UNIT-V JAVASCRIPT

What is JavaScript?

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

It is an interpreted programming language with object-oriented capabilities.

JavaScript was first known as LiveScript, but Netscape changed itsnameto JavaScript, possibly because of the excitement being generated by Java. JavaScript made its first appearance in Netscape 2.0 in 1995 with the name LiveScript.

The general-purpose core of the language has been embedded in Netscape, Internet Explorer, and oth er web browsers.The

ECMA-262 Specification defined a standard version of the core JavaScript language.

- JavaScript is a lightweight, interpreted programming language.
- Designed for creating network-centric applications.
- Complementary to and integrated with Java.
- Complementary to and integrated with HTML.
- Open and cross-platform.

Client-Side JavaScript

Client-side JavaScript is the most common form of the language. The script should be included in or referenced by an HTML document for the code to be interpreted by the browser.It means that a web page need not be a static HTML, but can include programs that interact with the user, control the browser, and dynamically create HTML content.

The JavaScript client-side mechanism provides many advantages over traditional CGI server-side scripts. For example, you might use JavaScript to check if the user has entered a valid e-mail address in a form field.The JavaScript code is executed when the user submits the form, and only if all the entries are valid,they would be submitted to the Web Server.JavaScript can be used to trap userinitiated events such as button clicks, link navigation, and other actions that the user initiates explicitly or implicitly

OVERVIEW

Javascript

Advantages of JavaScript

The merits of using JavaScript are:

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

Limitations of JavaScript

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

- Client-side JavaScript does not allow the reading or writing of files. This has been kept for security reason.
- JavaScript cannot be used for networking applications because there is no such support available.
- JavaScript doesn't have any multithreading or multiprocessor capabilities.Once again, JavaScript is a lightweight, interpreted programming language that allows you to build interactivity into otherwise static HTML pages.

JavaScript Development

Tools One of major strengths of JavaScript is that it does not require expensive development tools. You can start with a simple text editor such as Notepad.Since it is an interpreted language inside the context of a web browser, you don't even need to buy a compiler.To make our life simpler, various vendors have come up with very nice JavaScript editing tools.

Some of them are listed here:

Microsoft FrontPage:Microsoft has developed a popular HTML editor called FrontPage. FrontPage also provides web developers with a number of JavaScript tools to assist in the creation of interactive websites.

Macromedia Dreamweaver MX:Macromedia Dreamweaver MX is a very popular HTML and JavaScript editor in the professional web development crowd. It provides several handy prebuilt JavaScript

Javascript components, integrates well with databases, and conforms to new standards such as XHTML and XML.

Macromedia HomeSite 5:HomeSite 5 is a well-liked HTML and JavaScript editor from Macromedia that can be used to manage personal websites effectively.

Where is JavaScript Today?

The ECMAScript Edition 5 standard will be the first update to be released in over four years. JavaScript 2.0 conforms to Edition 5 of the ECMAScript standard, and the difference between the two is extremely minor.The specification for JavaScript 2.0 can be found on the following site: http://www.ecmascript.org/Today, Netscape's JavaScript and Microsoft's JScript conform to the ECMAScript standard, although both thelanguages still supportthe features that are not a part of the standard.

JavaScript Variables

Like many other programming languages, JavaScript has variables. Variables can be thought of as named containers. You can place data into these containers and then refer to the data simply by naming the container.Before you use a variable in a JavaScript program, you must declare it. Variables are declared with the var keyword as follows.

```
<script type="text/javascript">
<!--var money; var name;//-->
</script>
```

You can also declare multiple variables with the same var keyword as follows:

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

<!--var money, name;//-->
</script>

Storing a value in a variable is called variable initialization. You can do variable initialization at the time of variable creation or at a later point in time when you need that variable.

For instance, you might create a variable named money and assign the value 2000.50 to it later. For another variable,you can assign a value at the time of initialization as follows. <script type="text/javascript">

<!--var name = "Ali"; var money; money = 2000.50;//-->
</script>

Note:Use the var keyword only for declaration or initialization, once for the life of any variable name in a document. You should not re-declare same variable twice.JavaScript isuntypedlanguage. This means that a JavaScript variable can hold a value of any data type. Unlike many other languages, you don't have to tell JavaScript during variable declaration what type of value the variable will hold. The value type of a variable can change during the execution of a program and JavaScript takes care of it automatically.

JavaScript Variable ScopeThe scope of a variable is the region of your program in which it is defined. JavaScript variables have only two scopes.

- Global Variables:A global variable has global scope which means it can be defined Any where in your JavaScript code.
- Local Variables:A local variable will be visible only within a function where it is defined. Function parameters are always local to that function.Javascript Within the body of a function, a local variable takes precedence over a global variable with the same name. If you declare a local variable or function parameter with the same name as a global variable, you effectively hide the global variable .

Take a look into the following example.

<script type="text/javascript">
<!--var myVar = "global"; // Declare a global variable
function checkscope( ) { var myVar = "local"; //
Declare a local variable document.write(myVar);
}
//-->
</script>

It will produce the following result:

Local

JavaScript Variable Names While naming your variables in JavaScript, keep the following rules in mind.

You should not use any of the JavaScript reserved keywords as a variable name. These keywords are mentioned in the next section. For example, break or Boolean variable names are not valid.

JavaScript variable names should not start with a numeral (0-9). They must begin with a letter or an underscore character. For example,123test is an invalid variable name but_123test is a valid one.

JavaScript variable names are case-sensitive. For example, Name and nameare two different variables. Javascript

What is an Event?

JavaScript's interaction with HTML is handled through events that occur when the user or the browser manipulates a page.When the page loads, it is called an event. When the user clicks a button, that click too is an event.

Other

Examples include events like pressing any key, closing a window, resizing a window,etc.Developers can use these events to execute JavaScript coded responses, which cause buttons to close windows, messages to be displayed to users, data to be validated, and virtually any other type of response imaginable.

Events are a part of the Document Object Model (DOM) Level 3 and every HTML element containsa set of events which can trigger JavaScript Code.

. Here we will see a few examples to understand the relation between

Event and JavaScript. onclick Event TypeThis is the most frequently used event type which occurs when a user clicks the left buttonof his mouse. You can put your validation, warning etc.,against this event type.

Example

Try the following example.

<html>

<head>

<script type="text

```
/javascript"> <!-- function
sayHello() { document.write
("Hello World")
}
//
-->
```

UNIT 4-JSP

JSP

Overview

Jsp ava Server Pages (JSP) is a technology for developing web pages that support dynamic content which helps developers insert java code in HTML pages by making use of special JSP tags, most of which start with <% and end with %>.

A JavaServer Pages component is a type of Java servlet that is designed to fulfill the role of a user interface for a Java web application. Web developers write JSPs as text files that combine HTML or XHTML code, XML elements, and embedded JSP actions and commands. Using JSP, you can collect input from users t hrough web page forms, present records from a database or another source, and create web pages dynamically.

JSP tags can be used for a variety of purposes, such as retrieving information from a database or registering user preferences, accessing JavaBeans components, passing control between pages and sharing information between requests, pages etc.

Why Use JSP?

JavaServer Pages often serve the same purpose as programs implemented using the Common Gateway Interface (CGI). But JSP offer several advantages in comparison with the CGI.

- Performance is significantly better because JSP allows embedding Dynamic Elements in HTML Pages itself instead of having a separate CGI files.
- JSP are always compiled before it's processed by the server unlike CGI/Perl which requires the server to load an interpreter and the target script each time the page is requested.
- JavaServer Pages are built on top of the Java Servlets API, so like Servlets, JSP also has access to all the powerful Enterprise Java APIs, including JDBC, JNDI, EJB, JAXP etc.

JSP pages can be used in combination with servlets that handle the business logic, the model supported by Java servlet template engines.

Finally, JSP is an integral part of J2EE, a complete platform for enterprise class applications. This mean

s that JSP can play a part in the simplest applications to the most complex and demanding.C

Advantages of JSP:

Following is the list of other advantages of using JSP over other technologies:

vs. Active Server Pages (ASP):The advantages of JSP are twofold. First, the dynamic part is written in Java, not Visual Basic or other MS specific language, so it is more powerful and easier to use. Second, it is portable to other operating systems and non-Microsoft Web servers.

vs. Pure Servlets:It is more convenient to write (and to modify!) regular HTML than to have plenty of println statements that generate the HTML.

vs. Server-Side Includes (SSI):SSI is really only intended for simple inclusions, not for "real" programs that use form data, make database connections, and the like.

vs. JavaScript:JavaScript can generate HTML dynamically on the client but can hardly interact with the web server to perform complex tasks like database access and image processing etc.

vs. Static HTML:Regular HTML, of course, cannot contain dynamic information

JSP Declarations:

A declaration declares one or more variables or methods that you can use in Java code later in the JSP file. You must declare the variable or method before you use it in the JSP file.

Following is the syntax of JSP Declarations:

<%!declaration;[declaration;]+...%>

You can write XML equivalent of the above syntax as follows:

<jsp:declaration> code

fragment

</jsp:declaration>

Following is the simple example for JSP Declarations:

<%!int i =0;%>

<%!int a,b,c;%>

<%!Circle a =newCircle(2.0);%>

<\%Represents static <% literal.%\> Represents static %> literal.\'A single quote in an attribute that uses single quotes.\"A double quote in an attribute that uses double quotes.

JSP Directives:

A JSP directive affects the overall structure of the servlet class. It usually has the following form:

<%@directive attribute="value"%>

There are three types of directive tag:

Directive

Description

<%@ page ... %>

Defines page

-

dependent attributes, such as scripting language, error page, and buffering requirements.

<%@ include ... %>

Includes a file during the translation phase.

<%@ tag lib ... %>

Declares a tag library, containing custom actions, used in the page

The <jsp:forward> Action

The forward action terminates the action of the current page and forwards the request to another resource such as a static page, another JSP page, or a Java Servlet.

The simple syntax of this action is as follows:

<jsp:forward page="Relative URL"/>

Following is the list of required attributes associated with forward action:

Attribute

Description page

Should consist of a relative URL of another resource such as a static page, another JSP page, or a Java Servlet Example:

Let us reuse following two files (a) date.jps and (b) main.jsp as follows:

Following is the content of date.jsp file:

```
<p>
Today's date:
<%=(new java.util.Date()).toLocaleString()%>
</p>
```

Here is the content of main.jsp file:

```
<html>
<head>
<title>
The include Action Example
</title>
</head>
<body>
<center>
<h2>
The include action Example
</h2>
<jsp:forward page="date.jsp"/>
</center>
</body>
</html>
```

Now let us keep all these files in root directory and try to access main.jsp. This would display r esult something like as below. Here it discarded content from main page and displayed content from forwarded page only.

Today's date: 12-Sep-2010 14:54:22

The <jsp:plugin> Action

The plugin action is used to insert Java components into a JSP page. It dete rmines the type of browser and inserts the <object> or <embed> tags as needed.If the needed plugin is not present, it downloads the plugin and then executes the Java component. The Java component can be either an Applet or a JavaBean.

The plugin action has several attributes that correspond to common HTML tags used to format Java components.

The <param> element can also be used to send parameters to the Applet or Bean.

