, value2, value3 );

    document.writeln( "First number: " + value1 +
      "<br />Second number: " + value2 +
      "<br />Third number: " + value3 +
      "<br />Maximum is: " + maxValue );
```

JavaScript Functions

maximum method definition (called from line 25)

```
function maximum( x, y, z )  
{  
    return Math.max( x, Math.max( y, z ) );  
}
```

```
</script>
```

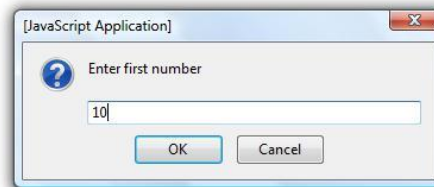
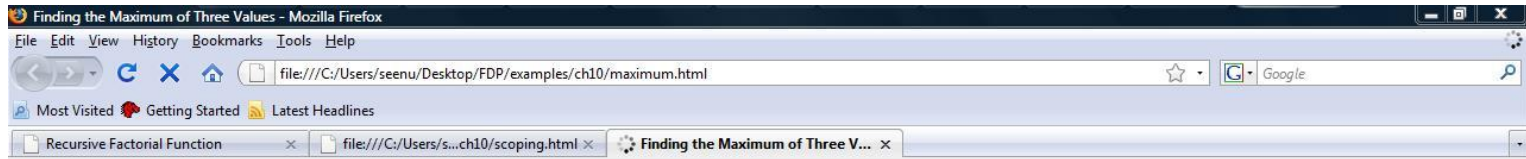
```
</head>
```

```
<body>
```

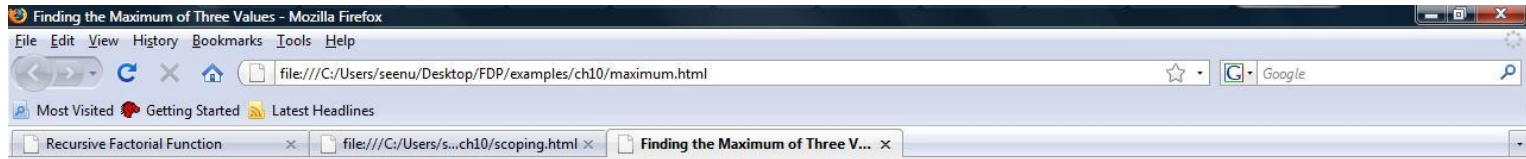
```
    <p>Click Refresh (or Reload) to run the script  
again</p> </body>
```

```
</html>
```

JavaScript Functions

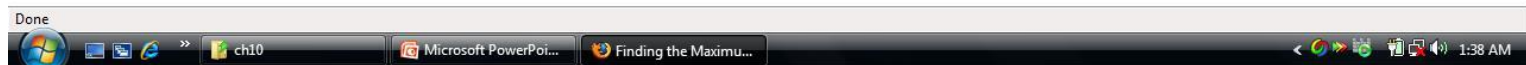


JavaScript Functions



First number: 10
Second number: 20
Third number: 5
Maximum is: 20

Click Refresh (or Reload) to run the script again



JavaScript Functions - factorial

```
<html >
  <head>
    <script type = "text/javascript">
      document.writeln("<table border = '1' width = '100%'"
    ); for ( var i = 0; i <= 10; i++ )
      document.writeln( "<tr><td>"+i+"!</td><td>"+
factorial(i)+"</td></tr>" );
      document.writeln( "</table>" );

      function factorial( number )
      {
        if ( number <= 1 ) // base case
          return 1;
        else
          return number * factorial( number - 1 );
      }
    </script> </head><body></body>
</html>
```


JavaScript Functions - factorial



Factorials of 1 to 10

0!	1
1!	1
2!	2
3!	6
4!	24
5!	120
6!	720
7!	5040
8!	40320
9!	362880
10!	3628800



JavaScript Functions-scope

```
<html >
  <head>
    <title>A Scoping Example</title>

    <script type = "text/javascript">
      var x = 1;    // global variable

      function start()
      {
        var x = 5; // variable local to function start

        document.writeln( "local x in start is " + x );
        functionA(); // functionA has local x
        functionB(); // functionB uses global variable x
        functionA(); // functionA reinitializes local x
        functionB(); // global variable x retains its value

        document.writeln(
          "<p>local x in start is " + x + "</p>" );
      }
    </script>
  </head>
</html >
```

JavaScript Functions-scope

```
function functionA()
{ var x = 25; // initialized each time
  document.writeln( "<p>local x in functionA is " +
    x + " after entering functionA" );
  ++x;
  document.writeln( "<br />local x in functionA is "
    + x + " before exiting functionA" + "</p>" ); }
function functionB()
{document.writeln( "<p>global variable x is " + x +
  " on entering functionB" );
  x *= 10;
  document.writeln( "<br />global variable x is "
    x + " on exiting functionB"
    "</p>" ); }
</script>
</head> <body onload = "start()"></body></html>
```

JavaScript Functions-scope

```
local x in start is 5

local x in functionA is 25 after entering functionA
local x in functionA is 26 before exiting functionA

global variable x is 1 on entering functionB
global variable x is 10 on exiting functionB

local x in functionA is 25 after entering functionA
local x in functionA is 26 before exiting functionA

global variable x is 10 on entering functionB
global variable x is 100 on exiting functionB

local x in start is 5
```

UNIT-II

Objects in JAVASCRIPT and XML

Array Object

An array is a special variable, which can hold more than one value, at a time.

```
<html>
  <head>
    <script type = "text/javascript">
      function initializeArrays()
      {
        var n1 = new Array( 5 ); // allocate 5-element Array
        var n2 = new Array(); // allocate empty Array

          for ( var i = 0; i < n1.length; ++i )
            n1[ i ] = i;
          for ( i = 0; i < 5; ++i )
            n2[ i ] = i;

        outputArray( "Array n1 contains", n1 );
        outputArray( "Array n2 contains", n2 );
      }
    </script>
  </head>
</html>
```

Array Object

```
function outputArray( header, theArray )
{
  document.writeln( "<h2>" + header + "</h2>" );
  document.writeln( "<table border = \"1\" width = " +           "\"100%\">" );
  document.writeln( "<thead><th width = \"100\"\" +
  \"align = \"left\">Subscript</th>" +
  "<th align = \"left\">Value</th></thead><tbody>" );
  for ( var i = 0; i < theArray.length; i++ )
  document.writeln( "<tr><td>" + i + "</td><td>" +
  theArray[ i ] + "</td></tr>" );
  document.writeln( "</tbody></table>" );
}
</script>
</head><body onload =
"initializeArrays()"></body> </html>
```

Array Object



Array n1 contains

Subscript	Value
0	0
1	1
2	2
3	3
4	4

Array n2 contains

Subscript	Value
0	0
1	1
2	2
3	3
4	4



Array Object

```
<html>
```

```
<head>
```

```
<title>Initializing an Array with a  
Declaration</title> <script type = "text/javascript">
```

```
function start()
```

```
{
```

```
    Initializer list specifies number of elements and  
    value for each element.
```

```
var colors = new Array( "cyan", "magenta",  
    "yellow", "black" );
```

```
    var integers1 = [ 2, 4, 6, 8 ];
```

```
var integers2 = [ 2, , , 8 ];
```

```
outputArray( "Array colors contains", colors );
```

```
    outputArray( "Array integers1 contains", integers1 );
```

```
outputArray( "Array integers2 contains", integers2 );
```

```
}
```

Array Object

```
function outputArray( header, theArray )
{
    document.writeln( "<h2>" + header + "</h2>" );
    document.writeln( "<table border = \"1\" \"
        \"width = \"100%\">" );
    document.writeln( "<thead><th width = \"100\" \"
        + \"align = \"left\">Subscript</th>\" +
        \"<th align = \"left\">Value</th></thead><tbody>" );
    for ( var i = 0; i < theArray.length; i++ )
        document.writeln( "<tr><td>" + i + "</td><td>" +
            theArray[ i ] + "</td></tr>" );
    document.writeln( "</tbody></table>" );
}
</script>
```

```
</head><body onload = "start()"></body>
</html>
```

Array Object



Array colors contains

Subscript	Value
0	cyan
1	magenta
2	yellow
3	black

Array integers1 contains

Subscript	Value
0	2
1	4
2	6
3	8

Array integers2 contains

Subscript	Value
0	2
1	undefined
2	undefined
3	8



Array Object

```
<html >
  <head>
    <title>Initializing Multidimensional Arrays</title>

    <script type = "text/javascript">
      function start()
      {
        var array1 = [ [ 1, 2, 3 ], // first row
                      [ 4, 5, 6 ] ]; // second row
        var array2 = [ [ 1, 2 ], // first row
                      [ 3 ], // second row
                      [ 4, 5, 6 ] ]; // third row

        outputArray( "Values in array1 by row", array1 );
        outputArray( "Values in array2 by row", array2 );
      }
    </script>
  </head>
</html >
```

Array Object

```
function outputArray( header, theArray )
{
    document.writeln( "<h2>" + header + "</h2><tt>" );

    for ( var i in theArray ) {

        for ( var j in theArray[ i ] )
            document.write( theArray[ i ][ j ] + " " );
        document.writeln( "<br />" );
    }

}
</script>

</head><body onload = "start()"></body>
</html>
```

Array Object



Values in array1 by row

1 2 3
4 5 6

Values in array2 by row

1 2
3
4 5 6



Array Object

```
<html>
  <head>
    <title>Sum the Elements of an Array</title>
    <script type = "text/javascript">
      function start()
      {
        var theArray = [ 1, 2, 3, 4, 5, 6, 7, 8, 9, 10 ];
        var total1 = 0, total2 = 0;
        for ( var i = 0; i < theArray.length; i++ )
          total1 += theArray[ i ];

          document.writeln( "Total using subscripts: " + total1
        ); for ( var element in theArray )
          total2 += theArray[ element ]; document.writeln(
          "<br />Total using for...in: " +
          total2 );
      }
    </script> </head><body onload = "start()"></body></html>
```

Array Object



Total using subscripts: 55
Total using for...in: 55



Array Object

```
<html>
  <head>
    <title>Passing Arrays and Individual Array
      Elements to Functions</title>
    <script type = "text/javascript">
      function start()
      { var a = [ 1, 2, 3, 4, 5 ];
        document.writeln( "<h2>Effects of passing entire "
          + "array by reference</h2>" );
        outputArray( "The values of the original array are: ", a );
        modifyArray( a ); // array a passed by reference
        outputArray( "The values of the modified array are: ", a );
        document.writeln( "<h2>Effects of passing array " +
          "element by value</h2>" + "a[3] before modifyElement: " + a[ 3 ]
          ); modifyElement( a[ 3 ] );
        document.writeln("<br />a[3] after modifyElement: " + a[ 3 ] );
      }
    </script>
  </head>
</html>
```

Array Object

```
function outputArray( header, theArray )
{
    document.writeln( header + theArray.join( " " ) + "<br />" );
}
function modifyArray( theArray )
{
    for ( var j in theArray )
        theArray[ j ] *= 2;
}
function modifyElement( e )
{
    e *= 2;
    document.writeln( "<br />value in modifyElement: " + e );
}
</script>
</head><body onload = "start()"></body>
</html>
```

Array Object

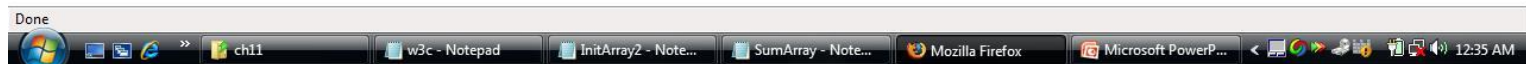


Effects of passing entire array by reference

The values of the original array are: 1 2 3 4 5
The values of the modified array are: 2 4 6 8 10

Effects of passing array element by value

a[3] before modifyElement: 8
value in modifyElement: 16
a[3] after modifyElement: 8



Array Object

```
<html >
  <head>
    <title>Sorting an Array with Array Method sort</title>

    <script type = "text/javascript">

      function start()
      {
        var a = [ 10, 1, 9, 2, 8, 3, 7, 4, 6, 5 ];

        document.writeln( "<h1>Sorting an Array</h1>" );
        outputArray( "Data items in original order: ", a );
        a.sort( compareIntegers ); // sort the array
        outputArray( "Data items in ascending order: ", a );
      }
    </script>
  </head>
</html >
```

Array Object

```
// outputs "header" followed by the contents of "theArray"
```

```
function outputArray( header, theArray )  
{  
    document.writeln( "<p>" + header +  
        theArray.join( " " ) + "</p>" );  
}
```

```
    comparison function for use with sort
```

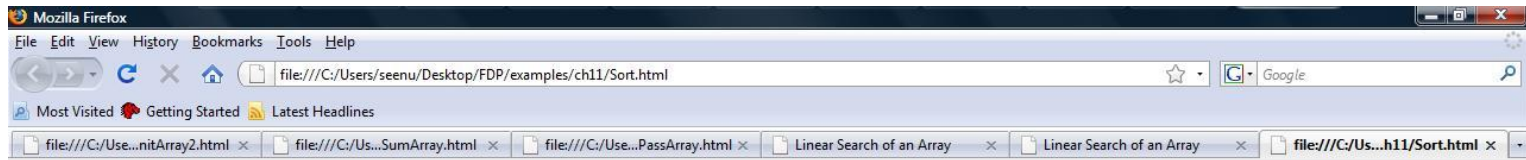
```
function compareIntegers( value1, value2 )  
{  
    return parseInt( value1 ) - parseInt( value2 );  
}
```

```
</script>
```

```
</head><body onload = "start()"></body>
```

```
</html>
```

Array Object



Sorting an Array

Data items in original order: 10 1 9 2 8 3 7 4 6 5

Data items in ascending order: 1 2 3 4 5 6 7 8 9 10

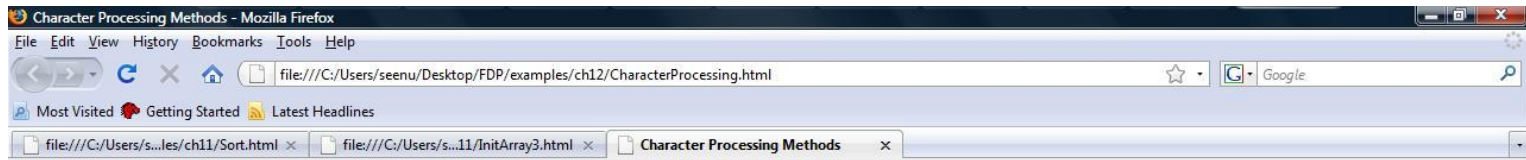


String object

```
<html >
  <head>
    <title>Character Processing Methods</title>
    <script type = "text/javascript">
      var s = "ZEBRA";
      var s2 = "AbCdEfG";

      document.writeln( "<p>Character at index 0 in '"
        + s + "' is " + s.charAt( 0 ) );
      document.writeln( "<br />Character code at index 0 in '"
        + s + "' is " + s.charCodeAt( 0 ) + "</p>" );
      document.writeln( "<p>" + String.fromCharCode( 87, 79, 82, 68 )
        + "' contains character codes 87, 79, 82 and 68</p>" )
      document.writeln( "<p>" + s2 + "' in lowercase is '"
        + s2.toLowerCase() + "'" );
      document.writeln( "<br />" + s2 + "' in uppercase is '"
        + s2.toUpperCase() + "'</p>" );
    </script> </head><body></body></html>
```

String object



Character at index 0 in 'ZEBRA' is Z
Character code at index 0 in 'ZEBRA' is 90

'WORD' contains character codes 87, 79, 82 and 68

'AbCdEfG' in lowercase is 'abcdefg'
'AbCdEfG' in uppercase is 'ABCDEFGG'

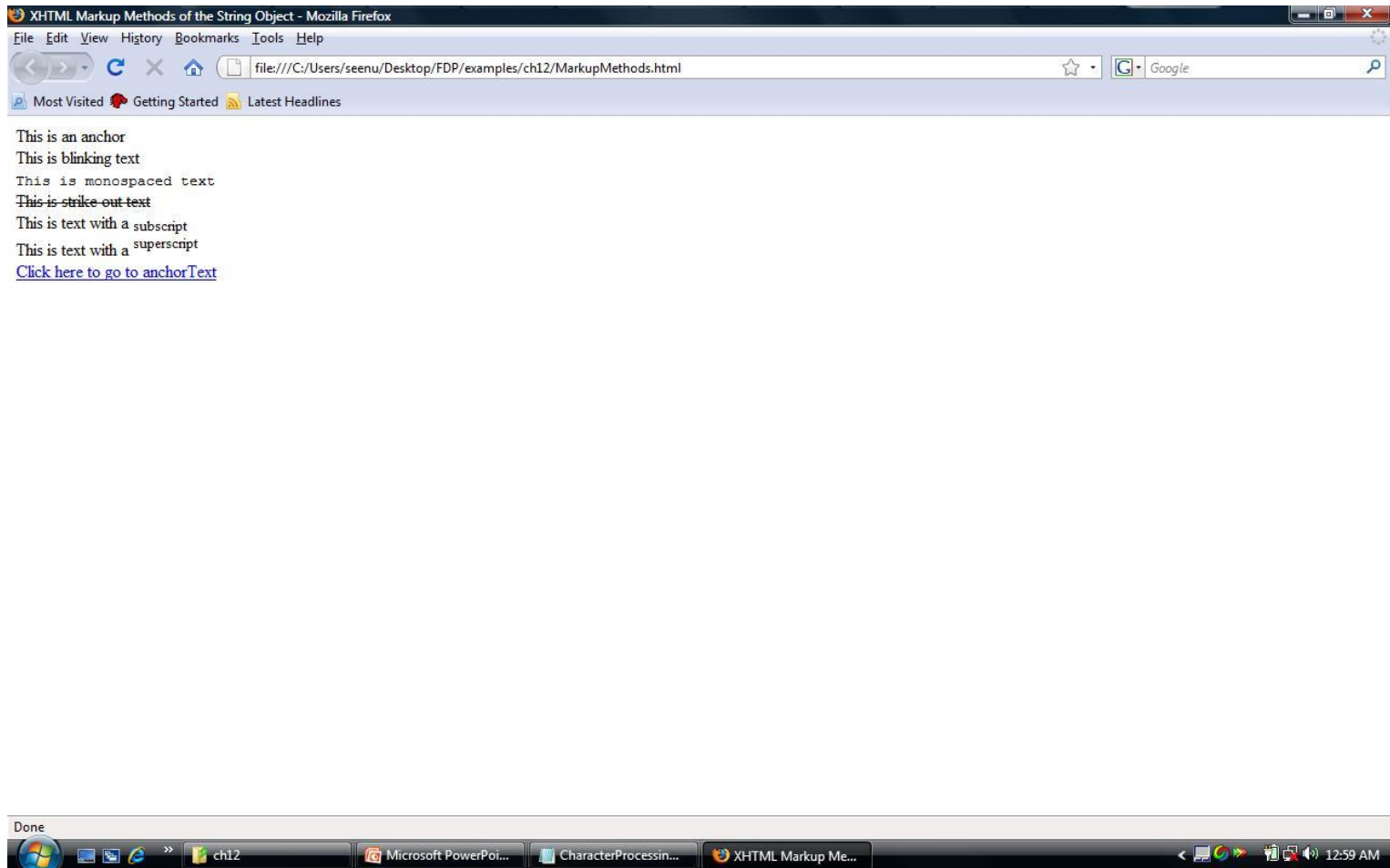


String object

```
<html >
```

```
<head><title>XHTML Markup Methods of the String  
Object</title> <script type = "text/javascript">  
    var anchorText = "This is an anchor",  
        blinkText = "This is blinking text",  
        fixedText = "This is monospaced text",  
        linkText = "Click here to go to anchorText",  
        strikeText = "This is strike out text",  
        subText = "subscript", supText = "superscript";  
    document.writeln( anchorText.anchor( "top" ) );  
    document.writeln( "<br />" + blinkText.blink() );  
    document.writeln( "<br />" + fixedText.fixed() );  
    document.writeln( "<br />" + strikeText.strike() );  
    document.writeln("<br />This is text with a " + subText.sub() );  
    document.writeln( "<br />This is text with a " + supText.sup() );  
    document.writeln( "<br />" + linkText.link( "#top" ) ); </script>  
</head><body></body></html>
```

String object



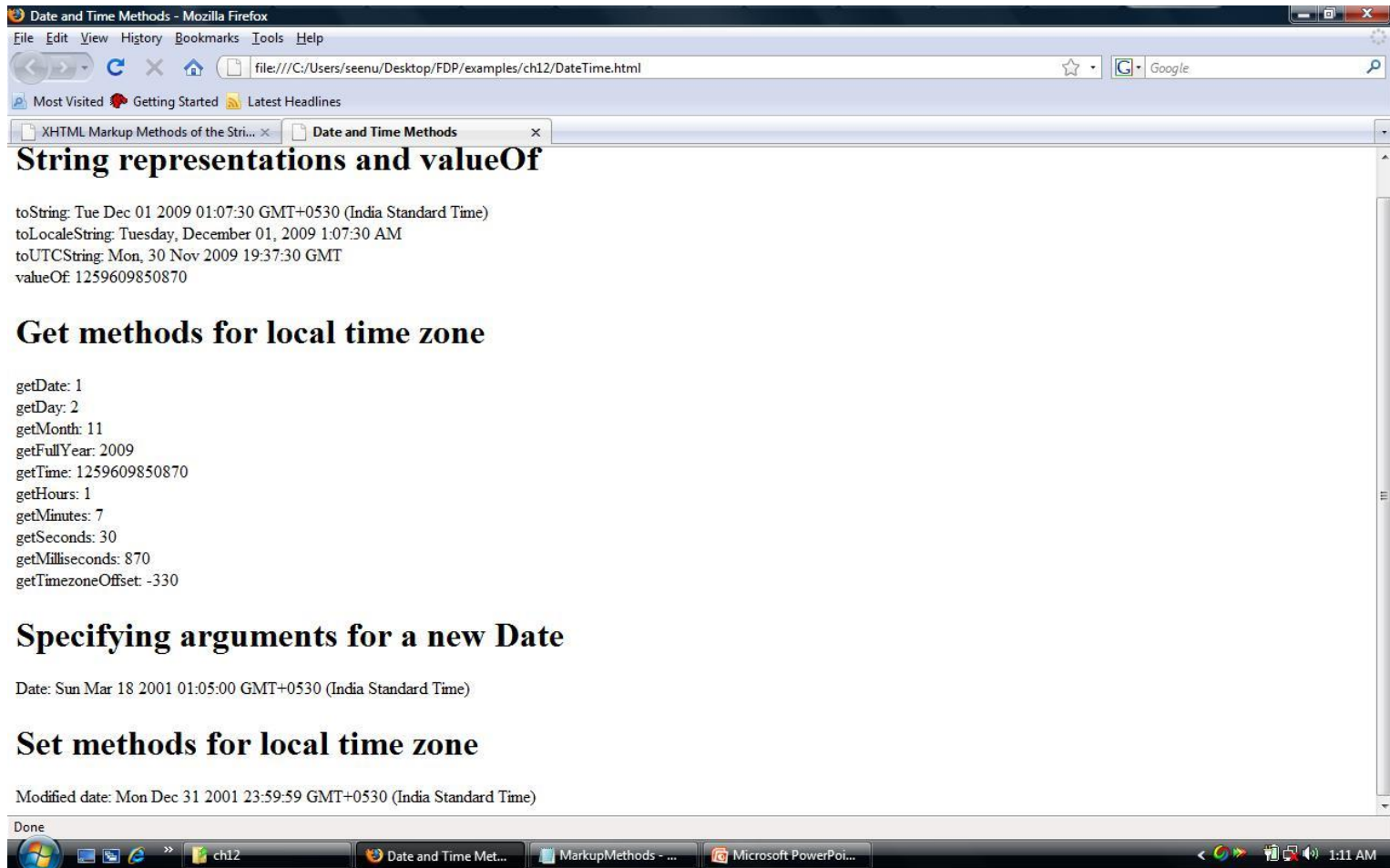
Date object

```
<html >
<head><title>Date and Time Methods</title>
<script type = "text/javascript">
  var current = new Date();
  document.writeln(
    "<h1>String representations and valueOf</h1>" );
  document.writeln( "toString: " + current.toString() +
    "<br />toLocaleString: " + current.toLocaleString() +
    "<br />toUTCString: " + current.toUTCString() +
    "<br />valueOf: " + current.valueOf() );
  document.writeln("<h1>Get methods for local time zone</h1>" );
  document.writeln( "getDate: " + current.getDate() +
    "<br />getDay: " + current.getDay() +
    "<br />getMonth: " + current.getMonth() +
    "<br />getFullYear: " + current.getFullYear() +
    "<br />getTime: " + current.getTime() +
    "<br />getHours: " + current.getHours() +
```

Date object

```
"<br />getMinutes: " + current.getMinutes() +
    "<br />getSeconds: " + current.getSeconds() + "<br
    />getMilliseconds: " + current.getMilliseconds() +
    "<br />getTimezoneOffset: " +
    current.getTimezoneOffset() ); document.writeln(
    "<h1>Specifying arguments for a new Date</h1>" );
var anotherDate = new Date( 2001, 2, 18, 1, 5, 0, 0 );
document.writeln( "Date: " + anotherDate );
document.writeln("<h1>Set methods for local time zone</h1>"
); anotherDate.setDate( 31 );
anotherDate.setMonth( 11 );    anotherDate.setFullYear( 2001 );
anotherDate.setHours( 23 );    anotherDate.setMinutes( 59 );
anotherDate.setSeconds( 59 );
document.writeln( "Modified date: " + anotherDate
); </script> </head><body></body></html>
```

Date object



Date and Time Methods - Mozilla Firefox

File Edit View History Bookmarks Tools Help

file:///C:/Users/seenu/Desktop/FDP/examples/ch12/DateTime.html

Most Visited Getting Started Latest Headlines

XHTML Markup Methods of the Stri... Date and Time Methods

String representations and valueOf

toString: Tue Dec 01 2009 01:07:30 GMT+0530 (India Standard Time)
toLocaleString: Tuesday, December 01, 2009 1:07:30 AM
toUTCString: Mon, 30 Nov 2009 19:37:30 GMT
valueOf: 1259609850870

Get methods for local time zone

getDate: 1
getDay: 2
getMonth: 11
getFullYear: 2009
getTime: 1259609850870
getHours: 1
getMinutes: 7
getSeconds: 30
getMilliseconds: 870
getTimezoneOffset: -330

Specifying arguments for a new Date

Date: Sun Mar 18 2001 01:05:00 GMT+0530 (India Standard Time)

Set methods for local time zone

Modified date: Mon Dec 31 2001 23:59:59 GMT+0530 (India Standard Time)

Done

ch12 Date and Time Met... MarkupMethods - ... Microsoft PowerPoi...

1:11 AM

Compare Two Dates

The Date object is also used to compare two dates.

The following example compares today's date with the 14th January 2010:

```
var myDate=new Date();
    myDate.setFullYear(2010,0,14);
    var today = new Date();

    if (myDate>today)
    {
        alert("Today is before 14th January 2010");
    }
    else
    {
        alert("Today is after 14th January 2010");
    }
```

Document object

```
<html >
  <head>
    <title>Using Cookies</title>
    <script type = "text/javascript">
      var now = new Date(); // current date and time
      var hour = now.getHours(); // current hour (0-
      23) var name;
      if ( hour < 12 ) // determine whether it is morning
        document.write( "<h1>Good Morning, " );
      else
      { hour = hour - 12; // convert from 24 hour clock to PM time
        determine whether it is afternoon or
        evening if ( hour < 6 )
          document.write( "<h1>Good Afternoon, "
          ); else
            document.write( "<h1>Good Evening, " );
      }
    }
  }
</script>
</head>
</html >
```

Document object

determine whether there is a

```
cookie if ( document.cookie )
```

```
{// convert escape characters in the cookie string to their
```

```
    english notation
```

```
    var myCookie = unescape( document.cookie );
```

```
        split the cookie into tokens using = as delimiter
```

```
    var cookieTokens = myCookie.split( "=" );
```

```
        set name to the part of the cookie that follows the =
```

```
    sign name = cookieTokens[ 1 ];
```

```
}
```

```
else
```

```
{ // if there was no cookie then ask the user to input a name name
```

```
    = window.prompt( "Please enter your name", "GalAnt" );
```

```
        escape non-alphanumeric characters in the name string
```

```
            and add name to the cookie
```

```
    document.cookie = "name=" + escape( name );
```

```
}
```

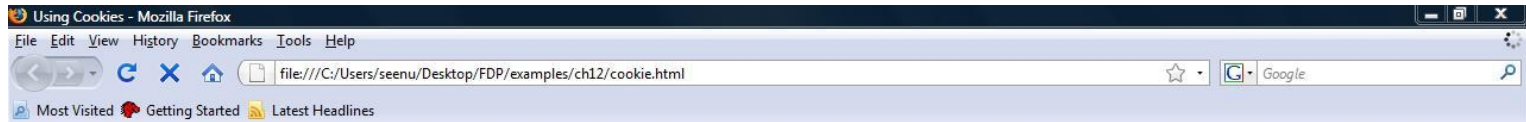

Document object

```
document.writeln(
    name + ", welcome to JavaScript programming! </h1>" );
document.writeln( "<a href= \" JavaScript:wrongPerson() \" > " +
    "Click here if you are not " + name + "</a>" );

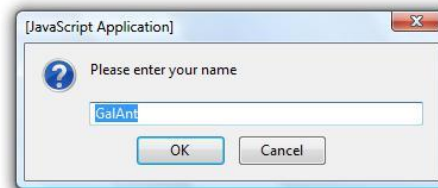
    reset the document's cookie if wrong
person function wrongPerson()
{
    reset the cookie
document.cookie= "name=null;" +
    expires=Thu, 01-Jan-95 00:00:01 GMT";

    after removing the cookie reload the page to get a new
name location.reload();
}
</script>
</head>
<body><p>Click Refresh (or Reload) to run the script
again</p> </body></html>
```

Document object



Good Morning,



Document object



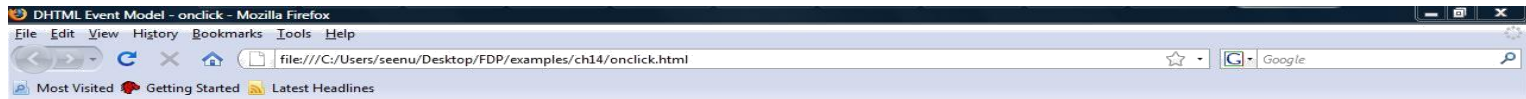
Good Morning, GalAnt, welcome to JavaScript programming!

[Click here if you are not GalAnt](#)

Click Refresh (or Reload) to run the script again



Event model-onclick



