Code No: 126EP

R13

JAWAHARLAL NEHRU TECHNOLOGICAL UNIVERSITY HYDERABAD

B. Tech III Year II Semester Examinations, May - 2017

WEB TECHNOLOGIES

(Common to CSE, IT)

Time: 3 hours **Max. Marks: 75**

Note: This question paper contains two parts A and B.
Part A is compulsory which carries 25 marks. Answer all questions in Part A. Part B consists of 5 Units. Answer any one full question from each unit. Each question carries 10 marks and may have a, b, c as sub questions.

PART - A

(25 Marks)

1.a) What is PHP? What are the common uses of PHP? [2]
b) How to declare a string in PHP? List various string functions in PHP. [3]
c) State rules to define tags in XML. [2]
d) What is DTD? Distinguish between internal DTD and external DTD. [3]
e) How is a Servlet different from an Applet? [2]
f) Write about servlet API. [3]
g) Provide an example for JSP expression. [2]
h) How are cookies used for session tracking in JSP? [3]
i) What is the scope of variables in JavaScript? [2]
j) What is an 'event'? How are events handled in JavaScript? [3]

PART - B

(50 Marks)

2.a) What are various operators supported by PHP. [3+7]
b) Explain about the control structures in PHP with illustrations.

OR

3.a) How to list the directories in PHP? [3+7]
b) Explain about various file operations on text files in PHP.

4.a) What is DOM? [2+8]
b) Compare and contrast DOM parser with SAX Parser.

OR

5.a) List any *two* XML tags with their attributes and values.
b) Collect the student's details such as register number, name, subject and marks using forms and generate a DTD for this XML document. Display the collected information in the descending order of marks. Write XML source code for the above. [4+6]

6.a) What is JDBC? What are various drivers of JDBC? [4+6]
b) Explain about the database connectivity using JDBC.

OR

7.a) Develop a servlet that handles an HTTP POST request. [5+5]
b) Discuss about Session tracking in Servlets with a suitable example.

8.a) Describe the anatomy of a JSP page. [7+3]

b) Write a in brief about JSP Declarations.

OR

9. Explain in detail of how to use Java Beans in JSP pages with suitable example. [10]

10.a) What is the need of scripting languages in Web Technologies?

b) Write a program in JavaScript to convert temperature from Celsius to Fahrenheit and vice versa or Height from centimeters to inches and vice-versa. [2+8]

OR

11. Write about the following with reference to Java Script with an example:

a) Functions

b) Form Validation [5+5]

WEB TECHNOLOGIES – May – 2017

PART – A

1. a. What is PHP ? What are the common uses of PHP

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. It is integrated with a number of popular databases, including MySQL, PostgreSQL, Oracle, Sybase, Informix, and Microsoft SQL Server

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, and 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.

b. How to declare string in PHP? List various string functions in PHP

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:

“ ace clg” or ‘ace clg’

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

PHP 5 String Functions

The PHP string functions are part of the PHP core. No installation is required to use these functions.

Function	Description
chr()	Returns a character from a specified ASCII value
count_chars()	Returns information about characters used in a string
crypt()	One-way string hashing
echo()	Outputs one or more strings
fprintf()	Writes a formatted string to a specified output stream
join()	Alias of implode()
localeconv()	Returns locale numeric and monetary formatting information
ltrim()	Removes whitespace or other characters from the left side of a string
parse_str()	Parses a query string into variables
print()	Outputs one or more strings
printf()	Outputs a formatted string
rtrim()	Removes whitespace or other characters from the right side of a string
setlocale()	Sets locale information
sprintf()	Writes a formatted string to a variable
sscanf()	Parses input from a string according to a format
str_split()	Splits a string into an array
str_word_count()	Count the number of words in a string
strcasecmp()	Compares two strings (case-insensitive)
strchr()	Finds the first occurrence of a string inside another string (alias of strstr())
strcmp()	Compares two strings (case-sensitive)
stristr()	Finds the first occurrence of a string inside another string (case-insensitive)
strlen()	Returns the length of a string
strnatcasecmp()	Compares two strings using a "natural order" algorithm (case-insensitive)