Following is the typical syntax of using plugin action:

<jsp:plugin type="applet"code base="dirname"code="MyApplet.class" width="60",height="80">

<jsp:paramname="fontcolor" value="red"/>

<jsp:param name="background"value="black"/>

<jsp:fallback>

Unable to initialize Java Plugin

</jsp:fallback>

</jsp:plugin>

You can try this action using some applet if you are interested. A new element, the <fallback> element, can be used to specify an error string to be sent to the user in case the component fails.

The <jsp:element> Action

The <jsp:attribute> Action

The <jsp:body> Action

The <jsp:element>, lt;jsp:attribute> and <jsp:body> actions are used to define XML elements dynamically. The word dynamically is important, because it means that the XML elements can be generated at request time rather than statically at compile time.

Following is a simple example to define XML elements dynamically:

<%@page language ="java" contentType="text/html"%>

<html xmlns=http://www.w3c.org/1999/xhtml xmlns:jsp="http://java.sun.com/JSP/Page">

<head><title>

Genera te XML

Element

</title></head>

<body>

<jsp:element name="xmlElement">

<jsp:attribute name="xmlElementAttr">

Value for the attribute

</jsp:attribute>

<jsp:body>

Body for XML element

</jsp:body>

</jsp:element>

</body>

</html>

This would produce followin g

HTML code at run time:

<html xmlns=http://www.w3c.org/1999/xhtml xmlns:jsp="http://java.sun.com/JSP/Page">

<head><title>

Generate XML Element

</title></head>

<body>

<xmlElement xmlElementAttr= "Value for the attribute">

Body for XML element

</xmlElement>

</body>

</html>

The <jsp:text> Action

The <jsp:text> action can be used to write template text in JSP pages and documents. Following is the simple syntax for this action:

<jsp:text>

Template data

</jsp:text>

The body fo the template cannot contain other elements; it can only contain text and EL expressions (Note: EL expressions are explained in subsequent chapter). Note that in XML files, you cannot use expressions such as ${whatever > 0}, because the greater than signs are illegal. Instead, use the gt form, such as ${whatever gt 0} or an alternative is to embed the value in a CDATA section.

Unit-3 Servlets

Java Servlets are programs that run on a Web or Application server and act as a middle layer between a request coming from a Web browser or other HTTP client and databases or applications on the HTTP server.

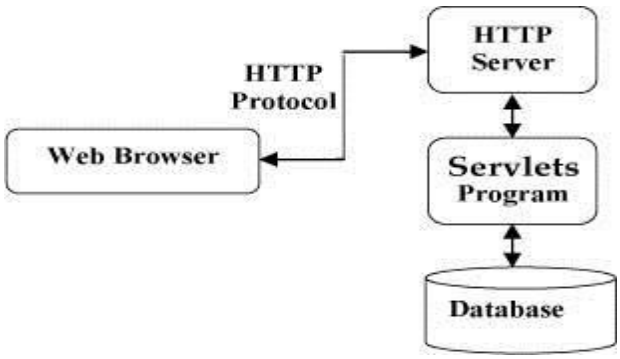
Using Servlets, you can collect input from users through web page forms, present records from a database or another source, and create web pages dynamically.

Java Servlets often serve the same purpose as programs implemented using the Common Gateway Interface (CGI). But Servlets offer several advantages in comparison with the CGI.

- Performance is significantly better.
- Servlets execute within the address space of a Web server. It is not necessary to create a separate process to handle each client request.
- Servlets are platform-independent because they are written in Java.
- Java security manager on the server enforces a set of restrictions to protect the resources on a server machine. So servlets are trusted.
- The full functionality of the Java class libraries is available to a servlet. It can communicate with applets, databases, or other software via the sockets and RMI mechanisms that you have seen already.

Servlets Architecture:

Following diagram shows the position of Servelts in a Web Application.



Servlets Tasks:

Servlets perform the following major tasks:

- Read the explicit data sent by the clients (browsers). This includes an HTML form on a Web page or it could also come from an applet or a custom HTTP client program.
- Read the implicit HTTP request data sent by the clients (browsers). This includes cookies, media types and compression schemes the browser understands, and so forth.
- Process the data and generate the results. This process may require talking to a database, executing an RMI or CORBA call, invoking a Web service, or computing the response directly.
- Send the explicit data (i.e., the document) to the clients (browsers). This document can be sent in a variety of formats, including text (HTML or XML), binary (GIF images), Excel, etc.
- Send the implicit HTTP response to the clients (browsers). This includes telling the browsers or other clients what type of document is being returned (e.g., HTML), setting cookies and caching parameters, and other such tasks.

Servlets Packages:

Java Servlets are Java classes run by a web server that has an interpreter that supports the Java Servlet specification.

Servlets can be created using the **javax.servlet** and **javax.servlet.http** packages, which are a standard part of the Java's enterprise edition, an expanded version of the Java class library that supports large-scale development projects.

These classes implement the Java Servlet and JSP specifications. At the time of writing this tutorial, the versions are Java Servlet 2.5 and JSP 2.1.

Java servlets have been created and compiled just like any other Java class. After you install the servlet packages and add them to your computer's Classpath, you can compile servlets with the JDK's Java compiler or any other current compiler.

Life cycle of servlet

A servlet life cycle can be defined as the entire process from its creation till the destruction. The following

are the paths followed by a servlet

- The servlet is initialized by calling the **init ()** method.
- The servlet calls **service()** method to process a client's request.
- The servlet is terminated by calling the **destroy()** method.
- Finally, servlet is garbage collected by the garbage collector of the JVM. Now let us discuss the life cycle methods in details.

The init() method :

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

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

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

The init method definition looks like this:

```
publicvoid init()throwsServletException{ //
Initialization code...
}
```

The service() method :

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

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

Here is the signature of this method:

```
publicvoid service(ServletRequest request, ServletResponse response)
throwsServletException,IOException{
}
```

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

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

The doGet() Method

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

```
publicvoid doGet(HttpServletRequest request, HttpServletResponse response)
throwsServletException,IOException{
// Servlet code
}
```

The doPost() Method

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

```
publicvoid doPost(HttpServletRequest request, HttpServletResponse response)
throwsServletException,IOException{
// Servlet code
}
```

The destroy() method :

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

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

```
publicvoid destroy(){ //
Finalization code...
}
```

Servlet Deployment:

By default, a servlet application is located at the path <Tomcat-installationdirectory>/webapps/ROOT and the class file would reside in <Tomcat-installationdirectory>/webapps/ROOT/WEB-INF/classes.

If you have a fully qualified class name of **com.myorg.MyServlet**, then this servlet class must be located in WEB- INF/classes/com/myorg/MyServlet.class.

For now, let us copy HelloWorld.class into <Tomcat-installation-directory>/webapps/ROOT/WEBINF/classes and create following entries in **web.xml** file located in <Tomcat-installationdirectory>/webapps/ROOT/WEB-INF/

```
<servlet>
<servlet-name>HelloWorld</servlet-name>
<servlet-class>HelloWorld</servlet-class>
</servlet>

<servlet-mapping>
<servlet-name>HelloWorld</servlet-name>
<url-pattern>/HelloWorld</url-pattern>
</servlet-mapping>
```

Above entries to be created inside <web-app>...</web-app> tags available in web.xml file. There could be various entries in this table already available, but never mind.

You are almost done, now let us start tomcat server using <Tomcat-installation-directory>\bin\startup.bat (on windows) or <Tomcat-installation-directory>/bin/startup.sh (on Linux/Solaris etc.) and finally type **http://localhost:8080/HelloWorld** in browser's address box. If everything goes fine, you would get following result:

UNIT 2-XML

XML stands for **E**xtensible **M**arkup **L**anguage. It is a text-based markup language derived from Standard Generalized Markup Language (SGML).

XML tags identify the data and are used to store and organize the data, rather than specifying how to display it like HTML tags, which are used to display the data. XML is not going to replace HTML in the near future, but it introduces new possibilities by adopting many successful features of HTML.

There are three important characteristics of XML that make it useful in a variety of systems and solutions:

- **XML is extensible:** XML allows you to create your own self-descriptive tags, or language, that suits your application.
- **XML carries the data, does not present it:** XML allows you to store the data irrespective of how it will be presented.
- **XML is a public standard:** XML was developed by an organization called the World Wide Web Consortium (W3C) and is available as an open standard.

XML Usage

A short list of XML usage says it all:

- XML can work behind the scene to simplify the creation of HTML documents for large web sites.
- XML can be used to exchange the information between organizations and systems.
- XML can be used for offloading and reloading of databases.
- XML can be used to store and arrange the data, which can customize your data handling needs.

- XML can easily be merged with style sheets to create almost any desired output.
- Virtually, any type of data can be expressed as an XML document.

What is Markup?

XML is a markup language that defines set of rules for encoding documents in a format that is both human-readable and machine-readable. So what exactly is a markup language? Markup is information added to a document that enhances its meaning in certain ways, in that it identifies the parts and how they relate to each other. More specifically, a markup language is a set of symbols that can be placed in the text of a document to demarcate and label the parts of that document.

Following example shows how XML markup looks, when embedded in a piece of text:

```
<message>
<text>Hello, world!</text>
</message>
```

This snippet includes the markup symbols, or the tags such as <message>...</message> and <text>...</text>. The tags <message> and </message> mark the start and the end of the XML code fragment. The tags <text> and </text> surround the text Hello, world!.

Is XML a Programming Language?

A programming language consists of grammar rules and its own vocabulary which is used to create computer programs. These programs instructs computer to perform specific tasks.

perform any computation or algorithms. It is usually stored in a simple text file and is processed by special software that is capable of interpreting XML.

Tags and Elements

An XML file is structured by several XML-elements, also called XML-nodes or XML- tags. XML-elements' names are enclosed by triangular brackets < > as shown below: <element>

Syntax Rules for Tags and Elements

Element Syntax: Each XML-element needs to be closed either with start or with end elements as shown below:

<element>....</element>

or in simple-cases, just this way:

<element/>

Nesting of elements: An XML-element can contain multiple XML-elements as its children, but the children elements must not overlap. i.e., an end tag of an element must have the same name as that of the most recent unmatched start tag.

Following example shows incorrect nested tags:

```
<?xml version="1.0"?>
<contact-info>
<company>IARE
<contact-info>
</company>
```

Following example shows correct nested tags:

```
<?xml version="1.0"?>
<contact-info>
<company>IARE</company>
<contact-info>
```

Let us learn about one of the most important part of XML, the XML tags. XML tags form the foundation of XML. They define the scope of an element in the XML. They can also be used to insert comments, declare settings required for parsing the environment and to insert special instructions.

We can broadly categorize XML tags as follows:

Start Tag

The beginning of every non-empty XML element is marked by a start-tag. An example of start-tag is:

<address>

End Tag

Every element that has a start tag should end with an end-tag. An example of end- tag is:

</address>

Note that the end tags include a solidus ("/") before the name of an element.

Empty Tag

The text that appears between start-tag and end-tag is called content. An element which has no content is termed as **empty**. An **empty** element can be represented in two ways as below:

(1) A start-tag immediately followed by an end-tag as shown below:

<hr></hr>

(2) A complete empty-element tag is as shown below:

<hr />

Empty-element tags may be used for any element which has no content.

XML Tags Rules

Following are the rules that need to be followed to use XML tags:

Rule 1

XML tags are case-sensitive. Following line of code is an example of wrong syntax </Address>, because of the case difference in two tags, which is treated as erroneous syntax in XML.