```
<html >
```

```
  <input type = "button" value = "Click Me!"
```

```
</input >
```

```
  <head>
```

```
    <title>DHTML Event Model - onclick</title>
```

```
    <script type = "text/javascript" for = "para"
```

```
      event = "onclick">
```

```
        alert( "Hi there" );
```

```
    </script>
```

```
  </head>
```

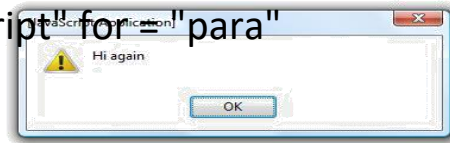
```
  <body>
```

```
    <p id = "para">Click on this text!</p>
```

```
    <input type = "button" value = "Click Me!"
```

```
      onclick = "alert( 'Hi again' )" />
```

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



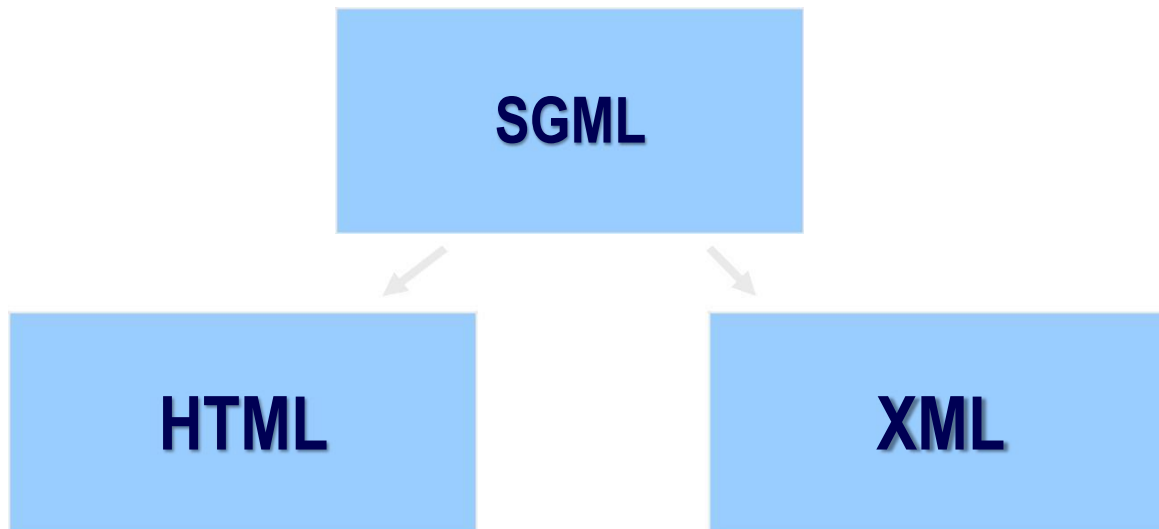
XML

Extensible Markup Language

XML

**XML is based on SGML: Standard
Generalized Markup Language**

HTML and XML are both based on SGML



Diff b/w HTML & XML

HTML was designed to display data and to focus on how data looks.

HTML is about displaying information, while XML is about describing information

XML was designed to describe data and to focus on what data is.

It is important to understand that XML was designed to store, carry, and exchange data. XML was not designed to display data.

What is XML?

XML stands for EXtensible Markup Language

XML is a markup language much like HTML

XML was designed to describe data

XML tags are not predefined. You must define your own tags

Extensible: can be extended to lots of different applications.

Markup language: language used to mark up data.

Meta Language: Language used to create other languages.

XML uses a Document Type Definition (DTD) or an XML Schema to describe the data

HTML File

```
<HTML>
```

```
<BODY>
```

```
<H1>Harry Potter</H1>
```

```
<H2>J. K. Rowling</H2>
```

```
<H3>1999</H3>
```

```
<H3>Scholastic</H3>
```

```
</BODY>
```

```
</HTML>
```

XML File

<BOOK>

<TITLE>Harry Potter</TITLE>

<AUTHOR>J. K. Rowling</AUTHOR>

<DATE>1999</DATE>

<PUBLISHER>Scholastic</PUBLISHER>

</BOOK>

XML ROLE



XML Advantages

1. XML is used to Exchange Data

- With XML, data can be exchanged between incompatible systems

2. XML and B2B

- With XML, financial information can be exchanged over the Internet.

3. XML can be used to Share Data

- With XML, plain text files can be used to share data.

XML Advantages

- 4. XML is free and extensible**
 - XML tags are not predefined. we must "invent" your own tags.
- 5. XML can be used to Store Data**
 - With XML, plain text files can be used to store data.
- 6. XML can be used to Create new Languages**
 - XML is the mother of WAP and WML.
- 7. HTML focuses on "look and feel"**
 - XML focuses on the structure of the data.

XML Example

`<!-- This is a comment -->`

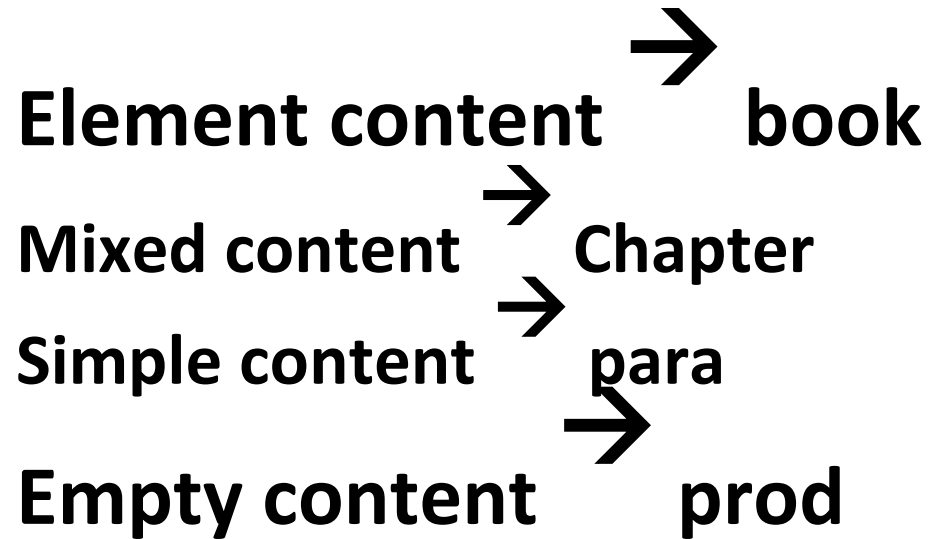
`<?xml version="1.0" encoding="ISO-8859-1"?>`

`<book>`

- `<title>My First XML</title>`
- `<prod id="33-657" Media="paper"></prod>`
- `<chapter>Introduction to XML`
 - `<para>What is HTML</para>`
 - `<para>What is XML</para>`
- `</chapter>`
- `<chapter>XML Syntax`
 - `<para>Elements must have a closing tag</para>`
 - `<para>Elements must be properly nested</para>`
- `</chapter>`

`</book>`

Elements can have different content types.



Element Naming rules

Names can contain letters, numbers, and other characters

Names must not start with a number or punctuation character

Names must not start with the letters xml (or XML, or Xml, etc)

Names cannot contain spaces

Avoid "-" and "." in names. For example, if you name something "first-name," it could be a mess if your software tries to subtract name from first. Or if you name something "first.name," your software may think that "name" is a property of the object "first."

Element Naming rules

Names should be short and simple

XML documents often have a corresponding database, in which fields exist corresponding to elements in the XML document.

Attribute values must always be enclosed in quotes, but either single or double quotes can be used.

Ex: <person sex="female">

XML Attributes

XML elements can have attributes.

Attributes are used to provide additional information about elements.

In HTML → **.**

- The SRC attribute provides additional information about the IMG element.

In XML →

- **<prod id="33-657" media="paper"></prod>**

Use of Elements vs. Attributes

Data can be stored in child elements or in attributes.

Attributes:

```
<person sex="female">  
  <firstname>Anna</firstname>  
  <lastname>Smith</lastname>  
</person>
```

Elements:

```
<person>  
  <sex>female</sex>  
  <firstname>Anna</firstname>  
  <lastname>Smith</lastname>  
</person>
```

Well-formedness

A well-formed XML document conforms to XML syntax rules and constraints, such as:

- The document must contain exactly one root element and all other elements are children of this root element.**
- All markup tags must be balanced; that is, each element must have a start and an end tag.**
- Elements may be nested but they must not overlap.**
- All attribute values must be in quotes.**

Validity

According to the XML specification, an XML document is considered valid if it has an associated DTD declaration and it complies with the constraints expressed in the DTD.

To be valid, an XML document must meet the following criteria:

- Be well-formed**
- Refer to an accessible DTD-based schema using a Document Type Declaration:**

<!DOCTYPE>

Document Type Definitions

The Document Type Definition (DTD) forms the basis of valid documents because it establishes the grammar of an XML vocabulary, which in turn determines the structure of XML documents.

A DTD is necessary for performing document validation, which is an important part of XML content development and deployment.

Syntax of DTD

ELEMENT is used to declare element names

*<!ELEMENT element_name
(subelement1,subelement2.....subelement(n-1))>*

– *Ex: <!ELEMENT product (name, type)>*

ATTLIST To declare attributes

*<!ATTLIST element_name attr1_name att_type constraints [att2_name
att_type constraints.....]>*

– *<!ATTLIST product name CDATA #REQUIRED>*

Attribute Types

- | • Types | Description |
|-----------------------|---------------------------------------|
| • CDATA | Unparsed character data |
| • Enumerated | a series of string values |
| • ID | A unique identifier |
| • IDREF | A reference to an ID declared |
| somewhere | |
| • NMTOKEN | A name consisting of XML token |
| characters | |
| • NMTOKENS | Multiple names consisting of |
| XML token | characters. |

Internal DTD

```
<!DOCTYPE root_ele_name  
[ DTD code  

```

InternalDTDs

Placing the DTD code in the DOCTYPE tag in this way

products.xml

```
<?xml version="1.0" encoding="UTF-8"?>  
<!DOCTYPE products [  
<!ELEMENT PRODUCTS (PRODUCT*)>  
<!ELEMENT PRODUCT (PID,PNAME,PRICE,DESCR,ISTOCK)>  
<!ELEMENT PID (#PCDATA)>  
<!ELEMENT PNAME (#PCDATA)>  
<!ELEMENT PRICE (#PCDATA)>  
<!ELEMENT DESCR (#PCDATA)>  
<!ELEMENT ISTOCK (#PCDATA)>  
>]
```

```
<PRODUCTS>  
<PRODUCT>  
<PID>1</PID>  
<PNAME>XYZ</PNAME>  
<PRICE>300.00</PRICE>  
<DESCR>XYZ descr</DESCR>  
<ISTOCK>1000</ISTOCK>  
</PRODUCT>  
</PRODUCTS>
```

External DTD

<!DOCTYPE root_ele_name SYSTEM 'dtd file name'>

– SYSTEM → the definitions are developed and used by the same comp

or

<!DOCTYPE root_ele_name PUBLIC ' fpi string ' ' dtd url '>

– PUBLIC → if the definition can be used by public

products.dtd

<!ELEMENT products (product)>

<!ELEMENT product (name, type)>

<!ELEMENT name (#PCDATA)>

<!ELEMENT type (#PCDATA)>

products.xml

<?xml version = "1.0"?>

***<!DOCTYPE products
SYSTEM "products.dtd">***

Qualifier Name Meaning

<i>Qualifier</i>	<i>Name</i>	<i>Meaning</i>
?	Question Mark	Optional (zero or one)
*	Asterisk	Zero or more
+	Plus Sign	One or more

Ex : `<!ELEMENT emp_details (emp+)>`

Attribute Values

<i>Specification</i>	<i>Specifies...</i>
#REQUIRED	The attribute value must be specified in the document.
#IMPLIED	The value need not be specified in the document. If it isn't, the application will have a default value it uses.
"defaultValue"	The default value to use, if a value is not specified in the document.
#FIXED "fixedValue"	The value to use. If the document specifies any value at all, it must be the same.

**<!ATTLIST product name CDATA
#REQUIRED>**

Prefix URI for namespaces

Xml

<http://www.w3.org/XML/1998/namespace>

Xsl

<http://www.w3.org/1999/XSL/Transform>

Xsd

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

```
<root>
<h:table xmlns:h="http://www.w3.org/TR/html4/"
>
  <h:tr>
    <h:td>Apples</h:td>
    <h:td>Bananas</h:td>
  </h:tr>
</h:table>
```

```
<f:table xmlns:f="http://www.w3.org/TR/html4/"
>
  <f:name>African Coffee Table</f:name>
  <f:width>80</f:width>
  <f:length>120</f:length>
</f:table>
</root>
```


XML Schema

XML Schema is an XML-based alternative to DTD.

An XML schema describes the structure of an XML document.

The XML Schema language is also referred to as XML Schema Definition (XSD).

XML Schema

An XML Schema:

defines elements that can appear in a document

defines attributes that can appear in a document

defines which elements are child elements

defines the order of child elements

defines the number of child elements

defines whether an element is empty or can include text

defines data types for elements and attributes

XML Schema elements

- 1.Simple elements
- 2.Complex elements

Simple Elements

A simple element is an XML element that can contain only text. It cannot contain any other elements or attributes.

Xml:

```
<lastname> abc </lastname>
```

Xml schema:

```
<xs:element name="lastname" type="xs:string"  
/>
```

Restrictions

```
<?xml version="1.0" encoding="utf-8" ?>
<xsd:schema
  xmlns:xsd=http://www.w3.org/2001/XMLSchema
  >
  <xsd:element name="age">
    <xsd:simpleType>
      <xsd:restriction base="xs:integer">
        <xsd:minInclusive value="0"/>
        <xsd:maxInclusive value="120"/>
      </xsd:restriction>
    </xsd:simpleType>
  </xsd:element >
```

```
<?xml version="1.0" encoding="utf-8" ?>
  <xsd:schema
    xmlns:xsd=http://www.w3.org/2001/XMLSchema>
    <xsd:element name="car">
      <xsd:simpleType>
        <xsd:restriction base="xsd:string">
          <xsd:enumeration value="Audi"/>
          <xsd:enumeration value="Golf"/>
          <xsd:enumeration value="BMW"/>
        </xsd:restriction>
      </xsd:simpleType>
    </xsd:element>
```

```
<?xml version="1.0" encoding="utf-8" ?>
<xsd:schema
  xmlns:xsd="http://www.w3.org/2001/XMLSchema">
<xsd:element name="initials">
<xsd:simpleType>
<xsd:restriction base="xsd:string">
<xsd:pattern value="[A-Z][A-Z][A-Z]"/>
</xsd:restriction>
</xsd:simpleType>
</xsd:element>
```

Restriction on length

```
<xsd:element name="password">  
  <xsd:simpleType>  
    <xsd:restriction base="xs:string">  
      <xsd:length value="8"/>  
    </xsd:restriction>  
  </xsd:simpleType>  
</xsd:element>
```


Complex Elements

A complex element is an XML element that contains other elements and/or attributes.

```
<xsd:element name="employee">  
  <xsd:complexType>  
    <xsd:sequence>  
      <xsd:element name="firstname"  
type="xsd:string"/> <xsd:element  
name="lastname" type="xsd:string"/>  
    </xsd:sequence>  
  </xsd:complexType>  
</xsd:element>
```

```
<xsd:element name="employee" type="personinfo"/>
<xsd:complexType name="personinfo">
  <xsd:sequence>
    <xsd:element name="firstname" type="xs:string"/>
    <xsd:element name="lastname"
      type="xs:string"/> </xsd:sequence>
  </xsd:complexType>
```

Xml file

```
<college name="snist">  
  <strength>1000</strength>  
  <branch>6</branch>  
  <bname>  
    <cse block="1">200</cse>  
    <it block="2">200</it>  
    <mech block="3">200</mech>  
    <ece block="4">200</ece>  
    <eee block="5">100</eee>  
    <ecm block="6">100</ecm>  
  </bname>  
</college>  
<college>  
..  
</college>
```

Diff b/w DTD & XSD

DTD supports types ID, IDREF, CDATA etc.,

Schema supports all primitive and user defined data types

DTD supports No specifier, ?, *, + sign

Schema have minOccurs and maxOccurs attributes

XML Schemas are extensible to future additions

XML Schemas are richer and more powerful than DTDs

XML Schemas are written in XML

Parsers

An XML parser is a piece of code that reads a document and analyzes its structure.

The parser is the engine for interpreting our XML documents

The parser reads the XML and prepares the information for your application.

How to use a parser

- 1. Create a parser object**
- 2. Pass your XML document to the parser**
- 3. Process the results**

Parser Example

```
import com.ibm.xml.parser.*;
import java.io.*;
public class SimpleParser
{ public static void main (String a[]) throws Exception
  { Parser p=new Parser("err");
    FileInputStream fis=new FileInputStream(a[0]);
    TXDocument doc=p.readStream(fis);
    doc.printWithFormat(new OutputStreamWriter(System.out)
      }
  }
}
```

Types of Parsers

There are two common APIs that parsers use.

- **DOM is the Document Object Model API**
- **SAX is the Simple API for XML**

Document Object Model (DOM)

DOM uses a **tree based structure**.

DOM reads an entire XML document and builds a **Document Object**.

- The Document object contains the tree structure.
- The top of the tree is the root node.
- The tree grows down from this root, defining the child elements.
- DOM is a W3C standard.

Using DOM, we can also perform insert nodes, update nodes, and deleting nodes.

DOM interfaces

Node: The base data type of the DOM.

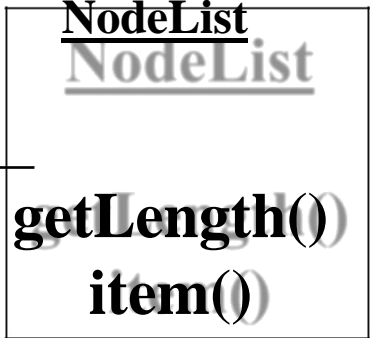
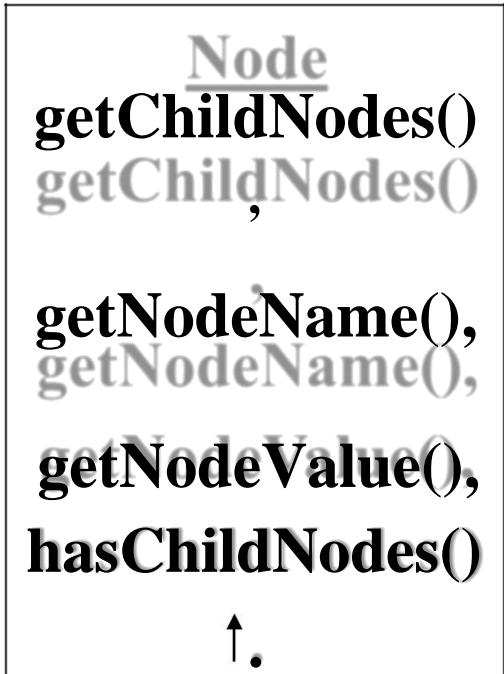
Methods:

Element:

- **Attr:** Represents an attribute of an element.
- **Text:** The actual content of an Element or Attribute

Document: Represents the entire XML document. A Document object is often referred to as a *DOM tree*.

Node



Element

Element

Document

Document

createElement(),
createAttribute(
),
createTextNode(
,),

Simple API for XML Parsing (SAX)

SAX parsers are event-driven

- **The parser fires an event as it parses each XML item.**
- **The developer writes a class that implements a handler interface for the events that the parser may fire.**

SAX Interface

DocumentHandler

– Functions in this interface

startDocument()

startElement()

endElement()

endDocument()

void setDocumentLocator(Locator)

void characters(char[],int start,int length)

– This event fires when text data is found in the XML Document

class HandlerBase is a sub class of DocumentHandler also called Adapter Class.

Diff b/w DOM & SAX

DOM	SAX
Uses more memory and has more functionality	Uses less memory and provides less functionality
The entire file is stored in an internal Document object. This may consume many resources	The developer must handle each SAX event before the next event is fired.
For manipulation of the document, DOM is best choice	For simple parsing and display SAX will work great

UNIT-III

Servlets

The Servlet Life Cycle

Overview of the Life Cycle

Birth of a Servlet

Life of a Servlet

Death of a Servlet

Overview of Servlet Life Cycle

Life of a Servlet

Birth: Create and initialize the servlet

Important method: `init()`

Life: Handle 0 or more client requests

Important method: `service()`

Death: Destroy the servlet

Important method: `destroy()`

Birth of a Servlet

The init() method

The init() method is called when the servlet is first requested by a browser.

It is not called again for each request.

Used for one-time initialization.

There are two versions of the init() method:

- Version 1: takes no arguments

- Version 2: takes a **servletConfig** object as an argument.

We will focus only on the first option.

Simple Example

The `init()` method is a good place to put any initialization variables.

For example, the following servlet records its Birth Date/time...

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

```
public class Birth extends HttpServlet {
    Date birthDate;
```

```
// Init() is called first
```

```
public void init() throws ServletException
    { birthDate = new Date();
}
```

```
// Handle an HTTP GET Request
public void doGet(HttpServletRequest request,
    HttpServletResponse response)
    throws IOException, ServletException {
    response.setContentType("text/plain");
    PrintWriter out = response.getWriter();
    out.println("I was born on:"+birthDate);
    "+birthDate); out.close();
}
}
```

Life of a Servlet

Life of a Servlet

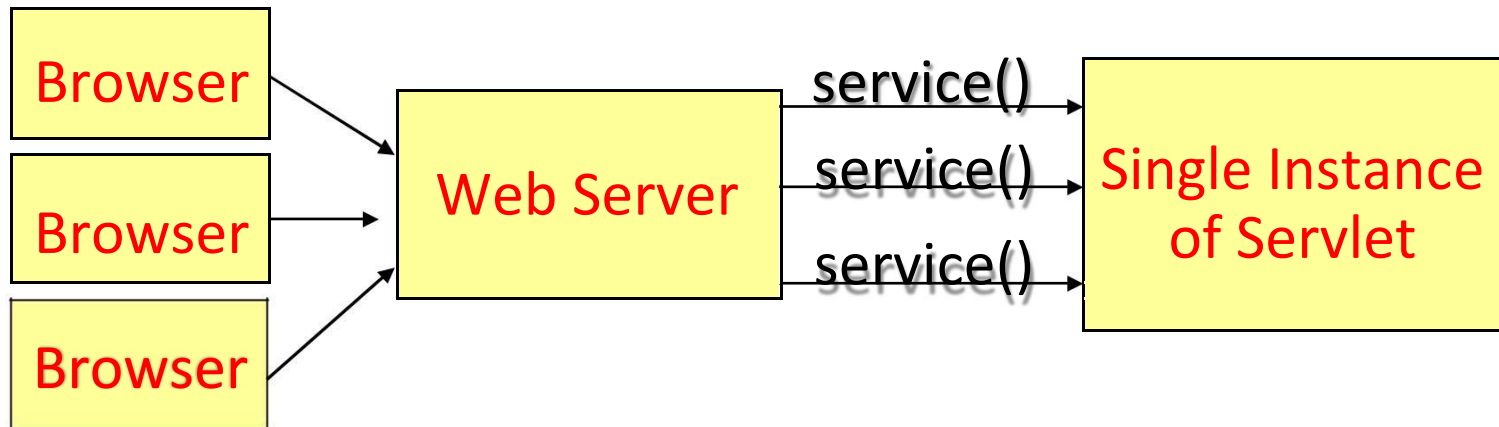
The first time a servlet is called, the Servlet is instantiated, and its `init()` method is called.

Only one instance of the servlet is instantiated.

This one instance handles all browser requests.

Service() Method

Each time the server receives a request for a servlet, the server spawns a new thread and calls the servlet's service () method.



Let's Prove it...

To prove that only one instance of a servlet is created, let's create a simple example.

The Counter Servlet keeps track of the number of times it has been accessed.

This example maintains a single instance variable, called count.

Each time the servlet is called, the count variable is incremented.

If the Server created a new instance of the Servlet for each request, count would always be 0!

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

```
public class Counter extends HttpServlet {
```

```
    // Create an instance variable
    int count = 0;
```

```
    // Handle an HTTP GET Request
```

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

```
ServletException {
```

```
        response.setContentType("text/plain");
```

```
        PrintWriter out = response.getWriter();
```

```
        count++;
```

```
        out.println ("Since loading, this servlet has "
```

```
            + "been accessed "+ count + " times.");
```

```
        times."); out.close();
```

```
    }
```

```
}
```

Only one instance of the
counter Servlet is created.

Each browser request is
therefore incrementing the
same count variable.

The Service Method

By default the `service()` method checks the HTTP Header.

Based on the header, `service` calls either `doPost()` or `doGet()`.

`doPost` and `doGet` is where you put the majority of your code.

If your servlets needs to handle both get and post identically, have your `doPost()` method call `doGet()` or vice versa.

Death of a Servlet

Death of a Servlet

Before a server shuts down, it will call the servlet's `destroy()` method.

You can handle any servlet clean up here. For example:

- Updating log files.

- Closing database connections.

- Closing any socket connections.

Example: Death.java

This next example illustrates the use of the `destroy()` method.

While alive, the servlet will say “I am alive!”.

When the server is stopped, the `destroy()` method is called, and the servlet records its time of death in a “rip.txt” text file.

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

```
public class Death extends HttpServlet
{
```

```
    // Handle an HTTP GET Request
```

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

```
    {
```

```
        response.setContentType("text/plain");
```

```
        PrintWriter out = response.getWriter();
```

```
        out.println ("I am alive!");
```

```
        out.close();
```

```
        out.close();
```

```
    }
```

Continued....
Continued....

}

```
//This method is called when one stops
```

```
//the Java Web Server
```

```
public void destroy()
```

```
{
```

```
    try
```

```
    {
```

```
        FileWriter fileWriter = new FileWriter ("rip.txt");
```

```
        ("rip.txt"); Date now = new Date();
```

```
        String rip = "I was destroyed at: "+now.toString();
```

```
        fileWriter.write (rip);
```

```
        fileWriter.close();
```

```
    } catch (IOException e)
```

```
    {
```

```
        e.printStackTrace();
```

```
    }
```

```
}
```

```
}
```

Putting it all together

A Persistent Counter

Now that we know all about the birth, life and death of a servlet, let's put this knowledge together to create a persistent counter.

The Counter.java example we covered earlier has a big problem:

When you restart the web server, counting starts all over at 0. It does not retain any persistent memory.

Persistent Counter

To create a persistent record, we can store the count value within a “counter.txt” file.

init(): Upon start-up, read in the current counter value from counter.txt.

destroy(): Upon destruction, write out the new counter value to counter.txt

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

```
public class CounterPersist extends HttpServlet {file.
    String fileName = "counter.txt";
    int count;
```

```
public void init() {
```

```
    try {
```

```
        FileReader fileReader = new FileReader (fileName);
```

```
        BufferedReader bufferedReader = new BufferedReader (fileRea
```

```
        String initial = bufferedReader.readLine();
```

```
        count = Integer.parseInt (initial);
```

```
    } catch (FileNotFoundException e) { count = 0; }
```

```
    } catch (IOException e) { count=0;}
```

```
    } catch (NumberFormatException e) { count = 0; }
```

```
    } catch (NumberFormatException e) { count = 0; } Continued...
}
```

At Start-up, load
the counter from
file.

In the event of any
exception, initialize
count to 0.

```
// Handle an HTTP GET Request
public void doGet(HttpServletRequest request, HttpServletResponse response)
    throws IOException, ServletException {
    response.setContentType("text/plain");
    PrintWriter out = response.getWriter();
    count++;
    out.println ("Since loading, this servlet has "
        + "been accessed "+ count + " times.");
    out.close();
}
```

Each time the
doGet() method is
called, increment
the count variable.

Continued....

```
// At Shutdown, store counter back to file
```

```
public void destroy() {
```

```
    try {
```

```
        FileWriter fileWriter = new FileWriter (fileName);
```

```
        (fileName); String countStr = Integer.toString
```

```
(count); fileWriter.write(countStr);
```

```
fileWriter.close();
```

```
    } catch (IOException e) {
```

```
        e.printStackTrace();
```

```
    }
```

```
}}
```

```
}
```

Can anybody
foresee

any problems with
this code?

When destroy() is
called, store new
counter variable
back to counter.txt.

Steps for implementing servlet programs

Import the required packages

```
Javax.servlet.http.*;
```

```
Javax.servlet.*;
```

```
Java.io.*;
```

Any other packages if required

Define a class extends with either generic servlet or `HttpServlet` class

Define the class with public access modifier hence save the file exactly same as class name

Override required lifecycle methods

Various phases of servlet during execution

Object instantiation phase

Object initialization

Executing the `init()` method of servlet

Request processing phase

Executing the `service()` method of servlet

Destruction phase

Executing `destroy()` method of servlet

5) unavailability

Web.xml

It is also known as web configuration file of deployment descriptor

Web.xml is the fixed name to be given in the web application development

It is always used to populate/ hide the technologies which are used in the web application development

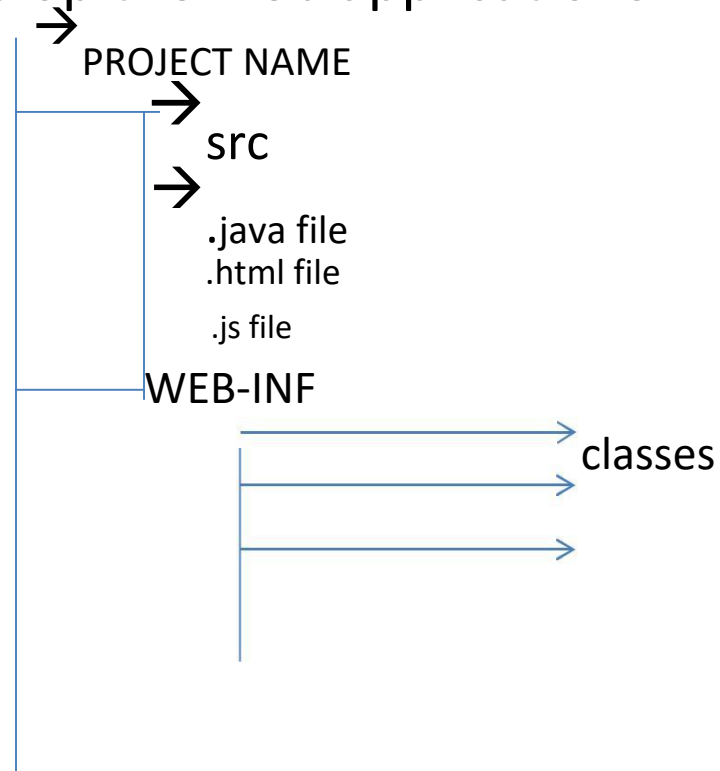
The web.xml file will be scanned by the server initially

Web.xml file contd.,

```
<web-app>
  <servlet>
    <servlet-name>logical name</servlet-name>
    <servlet-class>servlet class</servlet-class>
  </servlet>
  <servlet-mapping>
    <servlet-name>logical name</servlet-name> <url-
    pattern>/public url of web app</url-pattern>
  </servlet-mapping>
</web-app>
```

Deployment Structure

It is prescribed structure provided by sun microsystem in oracle to develop the web applications



Before compiling the servlet program we need to set the class path for **servlet-api.jar**

Set classpath=c:\program files\apache software foundatin\tomcat7.0\lib\servlet-api.jar;

After compiling the java program copy the .class file from src folder to classes folder.

Define web.xml file

Deployment

It is the process of copying the project from the current working directory to the context of server

Copy the project from current directory and paste it into webapp

- C:\program files\apache software foundation\tomcat 7.0\webapps

Start the server

- Start → programs → apache tomact 7.0 → configured tomcat → start
(ensure server status must be started)

Open the browser type the following url

- <http://localhost:8080/project> name/servlet-name

Thread Synchronization

After the Servlet is generated, one instance of it serves requests in different threads, just like any other Servlet

In particular, the service method (`_jspService`) may be executed by several concurrent threads

Thus, like Servlets, JSP programming requires concurrency management

Basic JSP Elements

Basic Elements in a JSP file

HTML code: `<html-tag>content</html-tag>`

JSP Comments: `<%-- comment --%>`

Expressions: `<%= expression %>`

Scriptlets: `<% code %>`

Declarations: `<%! code %>`

Directives: `<%@ directive attribute="value" %>`

Actions: `<jsp:forward.../>`, `<jsp:include.../>`

EL Expressions: `${expression}`

Covered Later...

JSP Expressions

A JSP **expression** is used to insert Java values directly into the output

It has the form: `<%= expression %>` , where *expression* can be a Java object, a numerical expression, a method call that returns a value, etc...

For example:

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

```
<%= "Hello"+" World" %>
```

```
<%= (int)(100*Math.random()) %>
```

The heading space and the following space are not created in the result.

Use "" if you want a real space

JSP Expressions

Within the generated Java code

- A JSP Expression is **evaluated**
- The result is **converted to a string**
- The string is **inserted into the page**

This evaluation is **performed at runtime** (when the page is requested), and thus has full access to information about the request, the session, etc...

Expression Translation

```
<h1>A Random Number</h1>  
<%= Math.random() %>
```



```
public void _jspService(HttpServletRequest  
    request, HttpServletResponse response)  
    throws java.io.IOException, ServletException {
```

...

```
response.setContentType("text/html");
```

...