strnatcmp()	Compares two strings using a "natural order" algorithm (case-sensitive)
strpos()	Returns the position of the first occurrence of a string inside another string (case-sensitive)
strrchr()	Finds the last occurrence of a string inside another string
strrev()	Reverses a string
strripos()	Finds the position of the last occurrence of a string inside another string (case-insensitive)
strrpos()	Finds the position of the last occurrence of a string inside another string (case-sensitive)
strspn()	Returns the number of characters found in a string that contains only characters from a specified charlist
strstr()	Finds the first occurrence of a string inside another string (case-sensitive)
strtok()	Splits a string into smaller strings
strtolower()	Converts a string to lowercase letters
strtoupper()	Converts a string to uppercase letters
strtr()	Translates certain characters in a string
substr()	Returns a part of a string
trim()	Removes whitespace or other characters from both sides of a string

c. State rules to define tags in XML

- All XML elements must have a closing tag
- XML tags are case sensitive
- XML Elements Must be Properly Nested
- XML Documents Must Have a Root Element
- Always Quote the XML Attribute Values
- With XML, White Space is Preserved
- Comments in XML <!-- This is a comment -->
- XML Elements have Relationships
- Elements in a xml document are related as parents and children.

d. What is DTD? Distinguish between internal DTD and external DTD

A Document Type Definition (DTD) defines the legal building blocks of an XML document. It defines the document structure with a list of legal elements and attributes. A DTD can be defined inside a XML document, or a external reference can be declared.

Using DTD we can declare and define:

I. Elements
II. Attributes
III. Entities
IV. Notations

Internal DTD

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

<!DOCTYPE root-element [element-declarations]>

External DTD

If the DTD is defined in an external file, it should be wrapped in a DOCTYPE definition with the following syntax:

<!DOCTYPE root-element SYSTEM "filename">

e. How is Servlet different from an Applet

Applet and servlet are the small Java programs or applications. But, both get processed in a different environment. The basic difference between an applet and a servlet is that an **applet** is executed on the client-side whereas, a **servlet** is executed on the server-side. Both of them differ in many contexts,

f. Write about Servlet API

Before creating the first servlet, you need to understand the Servlet API and Tomcat Servlet container. The Servlet API provides interfaces and classes that are required to built servlets. The javax.servlet and javax.servlet.http packages represent interfaces and classes for servlet api. The **javax.servlet** package contains many interfaces and classes that are used by the servlet or web container. These are not specific to any protocol. The **javax.servlet.http** package contains interfaces and classes that are responsible for http requests only.

g. Provide an example of JSP Expression

JSP expression tag

The code placed within JSP expression tag is *written to the output stream of the response* . So you need not write

out.print() to write data. It is mainly used to print the values of variable or method.

Syntax of JSP expression tag

<%= statement %>

Example of JSP expression tag
In this example of jsp expression tag, we are simply displaying a welcome message.

```
<html>
<body>
< %= "welcome to jsp" % >
</body>
</html>
```

Note: Do not end your statement with semicolon in case of expression tag.
Example of JSP expression tag that prints current time
index.jsp

```
<html>
<body>
Current Time: < %= java.util.Calendar.getInstance().getTime() % >
</body>
</html>
```

h. How are Cookies used for session tracking in JSP

HTTP is a "stateless" protocol which means each time a client retrieves a Web page, the client opens a new connection to the Web server and the server does not keep any record of previous client request.Session tracking is a mechanism that is used to maintain state about a series of requests from the same user(requests originating from the same browser) across some period of time. A session id is a unique token number assigned to a specific user for the duration of that user's session.