<address>This is wrong syntax</Address>

Following code shows a correct way, where we use the same case to name the start and the end tag.

<address>This is correct syntax</address>

Rule 2

XML tags must be closed in an appropriate order, i.e., an XML tag opened inside another element must be closed before the outer element is closed. For example: <outer_element>

```
<internal_element>
    This tag is closed before the outer_element
</internal_element>
</outer_element>
```

XML Elements

XML elements can be defined as building blocks of an XML. Elements can behave as containers to hold text, elements, attributes, media objects or all of these.

Each XML document contains one or more elements, the scope of which are either delimited by start and end tags, or for empty elements, by an emptyelement tag.

Syntax

Following is the syntax to write an XML element:

```
<element-name attribute1 attribute2>
....content
</element-name>
```

where

- **element-name** is the name of the element. The name its case in the start and end tags must match.
- **attribute1, attribute2** are attributes of the element separated by white spaces. An attribute defines a property of the element. It associates a name with a value, which is a string of characters. An attribute is written as:

```
name = "value"
```

The name is followed by an = sign and a string value inside double(" ") or single(' ') quotes.

Empty Element

An empty element (element with no content) has following syntax: <name attribute1 attribute2.../>

Example of an XML document using various XML element:

```
<?xml version="1.0"?>
<contact-info>
<address category="residence">
<name>Tanmay Patil</name>
<company>TutorialsPoint</company>
<phone>(011) 123-4567</phone>
<address/>
</contact-info>
```

XML Elements Rules

Following rules are required to be followed for XML elements:

- An element name can contain any alphanumeric characters. The only punctuation marks allowed in names are the hyphen (-), under-score (_) and period (.).
- Names are case sensitive. For example, Address, address, and ADDRESS are different names.
- Start and end tags of an element must be identical.
- An element, which is a container, can contain text or elements as seen in the above example.

Root element: An XML document can have only one root element. For example, following is not a correct XML document, because both the x and y elements occur at the top level without a root element:

```
<x>...</x>
<y>...</y>
```

The following example shows a correctly formed XML document: <root>

```
    <x>...</x>
    <y>...</y>
</root>
```

Case sensitivity: The names of XML-elements are case-sensitive. That means the name of the start and the end elements need to be exactly in the same case.

For example, **<contact-info>** is different from **<Contact-Info>**.

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

An XML DTD can be either specified inside the document, or it can be kept in a separate document and then liked separately.

Syntax

Basic syntax of a DTD is as follows: <!DOCTYPE

element DTD identifier

- **DTD identifier** is an identifier for the document type definition, which may be the path to a file on the system or URL to a file on the internet. If the DTD is pointing to external path, it is called **External Subset.**
- **The square brackets [ ]** enclose an optional list of entity declarations called Internal Subset.

Internal DTD

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

Syntax

The syntax of internal DTD is as shown:

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

Example

Following is a simple example of internal DTD:

```
<?xml version="1.0" encoding="UTF-8" standalone="yes" ?>
<!DOCTYPE address [
<!ELEMENT address (name,company,phone)>
<!ELEMENT name (#PCDATA)>
```

```
<!ELEMENT company (#PCDATA)>
<!ELEMENT phone (#PCDATA)>
]>
<address>
<name>Tanmay Patil</name>
<company>iare</company>
<phone>(011) 123-4567</phone>
  </address>
```

Let us go through the above code:

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

Start Declaration- Begin the XML declaration with following statement

DTD- Immediately after the XML header, the document type declaration follows, commonly referred to as the DOCTYPE:

```
<!DOCTYPE address [
```

The DOCTYPE declaration has an exclamation mark (!) at the start of the element name. The DOCTYPE informs the parser that a DTD is associated with this XML document. **DTD Body**- The DOCTYPE declaration is followed by body of the DTD, where you declare elements, attributes, entities, and notations:

```
<!ELEMENT address (name,company,phone)>
<!ELEMENT name (#PCDATA)>
<!ELEMENT company (#PCDATA)>
<!ELEMENT phone_no (#PCDATA)>
```

Several elements are declared here that make up the vocabulary of the <name> document. <!ELEMENT name (#PCDATA)> defines the element name to be of type "#PCDATA". Here #PCDATA means parse-able text data.

End Declaration - Finally, the declaration section of the DTD is closed using a closing bracket and a closing angle bracket (]>). This effectively ends the definition, and thereafter, the XML document follows immediately. **Rules**

- The document type declaration must appear at the start of the document (preceded only by the XML header) — it is not permitted anywhere else within the document.
- Similar to the DOCTYPE declaration, the element declarations must start with an exclamation mark.
- The Name in the document type declaration must match the element type of the root element.

External DTD

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

Syntax

Following is the syntax for external DTD:

```
<!DOCTYPE root-element SYSTEM "file-name">
```

where file-name is the file with .dtd extension.

Example

The following example shows external DTD usage:

```
<?xml version="1.0" encoding="UTF-8" standalone="no" ?>
<!DOCTYPE address SYSTEM "address.dtd">
<address>
<name>Tanmay Patil</name> <company>IARE</company>
<phone>(011) 123-4567</phone>
</address>
```

The content of the DTD file **address.dtd** are as shown:

```
<!ELEMENT address (name,company,phone)>
<!ELEMENT name (#PCDATA)>
<!ELEMENT company (#PCDATA)>
    <!ELEMENT phone (#PCDATA)>
```

Types

You can refer to an external DTD by using either **system identifiers** or **public identifiers**.

SYSTEM IDENTIFIERS

A system identifier enables you to specify the location of an external file containing DTD declarations. Syntax is as follows:

```
<!DOCTYPE name SYSTEM "address.dtd" [...]>
```

As you can see, it contains keyword SYSTEM and a URI reference pointing to the location of the document.

PUBLIC IDENTIFIERS

Public identifiers provide a mechanism to locate DTD resources and are written as below:

```
<!DOCTYPE name PUBLIC "-//Beginning XML//DTD Address Example//EN">
```

As you can see, it begins with keyword PUBLIC, followed by a specialized identifier. Public identifiers are used to identify an entry in a catalog. Public identifiers can follow any format, however, a commonly used format is called Formal Public Identifiers, or FPIs.

XML Schema is commonly known as XML Schema Definition (XSD). It is used to describe and validate the structure and the content of XML data. XML schema defines the elements, attributes and data types. Schema element supports Namespaces. It is similar to a database schema that describes the data in a database.

Syntax

You need to declare a schema in your XML document as follows:

```
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema">
```

Example

The following example shows how to use schema:

```
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema">
<xs:element name="contact">
<xs:complexType>
```

<xs:sequence>
<xs:element name="name" type="xs:string" />
<xs:element name="company" type="xs:string" />
<xs:element name="phone" type="xs:int" />
</xs:sequence>
</xs:complexType>
</xs:element>
</xs:schema>

The basic idea behind XML Schemas is that they describe the legitimate format that an XML document can take. Elements

As we saw in the chapter XML - Elements, elements are the building blocks of XML document. An element can be defined within an XSD as follows:

<xs:element name="x" type="y"/>

Definition Types

You can define XML schema elements in following ways:

<xs:element name="phone_number" type="xs:int" />

Simple Type - Simple type element is used only in the context of the text. Some of predefined simple types are: xs:integer, xs:boolean, xs:string, xs:date. For example:

Complex Type - A complex type is a container for other element definitions. This allows you to specify which child elements an element can contain and to provide some structure within your XML documents. For example:

<xs:element name="Address">
<xs:complexType>
<xs:sequence>
<xs:element name="name" type="xs:string" />
<xs:element name="company" type="xs:string" />
<xs:element name="phone" type="xs:int" />
</xs:sequence>
</xs:complexType>
</xs:element>

In the above example, Address element consists of child elements. This is a container for other <xs:element> definitions, that allows to build a simple hierarchy of elements in the XML document.

Global Types - With global type, you can define a single type in your document, which can be used by all other references. For example, suppose you want to generalize the person and company for different addresses of the company. In such case, you can define a general type as below:

```
<xs:element name="AddressType">
<xs:complexType>
<xs:sequence>
<xs:element name="name" type="xs:string" />
<xs:element name="company" type="xs:string" />
</xs:sequence>
</xs:complexType>
</xs:element>

<xs:element name="Address1">
<xs:complexType>
<xs:sequence>
<xs:element name="address" type="AddressType" />
<xs:element name="phone1" type="xs:int" />
</xs:sequence>
</xs:complexType>
</xs:element>
<xs:element name="Address2">
<xs:complexType>
<xs:sequence>
<xs:element name="address" type="AddressType" />
<xs:element name="phone2" type="xs:int" />
</xs:sequence>
</xs:complexType>
  </xs:element>
```

Instead of having to define the name and the company twice (once for Address1 and once for Address2), we now have a single definition. This makes maintenance simpler, i.e., if you decide to add "Postcode" elements to the address, you need to add them at just one place.

At ributes

Attributes in XSD provide extra information within an element. Attributes have name and type property as shown below:

<xs:attribute name="x" type="y"/>

Unit 1-PHP

The PHP Hypertext Preprocessor (PHP) is a programming language that allows web developers to create dynamic content that interacts with databases. PHP is basically used for developing web based software applications.

- PHP is a recursive acronym for "PHP: Hypertext Preprocessor".
- PHP is a server side scripting language that is embedded in HTML. It is used to manage dynamic content, databases, session tracking, even build entire e-commerce sites.

Common uses of PHP

- PHP performs system functions, i.e. from files on a system it can create, open, read, write, and close them.
- PHP can handle forms, i.e. gather data from files, save data to a file, thru email you can send data, return data to the user.
- You add, delete, modify elements within your database thru PHP.
- Access cookies variables and set cookies.
- Using PHP, you can restrict users to access some pages of your website.
- It can encrypt data.

Declaring Variables

In PHP, a variable starts with the $ sign, followed by the name of the variable:

```
<?php
$txt = "Hello world!";
$x = 5;
$y = 10.5;
?>
```

After the execution of the statements above, the variable **$txt** will hold the value **Hello world!**, the variable **$x** will hold the value **5**, and the variable **$y** will hold the value **10.5**.

PHP Variables

A variable can have a short name (like x and y) or a more descriptive name (age, carname, total_volume).

Rules for PHP variables:

- A variable starts with the $ sign, followed by the name of the variable
- A variable name must start with a letter or the underscore character
- A variable name cannot start with a number
- A variable name can only contain alpha-numeric characters and underscores (A-z, 0-9, and _)

- Variable names are case-sensitive ($age and $AGE are two different variables).

PHP is a Loosely Typed Language

In the example above, notice that we did not have to tell PHP which data type the variable is. PHP automatically converts the variable to the correct data type, depending on its value.In other languages such as C, C++, and Java, the programmer must declare the name and type of the variable before using it.

PHP Variables Scope

In PHP, variables can be declared anywhere in the script.

The scope of a variable is the part of the script where the variable can be referenced/used.