```
out.write("<h1>A Random Number</h1>\r\n" + "The generated");
```

```
out.print( Math.random() );
```

```
out.write("\r\n");
```

...

```
}  
}
```

Default
content-type

servlet calls
out.write() for
Strings, and

out.print() for

objects

objects

Predefined Variables (Implicit Objects)

The following predefined variables can be used:

- **request**: the `HttpServletRequest`
- **response**: the `HttpServletResponse`
- **session**: the `HttpSession` associated with the request
- **out**: the `PrintWriter` (a buffered version of type `JspWriter`) used to fill the response content
- **application**: The `ServletContext`
- **config**: The `ServletConfig`

These variables and more will be discussed in details

```
<html>
  <head>
    <title>JSP Expressions</title>
  </head>
  <body>
    <h2>JSP Expressions</h2>
    Expressions</h2> <ul>
      <li>Current time: <%= new java.util.Date() %></li>
      <li>Your hostname:<%= request.getRemoteHost() %></li>
      <li>Your session ID: <%= session.getId() %></li>
      <li>The <code>testParam</code> form parameter:
        <%= request.getParameter("testParam") %></li>
    </ul>
  </body>
</html>
```

Computer-
code style



Computer-

JSP Scriptlets

JSP **scriptlets** let you insert arbitrary code into the Servlet service method (`_jspService`)

Scriptlets have the form: `<% Java Code %>`

The code is inserted verbatim into the service method, according to the location of the scriptlet

Scriptlets have access to the same automatically defined variables as expressions

Scriptlet Translation

```
<%= foo () %>  
<% bar (); %>
```



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

```
...
```

```
response.setContentType("text/html");
```

```
...
```

```
out.print(foo());  
bar();
```

```
...
```

```
}
```

An Interesting Example

Scriptlets don't have to be complete code blocks:

```
<% if (Math.random() < 0.5) { %>  
%>You <b>won</b> the game!  
<% } else { %>  
You <b>lost</b> the game!  
<% } %>
```



```
if (Math.random() < 0.5) {  
    out.write("You <b>won</b> the game!");  
} else {  
    out.write("You <b>lost</b> the game!");  
}
```

JSP Declarations

A JSP **declaration** lets you define methods or members that get inserted into the Servlet class (**outside** of all methods)

It has the following form:

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

For example:

```
<%! private int someField = 5; %>  
<%! private void someMethod(...) {...} %>
```

JSPs are intended to contain a minimal amount of code so it is usually of better design to define methods in a separate Java class...

Declaration Example

Print the number of times the current page has been requested since the Servlet initialization:

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

```
<%! private synchronized int incAccess()  
        { return ++accessCount;  
        } %>
```

```
<h1>Accesses to page since Servlet init:
```

```
<%= incAccess() %> </h1>
```



```
public class serviceCount_jsp extends... implements...
```

```
throws... {
```

```
private int accessCount = 0;
```

```
private synchronized int incAccess() {
```

```
return ++accessCount;
```

```
}
```

```
public void _jspService(HttpServletRequest request,
```

```
HttpServletResponse response)
```

```
throws ServletException, IOException {
```

```
...
```

```
...
```

```
out.write("<h1>Accesses to page since Servlet init:  
"); out.print(incAccess());
```

```
... } ... }
```

Java permits
member

initialization

on
declaration,
even if the
location is
outside any
method's

scope

JSP Directives

A JSP directive affects the structure of the Servlet class that is generated from the JSP page

It usually has the following form:

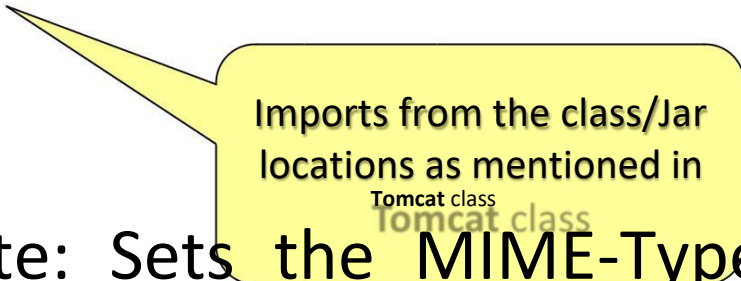
```
<%@ directive attribute1="value1" ...  
      attributeN="valueN" %>
```

Three important directives: **page**, **include** and **taglib**
include and **taglib** will be discussed later

page-Directive Attributes

import attribute: A comma separated list of classes/packages to import

```
<%@ page import="java.util.*, java.io.*" %>
```



Imports from the class/Jar locations as mentioned in

Tomcat class

Tomcat class

- **contentType** attribute: Sets the MIME-Type of the resulting document (default is `text/html` as already mentioned)

```
<%@ page contentType="text/plain" %>
```


page-Directive Attributes (cont)

What is the difference between setting the page contentType attribute, and writing

`<%response.setContentType("...");%> ?`

- In the latter case, the new servlet will call *response.setContentType()* twice
- The first, implicit (from the JSP point of view), call will be with the default content type.
- The second, explicit, call might even come after the buffer was flushed or after the writer was obtained...

Check: [double-contenttype.jsp](#) code using the explicit call, generated java code (servlet)

page-Directive Attributes (cont)

`session="true|false"` - use a session?

The underlined value is the default

• `buffer="sizekb|none|8kb"`

- Specifies the content-buffer (out) size to the client in kilobytes

If the JSP is defined as using a session, a session cookie will be

`autoFlush="true|false"`

- Specifies whether the buffer should be flushed when it fills, or throw an exception otherwise

`isELIgnored="true|false"`

- Specifies whether *JSP expression language* is used
- EL is discussed later

Web Application: Java Server Pages (JSP)

Outline

Introducing JavaServer Pages™ (JSP™)

JSP scripting elements

- Expressions, Scriptlets and declarations

The JSP page Directive:

- Structuring Generated Servlets™

Including Files in JSP Documents

Using JavaBeans™ components with JSP

Creating custom JSP tag libraries

Integrating servlets and JSP with the MVC architecture

The JSP Framework

Idea:

- Use regular HTML for most of page
- Mark servlet code with special tags
- Entire JSP page gets translated into a servlet (once), and servlet is what actually gets invoked (for each request)

Example:

- JSP

Thanks for ordering
<I><%= request.getParameter("title") %></I>

- URL

<http://host/OrderConfirmation.jsp?title=Core+Web+Programming>

- Result

Thanks for ordering Core Web Programming

Setting Up Your Environment

- ✘ Set your CLASSPATH.
- ✘ Compile your code.
- ✘ Use packages to avoid name conflicts.
- ✘ Put JSP page in special directory.
 - ✘ tomcat_install_dir/webapps/ROOT
- ✘ Use special URL to invoke JSP page.

Caveats

- Previous rules about CLASSPATH, install dirs, etc., still apply to regular classes used by JSP

```
<HTML><HEAD>
<TITLE>JSP Expressions</TITLE>
<META NAME="author" CONTENT="Marty Hall">
<META NAME="keywords"
  CONTENT="JSP,expressions,JavaServer,Pages,servlets
">
<META NAME="description"
  CONTENT="A quick example of JSP expressions.">
<LINK REL=STYLESHEET HREF="JSP-Styles.css"
  TYPE="text/css">
</HEAD>
<BODY>
```

```
<H2>JSP Expressions</H2> Example <UL>
  <LI>Current time: <%= new java.util.Date() %>
  <LI>Your hostname: <%= request.getRemoteHost() %>
  <LI>Your session ID: <%= session.getId() %> <LI>The
  <CODE>testParam</CODE> form parameter:
    <%= request.getParameter("testParam")
%> </UL></BODY></HTML>
```

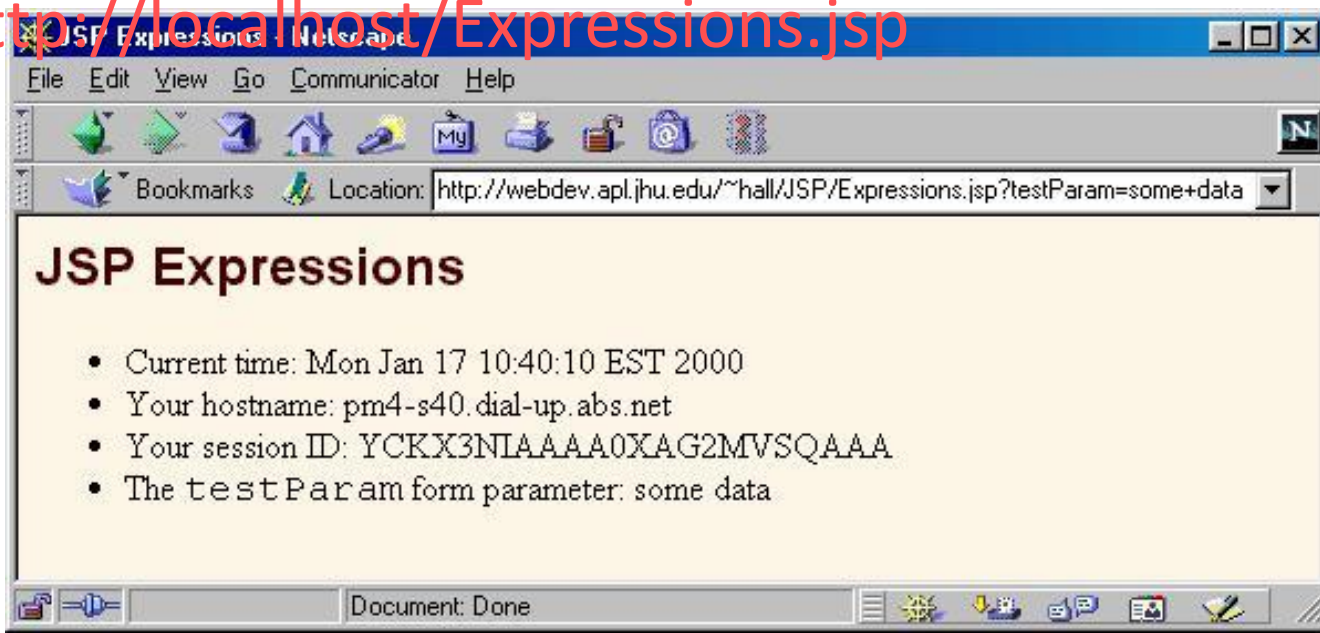

Example Result

With default setup, if location was

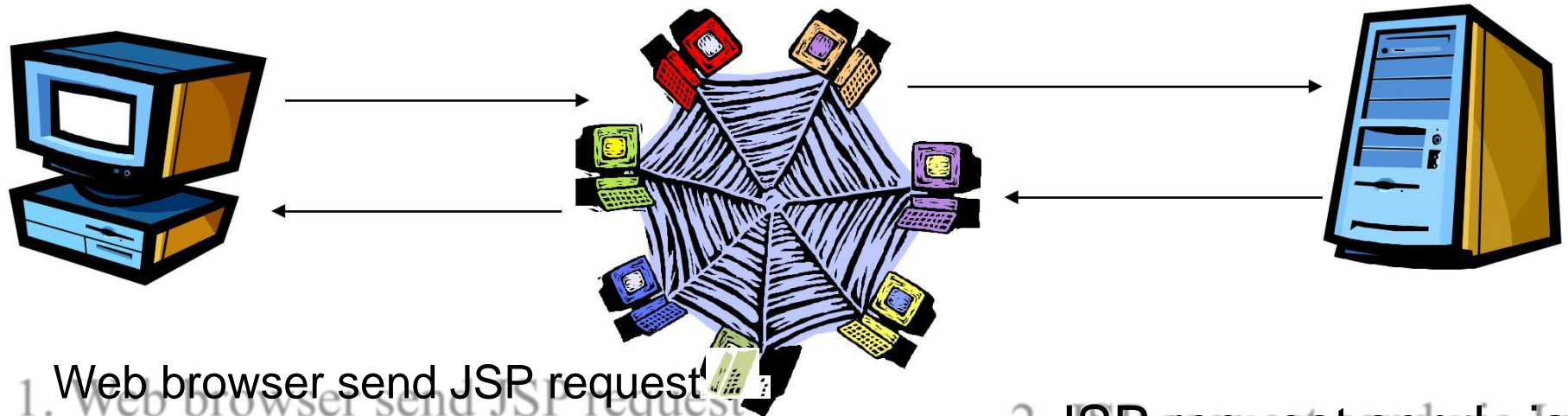
- C:\<tomcatHome>\webapps\ROOT\Expressions.jsp

URL would be

- <http://localhost/Expressions.jsp>



How JSP works?



1. Web browser send JSP request

2. JSP request send via Internet to the web server

3. The web server send the J file (template pages) to JS servlet engine

4. Parse JSP file

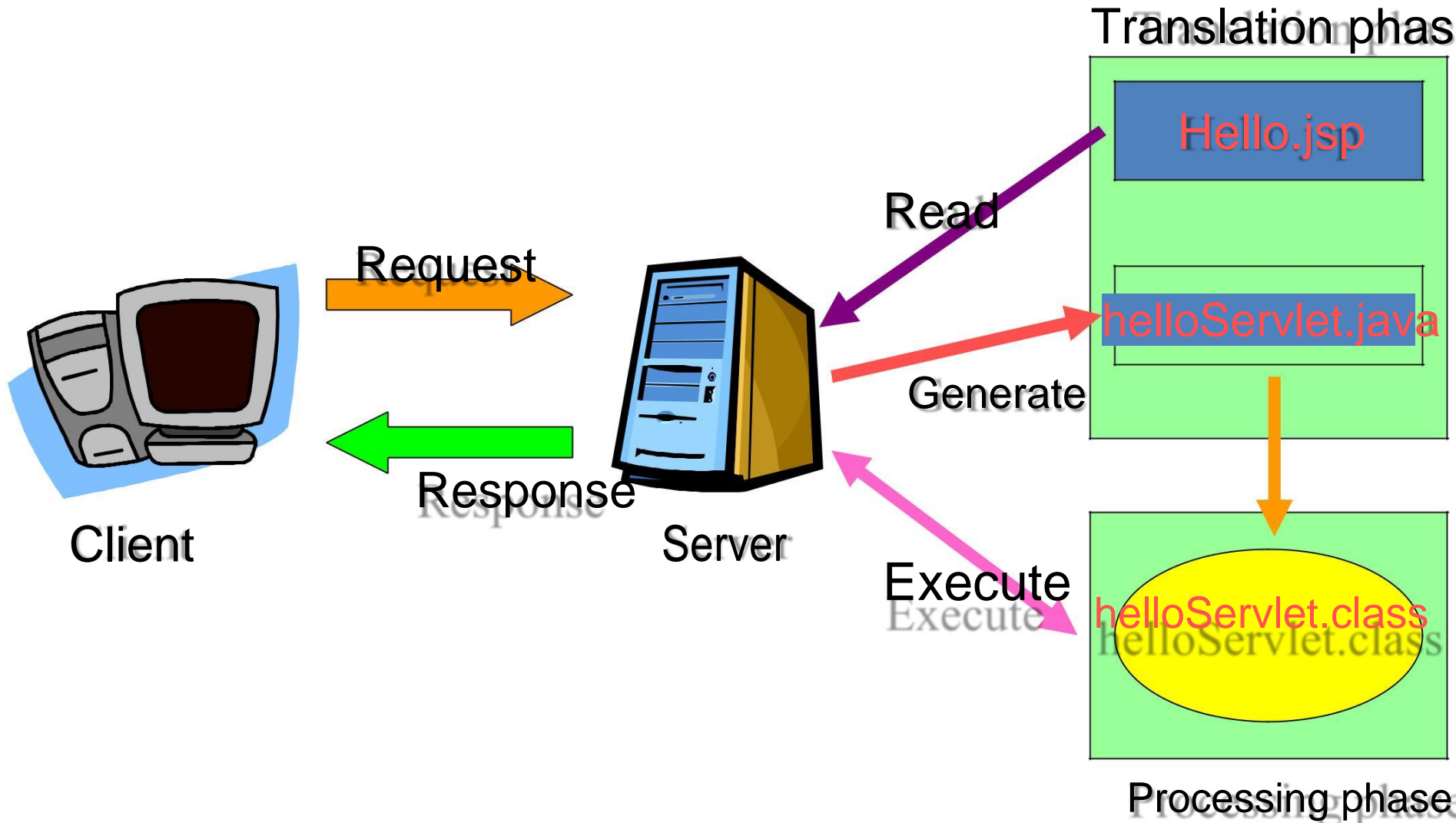
5. Generate servlet source code

6. Compile servlet to class

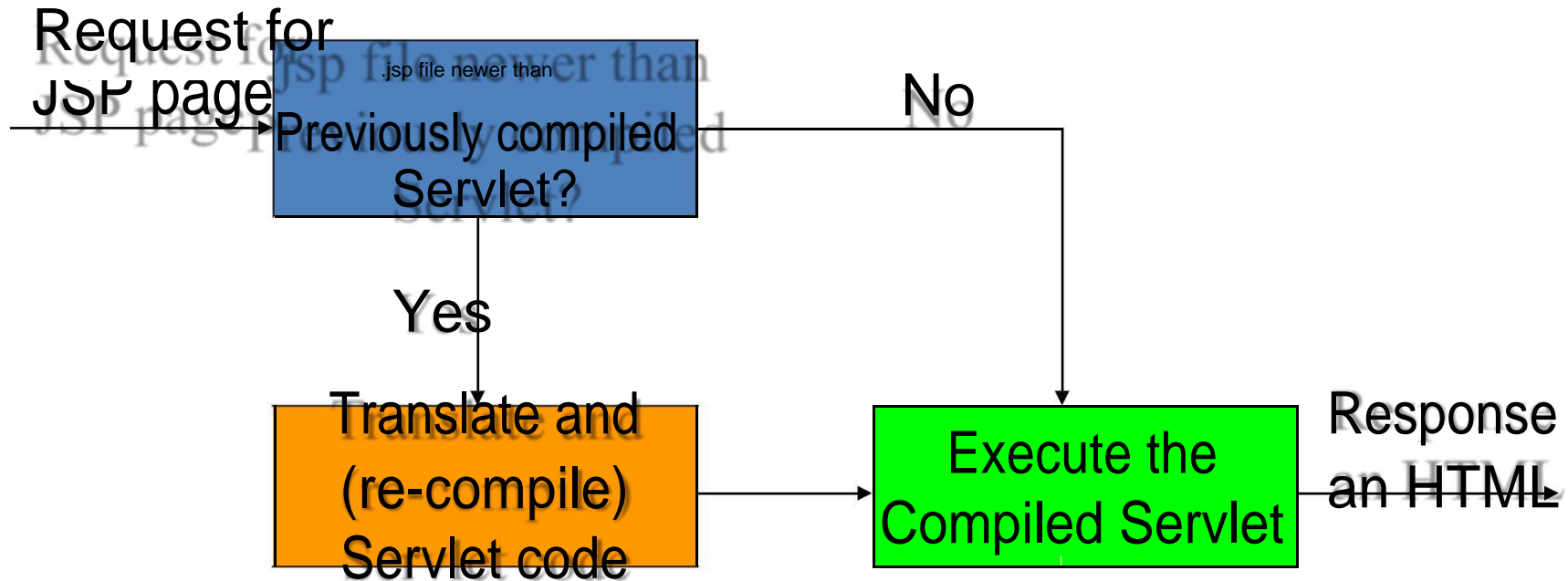
8. HTML send back to the browser

Compile servlet to class

JSP page translation and processing phases



JSP Life-cycle



Template Pages

Server Page Template

```
<html>
<title>
A simple example
</title>
</title>
<body color="#FFFFFF">
The time now is
<%= new java.util.Date() %>
</body>
</html>
```

translation
→
translation

Resulting HTML

```
<html>
<title>
A simple example
</title>
</title>
<body color="#FFFFFF">
The time now is
Tue Nov 5 16:15:11 PST 2002
</body>
</html>
```

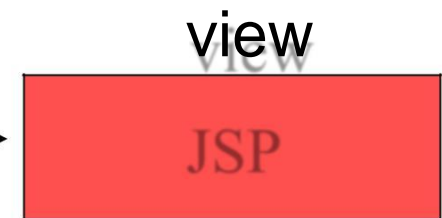
Dividing Pure Servlets

```
Public class MyServlet {  
    public void doGet(...){  
        if (isValid(..){  
            saveRecord();  
            out.println("<html>");  
            ....  
        }  
    }  
    private void isValid(...){...}  
    private void saveRecord(...){...}  
}
```

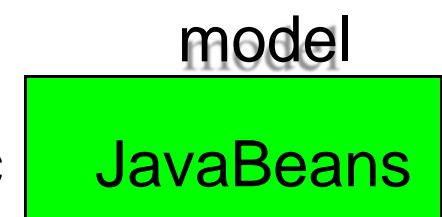
controller
Process request



Presentation
JSP



Business logic



Model-View-Controller (MVC)

Most Common Misunderstanding: Forgetting JSP is Server-Side Technology

Very common question

- I can't do such and such with HTML. Will JSP let me do it?

Similar questions

- How do I put an applet in a JSP page? Answer: send an `<APPLET...>` tag to the client
- How do I put an image in a JSP page? Answer: send an `` tag to the client
- How do I use JavaScript/Acrobat/Shockwave/Etc? Answer: send the appropriate HTML tags

2nd Most Common Misunderstanding: Translation/Request Time Confusion

What happens at page translation time?

- JSP constructs get translated into servlet code

What happens at request time?

- Servlet code gets executed. No interpretation of JSP occurs at request time. The original JSP page is ignored at request time; only the servlet that resulted from it is used

When does page translation occur?

- Typically, the first time JSP page is accessed after it is modified. This should never happen to real user (developers should test all JSP pages they install).

– Page translation does not occur for each request

JSP/Servlets in the Real World

Delta Airlines: entire Web site, including real-time schedule info

First USA Bank: largest credit card issuer in the world; most on-line banking customers

The image displays two side-by-side screenshots of web browsers. The left browser window is titled "First USA - Netscape" and shows the First USA Bank website. The right browser window is titled "Delta Air Lines - Welcome to Delta - Netscape" and shows the Delta Airlines website.

First USA - Netscape

Address: <http://www.firstusa.com/>

Navigation: File Edit View Go Communicator Help

Bookmarks: Netsite: <http://www.firstusa.com/>

Cardmember Log in | Contact Us

FIRST USA

Get great deals at **Connections**
Shop & compare at more than 200 merchants [CLICK HERE](#)

Inside First USA

- About First USA
- Help / FAQ
- Privacy Policy
- Career Opportunities
- Site Map

Become a Cardmember

Select a Card

Cardmember Services

Current Member? [LOG IN HERE](#)

Not a member? [ENROLL NOW](#)

Special Services

Select a Service

What's New

Cardmember Services

- Check your balance
- Make a payment
- View transactions

[Enroll here](#)

Manage your credit card account by [enrolling in](#) First USA's Cardmember Services Online. Our free service allows you to check account balances, see recent purchases, and pay your balance online.

You can relax when you shop at First USA Connections - our [Safe Shopping Guarantee](#) means you won't be responsible for unauthorized online charges to your First USA card.

Do you have questions about your First USA account? [Click here](#) to learn about payment options, cardmember tips, balance transfers and other useful FAQ's.

Copyright ©2000 First USA. All rights reserved.
Your [privacy](#) will be respected at all times.

Delta Air Lines - Welcome to Delta - Netscape

Address: <http://www.delta.com/home/index.jsp>

Navigation: File Edit View Go Communicator Help

Bookmarks: Netsite: <http://www.delta.com/home/index.jsp>

Delta
delta.com

HOME TRAVEL SKYMILES PROGRAMS & SERVICES INSIDE DELTA CUSTOMER CARE

SkyMiles log in

SkyMiles number and PIN

Start in: Home

[GO](#)

ROUND-TRIP RESERVATIONS

[One-way & multi-city reservations](#) | [Calendar](#) | [City codes](#)

Purchase online and earn up to 1,000 **Bonus SkyMiles!**

Leaving from: [] Select departure date and time: Sep 06 10 AM

Going to: [] Select return date and time: Sep 06 10 AM

Passengers and preferred cabin: 1 [] Coach (Restricted)

[GO](#)

ARRIVAL / DEPARTURE INFO

Get instant arrival and departure information.

Flight number: [] Yesterday Today Tomorrow

[GO](#)

DELTA FOR YOU

Wednesday, September 6, 2000

Specials

- Up to 10% off - [Offer Ends 9/8/00](#)

News

- Introducing the 767-400, our newest aircraft
- Celebrate SkyTeam's take-off with our [Million Miles Sweepstakes](#)
- More breaking [news](#)

[Stay connected with flight info on your Palm VII™ or cell phone.](#) [click here](#)

Navigation: File Edit View Go Communicator Help

Bookmarks: Netsite: <http://www.delta.com/home/index.jsp>

Links: [Reservations](#), [View Itineraries](#), [Upgrades](#), [Flight Schedules](#), [Special Offers](#), [Investor Relations](#), [E-mail Programs](#)

Address: <http://www.delta.com/home/index.jsp>

JSP/Servlets in the Real World

Excite: one of the top five Internet portals;
one of the ten busiest sites on the Web



wine.com ... the best of wine - Netscape

File Edit View Go Communicator Help

Location: http://www1.wine.com/

Home Wine Shop Rare Wines Wine Clubs Gift Shop

Welcome to wine.com! Please sign in, or create an account if you're a new user.

Search

Wine Selector:

Category [v]
Price [v]
Origin [v] **Go >**

Text Search: [] **Go >**

More Search Options

Short Cuts

Red Wines [v] **Go >**

Highlights

- Oxford Online
- W3 Tasting
- Peter's Tasting Chart
- Who's Peter
- Underage Drinking
- Save \$10
- How to Order

Live Help

Have a question for Customer Service? Use Live Help to get it

Oxford Online
New ones are popping up all the time in the wine country of Napa and Sonoma. Should you be afraid of these California cults?

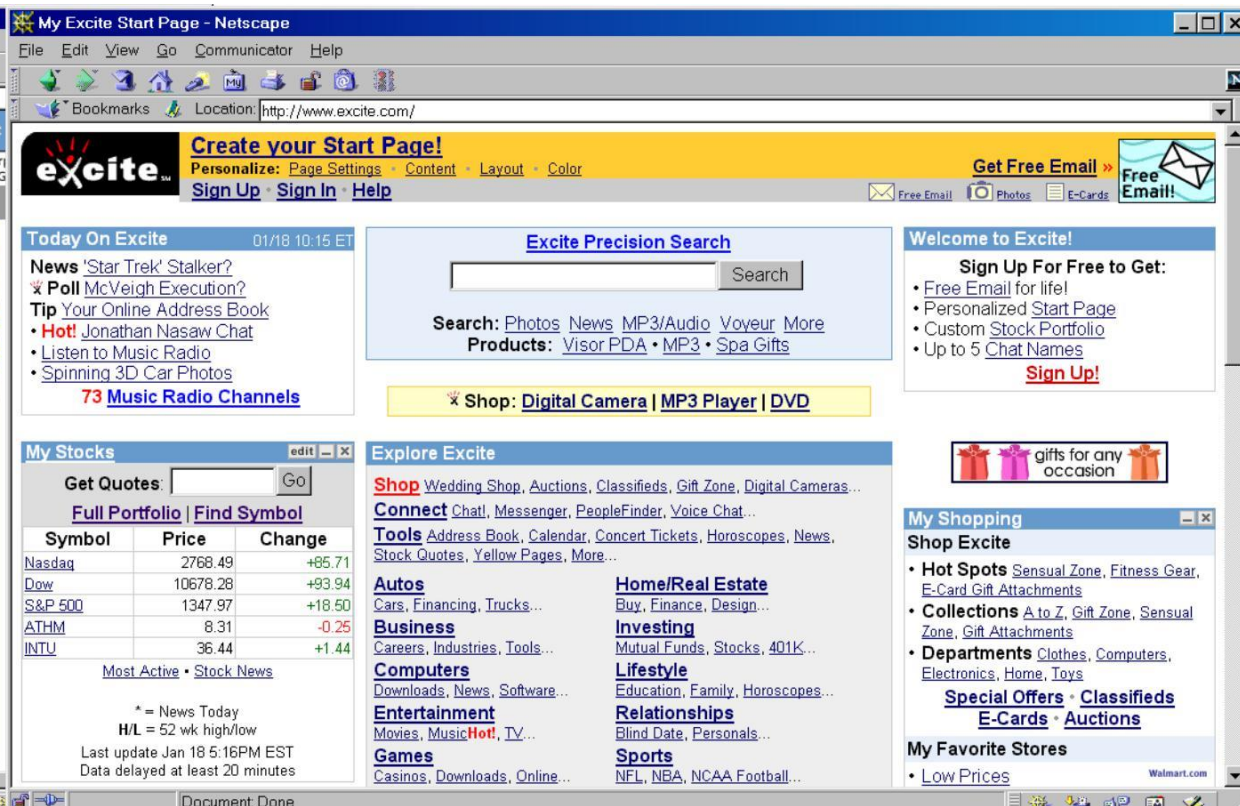
Wine Club Special

Join the wine.com Discovery Club for six months and receive an extra month free! Learn as you taste with our program of great wine selections and educational materials. Starts at only \$27.50 a month.

The People's Wine
With the approach of Labor Day -- the American people's holiday -- our thoughts naturally turn to the American people's grape: Zinfandel, our native contribution to the world of wine.

No question that White Zin is a wine for the masses. But as red Zin's popularity drives up its value and price, can it remain the wine of the people?

- Zinfandel goes upscale
- The truth about White Zin



My Excite Start Page - Netscape

File Edit View Go Communicator Help

Location: http://www.excite.com/

excite Create your Start Page!
Personalize: Page Settings · Content · Layout · Color
Sign Up · Sign In · Help

Get Free Email » Free Email! Photos E-Cards

Today On Excite 01/18 10:15 ET

News 'Star Trek' Stalker?
Poll McVeigh Execution?
Tip Your Online Address Book
Hot! Jonathan Nasaw Chat
Listen to Music Radio
Spinning 3D Car Photos
73 Music Radio Channels

Excite Precision Search

Search: [] Search

Search: Photos News MP3/Audio Voyeur More
Products: Visor PDA · MP3 · Spa Gifts

Shop: Digital Camera | MP3 Player | DVD

Welcome to Excite!

Sign Up For Free to Get:

- Free Email for life!
- Personalized Start Page
- Custom Stock Portfolio
- Up to 5 Chat Names

Sign Up!

gifts for any occasion

My Stocks edit [] x

Get Quotes: [] Go

Full Portfolio | Find Symbol

Symbol	Price	Change
Nasdaq	2768.49	+85.71
Dow	10678.28	+93.94
S&P 500	1347.97	+18.60
ATHM	8.31	-0.25
INTU	36.44	+1.44

Most Active · Stock News

* = News Today
H/L = 52 wk high/low
Last update Jan 18 5:16PM EST
Data delayed at least 20 minutes

Explore Excite

Shop Wedding Shop, Auctions, Classifieds, Gift Zone, Digital Cameras...
Connect Chat!, Messenger, PeopleFinder, Voice Chat...
Tools Address Book, Calendar, Concert Tickets, Horoscopes, News, Stock Quotes, Yellow Pages, More...

Autos
Cars, Financing, Trucks...

Business
Careers, Industries, Tools...

Computers
Downloads, News, Software...

Entertainment
Movies, MusicHot!, TV...

Games
Casinos, Downloads, Online...

Home/Real Estate
Buy, Finance, Design...

Investing
Mutual Funds, Stocks, 401K...

Lifestyle
Education, Family, Horoscopes...

Relationships
Blind Date, Personals...

Sports
NFL, NBA, NCAA Football...

My Shopping

Shop Excite

- Hot Spots Sensual Zone, Fitness Gear, E-Card Gift Attachments
- Collections A to Z, Gift Zone, Sensual Zone, Gift Attachments
- Departments Clothes, Computers, Electronics, Home, Toys
- Special Offers · Classifieds E-Cards · Auctions

My Favorite Stores

- Low Prices Walmart.com

Hidden / HTML Comment

An HTML comment is sent to the client's browser, but is not displayed. The information can be reviewed from the source code.

– `<!-- comment [<%= expression%>] -->`

A hidden comment is discarded before any processing of the JSP page and is not sent to the web browser.

– `<%-- comment -->`

JSP Components

There are three main types of JSP constructs that you embed in a page.

- **Scripting elements**

 - You can specify Java code

 - Expressions, Scriptlets, Declarations

- **Directives**

 - Let you control the overall structure of the servlet

 - Page, include, Tag library

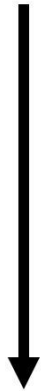
- **Actions**

 - Enable the use of server side Javabeans

 - Transfer control between pages

Uses of JSP Constructs: Use of Scripting elements

**Simple
Application**



**Complex
Application**

Scripting elements calling
servlet code directly

Scripting elements calling
servlet code indirectly (by
means of utility classes)

Beans

Custom tags

Servlet/JSP combo
(MVC architecture)

Types of Scripting Elements

You can insert code into the servlet that will be generated from the JSP page.

Expressions: `<%= expression %>`

- Evaluated and inserted into the servlet's output. i.e., results in something like `out.println(expression)`

Scriptlets: `<% code %>`

- Inserted verbatim into the servlet's `_jspService` method (called by service)

Declarations: `<%! code %>`

- Inserted verbatim into the body of the servlet class, outside of any existing methods

JSP Expressions

Format

- `<%= Java Expression %>`

Result

- Expression evaluated, converted to String, and placed into HTML page at the place it occurred in JSP page
- That is, expression placed in `_jspService` inside `out.print`

Examples

- Current time: `<%= new java.util.Date() %>`
- Your hostname: `<%= request.getRemoteHost() %>`

XML-compatible syntax

- `<jsp:expression>Java Expression</jsp:expression>`
- XML version not supported by Tomcat 3. Until JSP 1.2, servers are not required to support it.

JSP/Servlet Correspondence

Original JSP

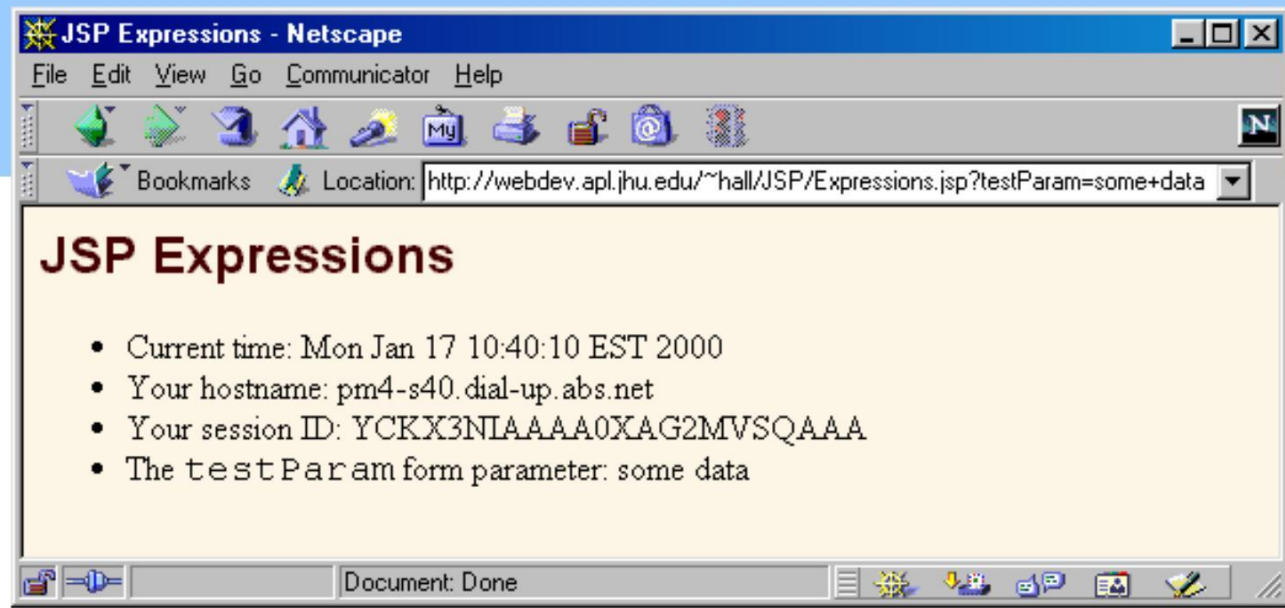
```
<H1>A Random Number</H1>  
<%= Math.random() %>
```

Possible resulting servlet code