Cookies :

Cookies mostly used for session tracking. Cookie is a key value pair of information, sent by the server to the browser. This should be saved by the browser in its space in the client computer. Whenever the browser sends a request to that server it sends the cookie along with it. Then the server can identify the client using the cookie.

This is not an effective way because many time browser does not support a cookie or users can opt to disable cookies using their browser preferences. In such case, the browser will not save the cookie at client computer and session tracking fails.

i. What is the scope of variable in Java Script

To work with JavaScript efficiently, one of the first things you need to understand is the concept of variable scope. The scope of a variable is controlled by the location of the variable declaration, and defines the part of the program where a particular variable is accessible.

Scoping rules vary from language to language. JavaScript has two scopes – *global* and *local*. Any variable declared outside of a function belongs to the global scope, and is therefore accessible from anywhere in your code. Each function has its own scope, and any variable declared within that function is only accessible from that function and any nested functions. Because local scope in JavaScript is created by functions, it's also called function scope. When we put a function inside another function, then we create nested scope.

```
var test = "I'm global";
function testScope() {
  var test = "I'm local";
  console.log (test);
}
testScope();           // output: I'm local
console.log(test);     // output: I'm global
```

j. What is an event? How are event handlers in Javascript

An ***event*** is defined as "something that takes place" and that is exactly what it means in web programming as well.
An ***event handler*** is JavaScript code that is designed to run each time a particular event occurs.

Table: JavaScript Events

Event	Handler	Description
blur	onBlur	The input focus is moved from the object
change	onChange	The value of a field in a form has been changed by the user entering or deleting data
click	onClick	The mouse is clicked over an element of a page
dblclick	onDblClick	A form element or link is clicked twice in rapid succession
dragdrop	onDragDrop	A system file is dragged with a mouse and dropped onto the browser
focus	onFocus	Input focus is given to an element. The reverse of blur
keydown	onKeyDown	A key is pressed but not released
keypress	onKeyPress	A key is pressed
keyup	onKeyUp	A pressed key is released
load	onLoad	The page is loaded by the browser
mousedown	onMouseDown	A mouse button is pressed
mousemove	onMouseMove	The mouse is moved
mouseout	onMouseOut	The mouse pointer moves off an element
mouseover	onMouseOver	The mouse pointer moves over an element
mouseup	onMouseUp	The mouse button is released
move	onMove	A window is moved,

PART – B

2 a. What are the various operators supported by PHP

This section lists the different operators used in PHP.

Arithmetic Operators

Operator	Description	Example	Result
+	Addition	x=2 x+2	4
-	Subtraction	x=2 5-x	3
*	Multiplication	x=4 x*5	20
/	Division	15/5 5/2	3 2.5
%	Modulus (division remainder)	5%2 10%8 10%2	1 2 0
++	Increment	x=5 x++	x=6
--	Decrement	x=5 x--	x=4

Assignment Operators

Operator	Example	Is The Same As
=	x=y	x=y
+=	x+=y	x=x+y
-=	x-=y	x=x-y
*=	x*=y	x=x*y
/=	x/=y	x=x/y
.=	x.=y	x=x.y
%=	x%=y	x=x%y

Comparison Operators

Operator	Description	Example
==	is equal to	5==8 returns false
!=	is not equal	5!=8 returns true
<>	is not equal	5<>8 returns true
>	is greater than	5>8 returns false
<	is less than	5<8 returns true

>=	is greater than or equal to	5>=8 returns false
<=	is less than or equal to	5<=8 returns true

Logical Operators

Operator	Description	Example
&&	and	x=6 y=3 (x < 10 && y > 1) returns true
||	or	x=6 y=3 (x==5 || y==5) returns false
!	not	x=6 y=3 !(x==y) returns true

Conditional operators:

Operator	Description	Example
? :	Conditional Expression	If Condition is true ? Then value X : Otherwise value Y

2 b. Explain about the control structures in PHP with illustrations

PHP Control Structures

Any PHP script is built out of a series of statements. A statement can be an assignment, a function call, a loop, a *conditional statement* of even a statement that does nothing (an empty statement). Statements usually end with a semicolon. In addition, statements can be grouped into a statement-group by encapsulating a group of statements with curly braces. A statement-group is a statement by itself as well.