PHP has three different variable scopes:

- local
- global
- static

Global And Local Scope

A variable declared **outside** a function has a GLOBAL SCOPE and can only be accessed outside a function:

```
<?php
$x = 5; // global scope

function myTest() {
   // using x inside this function will generate an error
echo "<p>Variable x inside function is: $x</p>";
}
myTest();

echo "<p>Variable x outside function is: $x</p>";
?>
```

A variable declared **within** a function has a LOCAL SCOPE and can only be accessed within that function:

```
<?php function
myTest() {     $x =
5; // local scope
    echo "<p>Variable x inside function is: $x</p>";
}
myTest();

// using x outside the function will generate an error echo
"<p>Variable x outside function is: $x</p>";
?>
```

PHP Data Types

Variables can store data of different types, and different data types can do different things.

PHP supports the following data types:

- String
- Integer
- Float (floating point numbers - also called double)
- Boolean
- Array
- Object □ NULL
- Resource

PHP String

A string is a sequence of characters, like "Hello world!".

A string can be any text inside quotes. You can use single or double quotes:

```
<?php
$x = "Hello world!";
$y = 'Hello world!';

echo $x; echo
"<br>";  echo
$y;
?>
```

PHP Integer

An integer is a whole number (without decimals). It is a number between -2,147,483,648 and +2,147,483,647.

Rules for integers:

- An integer must have at least one digit (0-9)
- An integer cannot contain comma or blanks
- An integer must not have a decimal point
- An integer can be either positive or negative
- Integers can be specified in three formats: decimal (10-based), hexadecimal (16-based - prefixed with 0x) or octal (8-based - prefixed with 0)

In the following example $x is an integer. The PHP var_dump() function returns the data type and value:

```
<?php  $x =
5985;
var_dump($x);
?>
```

PHP Float

A float (floating point number) is a number with a decimal point or a number in exponential form.

In the following example $x is a float. The PHP var_dump() function returns the data type and value:

```
<?php  $x =
10.365;
var_dump($x);
?>
```

PHP Boolean

A Boolean represents two possible states: TRUE or FALSE.

```
$x = true;
$y = false;
```

PHP Array

An array stores multiple values in one single variable.

In the following example $cars is an array. The PHP var_dump() function returns the data type and value:

```
<?php
```

$cars = array("Volvo","BMW","Toyota"); var_dump($cars);

?>

PHP NULL Value

Null is a special data type which can have only one value: NULL.

A variable of data type NULL is a variable that has no value assigned to it.

Array

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

If you have a list of items (a list of car names, for example), storing the cars in single variables could look like this:

$cars1 = "Volvo";

$cars2 = "BMW";

$cars3 = "Toyota";

However, what if you want to loop through the cars and find a specific one? And what if you had not 3 cars, but 300?

The solution is to create an array!

An array can hold many values under a single name, and you can access the values by referring to an index number.

Create an Array in PHP

In PHP, the array() function is used to create an array:

In PHP, there are three types of arrays:

- **Indexed arrays** - Arrays with a numeric index
- **Associative arrays** - Arrays with named keys
- **Multidimensional arrays** - Arrays containing one or more arrays

PHP Indexed Arrays

There are two ways to create indexed arrays:

The index can be assigned automatically (index always starts at 0), like this:

$cars = array("Volvo", "BMW", "Toyota");

The following example creates an indexed array named $cars, assigns three elements to it, and then prints a text containing the array values:

<?php

$cars = array("Volvo", "BMW", "Toyota");

```
echo "I like " . $cars[0] . ", " . $cars[1] . " and " . $cars[2] . ".";
?>
```

PHP Associative Arrays

Associative arrays are arrays that use named keys that you assign to them.

There are two ways to create an associative array:

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

Strings

A string is a sequence of characters, like "Hello world!".

Get The Length of a String

The PHP strlen() function returns the length of a string.

The example below returns the length of the string "Hello world!":

```
<?php echo strlen("Hello world!"); //
outputs 12
?>
```

Count The Number of Words in a String

The PHP str_word_count() function counts the number of words in a string:

```
<?php echo str_word_count("Hello world!"); //
outputs 2
?>
```

The output of the code above will be: 2.

Reverse a String

The PHP strrev() function reverses a string:

```
<?php echo strrev("Hello world!"); // outputs
!dlrowolleH
?>
```

String Concatenation Operator

To concatenate two string variables together, use the dot (.) operator −

```
<?php
```

```
$string1="Hello World";
$string2="1234";
echo $string1 . " " . $string2;
?>
```

This will produce the following result –

Hello World 1234

Using the strpos() function

The strpos() function is used to search for a string or character within a string.

If a match is found in the string, this function will return the position of the first match. If no match is found, it will return FALSE.

Let's see if we can find the string "world" in our string −

```
<?php echostrpos("Hello
world!","world"); ?>
```

This will produce the following result −6

(2)operators: An operator is a symbol that specifies a particular action in an expression.operators are classified into different types (a)Arithmetic operators

(b)Assignment operators

(c)string operators

(d)Increment and decrement operators

(e)Logical operators

(f)Equality operators

(g)Comparision operators

(h)Bitwise operators

a)Arithmetic operators :The arthimetic operators are

operators	Label	Example
+	addition	$a+$b
-	subtraction	$a-$b
*	Multiplication	$a*$b
/	Division	$a/$b
%	modulus	$a%$b

b)Assignment operators: Assignment operators mainly used for to assign a data value to a variable. The simplest form of assignment operator just assigns some value.

Operator	Label	Example
=	Assignment	$a=5
+=	Addition-assignment	$a+=5
*=	multiplication-assignment	$a*=5
/=	division-assignment	$a/=5
.=	concatenation-assignment	$a.=5

(c)String operators:php's string operators provide two operators that two operatorsusefull for concatenation the two strings.

Operator	label	example	output
.concatenation	$a= "abc". "def";	abcdef	
.=	concatenation-assighment	$a.= "ghijkl"	ghijkl

(d)increment and decrement operators:

Increment(++) and decrement operators increment by 1 and decrement by 1 from the current value of a variable

Operatorlabelexampleoutput

++ increment ++$a, $a++ increment $a by 1 --

Decrement --$a, $a-- decrement $a by 1 **(e) logical operators:**

Logical operators make it possible to direct the flow of a program and used frequently with control structures such as the if conditional and while and loops.

Operatorslabelexampleoutput

&& AND $a&&$b true if both $a and $b are true

AND AND $a AND $b true if both $a and $b are true

|| OR $a || $b true if either $a or $b is true

OR OR $a OR $b true if both $a and $b are true

! NOT !$a true if $a is not true

NOT NOTNOT $a true if $a is not true

XOR exclusive $a XOR $b true if only $a (or) only $b is true **(f)**

equality operators:

Equality operators are used to compare two values, testing for equivalence

Operatorslabelexample

<less than $a<$b

> Greater than $a>$b

<= less than or equal to $a<=$b >=

greator than or equal to $a=>$b **(g) bitwise**

operators:

Bitwise operators are used for variations on some of the logical operators

Operatorslabelexampleoutput

& AND $a&$b and together each bit contained in $a and $b

| OR $a|$b or together each bit contained in $a and $b

^ XOR $a^$b exclusive-or together each bit contained

in $a and $b

~ NOT ~$b negate each bit in $b

<<shift left $a<<$b $a will receive the value of $b

shifted left two values.

>> shift right $a>>$b $a will receive the value of $b shifted

right two values.

(3)expressions and statements:-

Expressions:-an expression is a phrase representing a particular action in a program.all expressions consists of a least one **operand** ana one (or) more operations

Ex:-

$a=5; //assign inter value 5 to the variable $a

$a="5"; //assign string value "5" to the variable $a

$a="abit"; //assign "abit" to the variable $a

➔ Here operends are the input expressions

Ex:-$a++; //$a is the operand

$sum=$val1+val2; //$sum,$val1,$val2 are operends

Statements:-php supports different types of statements like

(1)if statement

(2)else statement

(3)switch statement

(4)while statement

(5)do…while statement

(6)for statement

(7)for each statement

(8)break and goto statement

(9)continue statement

(1)switch statement:-

Syntax:-if (expression)

{

Statement

}

->An example,supposeuou want a cougratutory message displayed if the user guesses apredefimindservet number

```
<?php
$sescretnumber=143;
If($_POST[‘guess’]==$secretnumber)
{
Echo “<p>congratulation!</p>”;
}
```

(2)else statement: if the condition si true statement followed if will be execute other else statement will execute

```
<?php
$sescretnumber=143;
If($_POST[‘guess’]==$secretnumber)
{
Echo “<p>congratulation!</p>”;
}
Else
{
Echon “<p>sorry!</p>”;
}
?>
```

(4) switch statement:- switch statement can compare “ = “ operations only ex:-

```
<?php
        $x=1;
Switch($x)
        { case
1:
                Echo “number1”;
```

```
                Break;
            Case 2:echo "number2";
                    Break;
Case 3:
                Echo "number3";
                Break;
Default:
                Echo "no number b/w 1 and 3";
          }
         ?>
```

(5) while statement:- while loop check the condition then only excute the statement when the condition is true.

Syntax:- While(expression)

```
        {
            Statements
        }
```

Ex:-

```
<?php
    $count=1;
While($count <5)
    {
Printf("%d sqared=%d <br>",$count,pow($count,2));
      $count++;
    }
    ?>
```

o/p : 1 squared=1
2 squared=4
3 squared=9
4 squared=16

(6) do while:

It will execute the statement atleast once even condition false(or)true.

Syntax:

```
<?php
    $count=11;
```

```
Do
{
Printf("%d squared=%d<br/>",$count,pow($count,2));
}
While($count<10);
?>
```

(7) for statement: By using this loop we can run number of iteration.

Syntax: for(exp1;exp2;exp3)

```
{
  Statements;
}
```

There are afew rules to keep in mind when using php's for loops.

- ➔ The first expression,exp1,is evaluated by default at the first iteration of the loop
- ➔ The second expression,exp2 is evaluated at the beginning of each iteration.this expression determines wherher looping will continue.
- ➔ The third expression,exp3,is evaluated at conclusion of even loop.

Ex:

```
<?php
For($kilometers=1;$kilometers<=3;$kilometers++)
{
printf("%kilometers=%f miles<br/>",$kilometers,$kilometers*0.62140); }
?>
```

o/p:

1 kilometers= 0.6214 miles

2 kilometers=1.2428 miles

3 kilometers =1.8642 miles .

(a)**Break and goto statement:-**

Break statement:- break statement is end execution of a do while,for,foreach,switch,while block.

Goto statement:- In php goto statement,"BREAK" features was extend to support labels.

This means we can suddenly jump to a specific location outside of a looping or conditional construct. **(10) continue statement**:-

Continue statement execute the current loop iteration to the end.

(4) **string:-** string variable can hold collection of characters . in php we can assign values into the string variables '3' ways.

-> using single quotation

-> using double quotation -

>heradoc style.

->inphp offers approximately 100 function collectively.

->we introduce each function but we have to implement some of the functions of a string.

(1) determining string length

(2) comparing two strings

(3) manipulating string case

(4) alternatives for regular expression functions s(5) converting string to and form HTML

(6) padding and stripping a string

(7) counting characters and works

(1)**determining string length**:- here we are using strlen() function and this function returns the length of the string.

Ex:-intstrlen(string str)

(2)**comparing two string**:- in php provides four functions for performing this task.

1.strcmp ()

2.strcasecmp ()

3.strlen()

4.strcmp()

(1)strcmp():- the strcmp() function performs a binary numbers and compare two strings.