```
public void _jspService(HttpServletRequest request,  
                        HttpServletResponse response)  
    throws ServletException, IOException {  
    request.setContentType("text/html");  
    HttpSession session = request.getSession(true);  
    JspWriter out = response.getWriter();  
    out.println("<H1>A Random Number</H1>");  
    out.println(Math.random());  
    ...  
}
```

Example Using JSP Expressions

```
<BODY>
<H2>JSP Expressions</H2>
<UL>
  <LI>Current time: <%= new java.util.Date() %>
  <LI>Your hostname: <%= request.getRemoteHost() %>
  <LI>Your session ID: <%= session.getId() %> <LI>The
  <CODE>testParam</CODE> form parameter:
    <%= request.getParameter("testParam") %>
</UL>
</BODY>
```



Predefined Variables (Implicit Objects)

They are created automatically when a web server processes a JSP page.

request: The `HttpServletRequest` (1st arg to `doGet`)

response: The `HttpServletResponse` (2nd arg to `doGet`)

session

- The `HttpSession` associated with the request (unless disabled with the `session` attribute of the page directive)

out

- The stream (of type `JspWriter`) used to send output to the client

application

- The `ServletContext` (for sharing data) as obtained via `getServletConfig().getContext()`.

page, pageContext, config, exception

Implicit objects – Class files

application: javax.servlet.ServletContext

config: javax.servlet.ServletConfig

exception: java.lang.Throwable

out: javax.servlet.jsp.JspWriter

page: java.lang.Object

pageContext: javax.servlet.jsp.PageContext

request: javax.servlet.ServletRequest

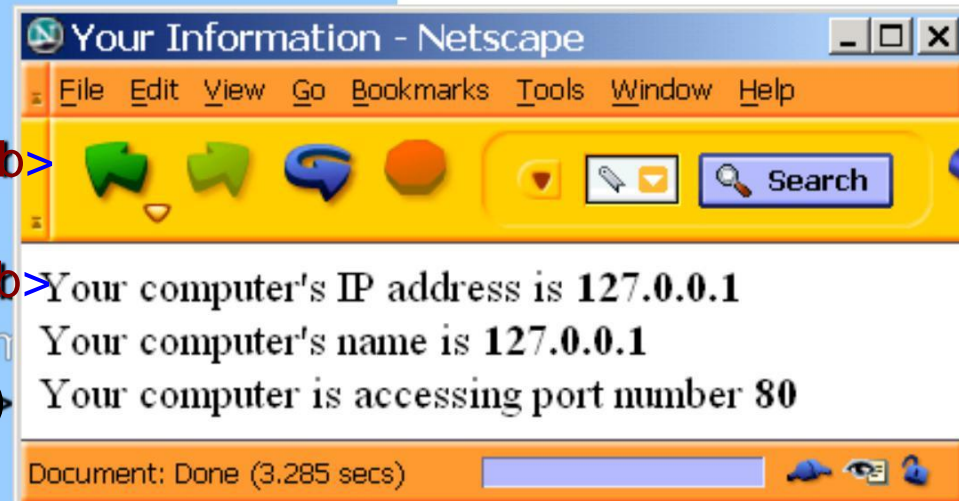
response: javax.servlet.ServletResponse

session: javax.servlet.http.HttpSession

Access Client Information

The `getRemoteHost` method of the request object allows a JSP to retrieve the name of a client computer.

```
<html><head>
<title>Your Information</title>
</head><body>
Your computer's IP address is
<b><%= request.getRemoteAddr() %></b>
<br>Your computer's name is
<b><%= request.getRemoteHost() %></b>
<br>Your computer is accessing port
number <b><%= request.getServerPort()
%></b>
</body></html>
```



Work with the Buffer

When the page is being processed, the data is stored in the buffer instead of being directly sent to the client browser.

```
<html>
This is a test of the buffer<br/>
<%
out.flush();
for (int x=0; x < 100000000; x++);
out.print("This test is generated about 5
seconds later.");
out.flush();
%>
</html>
```

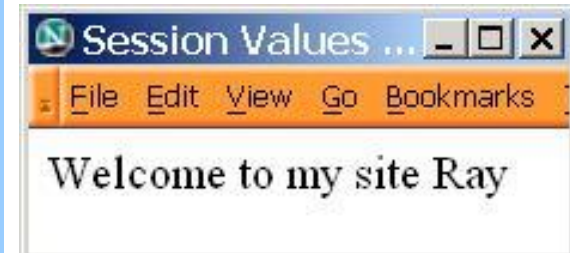
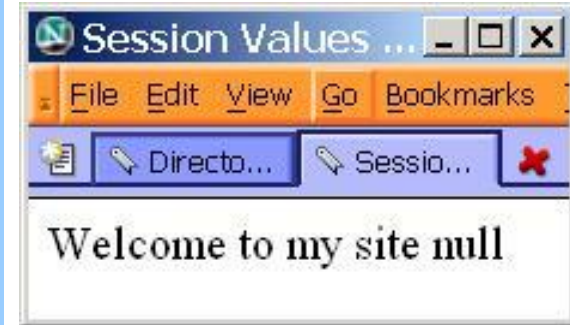


Working with Session object

The session object has many useful methods that can alter or obtain information about the current session.

– `setMaxInactiveInterval(second)`

```
<html><head>
<title>Session Values</title>
</head><body>
<%
session.setMaxInactiveInterval(10);
String name = (String)
    session.getAttribute("username");
out.print("Welcome to my site " + name + "<br>");
"%<br>"); %>
</body></html>
```



JSP Scriptlets

Format: `<% Java Code %>`

Result

- Code is inserted verbatim into servlet's `_jspService`

Example

- `<%
String queryData = request.getQueryString();
out.println("Attached GET data: " +
queryData); %>`
- `<% response.setContentType("text/plain"); %>`

XML-compatible syntax

- `<jsp:scriptlet>Java Code</jsp:scriptlet>`

JSP/Servlet Correspondence

Original JSP

```
<%= foo() %>  
<% bar(); %>
```

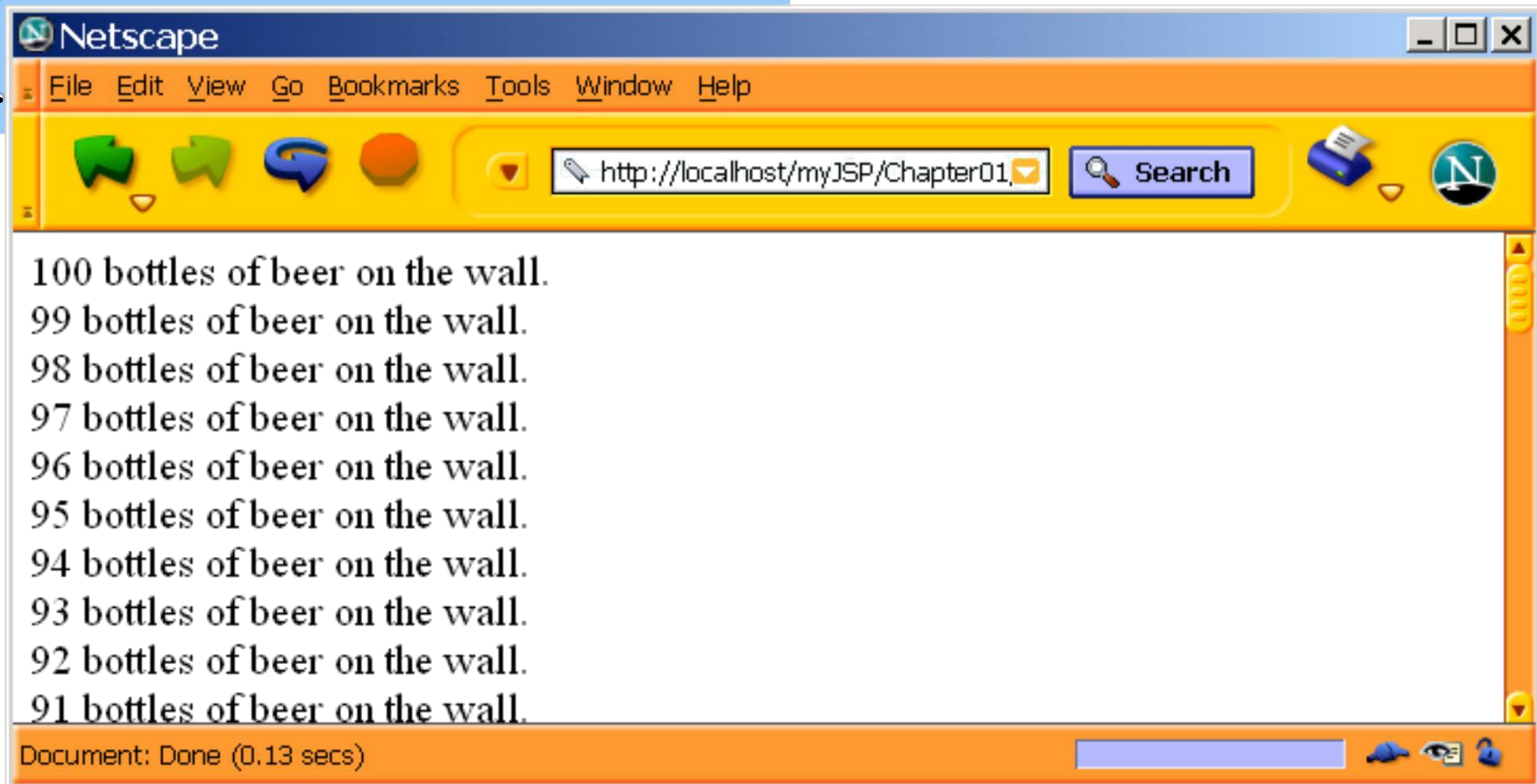
Possible resulting servlet code

```
public void _jspService(HttpServletRequest request,  
                        HttpServletResponse response)  
    throws ServletException, IOException {  
    request.setContentType("text/html");  
    HttpSession session =  
    request.getSession(true); JspWriter out =  
    response.getWriter(); out.println(foo());  
bar();  
    ...  
}
```

```
<%  
for (int i=100; i>=0; i--)  
{  
%>  
  <%= i %> bottles of beer on the  
  the wall.<br>
```

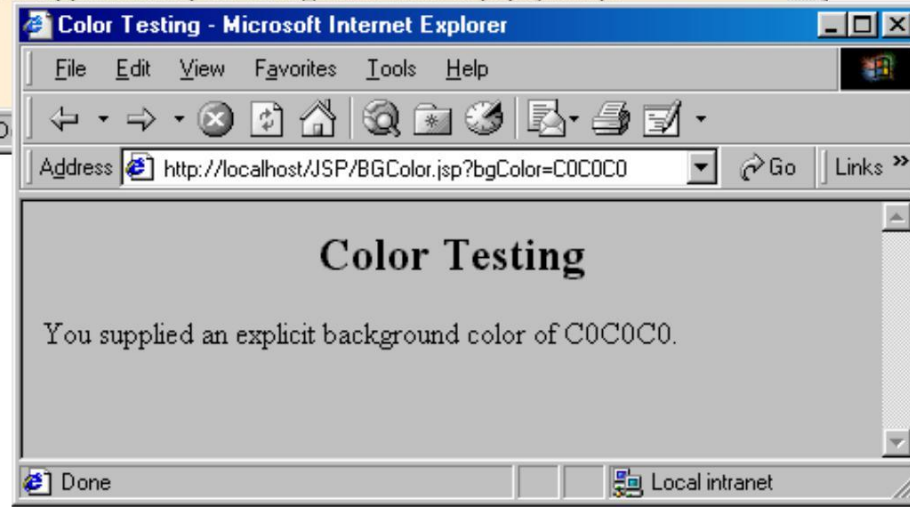
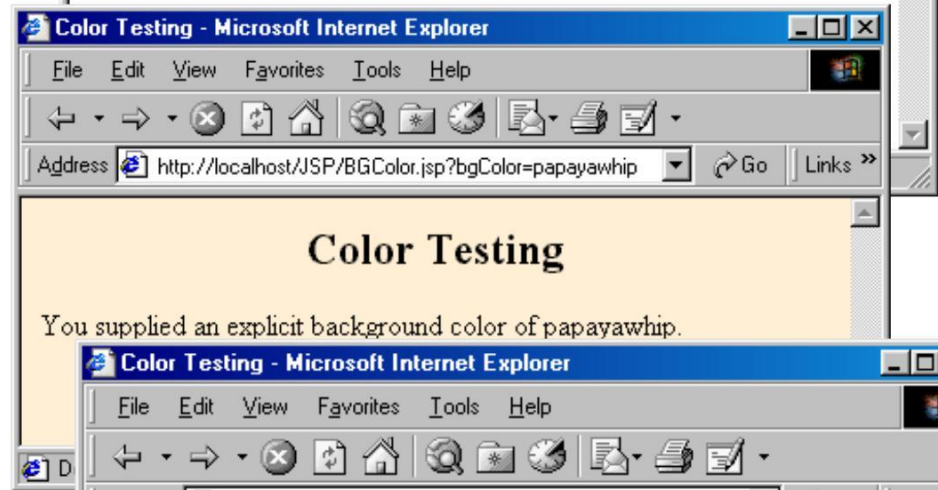
```
<%  
}  
%>
```

JSP Scriptlets Example



Example Using JSP Scriptlets

```
<HTML>
<HEAD>
  <TITLE>Color Testing</TITLE>
</HEAD>
<%
String bgColor =
  request.getParameter("bgColor");
boolean hasExplicitColor;
if (bgColor != null) {
  hasExplicitColor =
true; } else {
  hasExplicitColor = false;
  bgColor = "WHITE";
}
%>
<BODY BGCOLOR="<%= bgColor %>">
```



JSP Declarations

Format

- `<%! Java Code %>`

Result

- Code is inserted verbatim into servlet's class definition, outside of any existing methods

Examples

- `<%! private int someField = 5; %>`
- `<%! private void someMethod(...) {...} %>`

XML-compatible syntax

- `<jsp:declaration>Java Code</jsp:declaration>`

Scriptlets vs. Declarations

```
<%! int count=100; %>
%><%=++count %>
```

```
public final class
    _scopeExpermnt1_xjsp
{
    int count=100;

    public void _jspService
        (HttpServletRequest request,
         HttpServletResponse response)
        throws java.io.IOException
    {
        JspWriter out =
            pageContext.getOut();

        out.print( "\r\n" );

        out.print( String.valueOf( ++
            count ) );
        out.print( "\r\n" );
    }
}
```

```
<% int count=100; %>
%><%=++count %>
```

```
public final class
    _scopeExpermnt2_xjsp
{
    public void _jspService
        (HttpServletRequest request,
         HttpServletResponse response)
        throws java.io.IOException
    {
        JspWriter out =
            pageContext.getOut();

        int count=100;

        out.print( "\r\n" );
        out.print( String.valueOf(
            ++count ) ); out.print(
            "\r\n" );
    }
}
```

Original JSP

```
<H1>Some Heading</H1>
<%!
    private String randomHeading() {
        return("<H2>" + Math.random() + "</H2>");
    }
%>
<%= randomHeading() %>
```

Possible resulting servlet code

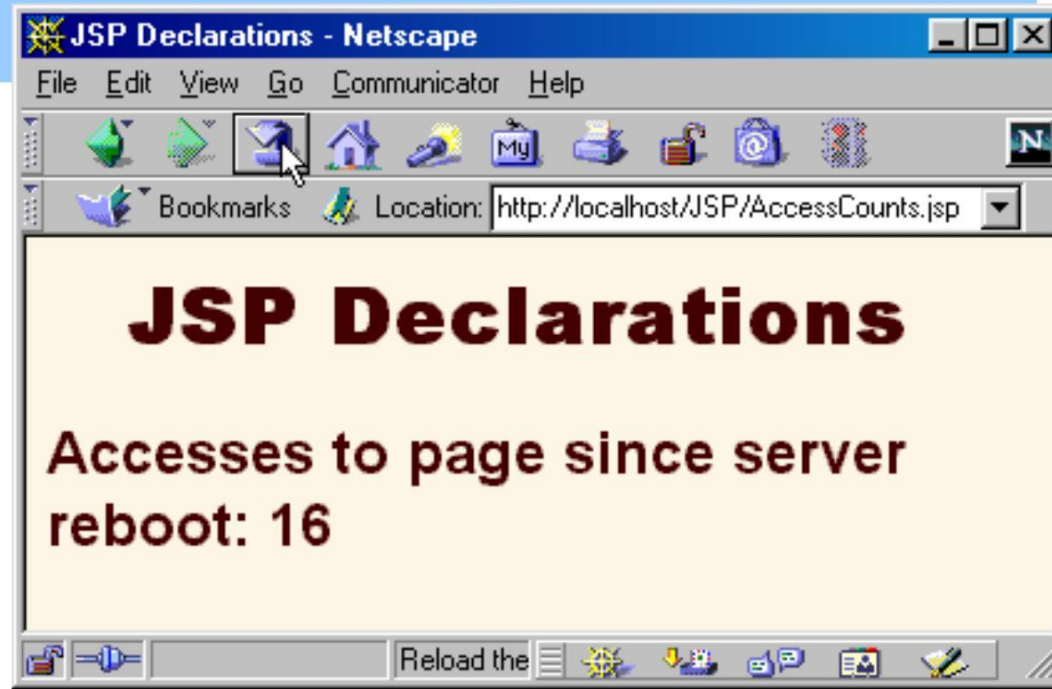
```
public class xxxx implements HttpJspPage
{ private String randomHeading() {
} return("<H2>" + Math.random() + "</H2>");
public void _jspService(HttpServletRequest request,
    HttpServletResponse response) throws ServletException, IOException {
    request.setContentType("text/html");
    HttpSession session =
    request.getSession(true); JspWriter out =
    response.getWriter(); out.println("<H1>Some
    Heading</H1>"); out.println(randomHeading());
} ...
```

JSP/Servlet Correspondence

Example Using JSP Declarations

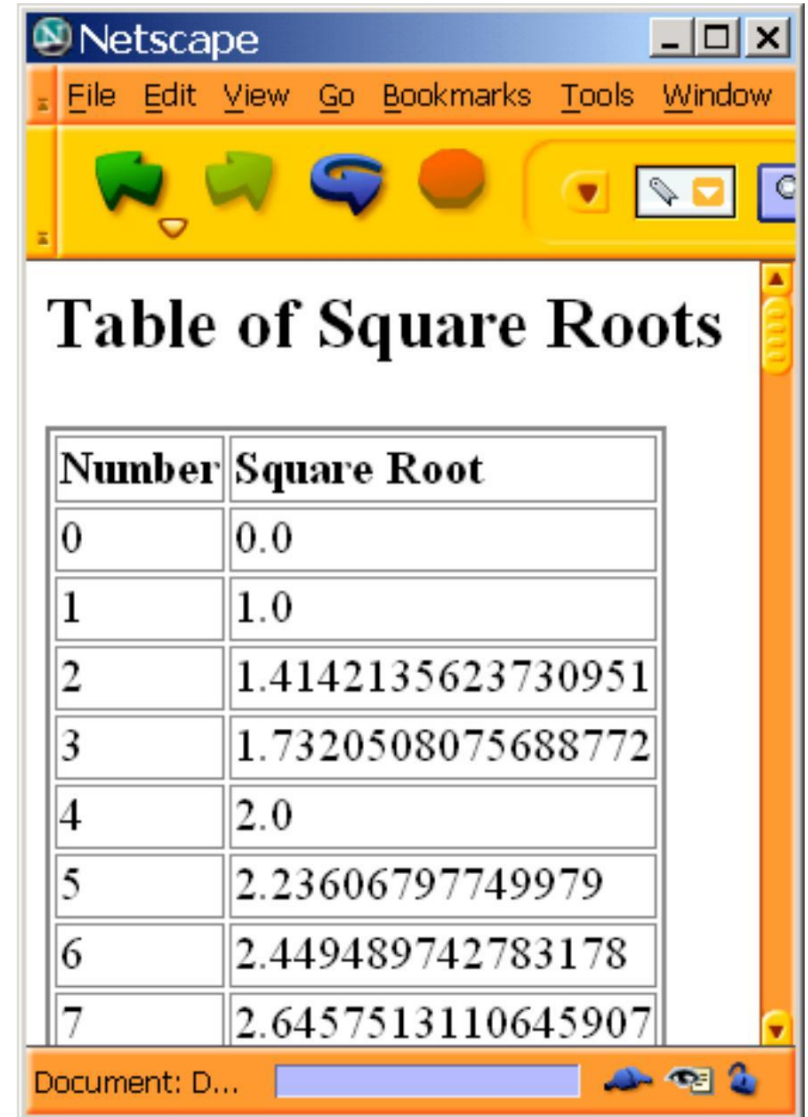
```
...  
<body>  
<h1>JSP Declarations</h1>  
<%! private int accessCount = 0; %>  
<h2>Accesses to page since server reboot:  
<%= ++accessCount %></h2>  
</body></html>
```

- After 15 total visits by an arbitrary number of different clients



JSP Tags + HTML Tags

```
<h2>Table of Square Roots</h2>
<table border=2>
  <tr>
    <td><b>Number</b></td>
    <td><b>Square Root</b></td>
  </tr>
  <%
for (int n=0; n<=100; n++)
{
%>
  <tr>
    <td><%=n%></td>
    <td><%=Math.sqrt(n)%></td>
  </tr>
  <%
}
%>
</table>
```



The screenshot shows a Netscape browser window with the title "Table of Square Roots". The browser's address bar is empty, and the document content is a table with two columns: "Number" and "Square Root". The table contains data for numbers 0 through 7, with the square root values rounded to 10 decimal places. The browser's interface includes a menu bar (File, Edit, View, Go, Bookmarks, Tools, Window), a toolbar with navigation buttons (back, forward, home, stop), and a status bar at the bottom showing "Document: D...".

Number	Square Root
0	0.0
1	1.0
2	1.4142135623730951
3	1.7320508075688772
4	2.0
5	2.23606797749979
6	2.449489742783178
7	2.6457513110645907

JSP Directives

Affect the overall structure of the servlet

Two possible forms for directives

- `<%@ directive attribute="value" %>`
- `<%@ directive attribute1="value1"
attribute2="value2"
....
attributeN="valueN" %>`

There are three types of directives

- Page, include, and taglib

Purpose of the page Directive

Give high-level information about the servlet that will result from the JSP page

Can control

- Which classes are imported
- What class the servlet extends
- What MIME type is generated
- How multithreading is handled
- If the servlet participates in sessions
- The size and behavior of the output buffer
- What page handles unexpected errors

The import Attribute

Format

- `<%@ page import="package.class" %>`
- `<%@ page import="package.class1,...,package.classN" %>`

Purpose

- Generate import statements at top of servlet

Notes

- Although JSP pages can be almost anywhere on server, classes used by JSP pages must be in normal servlet dirs
- For Tomcat, this is `install_dir/webapps/ROOT/WEB-INF/classes` or `.../ROOT/WEB-INF/classes/directoryMatchingPackage`

Example of import Attribute

```
...
<BODY><H2>The import Attribute</H2>
<%-- JSP page directive --%>

<%@ page import="java.util.*,cwp.*" %>
<%-- JSP Declaration --%>
<%!
private String randomID() {

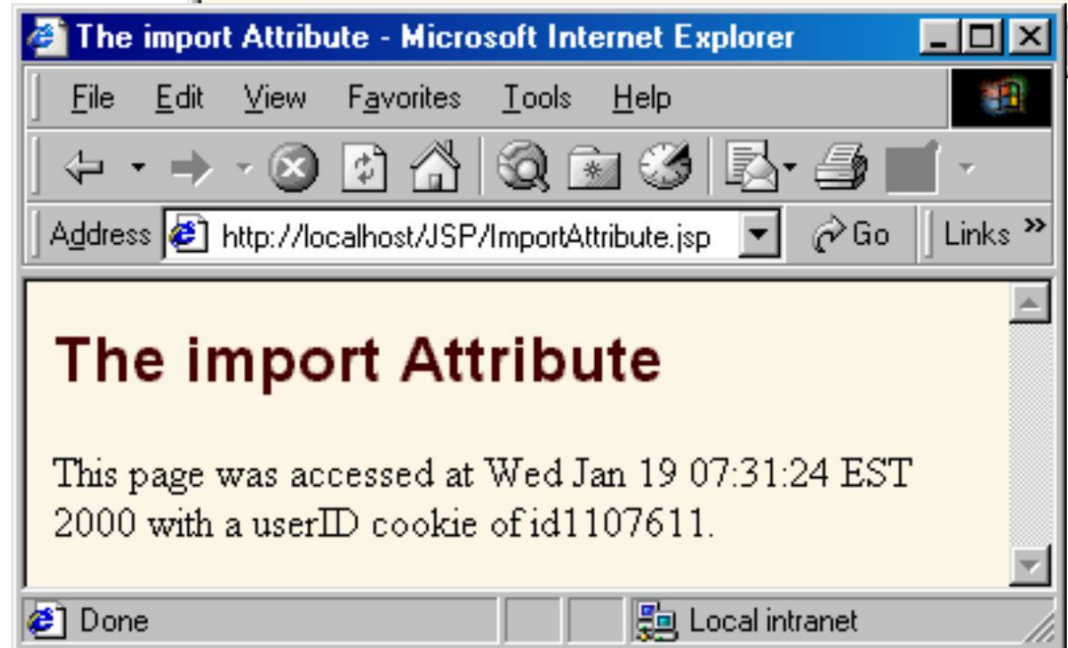
    int num = (int)(Math.random()*100000000.0); }
return("id" + num);
private final String NO_VALUE = "<I>No Value</I>";
%>
<%
Cookie[] cookies = request.getCookies();
String oldID = ServletUtilities.getCookieValue(cookies, "userID",
    NO_VALUE);
String newID;
if (oldID.equals(NO_VALUE)) { newID = randomID(); }
else { newID = oldID; }
LongLivedCookie cookie = new LongLivedCookie("userID", newID);
response.addCookie(cookie);
%>
<%-- JSP Expressions --%>
This page was accessed at <%= new Date() %> with a userID
cookie of <%= oldID %>.
</BODY></HTML>
```

Example of import Attribute

First access



Subsequent accesses



The contentType Attribute

Format

- `<%@ page contentType="MIME-Type" %>`
- `<%@ page contentType="MIME-Type;
charset=Character-Set"%>`

Purpose

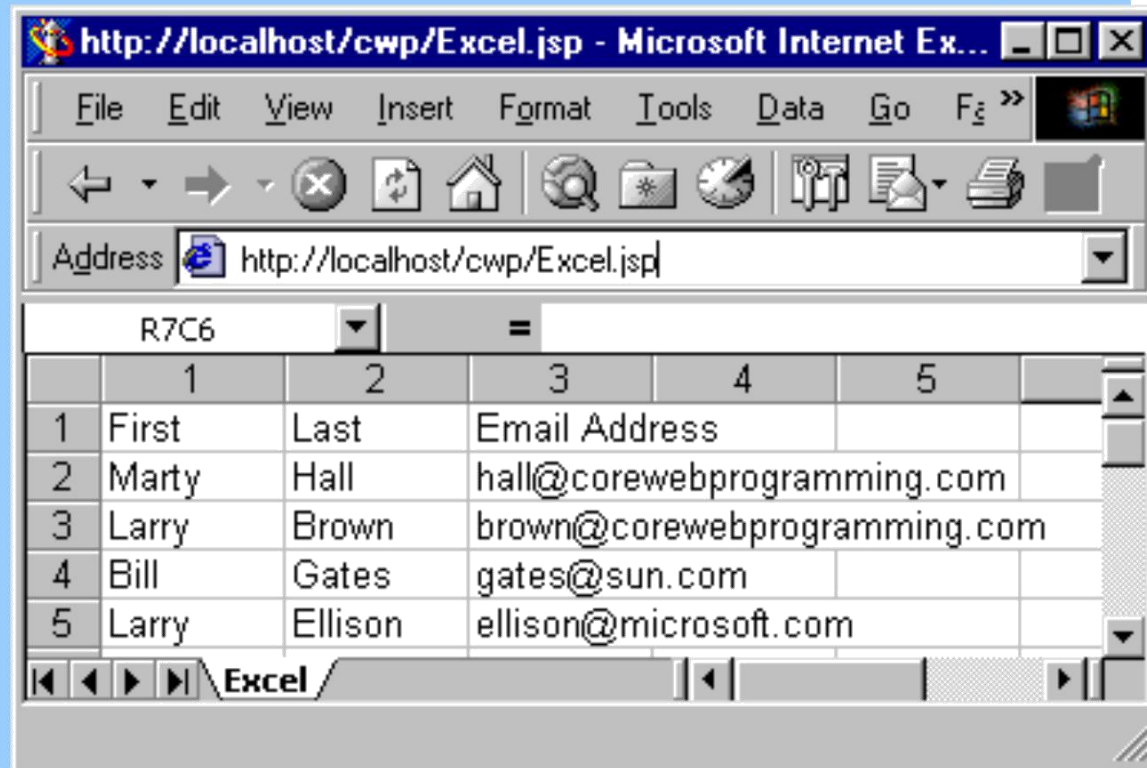
- Specify the MIME type of the page generated by the servlet that results from the JSP page