1). If Statement
2). Else Statement
3). Elseif Statement
4). Switch Statement

If Statement

An ***if statement*** is a way of controlling the execution of a statement that follows it (that is, a single statement or a block of code inside braces). The if statement evaluates an expression between parentheses. If this expression results in a true value, the statement is executed. Otherwise, the statement is skipped entirely. This enables scripts to make decisions based on any number of factors:

Syntax:-

```
if ( expression ) {
// code to execute if the expression evaluates to true }
```

PHP Example:- The following code would display a is bigger than b if *$a* is bigger than *$b*:

```
<?php
if ($a > $b)
  echo "a is bigger than b"; ?>
```

Else Statement

When working with the if statement, you will often want to define an alternative block of code that should be executed if the expression you are testing evaluates to false. You can do this by adding else to the if statement followed by a further block of code:

Syntax:-

```
if ( expression ) {
// code to execute if the expression evaluates to true
} else {
// code to execute in all other cases
}
```

PHP Example:- The following code would display a is bigger than b if *$a* is bigger than *$b*, and a is NOT bigger than b otherwise:

```
<?php
if ($a > $b) {
  echo "a is bigger than b";
} else {
   echo "a is NOT bigger than b";
}
?>
```

The else statement is only executed if the if expression evaluated to **FALSE**.

Go Top

Elseif Statement

As its name suggests, is a combination of if and else. Like else, it extends an if statement to execute a different statement in case the original if expression evaluates to **FALSE**. However, unlike else, it will execute that alternative expression only if the elseif conditional expression evaluates to **TRUE**.

Syntax:-

```
if ( expression ) {
// code to execute if the expression evaluates to true
} elseif (another expression) {
// code to execute if the previous expression failed
// and this one evaluates to true
} else { // code to execute in all other cases
}
```

PHP Example:- The following code would display a is bigger than b, a equal to b or a is smaller than b:

```
<?php
if ($a > $b) {
   echo "a is bigger than b";
} elseif ($a == $b) {
   echo "a is equal to b";
} else {
   echo "a is smaller than b";
}
?>
```

Go Top

Switch Statement

The *switch statement* is similar to a series of *IF statements* on the same expression. In many occasions, you may want to compare the same *variable* (or expression) with many different values, and execute a different piece of code depending on which value it equals to. This is exactly what the switch statement is for.

Tips:-

- That unlike some other languages, the continue statement applies to switch and acts similar to *break*. If you have a switch inside a loop and wish to *continue* to the next iteration of the outer loop, use continue 2.

Syntax:-

```
switch ( expression ) {
case result1:
// execute this if expression results in result1
break;
case result2:
// execute this if expression results in result2
break;
default:
// execute this if no break statement
// has been encountered hitherto
}
```

PHP Example:-

```
<?php
switch ($i) {
   case 0:
     echo "i equals 0";
   case 1:
      echo "i equals 1";
   case 2:
      echo "i equals 2";
```

```
}
?>
```

PHP Loops

1). While Statement
2). Do.. While Statement
3). For Statement
4). Foreach Statement
5). Break Statement
6). Continue Statement

```
while ( expression ) {
// do something }

do {
// code to be executed
} while ( expression );

for ( initialization expr1; test expr2; modification expr3 )
{ // code to be executed }

foreach (array_expression as $value) statement
foreach (array_expression as $key => $value) statement
```

Break Statemen

```
<?php
echo "<p><b>Example of using the Break statement:</b></p>";

for ($i=0; $i<=10; $i++) {
  if ($i==3){break;}
  echo "The number is ".$i;
  echo "<br />";
}
?>
```