Syntax:-(case-sensitive) Intstrcmp(string str1,string str2)

- ➔ 0, if str1 and str2 are equal.(s1==s2)
- ➔ -1, if str1 is less than str2(s1<s2)
- ➔ 1, if str2 is less than str1(s2<s1)

Ex:-<?php

$pwd="abitcse";

$pwd2="abitcse2"; if

Ex:-<?php

$pwd="abc123";

If(strcspn($pwd,"1234567890")==0)

Echo"pwd can't consists solely of numbers! }

```
?>
```

(2)strspn:- calculating the similarities between two strings.

Syntax: int strspn(string str1,string str2[, int start[, int length]])

Ex:-

```
<?php
    $pwd=”abc123”;
If(strspn($pwd,”1234567890”)==strlen($pwd))
Echo”thepwd  cannot consist solely of numbers!”;
?>
```

(3) mani

```
Ex:-<?php
    $pwd=”abitcse”;
    $pwd2=”abitcse2”;
If(strcmp($pwd,$pwd2)!=0)
    {
Echo”pwd do not match’;
    }
    Else
{
Echo”pwd match”;
    }
   ?>
```

(2) strcasecmp():- (case-insensitive0

The strcasecmp() function operates exactly like strcmp().

Syntax: intstrcasecmp(string str1,string str2)

```
Ex:-<?php
    $gmail1= ”abit @gmail .com ”;
    $gmail2= ” ABIT@ gmail.com ”;
If(!strcasecmp($gmail1,gmail20)
```

Echo" the gmail addresses are identical! ";

?>

(3) strspn():- calculating the simirality between two strings.

Syntax;-intstrspn(string str1,string str2 [, int start [, int length]]]

Ex:-<?php

$pwd="abc123";

If(strspn($pwd,"1234567890")==strlen($pwd))

Echo " thepwd cannot consist solely of numbers! ";

?>

(4) strcspn():- calculating difference between two strings.

Syntax:-intstrcspn(string str1,string str2 [, int start [, int length]])

Ex:-<?php

$pwd= " abc123 ";

If(strcspn($pwd, "1234567890 ")==0)

{

Echo " pwd can't consist solely of numbers! ";

}

?>

(3) manipulating string case:-

In this we have to mainely concentrated on four function.

1. Strtolower()
2. Strtoupper()
3. Ucfirst()
4. Ucwords()

(1) Strtolower:- (converting a string to all lowercase)

Ex:-<?php

$name= " bangaram ";

Echo str to lower($name);

?

(2) Strtoupper():- (converting a string to all uppercase)

Ex:-<?php

```
$name= " JAMALBASHA ";
 Echo strtoupper
?>
```

(3) Ucfirst():- (capitalizing the first letter of a string)

Ex:-<?php

$name=" abit college **";**

```
 Echo strucfirst ($name);
?>
```

(4) Ucwords():- (capitalizing the first letter of each words of a string).

Ex;-<?php

```
$name " abitengg college ";
  Echo ucwords($name)
 ?>
```

(4) **Alternatives for regular expression function**:- in this we have to describe different types of functions. There are

(a) strtok() (e) strrpos() (i) substr_count()

(b) explodc() (f) str_replace() (j) substr_replace()

(c) implodc() (g) strstr()

(d) strops() (h) substr()

(a) **strtok()**:- this function parses the string based on a predefind list of characters.

Syntax:- string strtok(string str,string tokens)

Ex:-<?php

```
 $into= " abit: abit@gmail.com\siddavatam,kdp ";
$tokens= " :\ , ";
$tokenized= strtok($into,$tokens);
```

```
While( $tokenized)
  {
   Echo " elements = $tokenized<br>";
    $tokenized=strtok($tokens);
  }
 ?>
```

(b)explode():-this function devides the string str into an array of substrings,in this we have to mainly concentrated areas are size of ( ) and stip-tags( ) to determine the total number of words.

Ex: <? Php

$summary==<<<summery

Php is a server side scripting language.

Summary;

$words=size of(explode(' ',strip-tags($summary)));

Echo" total words in summary;$words";

?>

(c)implode( ):-we concatenate array elements to form a single delimited string using the implode( ) function. **Ex:** <? Php

$cites=array("kdp","antpr","tirupathi".);

Echo implode("\",$cities);

?>

o/pkdp\antpr\tirupathi

(d)**strops():** in this function finds the position of the first case_sensitiveoccurance of a substring in a string.

(e)**strrpos():-** in this function finds the last occurance of a string returning it's numerical position.

(f) **str_replace**():-this function case sensitively replaces all instance of a string with another.

Ex:-<?php

$gmail= "abit@gmail.com ";

$gmail=str_replace("@", "(is)", $gmail);

Echo "college mail is $gmail;

?>

(g)**strstr():-** this function returns the remainder of a string beginning with the first occurance of a predefined string.

Ex:-<?php

$gmail= " abit@gmail.com ";

Echo ltrim (strstr ($gmail, "@"), "@");

?>

(h)**substr():-** this function returns the part of a string located between a predefined string offset and length positions.

Ex:-<?php

$car= " 1994 ford ";

Echo substr($car,5);

?>

Ex2:- <?php

$car= " 1944 ford ";

Echo substr($car,0,4);

?>

(i)substr-count() :thisfunctionreturns the no.of times one string excuss another

Ex:<?php

$intu=array("php", "XAMPP");

$talk=<<<talk

PHP is a scriptin language and php is server side

Programming language.XAMPP is a web server

Talk:

Foreach($info as $it)

{

Echo "the word $it appears".substr_count($talk,$it). "time(s)
/>";

}

?>

(j)substr-replace():replace the portion of a string with another string

Ex:<?php

$name="Abitcollege";

Echo substr_replace($name, "engg",0,4);

?>

(5**)converting string to HTML form:-** in this we are using different types of converting function .there are

-> Converting newline characters to HTML break tags.

Ex:-<?php

$info=”aaaaaaaaaaaaaaaaaaaaaa
Bbbbbbbbbbbbbbbbbbbbb
ccccccccccccccccccccccc
ddddddddddddddddddd”; echo
n12br($into);
?>

Here we are not use
 statement.

->using special HTML characters for other purpose. In this we using htmlspecialchars() function.

->& ->&

-> “ ->&quoit;

-> ‘ ->'

->< ->&1t;

->> ->>

Ex:-<?php
$input= “<php is “scripting” language>”;
Echo html\specialchars($input);
?>

(6**) padding and stripping a string**:- php provides no. of functions there are

(a) ltrim()

(b) rtrim()

(c) trim()

(d) str_pad()

(a) **ltrim()**:- this function removes various characters from the beginning of a string including white space, horizontal tab(\t), newline(\n), carriage return(\v), null(\o).

String ltrim(string str [, string [, string charlist])

(b) **rtrim**():- this function removes various characters from the end of the string and except designated characters.

String rtrim(string str [, string charlist])

(c) **trim**():-both l trim and r trim

(d) **str_pad**():- this function pads a string with a specified number of characters. Ex:-<?php

Echo str_pad(“salad”,10). “is good.”;

?>**output**:- salad is good

(7) counting characters and words:- it's mainely used for to determine the total number of characters or words in a given string. Php provides two functions. there are count_chars() and str_word_count.

(5) arrays and functions:- array is a collection of heterogeneous(different elements) data types in php. Because php is a loosely typed language.

Ex:-<?php

$arr=array(10,20,30);

Print_r($str);

?>**output**:- [0]=10,[1]=20,[2]=30

Ex2:- <?php

$arr=array(100->10, 101->20, ->102=>30);

Print_r($arr);

?>**output**:-[100]=10, [101]=20, [102]=30

Ex3:- <?php

$arr=array(100=>10,20,30, 106=>30);

Print_r($arr);

?>**output**:- [100]=10,[101]=20,[102]=30,[106]=30

Ex:-<?php

$arr=array(100=>10, 'city'=> 'hyd', 105=>30, 50=>40,70);

Print_r($arr);

?>**output**:-[100]=40,[city]=hyd,[105]=30,[50]=40,[106]=70 **Array functions**:-

➔ **Count**:- it returns total no. of elements

Ex:-<?php

$arr=array(10,20,30);

Echo count($arr);

?>**output**:- 3

➔ **Sort** :- it returns the elements of an array in assending order.

Ex:-<?php

$arr=array(60,20,30);

Sort($arr);

Print_r($arr);

?>**output**:- 20,30,60

➔ **rsort :-** it returns the elements of an array in descending order.

Ex:-<?php

$arr=array(101,104,102);

rsort_r($arr); print_r($arr);

?> output:-104,102,101

➔ **asort:-** it returns the original keys with assending order.

Ex**:-**<?php

$arr=array(104=>40, 101=>20, 108=>50, 102=>80);

assort($arr); print_r($arr);

?> output:- 101=20,104=40,108=50,102=80 ➔

arsort:-it returns the original key values with descending order.

Ex:-<?php

$arr=array(104=>40, 101=>20, 108=>50,

102=>80); arsort($arr); Print_r($arr);

?>ouput:-[102]=80,[108]=50,[104]=40,[101=20

➔ **ksort**:-it returns the array in assending order with based on the"keys"**.**

Ex:-<?php

$arr=array(104=>40,101=>20,108=>50,102=>80);

Ksort($arr);

Print_r($arr);

?> output:-[101]=20,[102]=80,[104]=40,[108]=50

➔ **krsort**:- it returns the array in dessending order with based on "keys".

Ex:-<?php

$arr=array(104=>40,101=>20,108=>50,102=>80);

Krsort($arr);

Print_r($arr);

?> output:-[108]=50,[104]=40,[102]=80,[101]=20

➔ **array push():-** this function adds an elements into the end of an array and returns the total no. of elements in that array.

```
Ex:-<?php
    $arr=array(10,20,30);
    Echo array_push($arr,40);
Print_r($arr);
  ?>                    output:-
```

➔ **array_pop()**:- remove the last element & return the value of that element.

```
Ex:-<?php
    $arr=array(10,20,30);
Echo array_pop($arr);
Print_r($arr);
   ?>                    output:-
```

➔ **array_shift**():-it removes the first element of an array and returns the value of that element.

```
Ex:-<?php
    $arr=array(10,20,30);
     Echo array_shift($arr);
Print_r($arr);
     ?>                    output:-
```

➔ **array_unshift():-** add an element at the beginning of an array and return size of an array.

```
Ex:-<?php
    $arr=array(10,20,30);
    Echo array_unshift($arr);
Print_r($arr);
    ?>                    output:-
```

➔ **array_change_key_case():-** it converts all keys of an array into lower case.

```
Ex:-<?php
    $arr=array('ABC'=>10,20,30);
Print_r(array_change_key_case($arr));
    ?>                    output:-
```

➔ **array_chunk():-** splits an array into chunk of an array.

Ex:-<?php

$arr=array(10,20,30,40,50,60);

Print_r(array_chunk($arr,2));

?> output:-

➔ **array_combine():-** creats an array by using one array one array for keys and another for it's value.

Ex:-<?php

$arr=array('abc'=>10,20,30,40,50);

$arr1=array(100,200,300,400,500);

Print_r(array_combine($arr,$arr1));

?> output:-

➔ **array_keys:-** it returns new array with keys as value of another array.