```
First    Last    Email Address
Marty   Hall    hall@corewebprogramming.com
Larry   Brown   brown@corewebprogramming.com
Bill    Gates   gates@sun.com
Larry   Ellison ellison@microsoft.com
```

```
<%@ page contentType="application/vnd.ms-excel" %>
```

```
<!-- There are tabs, not spaces, between columns. --%>
```

Generating Excel Spreadsheets



Another Example

Comparing Apples and Oranges - Microsoft Int...

File E » Back » Google

Address <http://localhost/jsp/ApplesAndOranges.jsp> Go

Comparing Apples and Oranges

	Apples
First Quarter	2307
Second Quarter	2982
Third Quarter	3011
Fourth Quarter	3055

Done

<http://localhost/jsp/ApplesAndOranges.jsp?for...>

File E » Back » Google

Address <http://localhost/jsp/ApplesAndOranges.jsp?format=excel> Go

A1 = Comparing Apples and Oranges

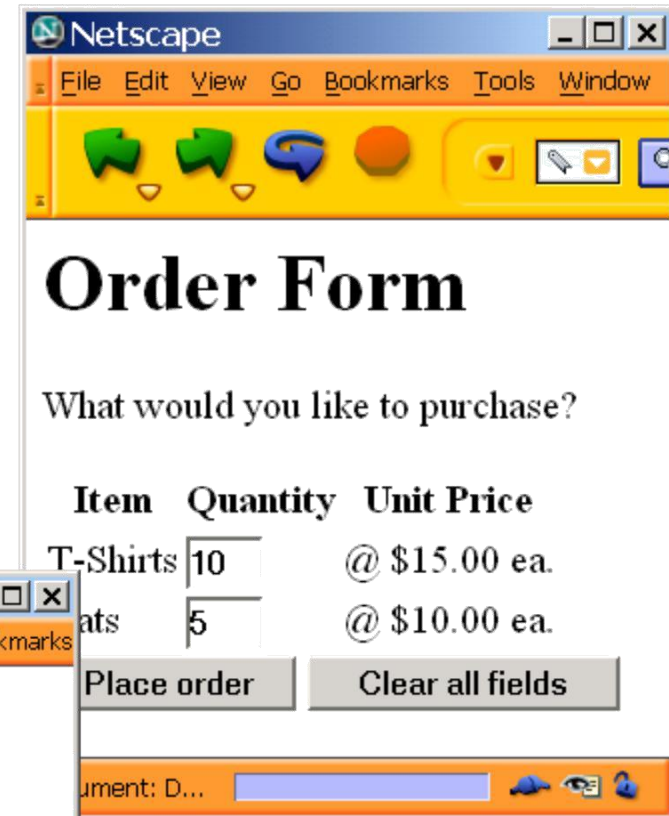
	A	B	C	D	E	F	G
1	Comparing Apples and Oranges						
2							
3		Apples	Oranges				
4	First Quarter	2307	4706				
5	Second Quarter	2982	5104				
6	Third Quarter	3011	5220				
7	Fourth Quarter	3055	5287				
8							
9							
10							

ApplesAndOranges.jsp format=ex

Unknown Zone

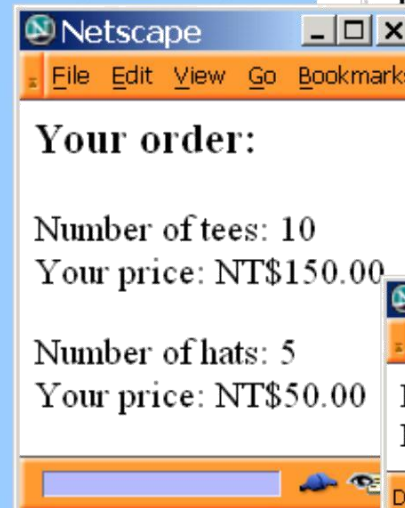
Form Processing

```
<%-- processOrder.jsp --%>
<%@ page errorPage="orderError.jsp"
    import="java.text.NumberFormat" %>
<h3 >Your order:
order:</h3> <%
    String numTees = request.getParameter("t-shirts");
    String numHats = request.getParameter("hats");
    NumberFormat currency =
        NumberFormat.getCurrencyInstance();
%>
Number of tees:
    <%= numTees %><br>
Your price:
    <%= currency.format(Integer.parseInt(numTees)*15.00)
    %><p>
Number of hats:
    <%= numHats %><br>
Your price:
    <%= currency.format(Integer.parseInt(numHats)*10.00)
    %><p>
<!-- orderForm.htm -->
<h1>Order Form</h1>
What would you like to purchase?<p>
<form name=orders
action=processOrder.jsp> <table border=0>
    <tr><th>Item</th>
    <th>Quantity</th>
    <th>Unit Price</th>
    <tr><tr>
```

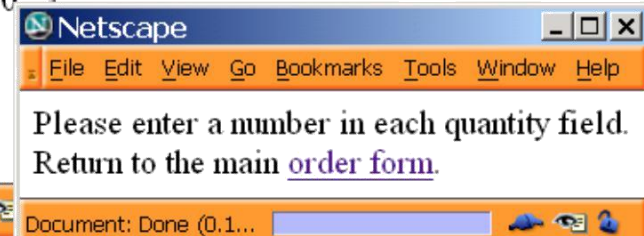


The screenshot shows a Netscape browser window displaying the 'Order Form'. The form has been successfully processed, and the results are shown below the heading 'Order Form'. The text reads: 'What would you like to purchase?'. Below this, there is a table with three columns: 'Item', 'Quantity', and 'Unit Price'. The table contains two rows: 'T-Shirts' with a quantity of 10 and a unit price of '\$15.00 ea.', and 'Hats' with a quantity of 5 and a unit price of '\$10.00 ea.'. Below the table, there are two buttons: 'Place order' and 'Clear all fields'. The browser's address bar shows 'Document: D...'.

Item	Quantity	Unit Price
T-Shirts	10	@ \$15.00 ea.
Hats	5	@ \$10.00 ea.



The screenshot shows a Netscape browser window displaying the results of the order form. The text reads: 'Your order:'. Below this, there are two lines of text: 'Number of tees: 10' and 'Your price: NT\$150.00'. Below this, there are two more lines of text: 'Number of hats: 5' and 'Your price: NT\$50.00'. The browser's address bar shows 'Document: Done (0.1...'.



The screenshot shows a Netscape browser window displaying an error message. The text reads: 'Please enter a number in each quantity field. Return to the main [order form](#).' The browser's address bar shows 'Document: Done (0.1...'.

Other Attributes of the page Directive

session

- Lets you choose not to participate in sessions

buffer

- Changes min size of buffer used by JspWriter

autoflush

- Requires developer to explicitly flush buffer

extends

- Changes parent class of generated servlet

errorPage

- Designates a page to handle unplanned errors

isErrorPage, isThreadSafe, language, ...

Break Time – 15 minutes



JSP Actions

There are *seven* standard JSP actions.

- Include, param, forward, plugin, ...
- Include action is similar to include directive.
- You can add additional parameters to the existing request by using the param action.
- The plugin action inserts object and embed tags (such as an applet) into the response to the client.
- In the coming slides, we will talk about “include” and “plugin” actions.

Including Pages at Request Time

Format

- `<jsp:include page="Relative URL" flush="true" />`

Purpose

- To reuse JSP, HTML, or plain text content
- JSP content cannot affect main page:
only output of included JSP page is used
- To permit updates to the included content
without changing the main JSP page(s)

Including Pages: Example Code

```
...
<BODY>
<TABLE BORDER=5 ALIGN="CENTER">
  <TR><TH CLASS="TITLE">
    What's New at JspNews.com</TABLE>
<P>
Here is a summary of our three most recent news stories:
<OL>
  <LI><jsp:include page="news/Item1.html" flush="true" />
  <LI><jsp:include page="news/Item2.html" flush="true" />
  <LI><jsp:include page="news/Item3.html" flush="true" />
</OL>
</BODY></HTML>
```

Including Pages: Result



Including Files at Page Translation Time

Format

- `<%@ include file="Relative URL" %>`

Purpose

- To reuse JSP content in multiple pages, where JSP content affects main page

Notes

- Servers are not required to detect changes to the included file, and in practice many don't
- Thus, you need to change the JSP files whenever the included file changes
- You can use OS-specific mechanisms such as the Unix "touch" command, or

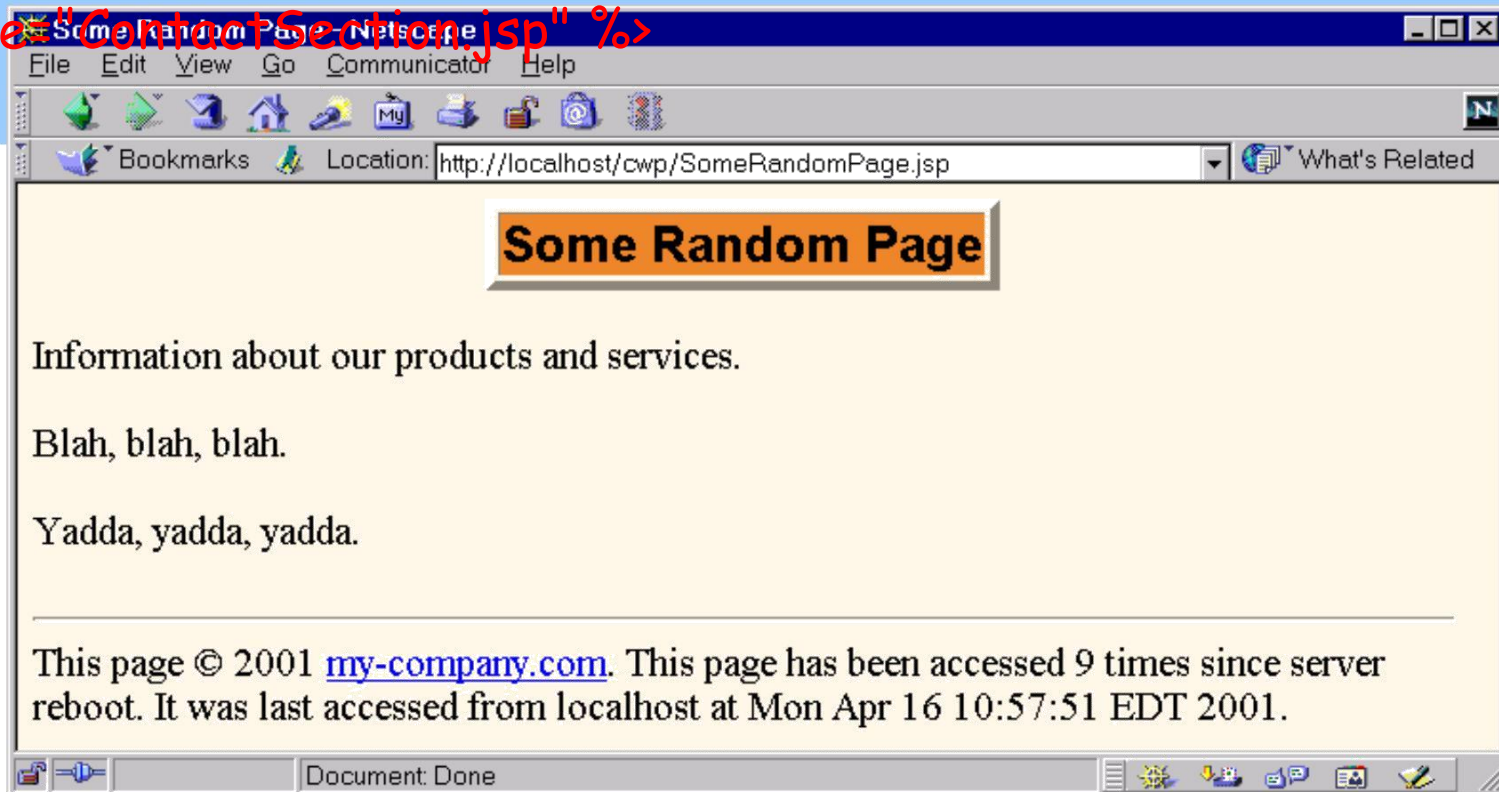
```
<%-- Navbar.jsp modified 3/1/02 --%>  
<%@ include file="Navbar.jsp" %>
```

Reusable JSP Content: ContactSection.jsp

```
<%@ page import="java.util.Date" %>
<!-- The following become fields in each servlet that
      results from a JSP page that includes this file. -->
<%!
private int accessCount = 0;
private Date accessDate = new Date();
private String accessHost = "<I>No previous access</I>";
%>
<P><HR>
This page &copy; 2000
<A HREF="http://www.my-company.com/">my-company.com</A>.
This page has been accessed <%= ++accessCount %>
times since server reboot. It was last accessed from
<%= accessHost %> at <%= accessDate %>.
<% accessHost = request.getRemoteHost(); %>
<% accessDate = new Date(); %>
```

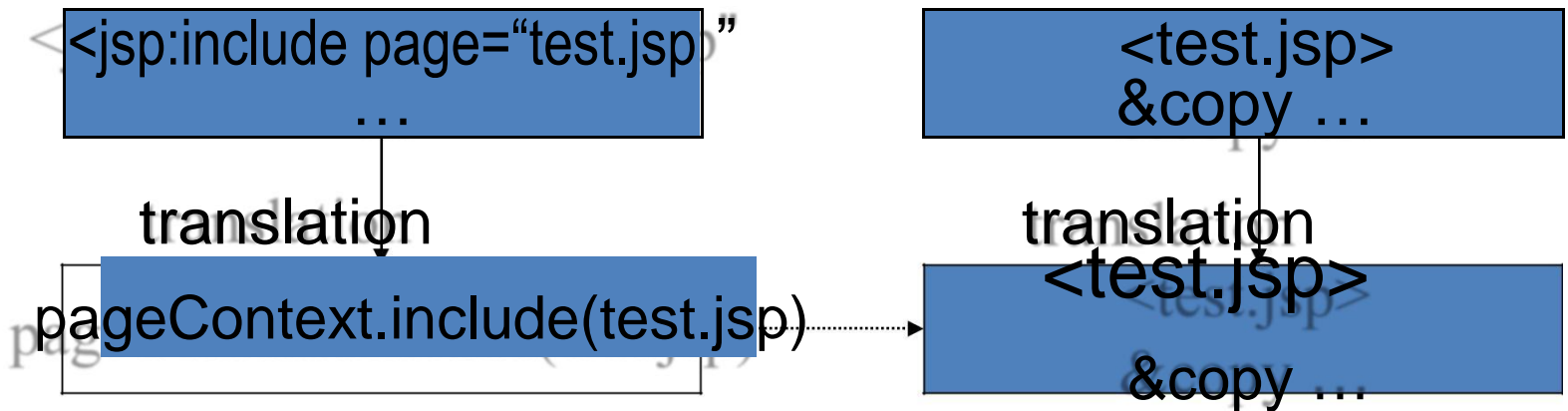
Using the JSP

```
...  
<BODY>  
<TABLE BORDER=5 ALIGN="CENTER">  
  
  <TR><TH CLASS="TITLE">Content Some Random  
    Page</TH></TR></TABLE>  
  
<P> Information about our products and services.  
<P> Blah, blah, blah.  
<P> Yadda, yadda, yadda.  
<%@ include file="ContentSection.jsp" %>  
</BODY>  
</HTML>
```

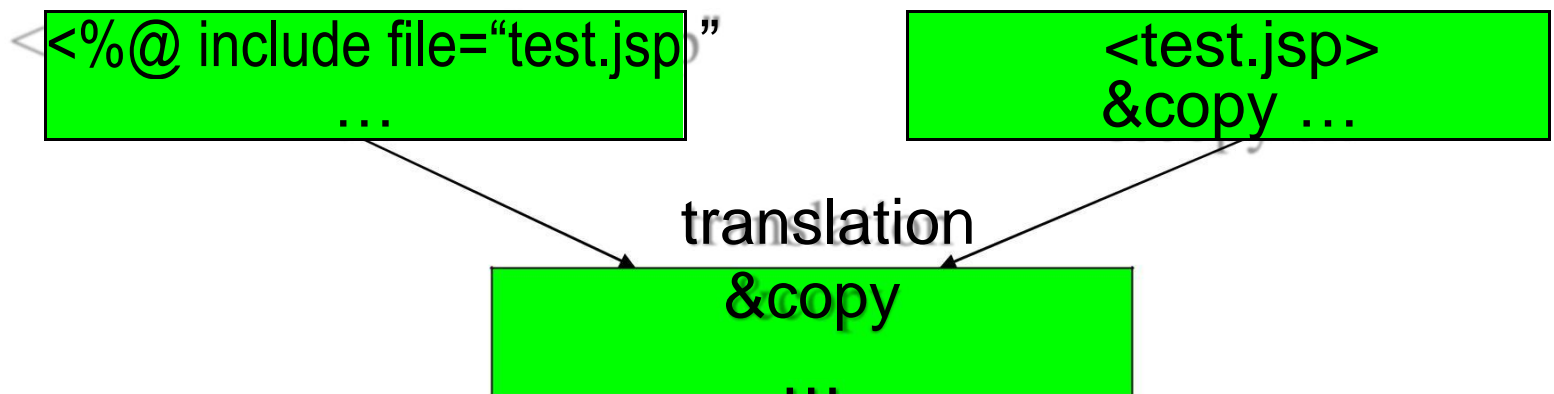


Include directive vs. Include action

A request-time include action



A translation-time include directive



The plugin action's attribute

```
<jsp:plugin type="applet" code="myBox"  
  codebase="path/myClass" width="200"  
  height=200">... params </jsp:plugin>
```

We usually use

- type: to specify we place an applet or others onto a web page.
- Code: to give the name of the Java class to be run.
- Width/Height: to define the size of the rectangle set aside for displaying the applet in the browser's window.

jsp:forward action

Used to instruct a web server to stop processing the current page and start another one.

```
<jsp:forward page="another.jsp">
```

```
<jsp:param name="callingPage"  
value="current.jsp"> </jsp:forward>
```

– Another page:

```
<%= request.getParameter("callingPage") %>
```

Returns "current.jsp"

Uses of JSP Constructs: Using JavaBeans

**Simple
Application**



**Complex
Application**

Scripting elements calling
servlet code directly

Scripting elements calling
servlet code indirectly (by
means of utility classes)

Beans

Custom tags

Servlet/JSP combo
(MVC architecture)

Background: What Are Beans?

Classes that follow certain conventions

- Must have a zero-argument (empty) constructor
- Should have no public instance variables (fields)
- Persistent values should be accessed through methods called getXxx and setXxx

If class has method getTitle that returns a String, class is said to have a String property named title

Boolean properties use isXxx instead of getXxx

For more on beans, see

<http://java.sun.com/beans/docs/>

Basic Bean Use in JSP

Format: `<jsp:useBean id="name"
class="package.Class" />`

Purpose: Allow instantiation of classes without explicit Java syntax

Notes

- Simple interpretation: JSP action
`<jsp:useBean id="book1" class="cwp.Book" />`
can be thought of as equivalent to the scriptlet
`<% cwp.Book book1 = new cwp.Book(); %>`
- But useBean has two additional features
 - Simplifies setting fields based on incoming request params
 - Makes it easier to share beans

Accessing Bean Properties

Format: `<jsp:getProperty name="name" property="property" />`

Purpose: Allow access to bean properties (i.e., calls to getXxx methods) without explicit Java code

Notes

– `<jsp:getProperty name="book1" property="title" />`

is equivalent to the following JSP expression

`<%= book1.getTitle() %>`

Setting Bean Properties: Simple Case

Format: `<jsp:setProperty name="name" property="property" value="value" />`

Purpose

- Allow setting of bean properties (i.e., calls to `setXxx`) without explicit Java code

Notes

- `<jsp:setProperty name="book1" property="title" value="Core Servlets and JSP" />`

is equivalent to the following scriptlet

```
<% book1.setTitle("Core Servlets and JSP"); %>
```

Example: StringBean

```
public class StringBean {  
    private String message = "No message specified";  
    public String getMessage() {  
        return(message);  
    }  
    public void setMessage(String message)  
        { this.message = message;  
    }  
}
```

Installed in normal servlet directory

```
<jsp:useBean id="stringBean" class="cwp.StringBean" />
```

```
<OL>
```

```
<LI>Initial value (getProperty):
```

```
<I><jsp:getProperty name="stringBean"  
    property="message" /></I>
```

```
<LI>Initial value (JSP expression): <I><%=  
    stringBean.getMessage() %></I>
```

```
<LI><jsp:setProperty name="stringBean"  
    property="message"  
    value="Best string bean: Fortex" />
```

```
Value after setting property with setProperty:
```

```
<I><jsp:getProperty name="stringBean"  
    property="message" /></I>
```

```
<LI>
```

```
<%= stringBean.setMessage("My favorite: Kentucky Wonder"); %>
```

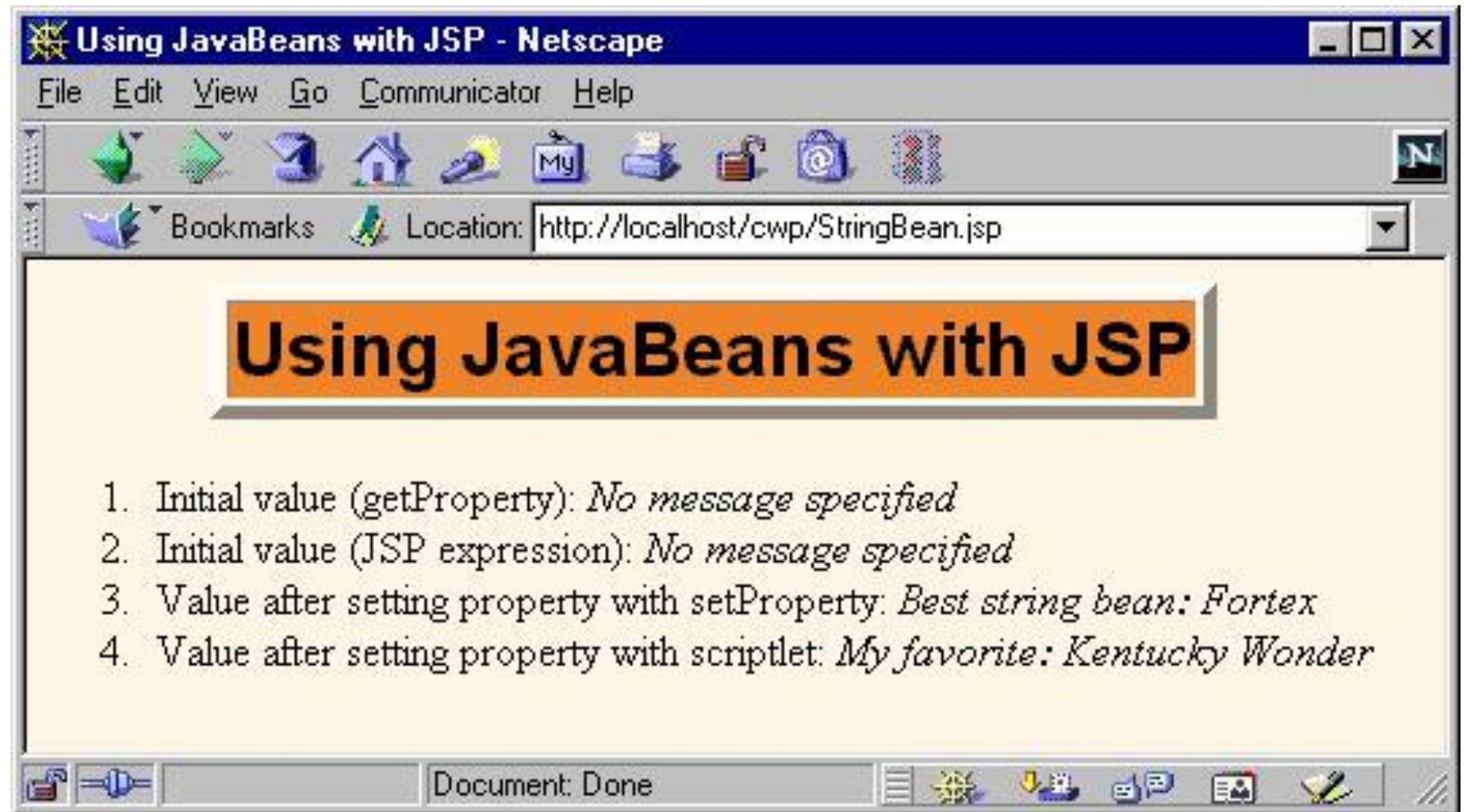
```
Value after setting property with scriptlet:
```

```
<I><%= stringBean.getMessage() %></I>
```

```
</OL>
```

JSP Page That
Uses
StringBean

JSP Page That Uses StringBean



Associating Bean Properties with Request (Form) Parameters

If property is a String, you can do

- `<jsp:setProperty ... value='<%= request.getParameter("...") %>' />`

Scripting expressions let you convert types, but you have to use Java syntax

The **param** attribute indicates that:

- Value should come from specified request param
- Simple automatic type conversion performed

Using **"*"** for the property attribute indicates that:

- Value should come from request parameter whose name matches property name
- Simple type conversion should be performed

Setting Bean Properties Case 1: Explicit Conversion & Assignment

```
<!DOCTYPE ...>
```

```
...
```

```
<jsp:useBean id="entry"  
             class="cwp.SaleEntry" />
```

```
<%-- getItemID expects a String --%>
```

```
<jsp:setProperty  
  name="entry"  
  property="itemID"  
  value='<%=
```



```
request.getParameter("itemID") %>' />
```

Setting Bean Properties Case 1: Explicit Conversion & Assignment

```
<%  
int numItemsOrdered = 1;  
try {  
    numItemsOrdered =  
        Integer.parseInt(request.getParameter("numItems"));  
} catch(NumberFormatException nfe) {}  
%>  
<%-- getNumItems expects an int --%>  
<jsp:setProperty  
    name="entry"  
    property="numItems"  
    value="<%= numItemsOrdered %>" />
```

Setting Bean Properties Case 1: Explicit Conversion & Assignment

```
<%  
double discountCode = 1.0;  
try {  
    String discountString =  
        request.getParameter("discountCode");  
    discountCode =  
        Double.valueOf(discountString).doubleValue();  
} catch(NumberFormatException nfe) {} %>  
  
<%-- getDiscountCode expects a double --%>  
<jsp:setProperty  
    name="entry"  
    property="discountCode"  
    value="<%= discountCode %>" />
```

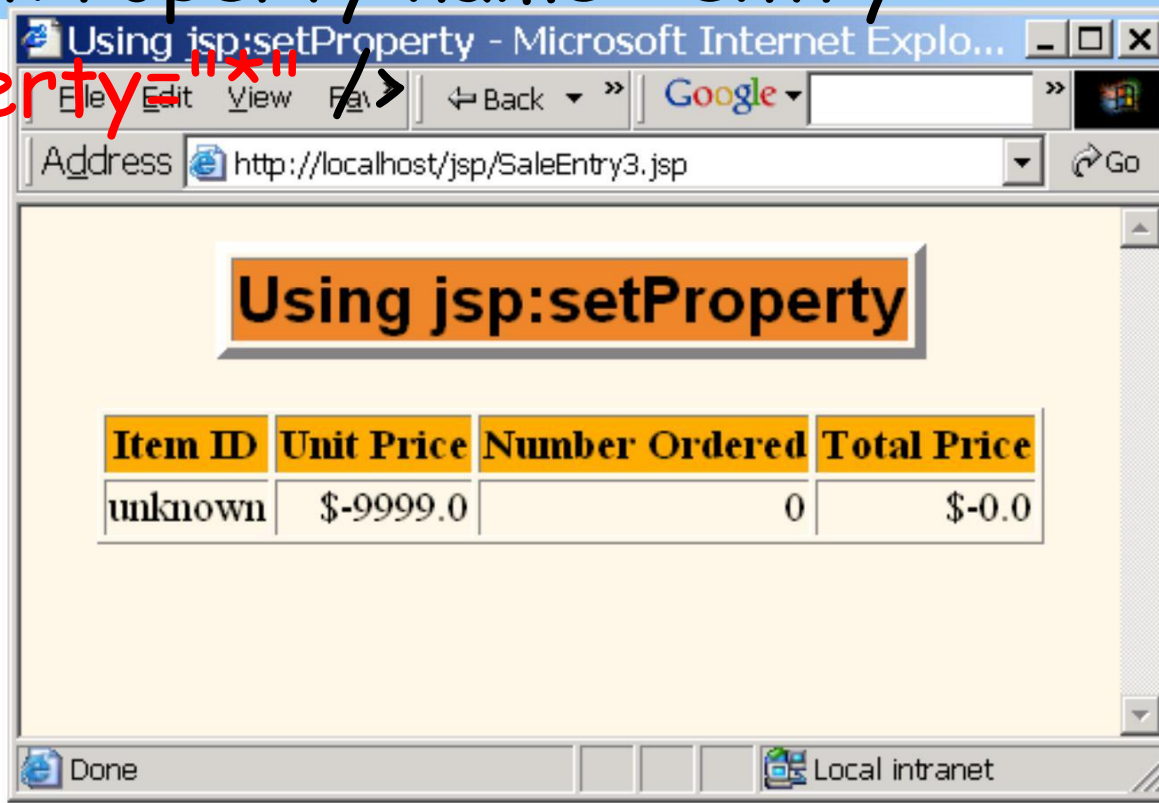
Case 2: Associating Individual Properties with Input Parameters

```
<jsp:useBean id="entry"  
            class="cwp.SaleEntry" />  
<jsp:setProperty  
    name="entry"  
    property="itemID"  
    param="itemID" />  
<jsp:setProperty  
    name="entry"  
    property="numItems"  
    param="numItems" />  
<jsp:setProperty  
    name="entry"  
    property="discountCode"  
    param="discountCode" />
```

Case 3: Associating All Properties with Input Parameters