Ex:-<?php

$arr=array('abc'=>10,20,30,40);

Print_r(array_keys($arr));

?> output:-

➔ **array_count_values():-** returns an array with no of occurance for each value. Ex:-<?php

$arr=array('ABC'=>10,20,30,40,50,10);

Printf_r(array_count_values($arr));

?>

➔ **array_values():=**return array with the values of an array

ex:- <?php

$arr=array('ABC'=10,20,30,40,50,60);

Printf_r(array_values($arr));

?>

➔ **array_flip():-**exchanges all keys with their associated values in array ex:-<?php

$arr=array('ABC'->10,20,30,40);

//$arr=array(10,200,400);

Printf_r(array-flip($arr));

?>

➔ **array_interest():-**compaves array values and returns the matches ex:-<?php

$arr=array(10,20,30,40);

$arr=array(100,200,300,400,10);

Printf_r(array_interest($arr)); //($arr,$arr1)

?>

➔ **array_interest_assoc():-**compaves array key and values and returns the matches ex:-<?php

$arr=array(10,20,30,40);

$arr1=array(100,200,300,400,0=>10);

Printf_r(array_interest_assoc($arr,$arr1));

?>

➔ **array_merge():-**merges one or more arrays into one array **ex:-**<?php

$arr=array('ABC'=>10,20,30,40);

$arr=array(100,200,300);

Printf_r(array-merge($arr,$arr1));

?>

➔ **array_product():-**returns the product of all array element values ex:-<?php

$arr=array('ABC'=>10,20,30,40);

Echo print_r(array_product($arr));

?>

➔ **array_sum():-**returns the sum of all elements of an array ex:-<?php

$arr=array(10,20,30,40);

Echo print_r(array_sum($arr));

?>

➔ **array_reverse():-**it revers the elements of an array ex:-<?php

$arr=array(10,20,30,40);

Print_r(array_reverse($arr));

?>

➔ **array_unique():-**removes the duplicate values and returns the values of an array ex:-<?php

$arr=('ABC'=>10,20,30,40);

```
Print_r(array_unique($arr));
    ?>
```

➔ **shuffle():-**shuffle the elements of an array ex:-

```
<?php
    $arr=array('ABC'=>10,20,30,40);
Shuffle($arr);
Print_r($arr);
    ?>
```

➔ **extract():-**divides the elements of an array as individual variables ex:-

```
<?php
    $arr=array('ABC'=>10,20,30,40);
Extract($arr);
    Echo $ABC;
    ?>
```

➔ **list():-**assign variables as it they were an array means that,we can assign the values of an array into variables

```
Ex:-<?php
List($x,$y,$z)=array(10,20,30);
        Echo $x;
        Echo $y;
        Echo $z;
```

UNIT 2-XML

XML stands for **E**xtensible **M**arkup **L**anguage. It is a text-based markup language derived from Standard Generalized Markup Language (SGML).

XML tags identify the data and are used to store and organize the data, rather than specifying how to display it like HTML tags, which are used to display the data. XML is not going to replace HTML in the near future, but it introduces new possibilities by adopting many successful features of HTML.

There are three important characteristics of XML that make it useful in a variety of systems and solutions:

- **XML is extensible:** XML allows you to create your own

self-descriptive tags, or language, that suits your application.

- **XML carries the data, does not present it:** XML allows you to store the data irrespective of how it will be presented.
- **XML is a public standard:** XML was developed by an organization called the World Wide Web Consortium (W3C) and is available as an open standard.

XML Usage

A short list of XML usage says it all:

- XML can work behind the scene to simplify the creation of HTML documents for large web sites.
- XML can be used to exchange the information between organizations and systems.
- XML can be used for offloading and reloading of databases.
- XML can be used to store and arrange the data, which can customize your data handling needs.
- XML can easily be merged with style sheets to create almost any desired output.
- Virtually, any type of data can be expressed as an XML document.

What is Markup?

XML is a markup language that defines set of rules for encoding documents in a format that is both human-readable and machine-readable. So what exactly is a markup language? Markup is information added to a document that enhances its meaning in certain ways, in that it identifies the parts and how they relate to each other. More specifically, a markup language is a set of symbols that can be placed in the text of a document to demarcate and label the parts of that document.

Following example shows how XML markup looks, when embedded in a piece of text:

```
<message>
<text>Hello, world!</text>
</message>
```

This snippet includes the markup symbols, or the tags such as <message>...</message> and <text>...</text>. The tags <message> and </message> mark the start and the end of the XML code fragment. The tags <text> and </text> surround the text Hello, world!.

Is XML a Programming Language?

A programming language consists of grammar rules and its own vocabulary which is used to create computer programs. These programs instructs computer to perform specific tasks. perform any computation or algorithms. It is usually stored in a simple text file and is processed by special software that is capable of interpreting XML.

Tags and Elements

An XML file is structured by several XML-elements, also called XML-nodes or XML- tags. XML-elements' names are enclosed by triangular brackets < > as shown below: <element>

Syntax Rules for Tags and Elements

Element Syntax: Each XML-element needs to be closed either with start or with end elements as shown below:

```
<element>....</element>
```

or in simple-cases, just this way:

```
<element/>
```

Nesting of elements: An XML-element can contain multiple XML-elements as its children, but the children elements must not overlap. i.e., an end tag of an element must have the same name as that of the most recent unmatched start tag.

Following example shows incorrect nested tags:

```
<?xml version="1.0"?>
<contact-info>
<company>IARE
```

<contact-info>
</company>

Following example shows correct nested tags:

<?xml version="1.0"?>
<contact-info>
<company>IARE</company>
<contact-info>

Let us learn about one of the most important part of XML, the XML tags. XML tags form the foundation of XML. They define the scope of an element in the XML. They can also be used to insert comments, declare settings required for parsing the environment and to insert special instructions.

We can broadly categorize XML tags as follows:

Start Tag

The beginning of every non-empty XML element is marked by a start-tag. An example of start-tag is:

<address>

End Tag

Every element that has a start tag should end with an end-tag. An example of end- tag is:

</address>

Note that the end tags include a solidus ("/") before the name of an element.

Empty Tag

The text that appears between start-tag and end-tag is called content. An element which has no content is termed as **empty**. An **empty** element can be represented in two ways as below:

(1) A start-tag immediately followed by an end-tag as shown below:

<hr></hr>

(2) A complete empty-element tag is as shown below:

<hr />

Empty-element tags may be used for any element which has no content.

XML Tags Rules

Following are the rules that need to be followed to use XML tags:

Rule 1

XML tags are case-sensitive. Following line of code is an example of wrong syntax </Address>, because of the case difference in two tags, which is treated as erroneous syntax in XML.

<address>This is wrong syntax</Address>

Following code shows a correct way, where we use the same case to name the start and the end tag.

<address>This is correct syntax</address>

Rule 2

XML tags must be closed in an appropriate order, i.e., an XML tag opened inside another element must be closed before the outer element is closed. For example: <outer_element>

<internal_element>

This tag is closed before the outer_element

</internal_element>

</outer_element>

XML Elements

XML elements can be defined as building blocks of an XML. Elements can behave as containers to hold text, elements, attributes, media objects or all of these.

Each XML document contains one or more elements, the scope of which are either delimited by start and end tags, or for empty elements, by an emptyelement tag.

Syntax

Following is the syntax to write an XML element:

<element-name attribute1 attribute2>

....content

</element-name>

where

- **element-name** is the name of the element. The name its case in the start and end tags must match.

- **attribute1, attribute2** are attributes of the element separated by white spaces. An attribute defines a property of the element. It associates a name with a value, which is a string of characters. An attribute is written as:

  name = "value"

The name is followed by an = sign and a string value inside double(" ") or single(' ') quotes.

Empty Element

An empty element (element with no content) has following syntax: <name attribute1 attribute2.../>

Example of an XML document using various XML element:

```
<?xml version="1.0"?>
<contact-info>
<address category="residence">
<name>Tanmay Patil</name>
<company>TutorialsPoint</company>
<phone>(011) 123-4567</phone>
<address/>
</contact-info>
```

XML Elements Rules

Following rules are required to be followed for XML elements:

- An element name can contain any alphanumeric characters. The only punctuation marks allowed in names are the hyphen (-), under-score (_) and period (.).
- Names are case sensitive. For example, Address, address, and ADDRESS are different names.
- Start and end tags of an element must be identical.
- An element, which is a container, can contain text or elements as seen in the above example.

Root element: An XML document can have only one root element. For example, following is not a correct XML document, because both the x and y elements occur at the top level without a root element:

<x>...</x>

<y>...</y>

The following example shows a correctly formed XML document: <root>

<x>...</x>

<y>...</y>

</root>

Case sensitivity: The names of XML-elements are case-sensitive. That means the name of the start and the end elements need to be exactly in the same case.

For example, **<contact-info>** is different from **<Contact-Info>**.

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

An XML DTD can be either specified inside the document, or it can be kept in a separate document and then liked separately.

Syntax

Basic syntax of a DTD is as follows: <!DOCTYPE

element DTD identifier

- **DTD identifier** is an identifier for the document type definition, which may be the path to a file on the system or URL to a file on the internet. If the DTD is pointing to external path, it is called **External Subset.**
- **The square brackets [ ]** enclose an optional list of entity declarations called Internal Subset.

Internal DTD

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

Syntax

The syntax of internal DTD is as shown:

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

Example

Following is a simple example of internal DTD:

```
<?xml version="1.0" encoding="UTF-8" standalone="yes" ?>
<!DOCTYPE address [
<!ELEMENT address (name,company,phone)>
<!ELEMENT name (#PCDATA)>
<!ELEMENT company (#PCDATA)>
<!ELEMENT phone (#PCDATA)>
]>
<address>
<name>Tanmay Patil</name>
<company>iare</company>
<phone>(011) 123-4567</phone>
  </address>
```

Let us go through the above code:

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

Start Declaration- Begin the XML declaration with following statement

DTD- Immediately after the XML header, the document type declaration follows, commonly referred to as the DOCTYPE:

<!DOCTYPE address [

The DOCTYPE declaration has an exclamation mark (!) at the start of the element name. The DOCTYPE informs the parser that a DTD is associated with this XML document. **DTD Body**- The DOCTYPE declaration is followed by body of the DTD, where you declare elements, attributes, entities, and notations:

<!ELEMENT address (name,company,phone)>

<!ELEMENT name (#PCDATA)>

<!ELEMENT company (#PCDATA)>

<!ELEMENT phone_no (#PCDATA)>

Several elements are declared here that make up the vocabulary of the <name> document. <!ELEMENT name (#PCDATA)> defines the element name to be of type "#PCDATA". Here #PCDATA means parse-able text data.

End Declaration - Finally, the declaration section of the DTD is closed using a closing bracket and a closing angle bracket (]>). This effectively ends the definition, and thereafter, the XML document follows immediately. **Rules**

- The document type declaration must appear at the start of the document (preceded only by the XML header) — it is not permitted anywhere else within the document.
- Similar to the DOCTYPE declaration, the element declarations must start with an exclamation mark.
- The Name in the document type declaration must match the element type of the root element.

External DTD

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

Syntax

Following is the syntax for external DTD:

<!DOCTYPE root-element SYSTEM "file-name">

where file-name is the file with .dtd extension.

Example

The following example shows external DTD usage:

<?xml version="1.0" encoding="UTF-8" standalone="no" ?>

<!DOCTYPE address SYSTEM "address.dtd">

<address>

<name>Tanmay Patil</name> <company>IARE</company>

<phone>(011) 123-4567</phone>

</address>

The content of the DTD file **address.dtd** are as shown:

<!ELEMENT address (name,company,phone)>

<!ELEMENT name (#PCDATA)>

<!ELEMENT company (#PCDATA)>

<!ELEMENT phone (#PCDATA)>

Types

You can refer to an external DTD by using either **system identifiers** or **public identifiers**.

SYSTEM IDENTIFIERS

A system identifier enables you to specify the location of an external file containing DTD declarations. Syntax is as follows:

<!DOCTYPE name SYSTEM "address.dtd" [...]>

As you can see, it contains keyword SYSTEM and a URI reference pointing to the location of the document.

PUBLIC IDENTIFIERS

Public identifiers provide a mechanism to locate DTD resources and are written as below:

<!DOCTYPE name PUBLIC "-//Beginning XML//DTD Address Example//EN">

As you can see, it begins with keyword PUBLIC, followed by a specialized identifier. Public identifiers are used to identify an entry in a catalog. Public identifiers can follow any format, however, a commonly used format is called Formal Public Identifiers, or FPIs.

XML Schema is commonly known as XML Schema Definition (XSD). It is used to describe and validate the structure and the content of XML data. XML schema defines the elements, attributes and data types. Schema element supports Namespaces. It is similar to a database schema that describes the data in a database.

Syntax

You need to declare a schema in your XML document as follows:

<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema">

Example

The following example shows how to use schema:

```
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema">
<xs:element name="contact">
<xs:complexType>
<xs:sequence>
<xs:element name="name" type="xs:string" />
<xs:element name="company" type="xs:string" />
<xs:element name="phone" type="xs:int" />
</xs:sequence>
</xs:complexType>
</xs:element>
</xs:schema>
```

The basic idea behind XML Schemas is that they describe the legitimate format that an XML document can take. Elements

As we saw in the chapter XML - Elements, elements are the building blocks of XML document. An element can be defined within an XSD as follows:

```
<xs:element name="x" type="y"/>
```

Definition Types

You can define XML schema elements in following ways:

```
<xs:element name="phone_number" type="xs:int" />
```

Simple Type - Simple type element is used only in the context of the text. Some of predefined simple types are: xs:integer, xs:boolean, xs:string, xs:date. For example:

Complex Type - A complex type is a container for other element definitions. This allows you to specify which child elements an element can contain and to provide some structure within your XML documents. For example:

```
<xs:element name="Address">
<xs:complexType>
```

```
<xs:sequence>
<xs:element name="name" type="xs:string" />
<xs:element name="company" type="xs:string" />
<xs:element name="phone" type="xs:int" />
</xs:sequence>
</xs:complexType>
</xs:element>
```

In the above example, Address element consists of child elements. This is a container for other <xs:element> definitions, that allows to build a simple hierarchy of elements in the XML document.

Global Types - With global type, you can define a single type in your document, which can be used by all other references. For example, suppose you want to generalize the person and company for different addresses of the company. In such case, you can define a general type as below:

```
<xs:element name="AddressType">
<xs:complexType>
<xs:sequence>
<xs:element name="name" type="xs:string" />
<xs:element name="company" type="xs:string" />
</xs:sequence>
</xs:complexType>
</xs:element>

<xs:element name="Address1">
<xs:complexType>
<xs:sequence>
<xs:element name="address" type="AddressType" />
<xs:element name="phone1" type="xs:int" />
</xs:sequence>
</xs:complexType>
</xs:element>
```

```
<xs:element name="Address2">
<xs:complexType>
<xs:sequence>
<xs:element name="address" type="AddressType" />
<xs:element name="phone2" type="xs:int" />
</xs:sequence>
</xs:complexType>
  </xs:element>
```

Instead of having to define the name and the company twice (once for Address1 and once for Address2), we now have a single definition. This makes maintenance simpler, i.e., if you decide to add "Postcode" elements to the address, you need to add them at just one place.

At ributes

Attributes in XSD provide extra information within an element. Attributes have name and type property as shown below:

```
<xs:attribute name="x" type="y"/>
```

Unit-3 Servlets

Java Servlets are programs that run on a Web or Application server and act as a middle layer between a request coming from a Web browser or other HTTP client and databases or applications on the HTTP server.

Using Servlets, you can collect input from users through web page forms, present records from a database or another source, and create web pages dynamically.

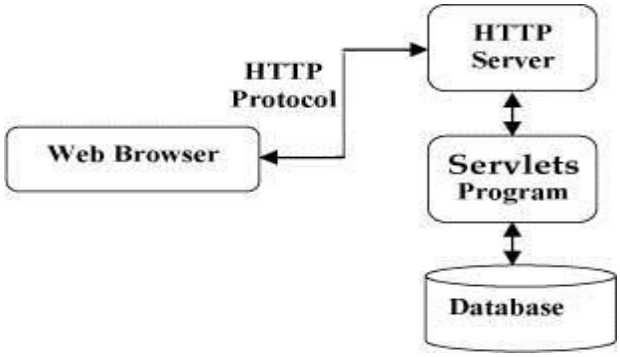
Java Servlets often serve the same purpose as programs implemented using the Common Gateway Interface (CGI). But Servlets offer several advantages in comparison with the CGI.

- Performance is significantly better.
- Servlets execute within the address space of a Web server. It is not necessary to create a separate process to handle each client request.
- Servlets are platform-independent because they are written in Java.

- Java security manager on the server enforces a set of restrictions to protect the resources on a server machine. So servlets are trusted.
- The full functionality of the Java class libraries is available to a servlet. It can communicate with applets, databases, or other software via the sockets and RMI mechanisms that you have seen already.

Servlets Architecture:

Following diagram shows the position of Servelts in a Web Application.



Servlets Tasks:

Servlets perform the following major tasks:

- Read the explicit data sent by the clients (browsers). This includes an HTML form on a Web page or it could also come from an applet or a custom HTTP client program.
- Read the implicit HTTP request data sent by the clients (browsers). This includes cookies, media types and compression schemes the browser understands, and so forth.
- Process the data and generate the results. This process may require talking to a database, executing an RMI or CORBA call, invoking a Web service, or computing the response directly.
- Send the explicit data (i.e., the document) to the clients (browsers). This document can be sent in a variety of formats, including text (HTML or XML), binary (GIF images), Excel, etc.
- Send the implicit HTTP response to the clients (browsers). This includes telling the browsers or other clients what type of document is being returned (e.g., HTML), setting cookies and caching parameters, and other such tasks.

Servlets Packages:

Java Servlets are Java classes run by a web server that has an interpreter that supports the Java Servlet specification.

Servlets can be created using the **javax.servlet** and **javax.servlet.http** packages, which are a standard part of the Java's enterprise edition, an expanded version of the Java class library that supports large-scale development projects.

These classes implement the Java Servlet and JSP specifications. At the time of writing this tutorial, the versions are Java Servlet 2.5 and JSP 2.1.

Java servlets have been created and compiled just like any other Java class. After you install the servlet packages and add them to your computer's Classpath, you can compile servlets with the JDK's Java compiler or any other current compiler.

Life cycle of servlet

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

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

JVM. Now let us discuss the life cycle methods in details.

The init() method :

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

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

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

The init method definition looks like this:

```
publicvoid init()throwsServletException{ //
Initialization code...
}
```

The service() method :

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

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

Here is the signature of this method:

```
publicvoid service(ServletRequest request, ServletResponse response)
throwsServletException,IOException{
}
```

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

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

The doGet() Method

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

```
publicvoid doGet(HttpServletRequest request, HttpServletResponse response)
throwsServletException,IOException{
// Servlet code
}
```

The doPost() Method

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

```
publicvoid doPost(HttpServletRequest request, HttpServletResponse response)
throwsServletException,IOException{
// Servlet code
}
```

The destroy() method :

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

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

```
publicvoid destroy(){ //
Finalization code...
}
```

Servlet Deployment:

By default, a servlet application is located at the path <Tomcat-installationdirectory>/webapps/ROOT and the class file would reside in <Tomcat-installationdirectory>/webapps/ROOT/WEB-INF/classes.

If you have a fully qualified class name of **com.myorg.MyServlet**, then this servlet class must be located in WEB- INF/classes/com/myorg/MyServlet.class.

For now, let us copy HelloWorld.class into <Tomcat-installation-directory>/webapps/ROOT/WEBINF/classes and create following entries in **web.xml** file located in <Tomcat-installationdirectory>/webapps/ROOT/WEB-INF/

```
<servlet>
<servlet-name>HelloWorld</servlet-name>
<servlet-class>HelloWorld</servlet-class>
</servlet>

<servlet-mapping>
<servlet-name>HelloWorld</servlet-name>
<url-pattern>/HelloWorld</url-pattern>
</servlet-mapping>
```

Above entries to be created inside <web-app>...</web-app> tags available in web.xml file. There could be various entries in this table already available, but never mind.

You are almost done, now let us start tomcat server using <Tomcat-installation-directory>\bin\startup.bat (on windows) or <Tomcat-installation-directory>/bin/startup.sh (on Linux/Solaris etc.) and finally type **http://localhost:8080/HelloWorld** in browser's address box. If everything goes fine, you would get following result:

UNIT 4-JSP

JSP

Overview

Jsp ava Server Pages (JSP) is a technology for developing web pages that support dynamic content which helps developers insert java code in HTML pages by making use of special JSP tags, most of which start with <% and end with %>.

A JavaServer Pages component is a type of Java servlet that is designed to fulfill the role of a user interface for a Java web application. Web developers write JSPs as text files that combine HTML or XHTML code, XML elements, and embedded JSP actions and commands. Using JSP, you can collect input from users t hrough web page forms, present records from a database or another source, and create web pages dynamically.

JSP tags can be used for a variety of purposes, such as retrieving information from a database or registering user preferences, accessing JavaBeans components, passing control between pages and sharing information between requests, pages etc.

Why Use JSP?

JavaServer Pages often serve the same purpose as programs implemented using the Common Gateway Interface (CGI). But JSP offer several advantages in comparison with the CGI.

- Performance is significantly better because JSP allows embedding Dynamic Elements in HTML Pages itself instead of having a separate CGI files.
- JSP are always compiled before it's processed by the server unlike CGI/Perl which requires the server to load an interpreter and the target script each time the page is requested.

JavaServer Pages are built on top of the Java Servlets API, so like Servlets, JSP also has access to all the powerful Enterprise Java APIs, including JDBC, JNDI, EJB, JAXP etc.

JSP pages can be used in combination with servlets that handle the business logic, the model supported by Java servlet template engines.