```
<jsp:useBean id="entry"  
             class="cwp.SaleEntry" />  
<jsp:setProperty name="entry"
```

property="*" />



Using jsp:setProperty

Item ID	Unit Price	Number Ordered	Total Price
unknown	\$-9999.0	0	\$-0.0

Sharing Beans

You can use scope attribute to specify where bean is stored

- `<jsp:useBean id="..." class="..." scope="..." />`
- Bean still also bound to local variable in `_jspService`

Lets multiple servlets or JSP pages share data

Also permits conditional bean creation

- Create new object only if you can't find existing one

Values of the scope Attribute

page

- Default value. Bean object should be placed in the PageContext object for the duration of the current request. Lets methods in same servlet access bean

application

- Bean will be stored in ServletContext (available through the application variable or by call to `getServletContext()`). ServletContext is shared by all servlets in the same Web application (or all servlets on server if no explicit Web applications are defined).

Values of the scope Attribute

session

- Bean will be stored in the HttpSession object associated with the current request, where it can be accessed from regular servlet code with `getAttribute` and `setAttribute`, as with normal session objects.

request

- Bean object should be placed in the ServletRequest object for the duration of the current request, where it is available by means of `getAttribute`

Conditional Bean Operations

Bean conditionally created

- `jsp:useBean` results in new bean object only if no bean with same id and scope can be found
- If a bean with same id and scope is found, the preexisting bean is simply bound to variable referenced by id

Bean properties conditionally set

- `<jsp:useBean ... />` replaced by `<jsp:useBean ...>statements</jsp:useBean>`
- The statements (`jsp:setProperty` elements) are executed only if a new bean is created, not if an existing bean is found

Conditional Bean Creation: AccessCountBean

```
public class AccessCountBean {  
    private String firstPage;  
    private int accessCount = 1;  
    public String getFirstPage() {  
        } return(firstPage);  
  
    public void setFirstPage(String firstPage) { }  
    this.firstPage = firstPage;  
    public int getAccessCount() {  
        } return(accessCount);  
    public void setAccessCountIncrement(int increment) {  
        } } accessCount = accessCount + increment;
```

Conditional Bean Creation: SharedCounts1.jsp

```
<jsp:useBean id="counter" class="coreservlets.AccessCountBean"  
            scope="application">  
  <jsp:setProperty name="counter"
```

```
</jsp:useBean>property="firstPage" value="SharedCounts1.jsp" />
```

Of SharedCounts1.jsp (this page),

SharedCounts2.jsp, and

SharedCounts3.jsp,

```
<jsp:getProperty name="counter" property="firstPage" />
```

was the first page accessed.

<p>

Collectively, the three pages have been accessed

```
<jsp:getProperty name="counter" property="accessCount"
```

```
</> times.
```

```
<jsp:setProperty name="counter"
```

```
  property="accessCountIncrement" value="1" />
```

Accessing SharedCounts1, SharedCounts2, SharedCounts3

SharedCounts2.jsp was accessed first.

Pages have been accessed twelve previous times by an arbitrary number of clients



Actions essentials

<jsp:forward> action

<jsp:setProperty> action

<jsp:getProperty> action

<jsp:plugin> action

<jsp:include> action

<jsp:useBean> action

Can you identify the use of each action?

Uses of JSP Constructs: Custom JSP Tag Libraries

**Simple
Application**



**Complex
Application**

Scripting elements calling
servlet code directly

Scripting elements calling
servlet code indirectly (by
means of utility classes)

Beans

Custom tags

Servlet/JSP combo
(MVC architecture)

Components That Make Up a Tag Library

The Tag Handler Class

Must implement `javax.servlet.jsp.tagext.Tag`

Usually extends `TagSupport` or `BodyTagSupport`

Goes in same directories as servlet class files and beans

The Tag Library Descriptor File

XML file describing tag name, attributes, and implementing tag handler class

Goes with JSP file or at arbitrary URL

The JSP File

Imports a tag library (referencing descriptor file)

Defines tag prefix, uses tags

Defining a Simple Tag Handler Class

Extend the TagSupport class

Import needed packages

- `import javax.servlet.jsp.*; import javax.servlet.jsp.tagext.*; import java.io.*;`

Override `doStartTag`

- Obtain the `JspWriter` by means of `pageContext.getOut()`
- Use the `JspWriter` to generate JSP content
- Return `SKIP_BODY`
- Translated into servlet code at page-translation time
- Code gets called at request time


```
package cwp.tags;
import javax.servlet.jsp.*; import
javax.servlet.jsp.tagext.*; import
java.io.*;
import java.math.*; import cwp.*;
```

Defining a Simple Tag Handler Class

```
public class SimplePrimeTag extends TagSupport
{ protected int len = 50;

public int doStartTag() {
    try {
        JspWriter out = pageContext.getOut();
        BigInteger prime =
            Primes.nextPrime(Primes.random(len));
        out.print(prime);
    } catch(IOException ioe) {

        } System.out.println("Error generating prime: " +
ioe); } return(SKIP_BODY);
}
```

Defining a Simple Tag Library Descriptor

Start with XML header and DOCTYPE

Top-level element is **taglib**

Each tag defined by tag element containing:

- **name**, whose body defines the base tag name.
In this case, I use `<name>simplePrime</name>`
- **tagclass**, which gives the fully qualified class name of the tag handler.
In this case, I use `<tagclass>cwp.tags.SimplePrimeTag</tagclass>`
- **bodycontent**, which gives hints to development environments.
Optional.
- **info**, which gives a short description. Here, I use
`<info>Outputs a random 50-digit prime.</info>`

```
<?xml version="1.0" encoding="ISO-8859-1"
```

```
?> <!DOCTYPE taglib ...>
```

```
<taglib>
```

TLD File for

```
<tlibversion>1.0</tlibversion> <jspversion>1.1</jspversion>
```

```
SimplePrimeTag
```

```
<shortname>cwp</shortname>
```

```
<info>
```

A tag library from Core Web Programming 2nd Edition, <http://www.corewebprogramming.com/>.

```
</info>
```

```
<tag>
```

```
<name>simplePrime</name>
```

```
<tagclass>cwp.tags.SimplePrimeTag</tagclass>
```

```
<info>Outputs a random 50-digit prime.</info>
```

```
</tag>
```

```
</taglib>
```

Accessing Custom Tags From JSP Files

Import the tag library

- Specify location of TLD file

```
<%@ taglib uri= "cwp-taglib.tld" prefix= "cwp" %>
```

- Define a tag prefix (namespace)

```
<%@ taglib uri="cwp-taglib.tld" prefix= "cwp" %>
```

Use the tags

- `<prefix:tagName />`

Tag name comes from TLD file

Prefix comes from taglib directive

- E.g., `<cwp:simplePrime />`

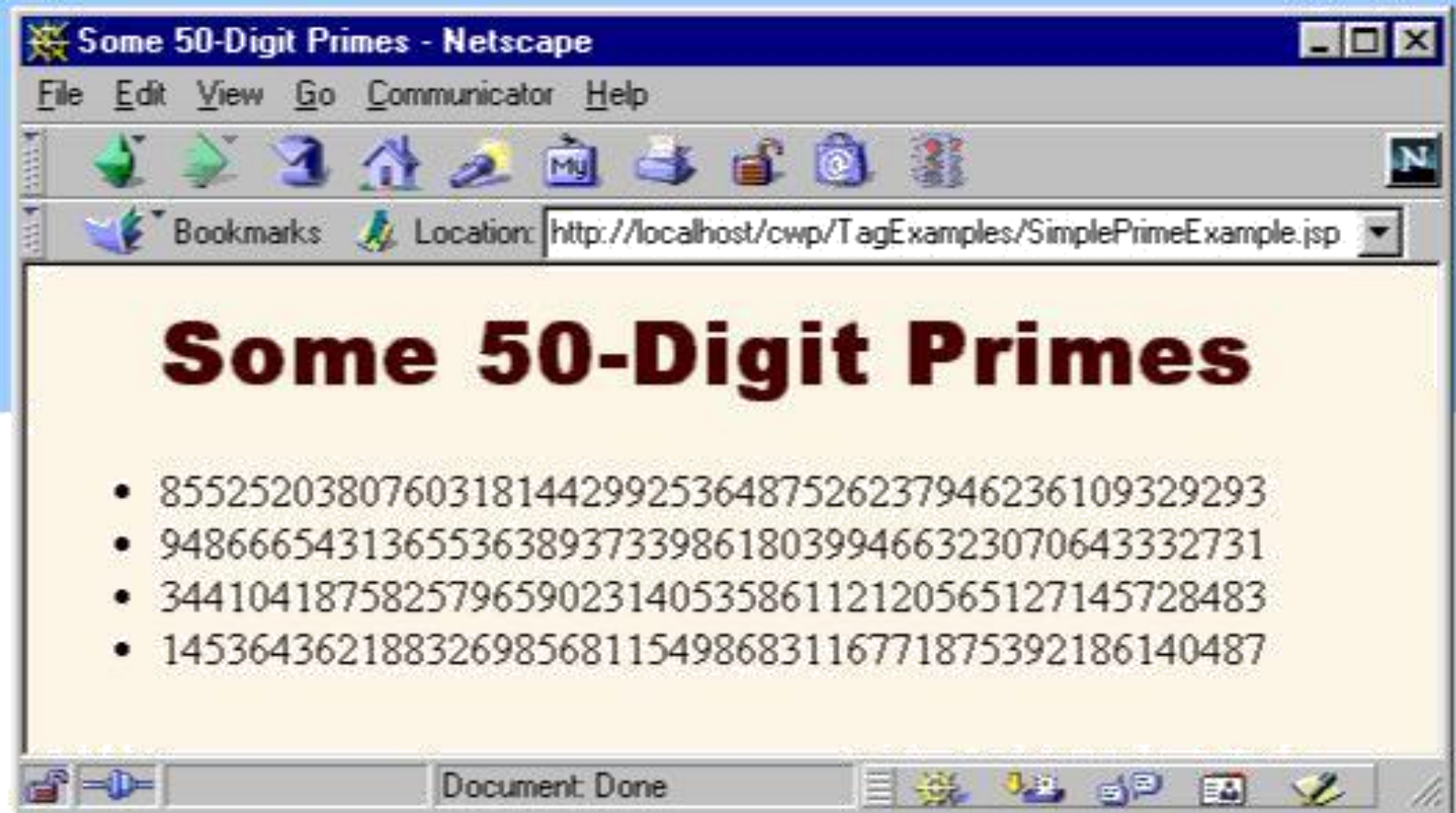
...
<H1>Some 50-Digit Primes</H1>

<%@ taglib uri="cwp-taglib.tld" prefix="cwp"%>

<cwp:simplePrime /> <cwp:simplePrime />

<cwp:simplePrime /> <cwp:simplePrime />

Using
simplePrime
Tag



Intermediate and Advanced Custom Tags

Tags with attributes

Tags that include their body content

Tags that optionally include their body

Tags that manipulate their body

Tags that manipulating their body multiple times (looping tags)

Nested tags

See book for details (related chapter online in PDF at Java Developer's Connection)

– <http://developer.java.sun.com/developer/Books/cservletsjsp/>

...

```
<H1>Some N-Digit Primes</H1>
```

```
<%@ taglib uri="cwp-taglib.tld" prefix="cwp"
```

```
%> <UL>
```

```
<LI>20-digit: <cwp:prime length="20" />
```

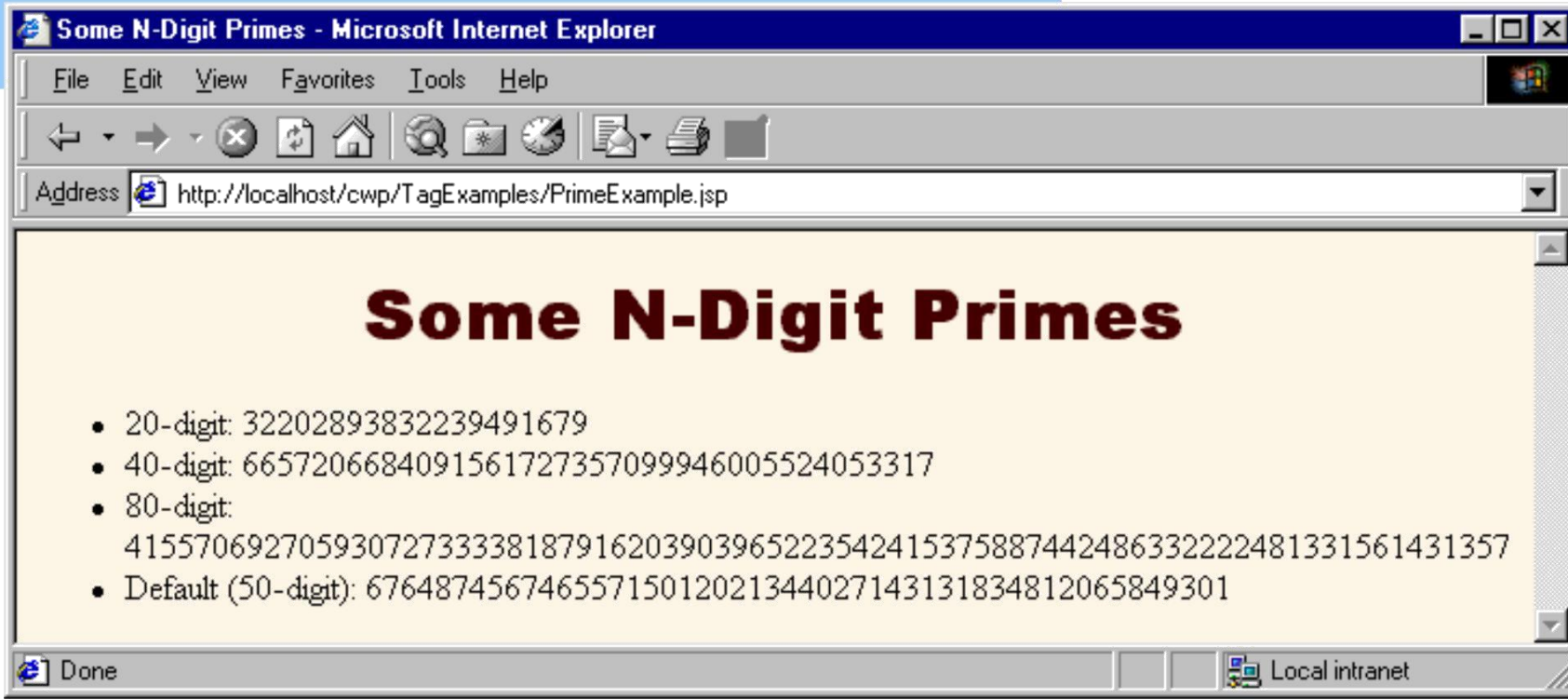
```
<LI>40-digit: <cwp:prime length="40" />
```

```
<LI>80-digit: <cwp:prime length="80" />
```

```
<LI>Default (50-digit): <cwp:prime />
```

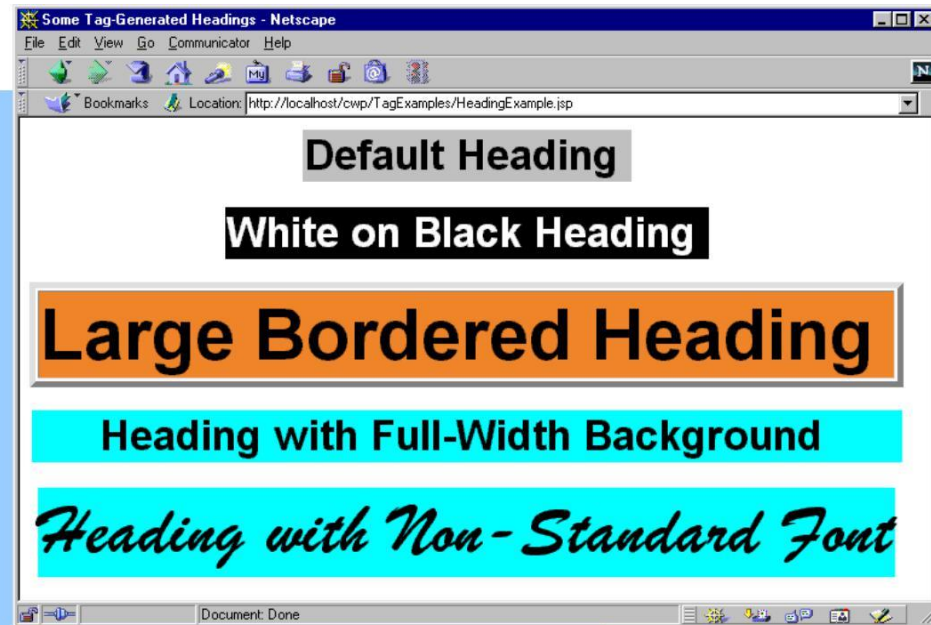
```
</UL>
```

Tags with
Attributes:
Prime Tag



Including Body Content: heading Tag

```
...  
<%@ taglib uri="cwp-taglib.tld" prefix="cwp"  
%> <cwp:heading bgColor="#COCOCO">  
Default Heading  
</cwp:heading>  
<P>  
<cwp:heading bgColor="BLACK"  
color="WHITE"> White on Black Heading  
</cwp:heading>  
<P>  
<cwp:heading bgColor="#EF8429" fontSize="60"  
border="5"> Large Bordered Heading  
</cwp:heading>  
...
```



Optionally Including Tag Body: debug Tag

```
<%@ taglib uri="cwp-taglib.tld" prefix="cwp" %>
```

Top of regular page. Blah, blah, blah. Yadda, yadda, yadda.

```
<P>
```

```
<cwp:debug>
```

```
<B>Debug:</B>
```

```
<UL>
```

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

```
<LI>Requesting hostname: <%= request.getRemoteHost()%>
```

```
<LI>Session ID: <%= session.getId() %>
```

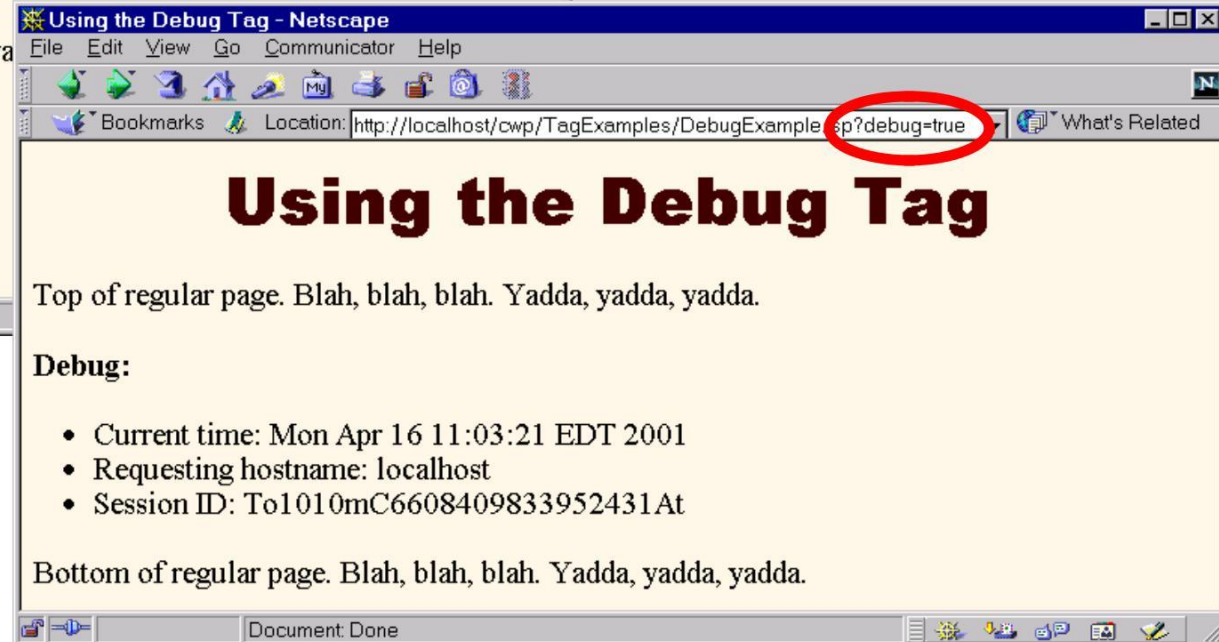
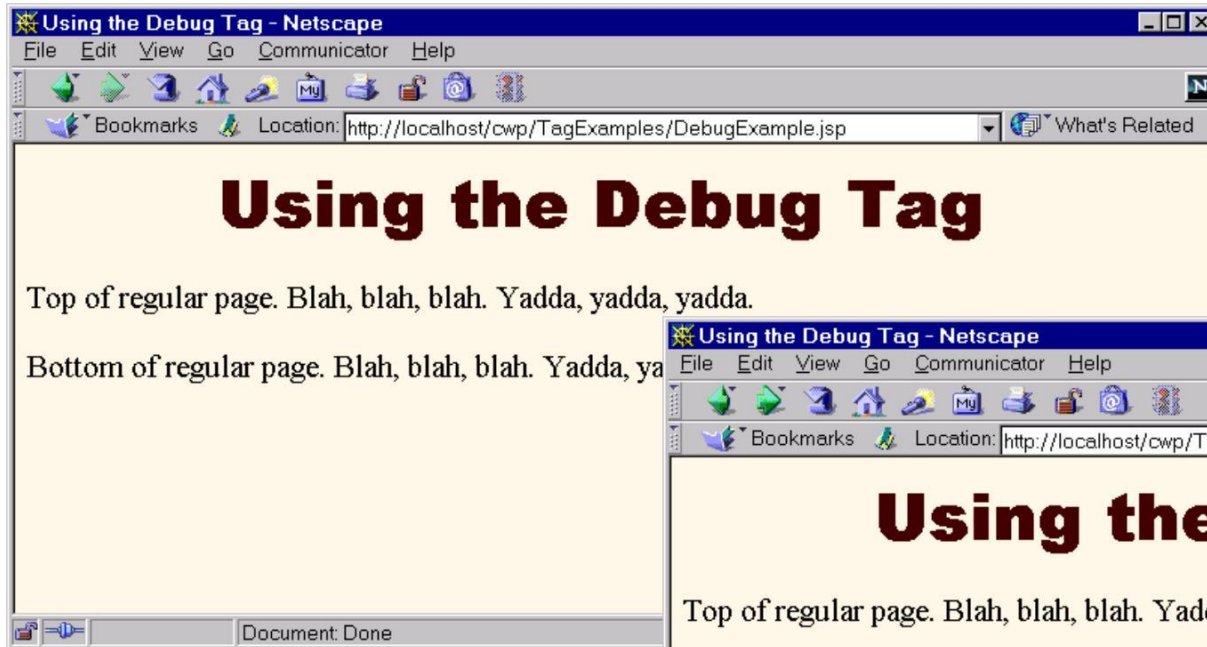
```
</UL>
```

```
</cwp:debug>
```

```
<P>
```

Bottom of regular page. Blah, blah, blah.
Yadda, yadda, yadda.

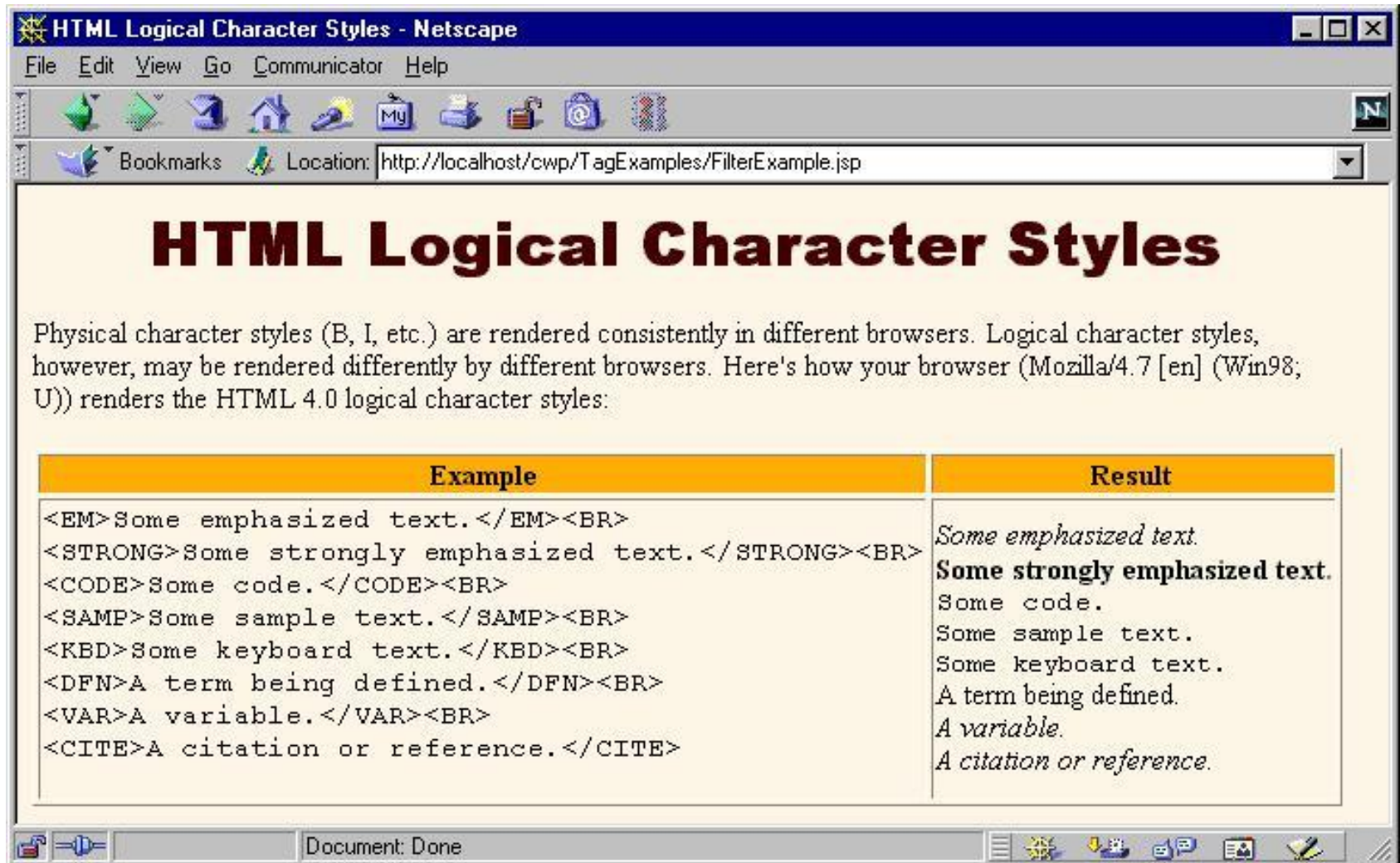
Using debug Tag: Results



Manipulating Tag Body: the filter Tag

```
<%@ taglib uri="cwp-taglib.tld" prefix="cwp"
%> <TABLE BORDER=1 ALIGN="CENTER">
<TR CLASS="COLORED"><TH>Example<TH>Result
<TR>
<TD><PRE><cwp:filter>
<EM>Some emphasized text.</EM><BR>
<STRONG>Some strongly emphasized text.</STRONG><BR>
<CODE>Some code.</CODE><BR>
...
</cwp:filter></PRE>
<TD>
<EM>Some emphasized text.</EM><BR>
<STRONG>Some strongly emphasized text.</STRONG><BR>
<CODE>Some code.</CODE><BR>
...
</TABLE>
```

Using the filter Tag: Results



The screenshot shows a Netscape browser window titled "HTML Logical Character Styles - Netscape". The address bar contains "http://localhost/cwp/TagExamples/FilterExample.jsp". The main content area displays the title "HTML Logical Character Styles" in a large, bold, dark red font. Below the title is a paragraph of text explaining that physical character styles are rendered consistently, while logical character styles may be rendered differently by different browsers. The text mentions "Mozilla/4.7 [en] (Win98; U)". Below the text is a table with two columns: "Example" and "Result". The table lists various HTML logical character styles and their rendered appearance in the browser.

Example	Result
<code>Some emphasized text.
</code>	<i>Some emphasized text.</i>
<code>Some strongly emphasized text.
</code>	Some strongly emphasized text.
<code><CODE>Some code.</CODE>
</code>	Some code.
<code><SAMP>Some sample text.</SAMP>
</code>	Some sample text.
<code><KBD>Some keyboard text.</KBD>
</code>	Some keyboard text.
<code><DFN>A term being defined.</DFN>
</code>	A term being defined.
<code><VAR>A variable.</VAR>
</code>	<i>A variable.</i>
<code><CITE>A citation or reference.</CITE></code>	<i>A citation or reference.</i>

```
<%@ taglib uri="cwp-taglib.tld" prefix="cwp" %>
```

```
<OL>
```

<!-- Repeats N times. A null reps value means repeat once. -->

```
<cwp:repeat
```

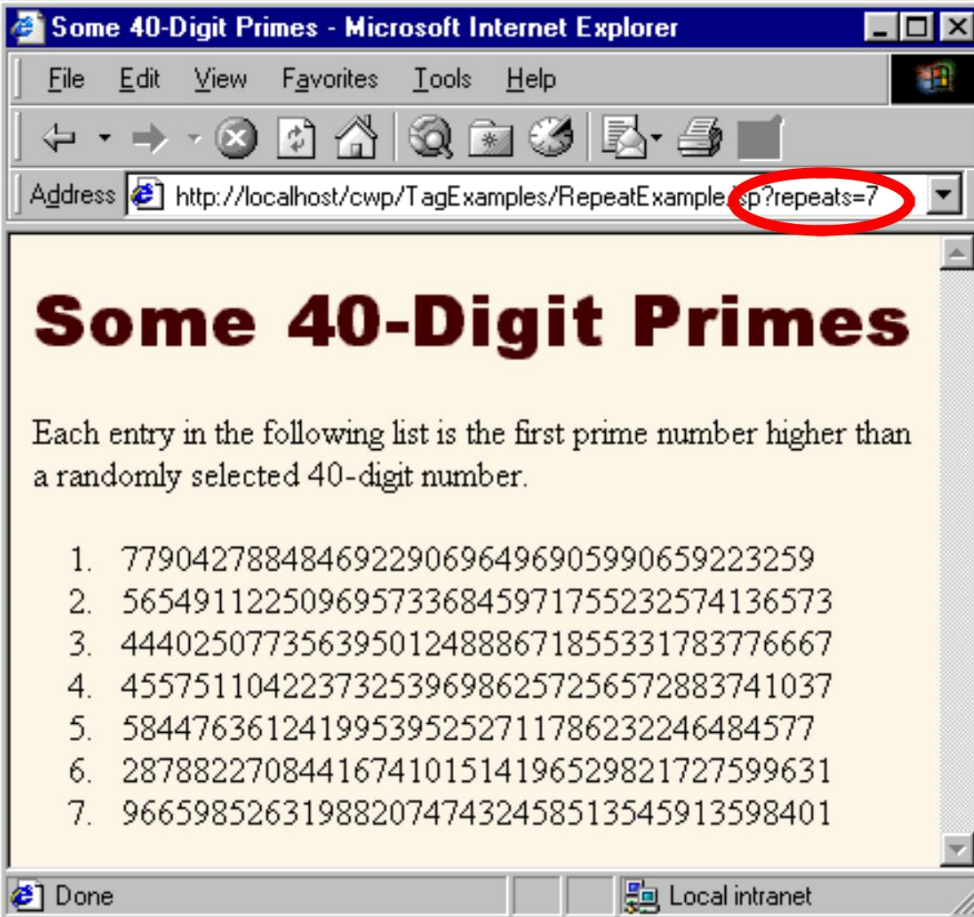
```
  reps='<%= request.getParameter("repeats") %>'
```

```
  <LI><cwp:prime length="40" />
```

```
</cwp:repeat>
```

```
</OL>
```

Manipulating the
Body Multiple Times:
the repeat Tag



Some 40-Digit Primes - Microsoft Internet Explorer

File Edit View Favorites Tools Help

Address <http://localhost/cwp/TagExamples/RepeatExample.jsp?repeats=7>

Some 40-Digit Primes

Each entry in the following list is the first prime number higher than a randomly selected 40-digit number.

1. 779042788484692290696496905990659223259
2. 5654911225096957336845971755232574136573
3. 4440250773563950124888671855331783776667
4. 4557511042237325396986257256572883741037
5. 584476361241995395252711786232246484577
6. 2878822708441674101514196529821727599631
7. 9665985263198820747432458513545913598401

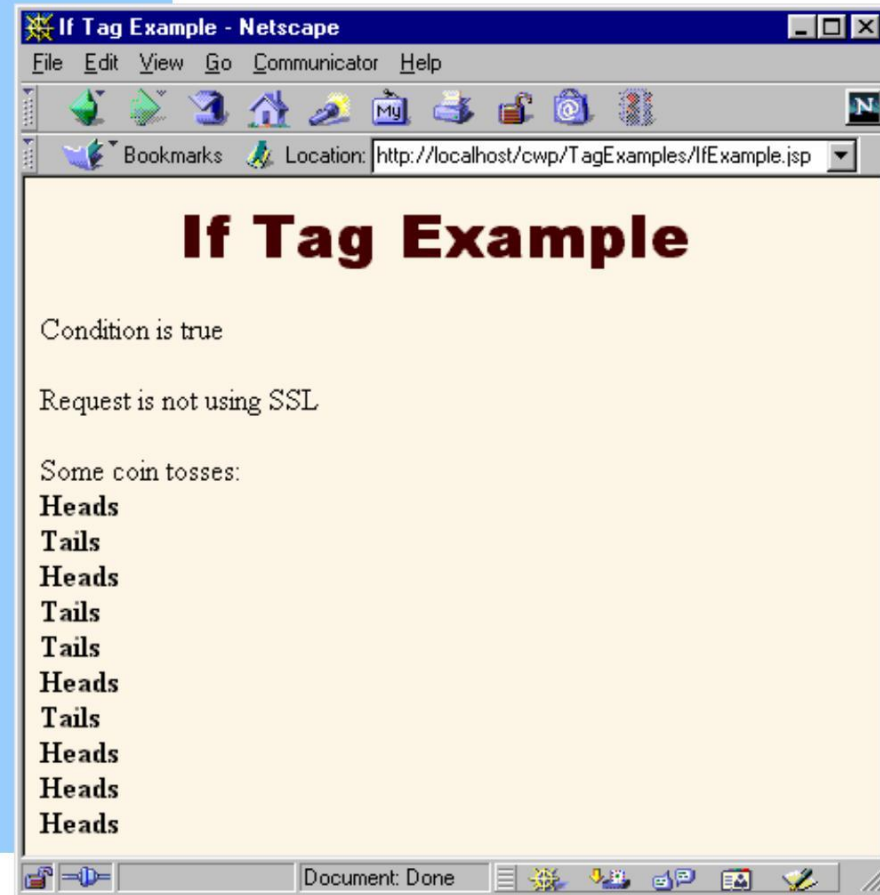
Done Local intranet

Nested Tags: the if Tag

```
<%@ taglib uri="cwp-taglib.tld" prefix="cwp" %>
<cwp:if>
  <cwp:condition>true</cwp:condition>
  <cwp:then>Condition is true</cwp:then>
  <cwp:else>Condition is false</cwp:else>
</cwp:if>
```

...