Finally, JSP is an integral part of J2EE, a complete platform for enterprise class applications. This mean

s that JSP can play a part in the simplest applications to the most complex and demanding.C

Advantages of JSP:

Following is the list of other advantages of using JSP over other technologies:

vs. Active Server Pages (ASP):The advantages of JSP are twofold. First, the dynamic part is written in Java, not Visual Basic or other MS specific language, so it is more powerful and easier to use. Second, it is portable to other operating systems and non-Microsoft Web servers.

vs. Pure Servlets:It is more convenient to write (and to modify!) regular HTML than to have plenty of println statements that generate the HTML.

vs. Server-Side Includes (SSI):SSI is really only intended for simple inclusions, not for "real" programs that use form data, make database connections, and the like.

vs. JavaScript:JavaScript can generate HTML dynamically on the client but can hardly interact with the web server to perform complex tasks like database access and image processing etc.

vs. Static HTML:Regular HTML, of course, cannot contain dynamic information

JSP Declarations:

A declaration declares one or more variables or methods that you can use in Java code later in the JSP file. You must declare the variable or method before you use it in the JSP file.

Following is the syntax of JSP Declarations:

<%!declaration;[declaration;]+...%>

You can write XML equivalent of the above syntax as follows:

<jsp:declaration> code fragment </jsp:declaration>

Following is the simple example for JSP Declarations:

<%!int i =0;%>

<%!int a,b,c;%>

<%!Circle a =newCircle(2.0);%>

<\%Represents static <% literal.%\> Represents static %> literal.\'A single quote in an attribute that uses single quotes.\"A double quote in an attribute that uses double quotes.

JSP Directives:

A JSP directive affects the overall structure of the servlet class. It usually has the following form:

<%@directive attribute="value"%>

There are three types of directive tag:

Directive

Description

<%@ page ... %>

Defines page

-

dependent attributes, such as scripting language, error page, and buffering

requirements.

<%@ include ... %>

Includes a file during the translation phase.

<%@ tag lib ... %>

Declares a tag library, containing custom actions, used in the page

The <jsp:forward> Action

The forward action terminates the action of the current page and forwards the request to another resource such as a static page, another JSP page, or a Java Servlet.

The simple syntax of this action is as follows:

<jsp:forward page="Relative URL"/>

Following is the list of required attributes associated with forward action:

Attribute

Description page

Should consist of a relative URL of another resource such as a static page, another

JSP page, or a Java Servlet Example:

Let us reuse following two files (a) date.jps and (b) main.jsp as follows:

Following is the content of date.jsp file:

```
<p>
Today's date:
<%=(new java.util.Date()).toLocaleString()%>
</p>
```

Here is the content of main.jsp file:

```
<html>
<head>
<title>
The include Action Example
</title>
</head>
<body>
<center>
<h2>
The include action Example
</h2>
<jsp:forward page="date.jsp"/>
</center>
</body>
</html>
```

Now let us keep all these files in root directory and try to access main.jsp. This would display r esult something like as below. Here it discarded content from main page and displayed content from forwarded page only.

Today's date: 12-Sep-2010 14:54:22

The <jsp:plugin> Action

The plugin action is used to insert Java components into a JSP page. It dete rmines the type of browser and inserts the <object> or <embed> tags as needed.If the needed plugin is not present, it downloads the plugin and then executes the Java component. The Java component can be either an Applet or a JavaBean.

The plugin action has several attributes that correspond to common HTML tags used to format Java components.

The <param> element can also be used to send parameters to the Applet or Bean.

Following is the typical syntax of using plugin action:

<jsp:plugin type="applet"code base="dirname"code="MyApplet.class" width="60",height="80">
<jsp:paramname="fontcolor" value="red"/>
<jsp:param name="background"value="black"/>
<jsp:fallback>
Unable to initialize Java Plugin
</jsp:fallback>
</jsp:plugin>

You can try this action using some applet if you are interested. A new element, the <fallback> element, can be used to specify an error string to be sent to the user in case the component fails.

The <jsp:element> Action

The <jsp:attribute> Action

The <jsp:body> Action

The <jsp:element>, lt;jsp:attribute> and <jsp:body> actions are used to define XML elements dynamically. The word dynamically is important, because it means that the XML elements can be generated at request time rather than statically at compile time.

Following is a simple example to define XML elements dynamically:

<%@page language ="java" contentType="text/html"%>
<html xmlns=http://www.w3c.org/1999/xhtml xmlns:jsp="http://java.sun.com/JSP/Page">
<head><title>
Genera te XML
Element
</title></head>
<body>
<jsp:element name="xmlElement">
<jsp:attribute name="xmlElementAttr">
Value for the attribute

</jsp:attribute>

<jsp:body>

Body for XML element

</jsp:body>

</jsp:element>

</body>

</html>

This would produce followin g

HTML code at run time:

<html xmlns=http://www.w3c.org/1999/xhtml xmlns:jsp="http://java.sun.com/JSP/Page">

<head><title>

Generate XML Element

</title></head>

<body>

<xmlElement xmlElementAttr= "Value for the attribute">

Body for XML element

</xmlElement>

</body>

</html>

The <jsp:text> Action

The <jsp:text> action can be used to write template text in JSP pages and documents. Following is the simple syntax for this action:

<jsp:text>

Template data

</jsp:text>

The body fo the template cannot contain other elements; it can only contain text and EL expressions (Note: EL expressions are explained in subsequent chapter). Note that in XML files, you cannot use expressions such as ${whatever > 0}, because the greater than signs are illegal. Instead, use the gt form, such as ${whatever gt 0} or an alternative is to embed the value in a CDATA section.

UNIT-V JAVASCRIPT

What is JavaScript?

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

It is an interpreted programming language with object-oriented capabilities.

JavaScript was first known as LiveScript, but Netscape changed itsnameto JavaScript, possibly because of the excitement being generated by Java. JavaScript made its first appearance in Netscape 2.0 in 1995 with the name LiveScript.

The general-purpose core of the language has been embedded in Netscape, Internet Explorer, and oth er web browsers.The

ECMA-262 Specification defined a standard version of the core JavaScript language.

- JavaScript is a lightweight, interpreted programming language.
- Designed for creating network-centric applications.
- Complementary to and integrated with Java.
- Complementary to and integrated with HTML.
- Open and cross-platform.

Client-Side JavaScript

Client-side JavaScript is the most common form of the language. The script should be included in or referenced by an HTML document for the code to be interpreted by the browser.It means that a web page need not be a static HTML, but can include programs that interact with the user, control the browser, and dynamically create HTML content.

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

in a form field.The JavaScript code is executed when the user submits the form, and only if all the entries are valid,they would be submitted to the Web Server.JavaScript can be used to trap userinitiated events such as button clicks, link navigation, and other actions that the user initiates explicitly or implicitly

OVERVIEW

Javascript

Advantages of JavaScript

The merits of using JavaScript are:

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

Limitations of JavaScript

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

- Client-side JavaScript does not allow the reading or writing of files. This has been kept for security reason.
- JavaScript cannot be used for networking applications because there is no such support available.
- JavaScript doesn't have any multithreading or multiprocessor capabilities.Once again, JavaScript is a lightweight, interpreted programming language that allows you to build interactivity into otherwise static HTML pages.

JavaScript Development

Tools One of major strengths of JavaScript is that it does not require expensive development tools. You can start with a simple text editor such as Notepad.Since it is an interpreted language inside the

context of a web browser, you don't even need to buy a compiler.To make our life simpler, various vendors have come up with very nice JavaScript editing tools.

Some of them are listed here:

Microsoft FrontPage:Microsoft has developed a popular HTML editor called FrontPage. FrontPage also provides web developers with a number of JavaScript tools to assist in the creation of interactive websites.

Macromedia Dreamweaver MX:Macromedia Dreamweaver MX is a very popular HTML and JavaScript editor in the professional web development crowd. It provides several handy prebuilt JavaScript

Javascript components, integrates well with databases, and conforms to new standards such as XHTML and XML.

Macromedia HomeSite 5:HomeSite 5 is a well-liked HTML and JavaScript editor from Macromedia that can be used to manage personal websites effectively.

Where is JavaScript Today?

The ECMAScript Edition 5 standard will be the first update to be released in over four years. JavaScript 2.0 conforms to Edition 5 of the ECMAScript standard, and the difference between the two is extremely minor.The specification for JavaScript 2.0 can be found on the following site: http://www.ecmascript.org/Today, Netscape's JavaScript and Microsoft's JScript conform to the ECMAScript standard, although both thelanguages still supportthe features that are not a part of the standard.

JavaScript Variables

Like many other programming languages, JavaScript has variables. Variables can be thought of as named containers. You can place data into these containers and then refer to the data simply by naming the container.Before you use a variable in a JavaScript program, you must declare it. Variables are declared with the var keyword as follows.

```
<script type="text/javascript">
<!--var money; var name;//-->
</script>
```

You can also declare multiple variables with the same var keyword as follows:

```
<script type="text/javascript">
<!--var money, name;//-->
</script>
```

Storing a value in a variable is called variable initialization. You can do variable initialization at the time of variable creation or at a later point in time when you need that variable.

For instance, you might create a variable named money and assign the value 2000.50 to it later. For another variable,you can assign a value at the time of initialization as follows. <script type="text/javascript">

```
<!--var name = "Ali"; var money; money = 2000.50;//-->
</script>
```

Note:Use the var keyword only for declaration or initialization, once for the life of any variable name in a document. You should not re-declare same variable twice.JavaScript isuntypedlanguage. This means that a JavaScript variable can hold a value of any data type. Unlike many other languages, you don't have to tell JavaScript during variable declaration what type of value the variable will hold. The value type of a variable can change during the execution of a program and JavaScript takes care of it automatically.

JavaScript Variable ScopeThe scope of a variable is the region of your program in which it is defined. JavaScript variables have only two scopes.

- Global Variables:A global variable has global scope which means it can be defined Any where in your JavaScript code.
- Local Variables:A local variable will be visible only within a function where it is defined. Function parameters are always local to that function.Javascript Within the body of a function, a local variable takes precedence over a global variable with the same name. If you declare a local variable or function parameter with the same name as a global variable, you effectively hide the global variable .

Take a look into the following example.

```
<script type="text/javascript">
<!--var myVar = "global"; // Declare a global variable
function checkscope( ) { var myVar = "local"; //
Declare a local variable document.write(myVar);
}
```

//-->

</script>

It will produce the following result:

Local

JavaScript Variable Names While naming your variables in JavaScript, keep the following rules in mind.

You should not use any of the JavaScript reserved keywords as a variable name. These keywords are mentioned in the next section. For example, break or Boolean variable names are not valid.

JavaScript variable names should not start with a numeral (0-9). They must begin with a letter or an underscore character. For example,123test is an invalid variable name but_123test is a valid one.

JavaScript variable names are case-sensitive. For example, Name and nameare two different variables. Javascript

What is an Event?

JavaScript's interaction with HTML is handled through events that occur when the user or the browser manipulates a page.When the page loads, it is called an event. When the user clicks a button, that click too is an event.

Other

Examples include events like pressing any key, closing a window, resizing a window,etc.Developers can use these events to execute JavaScript coded responses, which cause buttons to close windows, messages to be displayed to users, data to be validated, and virtually any other type of response imaginable.

Events are a part of the Document Object Model (DOM) Level 3 and every HTML element containsa set of events which can trigger JavaScript Code.

. Here we will see a few examples to understand the relation between

Event and JavaScript. onclick Event TypeThis is the most frequently used event type which occurs when a user clicks the left buttonof his mouse. You can put your validation, warning etc.,against this event type.

Example

Try the following example.

<html>

<head>

```
<script type="text
/javascript"> <!-- function
sayHello() { document.write
("Hello World")
}
//
-->
```