```
Some coin tosses:<BR>
<cwp:repeat reps="10">
  <cwp:if>
    <cwp:condition>
      <%= Math.random() < 0.5 %>
    </cwp:condition>
    <cwp:then><B>Heads</B><BR></cwp:then>
    <cwp:else><B>Tails</B><BR></cwp:else>
  </cwp:if>
</cwp:repeat>
```



Open Source Tag Libraries

<http://jakarta.apache.org/taglibs/>

Internationalization (I18N)

Database access

Sending email

JNDITM

Date/time

Populating/validating form fields

Perl regular expressions

Extracting data from other Web pages

XSL transformations

Etc

Break Time – 15 minutes



Uses of JSP Constructs: Integrating Servlets and JSP

**Simple
Application**



**Complex
Application**

Scripting elements calling
servlet code directly

Scripting elements calling
servlet code indirectly (by
means of utility classes)

Beans

Custom tags

Servlet/JSP combo
(MVC architecture)

Why Combine Servlets & JSP?

Typical picture: use JSP to make it easier to develop and maintain the HTML content

- For simple dynamic code, call servlet code from scripting expressions
- For moderately complex cases, use custom classes called from scripting expressions
- For more complicated cases, use beans and custom tags

But, that's not enough

- For complex processing, JSP is awkward
- Despite the convenience of separate classes, beans, and custom tags, *the assumption behind JSP is that a single page gives a single basic look*

Integrating Servlets and JSP :

Architecture

Approach

- Original request is answered by a servlet
- Servlet processes request data, does database lookup, accesses business logic, etc.
- Results are placed in beans
- Request is forwarded to a JSP page to format result
- Different JSP pages can be used to handle different types of presentation

Terminology

- Often called the “Model View Controller” architecture or “Model 2” approach to JSP
- Formalized further with Apache “Struts” framework

See <http://jakarta.apache.org/struts/>

Dispatching Requests

First, call the `getRequestDispatcher` method of `ServletContext`

- Supply a URL relative to the Web application root
- Example

```
String url = "/presentations/presentation1.jsp";  
RequestDispatcher dispatcher =  
    getServletContext().getRequestDispatcher(url);
```

Second

- Call `forward` to completely transfer control to destination page. See following example
- Call `include` to insert output of destination page and then continue on.

```

public void doGet(HttpServletRequest request,
                 HttpServletResponse response)
    throws ServletException, IOException {
    String operation = request.getParameter("operation"); if
    (operation == null) {
    } operation = "unknown"; Forwarding if (operation.equals("operation1")) {
    gotoPage("/operations/presentation1.jsp", Requests request,
            response);
    } else if (operation.equals("operation2")) {
    gotoPage("/operations/presentation2.jsp",
    } else { request, response);
    gotoPage("/operations/unknownRequestHandler.jsp",
            request, response);
    } }
private void gotoPage(String address,
                    HttpServletRequest request, HttpServletResponse response)
    throws ServletException, IOException {
    RequestDispatcher dispatcher =
    getServletContext().getRequestDispatcher(address);
    } dispatcher.forward(request, response);

```

Reminder: JSP useBean Scope Alternatives

request

- `<jsp:useBean id="..." class="..." scope="request" />`

session

- `<jsp:useBean id="..." class="..." scope="session" />`

application

- `<jsp:useBean id="..." class="..." scope="application" />`

page

- `<jsp:useBean id="..." class="..." scope="page" />`

or just

```
<jsp:useBean id="..." class="..." />
```

- This scope is not used in MVC architecture

Storing Data for Later Use: The Servlet Request

Purpose

- Storing data that servlet looked up and that JSP page will use only in this request.

Servlet syntax to store data

```
SomeClass value = new SomeClass(...);  
request.setAttribute("key", value);
```

Use RequestDispatcher to forward to JSP page

JSP syntax to retrieve data

```
<jsp:useBean id="key"  
  class="SomeClass" scope="request" />
```

Storing Data for Later Use: The Session Object

Purpose

- Storing data that servlet looked up and that JSP page will use in this request and in later requests from same client.

Servlet syntax to store data

```
SomeClass value = new SomeClass(...);
```

```
HttpSession session = request.getSession(true);
```

```
session.setAttribute("key", value);
```

Use RequestDispatcher to forward to JSP page

JSP syntax to retrieve data

```
<jsp:useBean id="key"
```

```
class="SomeClass" scope="session" />
```


Storing Data for Later Use: The Servlet Context

Purpose

- Storing data that servlet looked up and that JSP page will use in this request and in later requests from any client.

Servlet syntax to store data

```
SomeClass value = new SomeClass(...);  
getServletContext().setAttribute("key", value);
```

Use RequestDispatcher to forward to JSP page

JSP syntax to retrieve data

```
<jsp:useBean  
  id="key"  
  class="SomeClass"  
  scope="application" />
```

An On-Line Travel Agent

Online Travel Quick Search - Microsoft Internet Explorer

File Edit View Favorites Tools Help

Address <http://localhost/travel/quick-search.html> Go Links >>

Online Travel Quick Search

Email address:





Password:

Origin:

Destination:

Start date (MM/DD/YY):

End date (MM/DD/YY):

Not yet a member? Get a free account [here](#).

Done Local intranet

Best Available Flights - Microsoft Internet Explorer

File Edit View Favorites Tools Help

Address <http://localhost/servlet/coreservlets.Travel> Go Links >>

Best Available Flights

Finding flights for Joe Hacker

Java Airways Flight 1522 (\$455.95)

Outgoing: Leaves Baltimore at 9:00 AM on 3/20/00, arriving in Los Angeles at 3:15 PM (1 stop -- Java, Indonesia).
Return: Leaves Los Angeles at 9:00 AM on 3/25/00, arriving in Baltimore at 3:15 PM (1 stop -- Sun Microsystems).

Servlet Express Flight 2622 (\$505.95)

Outgoing: Leaves Baltimore at 9:30 AM on 3/20/00, arriving in Los Angeles at 4:15 PM (1 stop -- New Atlanta).
Return: Leaves Los Angeles at 9:30 AM on 3/25/00, arriving in Baltimore at 4:15 PM (1 stop -- New Atlanta).

Geek Airlines Flight 3.14159 (\$675.00)

Outgoing: Leaves Baltimore at 10:02:37 AM on 3/20/00, arriving in Los Angeles at 2:22:19 PM (1 stop -- JHU).
Return: Leaves Los Angeles at 10:02:37 AM on 3/25/00, arriving in Baltimore at 2:22:19 PM (1 stop -- MIT).

Airline	Frequent Flyer Number
Java Airways	321-9299-J
United	442-2212-U
Southwest	1A345

Credit Card: JavaSmartCard (XXXX-XXXX-XXXX-3120)

Done Local intranet

Review: JSP Introduction

JSP makes it easier to create/maintain HTML, while still providing full access to servlet code

JSP pages get translated into servlets

- It is the servlets that run at request time
- Client does not see anything JSP-related

You still need to understand servlets

- Understanding how JSP really works
- Servlet code called from JSP
- Knowing when servlets are better than JSP
- Mixing servlets and JSP

Uses of JSP Constructs

**Simple
Application**



**Complex
Application**

Scripting elements calling
servlet code directly

Scripting elements calling
servlet code indirectly (by
means of utility classes)

Beans

Custom tags

Servlet/JSP combo
(MVC architecture)

Review: Calling Java Code Directly: JSP Scripting Elements

JSP Expressions

Format: `<%= expression %>`

Evaluated and inserted into the servlet's output.

JSP Scriptlets

Format: `<% code %>`

Inserted verbatim into the `_jspService` method

JSP Declarations

Format: `<%! code %>`

Inserted verbatim into the body of the servlet class

Predefined variables

request, response, out, session, application

Limit the Java code in page

Use helper classes, beans, custom tags, servlet/JSP combo

Review: The JSP page Directive: Structuring Generated Servlets

The import attribute

- Changes the packages imported by the servlet that results from the JSP page

The contentType attribute

- Specifies MIME type of result
- Cannot be used conditionally

Review: Including Files in JSP Documents

```
<jsp:include page="Relative URL" flush="true" />
```

Output inserted into JSP page at request time

Cannot contain JSP content that affects entire page

Changes to included file do not necessitate changes to pages that use it

```
<%@ include file="Relative URL" %>
```

File gets inserted into JSP page prior to page translation

Thus, file can contain JSP content that affects entire page (e.g., import statements, declarations)

Changes to included file might require you to manually update pages that use it

Review: Using JavaBeans Components with JSP

Benefits of jsp:useBean

- Hides the Java programming language syntax
- Makes it easier to associate request parameters with objects (bean properties)
- Simplifies sharing objects among multiple requests or servlets/JSPs

jsp:useBean

- Creates or accesses a bean

jsp:getProperty

- Puts bean property (i.e. getXxx call) into output

jsp:setProperty

- Sets bean property (i.e. passes value to setXxx)

Review: Creating Custom JSP Tag Libraries

For each custom tag, you need

- A tag handler class (usually extending TagSupport or BodyTagSupport)
- An entry in a Tag Library Descriptor file
- A JSP file that imports library, specifies prefix, and uses tags

Simple tags

- Generate output in doStartTag, return SKIP_BODY

Attributes

- Define setAttributeName method. Update TLD file

Body content

- doStartTag returns EVAL_BODY_INCLUDE
- Add doEndTag method

Review: Integrating Servlets and JSP

Use MVC (Model 2) approach when:

- One submission will result in multiple basic looks
- Several pages have substantial common processing

Architecture

- A servlet answers the original request
- Servlet does the real processing & stores results in beans
 - Beans stored in `HttpServletRequest`, `HttpSession`, or `ServletContext`
- Servlet forwards to JSP page via `forward` method of `RequestDispatcher`
- JSP page reads data from beans by means of `jsp:useBean` with appropriate scope (request, session, or application)

UNIT-IV

Introduction to PHP

A short history of php

Parsing

Variables

Arrays

Operators

Functions

Control Structures

External Data Files

Background

PHP is server side scripting system

- PHP stands for "PHP: Hypertext Preprocessor"
- Syntax based on Perl, Java, and C
- Very good for creating dynamic content
- Powerful, but somewhat risky!
- If you want to focus on one system for dynamic content, this is a good one to choose

History

Started as a Perl hack in 1994 by Rasmus Lerdorf (to handle his resume), developed to PHP/FI 2.0

By 1997 up to PHP 3.0 with a new parser engine by Zeev Suraski and Andi Gutmans

Version 5.2.4 is current version, rewritten by Zend (www.zend.com) to include a number of features, such as an object model

Current is version 5

php is one of the premier examples of what an open source project can be

About Zend

A Commercial Enterprise

Zend provides Zend engine for PHP for free

They provide other products and services for a fee

- Server side caching and other optimizations
- Encoding in Zend's intermediate format to protect source code
- IDE-a developer's package with tools to make life easier
- Support and training services

Zend's web site is a great resource

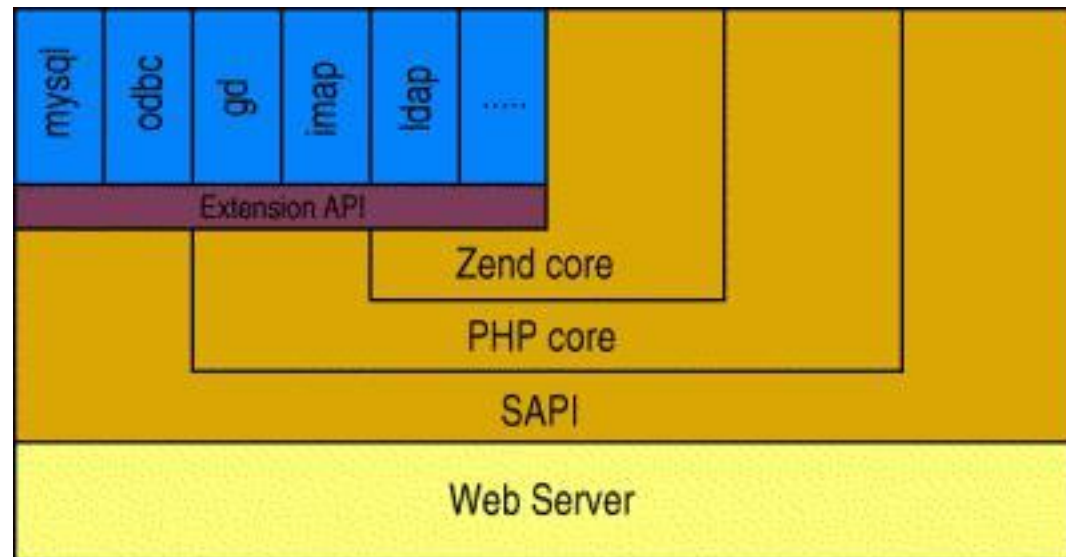
PHP 5 Architecture

Zend engine as parser (Andi Gutmans and Zeev Suraski)

SAPI is a web server abstraction layer

PHP components now self contained (ODBC, Java, LDAP, etc.)

This structure is a good general design for software (compare to OSI model, and middleware applications)



PHP Scripts

Typically file ends in .php--this is set by the web server configuration

- Separated in files with the `<?php ?>` tag
php commands can make up an entire file, or can be contained in html--this is a choice....
Program lines end in ";" or you get an error
Server recognizes embedded script and executes
Result is passed to browser, source isn't visible

```
<P>  
<?php $myvar = "Hello  
World!"; echo $myvar;  
>  
</P>
```

Parsing

We've talk about how the browser can read a text file and process it, that's a basic parsing method

Parsing involves acting on relevant portions of a file and ignoring others

Browsers parse web pages as they load

Web servers with server side technologies like php parse web pages as they are being passed out to the browser

Parsing does represent work, so there is a cost

Two Ways

You can embed sections of php inside html:

```
<BODY>  
<P>  
<?php $myvar = "Hello World!";  
echo $myvar;  
</BODY>
```

Or you can call html from php:

```
<?php  
echo "<html><head><title>Howdy</title>  
...  
>
```

What do we know already?

Much of what we learned about javascript holds true in php (but not all!), and other languages as well

```
$name = "bil";  
echo "Howdy, my name is $name";  
echo "What will $name be in this  
line?"; echo 'What will $name be in  
this line?'; echo 'What's wrong with  
this line?'; if ($name == "bil")  
{  
    Hey, what's this?  
    echo "got a match!";  
}
```

Variables

Typed by context (but one can force type), so it's loose

Begin with "\$" (unlike javascript!)

Assigned by value

- *\$foo = "Bob"; \$bar = \$foo;*

Assigned by reference, this links vars

- *\$bar = &\$foo;*

Some are preassigned, server and env vars

- For example, there are PHP vars, eg. *PHP_SELF*,
HTTP_GET_VARS

phpinfo()

The `phpinfo()` function shows the php environment

Use this to read system and server variables, setting stored in `php.ini`, versions, and modules

Notice that many of these data are in arrays

This is the first script you should write...

Variable Variables

Using the value of a variable as the **name** of a second variable)

```
$a = "hello";  
$$a = "world";
```

Thus:

```
echo "$a ${$a}";
```

Is the same as:

```
echo "$a $hello";
```

But `$$a` echoes as `"$hello"....`

Operators

- **Arithmetic (+, -, *, /, /%) and String (.).**

- **Assignment (=) and combined assignment**

```
$a = 3;  
$a += 5; // sets $a to 8;  
$b = "Hello ";  
$b .= "There!"; // sets $b to "Hello There!";
```

- **Bitwise (&, |, ^, ~, <<, >>)**

- `$a ^ $b` (Xor: Bits that are set in `$a` or `$b` but not both are set.)
- `~ $a` (Not: Bits that are set in `$a` are not set, and vice versa.)

- **Comparison (==, ===, !=, !==, <, >, <=, >=)**

Coercion

Just like javascript, php is loosely typed

Coercion occurs the same way

If you concatenate a number and string,
the number becomes a string

Operators: The Movie

- **Error Control (@)**

- When this precedes a command, errors generated are ignored (allows custom messages)

- **Execution (` is similar to the shell_exec() ~~shell_exec()~~ function)**

- You can pass a string to the shell for execution:

```
$output = `ls -al`;  
$output = shell_exec("ls -al");
```

- This is one reason to be careful about user set variables!

- **Incrementing/Decrementing**

```
++$a (Increments by one, then returns $a.)  
$a++ (Returns $a, then increments $a by one.)  
--$a (Decrements $a by one, then returns $a.)  
$a-- (Returns $a, then decrements $a by one.)
```

Son of the Valley of Operators

Logical

<i>\$a and \$b</i>	<i>And</i>	<i>True if both \$a and \$b are true.</i>
<i>\$a or \$b</i>	<i>Or</i>	<i>True if either \$a or \$b is true.</i>
<i>\$a xor \$b</i>	<i>Xor</i>	<i>True if either \$a or \$b is true, but not both.</i>
<i>! \$a</i>	<i>Not</i>	<i>True if \$a is not true.</i>
<i>\$a && \$b</i>	<i>And</i>	<i>True if both \$a and \$b are true.</i>
<i>\$a \$b</i>	<i>Or</i>	<i>True if either \$a or \$b is true.</i>

The two ands and ors have different precedence rules, "and" and "or" are lower precedence than "&&" and "||"

Use parentheses to resolve precedence problems or just to be clearer

Control Structures

Wide Variety available

- if, else, elseif
- while, do-while
- for, foreach
- break, continue, switch
- require, include, require_once, include_once

Control Structures

Mostly parallel to what we've covered already in javascript

if, elseif, else, while, for, foreach, break and continue

Switch

Switch, which we've seen, is very useful

These two do the same things....

```
if ($i == 0) {  
    echo "i equals 0";  
} elseif ($i == 1) {  
    echo "i equals 1";  
} elseif ($i == 2) {  
    echo "i equals 2";  
}
```

```
switch ($i) {  
case 0:  
    echo "i equals  
    0"; break;  
case 1:  
    echo "i equals  
    1"; break;  
case 2:  
    echo "i equals  
    2"; break;  
}
```

Nesting Files

- `require()`, `include()`, `include_once()`, `require_once()` are used to bring in an external file
- This lets you use the same chunk of code in a number of pages, or read other kinds of files into your program
- Be VERY careful of using these anywhere close to user input--if a hacker can specify the file to be included, that file will execute within your script, with whatever rights your script has (`readfile` is a good alternative if you just want the file, but don't need to execute it)
- Yes, Virginia, remote files can be specified

Example: A Dynamic Table

I hate writing html tables

You can build one in php

This example uses pictures and builds a table with pictures in one column, and captions in another

The captions are drawn from text files

I'm using tables, but you could use css for placement easily...

Arrays

You can create an array with the array function, or use the explode function (this is very useful when reading files into web programs...)

```
$my_array = array(1, 2, 3, 4, 5);  
$pizza = "piece1 piece2 piece3 piece4 piece5 piece6";  
$pieces = explode(" ", $pizza);
```

An array is simply a variable representing a keyed list

- A list of values or variables
- If a variable, that var can also be an array
- Each variable in the list has a key
- The key can be a number or a text label

Arrays

Arrays are lists, or lists of lists, or list of lists of lists, you get the idea--Arrays can be multi-dimensional

Array elements can be addressed by either by number or by name (strings)

If you want to see the structure of an array, use the `print_r` function to recursively print an array inside of pre tags

Text versus Keys

Text keys work like number keys (well, really, it's the other way around--number keys are just labels)

You assign and call them the same way, except you have to assign the label to the value or variables, eg:

```
echo "$my_text_array[third]";
```

```
$my_text_array = array(first=>1, second=>2, third=>3);  
echo "<pre>";  
print_r($my_text_array);  
echo "</pre>";
```

Walking Arrays

Use a loop, eg a foreach loop to walk through an array

while loops also work for arrays with numeric keys--just set a variable for the loop, and make sure to increment that variable within the loop

```
$colors = array('red', 'blue', 'green', 'yellow');  
  
foreach ($colors as $color) {  
    echo "Do you like $color?\n";  
}
```

05_arrays.php

You can't echo an array directly...

- You can walk through an echo or print() line by line
- You can use print_r(), this will show you the structure of complex arrays--that output is to the right, and it's handy for learning the structure of an array

```
Array
(
    => Array
    (
        [sku] => A13412
        [quantity] => 10
        [item] => Whirly
        Widgets [price] => .50
    )

    => Array
    (
        [sku] => A43214
        [quantity] => 142
        [item] => Widget
        Nuts [price] => .05
    )
)
```

Multidimensional Arrays

A one dimensional array is a list, a spreadsheet or other columnar data is two dimensional...

Basically, you can make an array of arrays

```
$multiD = array  
(  
    "fruits" => array("myfavorite" => "orange", "yuck" => "banana", "yum" => "apple"),  
    "numbers" => array(1, 2, 3, 4, 5, 6),  
    "holes"  => array("first", 5 => "second", "third")  
);
```

The structure can be built array by array, or declared with a single statement

You can reference individual elements by nesting:

```
echo "<p>Yes, we have no " . $multiD["fruits"]["yuck"] . " (ok by me).</p>";
```

`print_r()` will show the entire structure, but don't forget the pre tags

Getting Data into arrays

You can directly read data into individual array slots via a direct assignment:

```
$pieces[5] = "poulet resistance";
```

From a file:

- Use the file command to read a delimited file (the delimiter can be any unique char):

```
$pizza = file(./our_pizzas.txt)
```

- Use explode to create an array from a line within a loop:

```
$pieces = explode(" ", $pizza);
```

The Surface

The power of php lies partially in the wealth of functions---for example, the 40+ array functions

- `array_flip()` swaps keys for values
- `array_count_values()` returns an associative array of all values in an array, and their frequency
- `array_rand()` pulls a random element
- `array_unique()` removes duppies
 - `array_walk()` applies a user defined function to each element of an array (so you can dice all of a dataset)
- `count()` returns the number of elements in an array
- `array_search()` returns the key for the first match in an array

Using External Data

You can build dynamic pages with just the information in a php script

But where php shines is in building pages out of external data sources, so that the web pages change when the data does

Most of the time, people think of a database like MySQL as the backend, but you can also use text or other files, LDAP, pretty much anything....

Standard data files

Normally you'd use a tab delimited file, but you can use pretty much anything as a delimiter

Files get read as arrays, one line per slot

Remember each line ends in `\n`, you should clean this up, and be careful about white space

Once the file is read, you can use `explode` to break the lines into fields, one at a time, in a loop....

Standard data files

You can use `trim()` to clean white space and returns instead of `str_replace()`

Notice that this is building an array of arrays

```
$items=file("./mydata.txt");  
foreach ($items as $line)  
{  
  $line ≡ str_replace("\n", "", $line);  
  $line = explode("\t", $line);  
  // do something with $line array  
}
```

Useful string functions

`str_replace()`

`trim()`, `ltrim()`, `rtrim()`

`implode()`, `explode()`

`addslashes()`, `stripslashes()`

`htmlentities()`, `html_entity_decode()`,
`htmlspecialchars()`

`striptags()`

06_more_arrays.php

This is a simple script to read and process a text file

The data file is tab delimited and has the column titles as the first line of the file

How it works

The script uses the first line to build text labels for the subsequent lines, so that the array elements can be called by the text label

- If you add a new column, this script compensates
- Text based arrays are not position dependent...
- This script could be the basis of a nice function

There are two version of this, calling two different datafiles, but that's the only difference

06a_more_arrays.php

This version shows how to dynamically build a table in the html output

Alternative syntax

Applies to if, while, for, foreach, and switch

Change the opening brace to a colon

Change the closing brace to an endxxx

statement

```
<?php if ($a == 5): ?>
```

A is equal to 5

```
<?php endif; ?>
```

```
<?php
```

```
if ($a == 5):
```

```
    echo "a equals 5";
```

```
    echo "...";
```

```
else:
```

```
    echo "a is not 5";
```

```
endif;
```


Forms

(Getting data from users)

Forms: how they work

We need to know..

1. How forms work.
2. How to write forms in XHTML.
3. How to access the data in PHP.

How forms work

User requests a particular URL



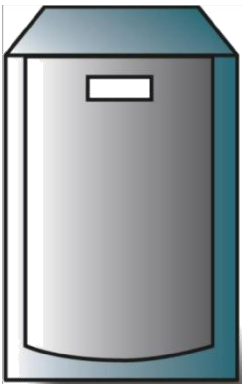
XHTML Page supplied with Form



User fills in form and submits. Another URL is requested and the Form data is sent to this page either in URL or as a separate piece of data.



XHTML Response



Web Server



User

XHTML Form

The form is enclosed in form tags..

```
<form  
  action="path/to/submit/page"  
  method="get">  
<!-- form contents -->  
</form>
```

Form tags

action="..." is the page that the form should submit its data to.

method="..." is the method by which the form data is submitted. The options are either **get** or **post**. If the method is **get** the data is passed in the url string, if the method is **post** it is passed as a separate file.

Form fields: text input

Use a text input within form tags for a single line freeform text input.

```
<label for="fn">First  
Name</label> <input type="text"  
    name="firstname"  
    id="fn"  
    size="20" />
```

Form tags

name="..." is the name of the field. You will use this name in PHP to access the data.

id="..." is label reference string – this should be the same as that referenced in the **<label>** tag.

size="..." is the length of the displayed text box (number of characters).

Form fields: password input

Use a starred text input for passwords.

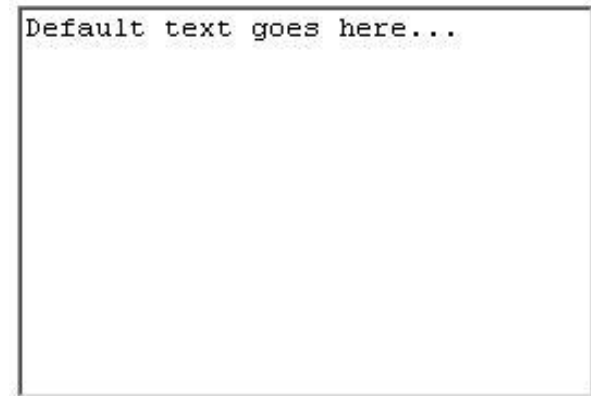
```
<label for="pw">Password</label>  
<input type="password"  
      name="passwd"  
      id="pw"  
      size="20" />
```


Form fields: text input

If you need more than 1 line to enter data, use a textarea.

```
<label for="desc">Description</label>  
<textarea name="description"  
          id="desc"  
          rows="10" cols="30">
```

```
Default text goes  
here... </textarea>
```



Default text goes here...

Form fields: text area

name="..." is the name of the field. You will use this name in PHP to access the data.

id="..." is label reference string – this should be the same as that referenced in the **<label>** tag.

rows="..." cols="..." is the size of the displayed text box.

Form fields: drop down

```
<label for="tn">Where do you live?</label>  
<select name="town" id="tn">  
<option value="swindon">Swindon</option>  
<option value="london"  
        selected="selected">London</option>  
<option value="bristol">Bristol</option>  
</select>
```

Form fields: drop down

name="..." is the name of the field.

id="..." is label reference string.

<option value="..." is the actual data sent back to PHP if the option is selected.

<option>...</option> is the value displayed to the user.

selected="selected" this option is selected by default.

Form fields: radio buttons

```
<input type="radio"
      name="age"
      id="u30"
      checked=""
      value="Under30" />
```

```
<label for="u30">Under
30</label> <br />
```

```
<input type="radio"
      name="age"
      id="thirty40"
      value="30to40" />
```

```
<label for="thirty40">30 to 40</label>
```

Form fields: radio buttons

name="..." is the name of the field. All radio boxes with the same name are grouped with only one selectable at a time.

id="..." is label reference string.

value="..." is the actual data sent back to PHP if the option is selected.

checked="checked" this option is selected by default.

Form fields: check boxes

```
What colours do you like?<br
/> <input type="checkbox"
      name="colour[]"
      id="r"
      checked="checked"
      value="red" />
<label for="r">Red</label>
<br />
<input type="checkbox"
      name="colour[]"
      id="b"
      value="blue" />
<label for="b">Blue</label>
```

Form fields: check boxes

name="..." is the name of the field. Multiple checkboxes can be selected, so if the buttons are given the same name, they will overwrite previous values. The exception is if the name is given with square brackets – an array is returned to PHP.

id="..." is label reference string.

value="..." is the actual data sent back to PHP if the option is selected.

checked="checked" this option is selected by default.

Hidden Fields

```
<input type="hidden"  
    name="hidden_value"  
    value="My Hidden Value" />
```

name="..." is the name of the field.

value="..." is the actual data sent back to PHP.

Submit button..

A submit button for the form can be created with the code:

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

Fieldset

In XHTML 1.0, all inputs must be grouped within the form into fieldsets. These represent logical divisions through larger forms. For short forms, all inputs are contained in a single fieldset.

```
<form>  
<fieldset>  
<input ... />  
<input ... />  
</fieldset>  
<fieldset>  
<input ... />  
<input ... />  
</fieldset>  
</form>
```

In PHP...

The form variables are available to PHP in the page to which they have been submitted.

The variables are available in two superglobal arrays created by PHP called `$_POST` and `$_GET`.

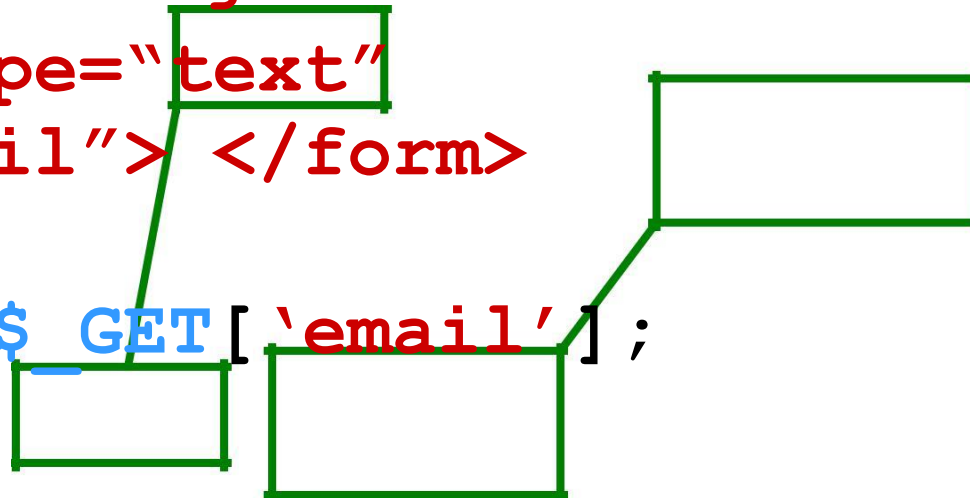
Access data

Access submitted data in the relevant array for the submission type, using the input name as a key.

```
<form action="path/to/submit/page"  
      method="get">
```

```
<input type="text"  
name="email"> </form>
```

```
$email = $_GET['email'];
```



A warning..

NEVER TRUST USER INPUT

Always check what has been input.

Validation can be undertaken using Regular expressions or in-built PHP functions.

A useful tip..

I find that storing the validated data in a different array to the original useful.

I often name this array 'clean' or something similarly intuitive.

I then **only** work with the data in \$clean, and never refer to \$_POST/\$_GET again.

Example

```
$clean = array();  
  
if (ctype_alnum($_POST['username']))  
{  
  
$clean['username'] = $_POST['username'];  
  
}
```


Filter example

```
$clean = array ();
```

```
$clean = array ();
```

```
if (ctype_alnum($_POST['username']))
```

```
{
```

```
$clean['username'] = $_POST['username'];
```

```
}
```

Initialise an array to store filtered data.

Filter example

```
$clean = array();
```

```
if (ctype_alnum($_POST['username']))
```

```
{
```

```
$clean['username'] = $_POST['username'];
```

```
}
```

Inspect username to make sure that it is alphanumeric.

Filter example

```
$clean = array();  
  
if (ctype_alnum($_POST['username']))  
{  
    $clean['username'] = $_POST['username'];  
}
```

If it is, store it in the array.

Is it submitted?

We also need to check before accessing data to see if the data is submitted, use `isset()` function.

```
if (isset($_POST['username'])) {  
    // perform validation  
}
```

Form Validation

What is form validation?

validation: ensuring that form's values are correct

some types of validation:

- preventing blank values (email address)
- ensuring the type of values
 - integer, real number, currency, phone number, Social Security number, postal
- address, email address, date, credit card number, ...
- ensuring the format and range of values (ZIP code must be a 5-digit integer)
- ensuring that values fit together (user types email twice, and the two must match)

A real Form that uses validation

The screenshot shows a Firefox browser window with several tabs open. The active tab is 'WeightWatchers.com: SI...'. The address bar shows the URL 'https://signup.weightwatchers.com/Signup/Version/registration/StepOne.aspx'. The page content includes a sidebar on the left with the heading 'Become a Registered User!' and a main form area on the right titled 'Create Your Registered User Account Login'. The form contains several fields with red 'X' icons and error messages:

- First name:** Xena
- Last name:** [Empty] **X**
Please enter your last name. It's required information.
- Your birthdate:** Month - Day - Year **X**
Please select your month, day, and year of birth. It's required information.
- Your gender:** Female Male **X**
Please enter your gender. It's required information.
- State:** (Select One) **X**
Please choose a state. It's required information.
- Zip code:** [Empty] **X**
Please enter a 5-digit ZIP code. It's required information.
- E-mail:** [Empty] **X**
Please enter your email address in the following format: abc@example.com. It's required information.
- Re-enter e-mail:** [Empty]

At the bottom of the browser window, there is a search bar with the text 'Find: preneq' and navigation buttons for 'Next', 'Previous', 'Highlight all', and 'Match case'.

Client vs. server-side validation

Validation can be performed:

- **client-side** (before the form is submitted)
 - can lead to a better user experience, but not secure (why not?)
- **server-side** (in PHP code, after the form is submitted)
 - needed for truly secure validation, but slower
- both
- best mix of convenience and security, but requires most effort to program

An example form to be validated


```
<form action="http://foo.com/foo.php"
method="get">
  <div>
    City: <input name="city" /> <br
  />
    State: <input name="state"
size="2" maxlength="2" /> <br />
    ZIP: <input name="zip" size="5"
size="5" maxlength="5" /> <br />
    <input type="submit" />
  </div>
</form>
```

- Let's validate this form's data on the server...

HTML

Basic server-side validation code

```
$city = $_REQUEST["city"];
$state = $_REQUEST["state"];
$zip = $_REQUEST["zip"];
if (!$city || strlen($state) != 2 || strlen($zip) !=
5) {
?>
    <h2>Error, invalid city/state submitted.</h2>
<?php
}
?>
```



basic idea: examine parameter values, and if they are bad, show an error message and abort

Basic server-side validation code

validation code can take a lot of time / lines to write

- How do you test for integers vs. real numbers vs. strings?
- How do you test for a valid credit card number?
- How do you test that a person's name has a middle initial?
- How do you test whether a given string matches a particular complex format?

Regular expressions

```
[a-z]at           #cat, rat, bat...
[aeiou]
[a-zA-Z]
[^a-z]           #not a-z
[[:alnum:]]+     #at least one alphanumeric char
(very) *large    #large, very very very large...
(very){1, 3}     #counting "very" up to 3
^bob             #bob at the beginning
com$            #com at the end
```

PHPRegExp•Regulareression: a pattern in a piece of text

PHP has:

- POSIX
- **Perl regular expressions**

Delimiters

```
/[a-z]/at          #cat, rat, bat...
#[aeiou]#
/[a-zA-Z]/
~[^a-z]~          #not a-z
/[[[:alnum:]]+]/  #at least one alphanumeric char
#(very) *#large   #large, very very very large...
~(very){1, 3}~    #counting "very" up to 3
/^bob/           3#bob at the beginning
/com$/          #com at the end

/http:// //
#http://# // #   #better readability
```

Used for Perl regular expressions (preg)

PHPRegExp

Basic Regular Expression

```
/abc/
```

in PHP, regexes are strings that begin and end with /

the simplest regexes simply match a particular substring

the above regular expression matches any string containing "abc":

- YES: "abc", "abcdef", "defabc", "=".abc.=" , ...
- NO: "fedcba", "ab c", "PHP", ...

Wildcards

A dot `.` matches any character except a `\n` line break

- `"/.oo.y/"` matches "Doocy", "goofy", "LooNy", ...

A trailing `i` at the end of a regex (after the closing `/`) signifies a case-insensitive match

- `"/xen/i"` matches "Xenia", "xenophobic", "Xena the warrior princess", "XEN technologies" ...

Special characters: |, (), ^, \

| means *OR*

- `"/abc|def|g/"` matches "abc", "def", or "g"
- There's no *AND* symbol. Why not?

() are for grouping

- `"/(Homer|Marge) Simpson/"` matches "Homer Simpson" or "Marge Simpson"

^ matches the beginning of a line; \$ the end

- `"/^<!--$/"` matches a line that consists entirely of "`<!--`"

Special characters: |, (), ^, \

\ starts an escape sequence

– many characters must be escaped to match them literally: / \ \$. [] () ^ * + ?

– "/<br \/>/" matches lines containing
 tags

Quantifiers: *, +, ?

* means 0 or more occurrences

- `/abc*/` matches "ab", "abc", "abcc", "abccc", ...
- `/a(bc)*/` matches "a", "abc", "abcbc", "abcbcbc", ...
- `/a.*a/` matches "aa", "aba", "a8qa", "a!?!_a", ...

+ means 1 or more occurrences

- `/a(bc)+/` matches "abc", "abcbc", "abcbcbc", ...
- `/Goo+gle/` matches "Google", "Gooogle", "Gooooogle", ...

? means 0 or 1 occurrences

- `/a(bc)?/` matches "a" or "abc"

More quantifiers: {min,max}

{min,max} means between min and max occurrences (inclusive)

- `"/a(bc){2,4}/"` matches `"abcbc"`, `"abcbcbc"`, or `"abcbcbcbc"`

min or max may be omitted to specify any number

- `{2,}` means 2 or more
- `{,6}` means up to 6
- `{3}` means exactly 3

Character sets: []

[] group characters into a character set; will match any single character from the set

- `"/[bcd]art/"` matches strings containing "bart", "cart", and "dart"
- equivalent to `"/(b|c|d)art/"` but shorter

inside [], many of the modifier keys act as normal characters

- `"/what[!*?]*/"` matches "what", "what!", "what?*!", "what??!",

What regular expression matches DNA (strings of A, C, G, or T)?

Character ranges: [start-end]

inside a character set, specify a range of characters with -

- `"/[a-z]/"` matches any lowercase letter
- `"/[a-zA-Z0-9]/"` matches any lower- or uppercase letter or digit

an initial `^` inside a character set negates it

- `"/[^abcd]/"` matches any character other than a, b, c, or d

Character ranges: [start-end]

inside a character set, - must be escaped to be matched

- `"/[+\-]?[0-9]+/"` matches an optional + or - , followed by at least one digit

What regular expression matches letter grades such as A, B+, or D- ?

Escape sequences

special escape sequence character sets:

- `\d` matches any digit (same as `[0-9]`); `\D` any non-digit (`[^0-9]`)
- `\w` matches any “word character” (same as `[a-zA-Z_0-9]`); `\W` any non-word

char

- `\s` matches any whitespace character (, `\t`, `\n`, etc.); `\S` any non-whitespace

What regular expression matches dollar amounts of at least \$100.00 ?

Regular expressions in PHP (PDF)

regex syntax: strings that begin and end with /, such as `"/[AEIOU]+/"`

function	description
<u>preg_match</u> (regex, string)	returns TRUE if string matches regex
<u>preg_replace</u> (regex, replacement, string)	returns a new string with all substrings that match regex replaced by replacement
<u>preg_split</u> (regex, string)	returns an array of strings from given string broken apart using the given regex as the delimiter (similar to explode but more

powerful)

Regular expressions example

```
echo preg_match ("/test/", "a test of preg_match");  
echo preg_match ('/tutorial/', "a test of test of  
preg_match ");
```

```
$matchesarray[0] = "http://www.tipsntutorials.com/"  
$matchesarray[1] = "http://"  
$matchesarray[2] = "www.tipsntutorials.com/"  
preg_match ('/(http://) (.*)/', "http://www.tipsntuto  
rials.com/", $matchesarray)
```

PHP

Regular expressions example

```
# replace vowels with stars
$str = "the quick brown fox";
$str = preg_replace("/[aeiou]/", "*", $str);
# "th* q**ck br*wn f*x"
# break apart into words
$words = preg_split("/[ ]+/", $str);
# ("th*", "q**ck", "br*wn", "f*x")
# capitalize words that had 2+ consecutive vowels
for ($i = 0; $i < count($words); $i++) {
    if (preg_match("/\\{2,}/", $words[$i])) {
        $words[$i] = strtoupper($words[$i]);
    }
}
# ("th*", "Q**CK", "br*wn", "f*x")
```

PHP

PHP form validation w/ regexes

```
$state = $_REQUEST["state"];  
if (!preg_match("/[A-Z]{2}/", $state)) {  
>?>  
<h2>Error, invalid state submitted.</h2>  
submitted.</h2> <?php  
}
```

PHP

using `preg_match` and well-chosen regexes allows you to quickly validate query parameters against complex patterns

Another PHP experiment

Write a PHP script that tests whether an e-mail address is input correctly. Test using valid and invalid addresses

Use array

Use function.

UNIT-5

PHP and Database Access

Connecting to a Database Using PHP

Most Web applications:

- Retrieve information from a database to alter their on-screen display

- Store user data such as orders, tracking, address, credit card, etc. in a database

Permits them to adapt to individual users, and provide fresh, changing content

PHP: Built-in Database Access

PHP provides built-in database connectivity for a wide range of databases

- MySQL, PostgreSQL, Oracle, Berkeley DB, Informix, mSQL, Lotus Notes, and more

- Starting support for a specific database may involve PHP configuration steps

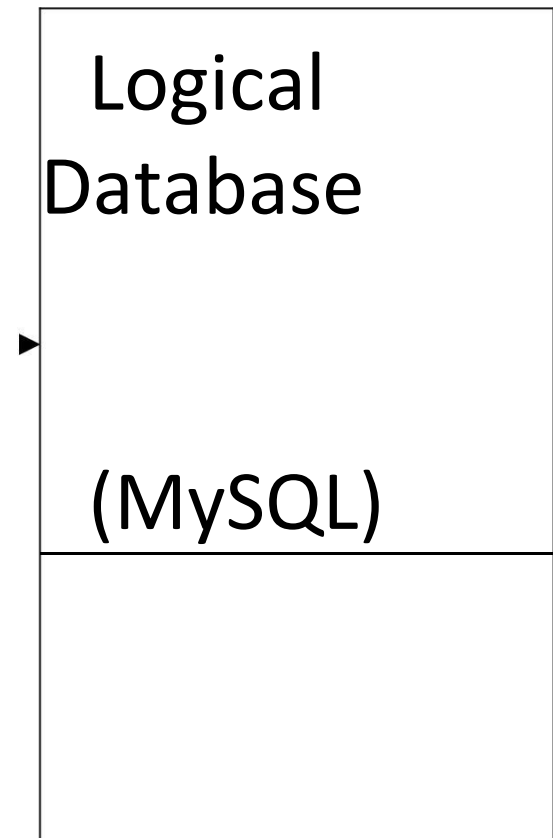
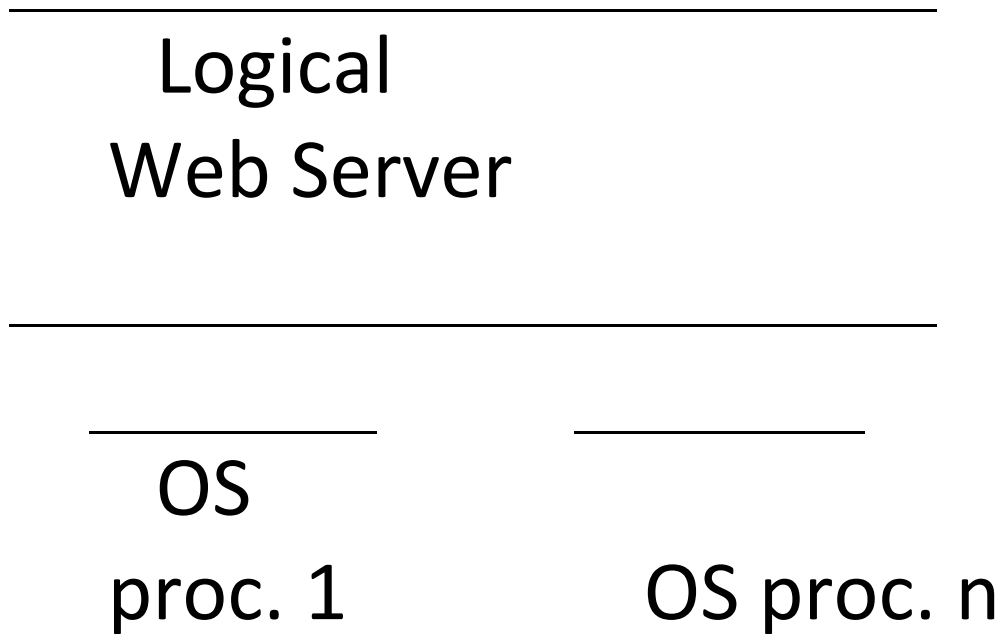
Another advantage of using a programming language that has been designed for the creation of web apps.

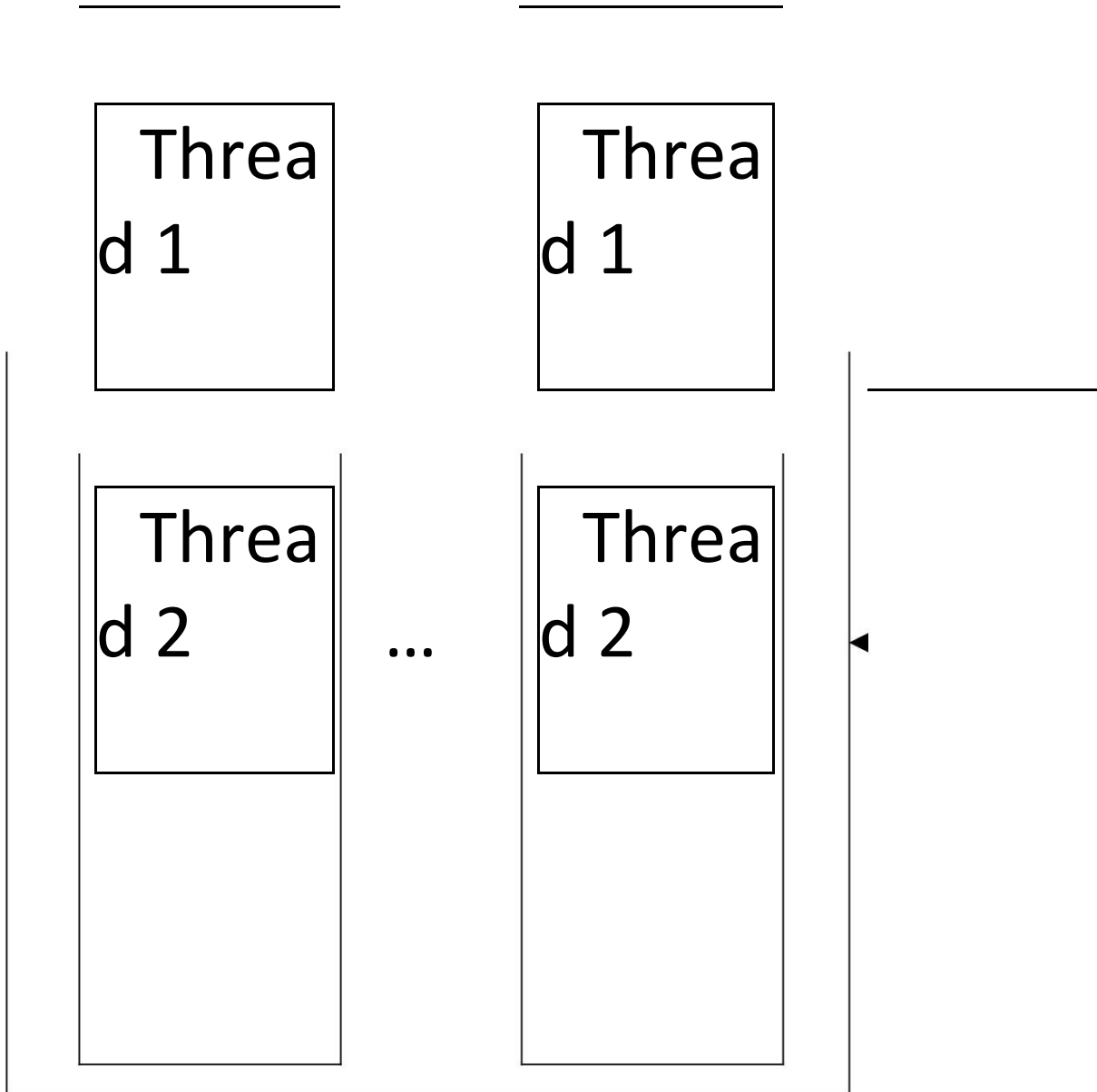
Support for each database is described in the PHP manual at:

– <http://www.php.net/manual/en/>

MySQL and PHP

- Architecture diagram





... ..

Threa
d n

Threa
d n

Connecting to MySQL

To connect to a database, need to create a connection

- At lowest level, this is a network connection
- Involves a login sequence (username/password)

Since this is a relatively expensive step, web application environments:

- Share connections
- Have multiple connections

Whether, and how many, are typical configuration items. In MySQL:

- `Allow_persistent`: whether to allow persistent connections

- Max_persistent: the maximum number of persistent connections

- Max_links: max number of connections, persistent and not

- Connection_timeout: how long the persistent connection is left open

- Can also use SSL to encrypt connection

High-Level Process of Using

MySQL from PHP

Create a database connection

Select database you wish to use

Perform a SQL query

Do some processing on query results

Close database connection

Creating Database Connection

Use either `mysql_connect` or `mysql_pconnect` to create database connection

- `mysql_connect`: connection is closed at end of script (end of page)

- `mysql_pconnect`: creates persistent connection

connection remains even after end of the page

Parameters

- Server – hostname of server
- Username – username on the database
- Password – password on the database

– New Link (mysql_connect only) – reuse database connection created by previous call to mysql_connect

– Client Flags

MYSQL_CLIENT_SSL :: Use SSL

MYSQL_CLIENT_COMPRESS :: Compress data sent to MySQL

Security Note

Username and password fields imply that database password is sitting there in the source code

- If someone gains access to source code, can compromise the database

- Servers are sometimes configured to view PHP source code when a resource is requested with “.phps” instead of “.php”

- One approach to avoid this: put this information in Web server config. File

Then ensure the Web server config. file is not externally accessible

Selecting a Database

`mysql_select_db()`

- Pass it the database name

Related:

- `mysql_list_dbs()`

List databases available

– `Mysql_list_tables()`

List database tables available

Perform SQL Query

Create query string

– `$query = 'SQL formatted string'`

– `$query = 'SELECT * FROM table'`

Submit query to database for processing

– `$result = mysql_query($query);`

- For UPDATE, DELETE, DROP, etc, returns TRUE or FALSE

- For SELECT, SHOW, DESCRIBE or EXPLAIN, \$result is an identifier for the results, and does not contain the results themselves

 - \$result is called a “resource” in this case

A result of FALSE indicates an error

If there is an error

- `mysql_error()` returns error string from last MySQL call

Process Results

Many functions exist to work with database results

`mysql_num_rows()`

- Number of rows in the result set

- Useful for iterating over result set

`mysql_fetch_array()`

- Returns a result row as an array
- Can be associative or numeric or both (default)
- `$row = mysql_fetch_array($result);`

- `$row['column name']` :: value comes from database row with specified column name
- `$row[0]` :: value comes from first field in result set

Process Results Loop

- Easy loop for processing results:

```
$result = mysql_query($qstring);
```

```
$num_rows = mysql_num_rows($result); for  
($i=0; $i<$num_rows; $i++) {
```

```
$row = mysql_fetch_array($result);  
  
// take action on database results here  
  
}
```

Closing Database Connection

`mysql_close()`

- Closes database connection
- Only works for connections opened with `mysql_connect()`

- Connections opened with `mysql_pconnect()` ignore this call
- Often not necessary to call this, as connections created by `mysql_connect` are closed at the end of the script anyway

Model View Controller

Bayu Priyambadha,
S.Kom

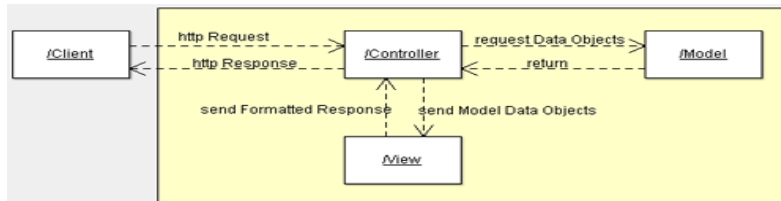
PHP & MVC

- The model view controller pattern is the most used pattern for today's world web applications
- It has been used for the first time in Smalltalk and then adopted and popularized by Java
- At present there are more than a dozen PHP web frameworks based on MVC pattern

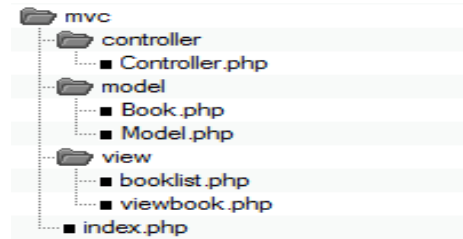
PHP & MVC

- The **model** is responsible to manage the data
- The **view (presentation)** is responsible to display the data provided by the model in a specific format
- The **controller** handles the model and view layers to work together

PHP & MVC



PHP & MVC



model/Book.php

```
<?php
class Book {
    public $title;
    public $author;
    public $description;

    public function __construct($title, $author,
    $description)
    {
        $this->title = $title;
        $this->author = $author;
        $this->description = $description;
    }
}
?>
```

model/Model.php

```
<?php
include_once("model/Book.php");

class Model {
    public function getBookList()
    {
        // here goes some hardcoded values to simulate the database
        return array(
            "Jungle Book" => new Book("Jungle Book", "R. Kipling", "A classic book."),
            "Moonwalker" => new Book("Moonwalker", "J. Walker", ""),
            "PHP for Dummies" => new Book("PHP for Dummies", "Some Smart Guy", "")
        );
    }
    public function getBook($title)
    {
        // we use the previous function to get all the books
        // and then we return the requested one.
        // in a real life scenario this will be done through
        // a database select command
        $allBooks = $this->getBookList();
        return $allBooks[$title];
    }
}
?>
```

view/viewbook.php

```
<html>
<head></head>
<body>
  <?php
    echo 'Title:' . $book->title . '<br/>';
    echo 'Author:' . $book->author . '<br/>';
    echo 'Description:' . $book->description . '<br/>';
  ?>
</body>
</html>
```

view/booklist.php

```
<html>
<head></head>
<body>
  <table>
    <tbody>
      <tr><td>Title</td><td>Author</td><td>Description</td></tr>
    </tbody>
  </table>
  <?php
    foreach ($books as $book)
    {
      echo '<tr><td><a href="index.php?book=' .
        $book->title . '>' . $book->title .
      '</a></td><td>' .
        $book->author . '</td><td>' . $book-
      >description . '</td></tr>';
    }
  ?>
</table>
</body>
</html>
```

controller/Controller.php

```
<?php
include_once("model/Model.php");

class Controller {
    public $model;

    public function __construct()
    {
        $this->model = new Model();
    }
}
```

controller/Controller.php

```
public function invoke()
{
    if (!isset($_GET['book']))
    {
        // no special book is requested, we'll show a
list of all available books
        $books = $this->model->getBookList();
        include 'view/booklist.php';
    }
    else
    {
        // show the requested book
        $book = $this->model->getBook($_GET['book']);
        include 'view/viewbook.php';
    }
}
?>
```


index.php

```
<?php

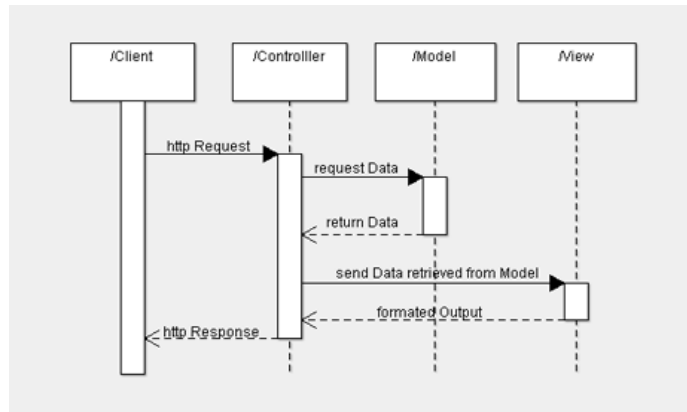
// All interaction goes through the index and is
forwarded
// directly to the controller

include_once("controller/Controller.php");

$controller = new Controller();
$controller->invoke();

?>
```

PHP & MVC





What is AJAX ?

- AJAX is not a new programming language, but a new technique for creating better, faster, and more interactive web applications.
- With AJAX, a JavaScript can communicate directly with the server, with the **XMLHttpRequest** object. With this object, a JavaScript can trade data with a web server, without reloading the page.
- AJAX uses asynchronous data transfer (HTTP requests) between the browser and the web server, allowing web pages to request small bits of information from the server instead of whole pages.
- The AJAX technique makes Internet applications smaller, faster and more user-friendly.

AJAX example



```
var xmlhttp
function showHint(str) {
  if (str.length==0) {
    document.getElementById("txtHint").innerHTML="";
    return;
  }
  xmlhttp=GetXmlHttpRequest();
  if (xmlhttp==null) {
    alert ("Your browser does not support XMLHttpRequest!");
    return;
  }
  var url="submit.php";
  url=url+"?q="+str;
  url=url+"&sid="+Math.random();
  xmlhttp.onreadystatechange=stateChanged;
  xmlhttp.open("GET",url,true);
  xmlhttp.send(null);
}
function stateChanged() {
  if (xmlhttp.readyState==4) {
    document.getElementById("txtHint").innerHTML=xmlhttp.responseText;
  }
}
function GetXmlHttpRequest() {
  if (window.XMLHttpRequest) { // code for IE7+, Firefox, Chrome, Opera, Safari
    return new XMLHttpRequest();
  }
  if (window.ActiveXObject) { // code for IE6, IE5
    return new ActiveXObject("Microsoft.XMLHTTP");
  }
  return null;
}
```

PHP AND XML:

XML is a markup language that looks a lot like HTML. An XML document is plain text and contains tags delimited by < and >. There are two big differences between XML and HTML –

- XML doesn't define a specific set of tags you must use.
- XML is extremely picky about document structure.

XML gives you a lot more freedom than HTML. HTML has a certain set of tags: the <a> tags surround a link, the <p> starts paragraph and so on. An XML document, however, can use any tags you want. Put <rating></rating> tags around a movie rating, <height></height> tags around someone's height. Thus XML gives you option to device your own tags.

XML is very strict when it comes to document structure. HTML lets you play fast and loose with some opening and closing tags. But this is not the case with XML.

HTML list that's not valid XML

```
<ul>
  <li>Braised Sea Cucumber
  <li>Baked GIBLETS with Salt
  <li>Abalone with Marrow and Duck Feet
</ul>
```

This is not a valid XML document because there are no closing tags to match up with the three opening tags. Every opened tag in an XML document must be closed.

HTML list that is valid XML

```
<ul>
  <li>Braised Sea Cucumber</li>
  <li>Baked Giblets with Salt</li>
  <li>Abalone with Marrow and Duck Feet</li>
</ul>
```

Parsing an XML Document

PHP 5's new **SimpleXML** module makes parsing an XML document, well, simple. It turns an XML document into an object that provides structured access to the XML.

X
by Counterflix

To create a SimpleXML object from an XML document stored in a string, pass the string to **simplexml_load_string()**. It returns a SimpleXML object.

Example

Try out following example –

```
<html>
  <body>

    <?php
      $note=<<<XML
```

```
<note>
  <to>Gopal K Verma</to>
  <from>Sairamkrishna</from>
  <heading>Project submission</heading>
  <body>Please see clearly </body>
</note>
```

XML;

```
$xml=simplexml_load_string($note);
```

```
print_r($xml);
```

```
?>
```

```
</body>
```

```
</html>
```

It will produce the following result –

```
SimpleXMLElement Object ( [to] => Gopal K Verma [from] => Sairamkrishna [heading]
=> Project submission [body] => Please see clearly )
```

NOTE – You can use function **simplexml_load_file(filename)** if you have XML content in a file.

For a complete detail of XML parsing function check PHP Function Reference.

Generating an XML Document

SimpleXML is good for parsing existing XML documents, but you can't use it to create a new one from scratch.

X
by Counterflix

The easiest way to generate an XML document is to build a PHP array whose structure mirrors that of the XML document and then to iterate through the array, printing each element with appropriate formatting.

Example

Try out following example –

```
<?php
$channel = array('title' => "What's For Dinner",
    'link' => 'http://menu.example.com/',
    'description' => 'Choose what to eat tonight.');
```



```
print "<channel>\n";
```



```
foreach ($channel as $element => $content) {
    print " <$element>";
    print htmlentities($content);
    print "</$element>\n";
}
```



```
print "</channel>";
```

```
?>
```

It will produce the following result –

```
<channel>  
  <title>What's For Dinner</title>  
  <link>http://menu.example.com/</link>  
  <description>Choose what to eat tonight.</description>  
</channel>
```