Continue Statement

```
<?php
echo "<p><b>Example of using the Continue statement:</b><p>";

for ($i=0; $i<=10; $i++) {
  if (i==3){continue;}
  echo "The number is ".$i;
  echo "<br />";
```

```
}
?>
```

3 a. How to List the Directories in PHP

A common request is to be able to produce a list of some or all of the files in a directory - similar to the default index page provided by most web servers, but with more control over the content and formatting.
Another goal might be to take some action on the files using PHP. In either case the first step is to query the file system to return a list of directories and files.
The functions presented below provide the means to extract the file names and other properties from a single directory, or to explore subdirectories recursively.
PHP 5 introduces the scandir function which will "List files and directories inside the specified path" but won't recurse or provide additional information about the listed files.

PHP scandir() Function

Example

List files and directories inside the images directory:

```
<?php
$dir = "/images/";
// Sort in ascending order - this is default
$a = scandir($dir);
// Sort in descending order
$b = scandir($dir,1);
print_r($a);
print_r($b);
?>
```

Result:

```
Array
(
[0] => .
[1] => ..
[2] => cat.gif
[3] => dog.gif
[4] => horse.gif
[5] => myimages
)
Array
(
[0] => myimages
[1] => horse.gif
```

[2] => dog.gif
[3] => cat.gif
[4] => ..
[5] => .
)

3 b. Explain about various file operations on text files in PHP

PHP File Operations

Summary: in this tutorial, you will learn some common used PHP file operations such as copying, renaming and deleting files in PHP.

PHP copying a file

To copy a file, you use the copy() function. First, you need to determine which file to copy by passing the file path to the first parameter of the copy() function. Second, you need to specify the file path to copy the file to. The copy() function returns true if the file was copied successfully, otherwise it returns false.

The following example illustrates how to copy the test.txt file to test2.txt file, which locates in the same directory as the script.

```
1 <?php
2 $fn = './test.txt';
3 $newfn = './test2.txt';
4
5 if(copy($fn,$newfn))
6 echo 'The file was copied successfully';
7 else
8 echo 'An error occurred during copying the file';
```

PHP renaming a file

To rename a file, you use the rename() function. This function also allows you to move a file to a different directory. For example, to rename the test.txt file to test.bak file, you use the following code:

```
1 <?php
2 $fn = './test.txt';
3 $newfn = './test.bak';
4
5 if(rename($fn,$newfn)){
6 echo sprintf("%s was renamed to %s",$fn,$newfn);
7 }else{
8 echo 'An error occurred during renaming the file';
9 }
```

Notice that the rename() function returns true if the file is renamed successfully, otherwise it returns false.

The following example shows you how to use the rename() function to move the test.bak to the backup directory:

```
1 <?php
2 $fn = './test.bak';
3 $newfn = './backup/test.bak';
4
5 if(rename($fn,$newfn)){
6 echo sprintf("%s was moved to %s",$fn,$newfn);
7 }else{
8 echo 'An error occurred during moving the file';
9 }
```

PHP deleting a file

To delete a file, you use the unlink() function. You need to pass the file name that you want to delete to the unlink() function. The function returns true on success or false on failure.

```
1 <?php
2 $fn = './backup/test.bak';
3 if(unlink($fn)){
4 echo sprintf("The file %s deleted successfully",$fn);
5 }else{
6 echo sprintf("An error occurred deleting the file %s",$fn);
7 }
```

PHP checking file permissions

PHP gives you three handy functions that allow you to check file permissions:

- *is_readable()* returns *true* if you are allowed to read the file, otherwise returns *false*.
- *is_writable()* returns *true* if you are allowed to write the file, otherwise returns *false*.
- *is_executable()* returns *true* if you are allowed to execute the file, otherwise returns *false*.

Let's take a look at the following example:

```
1  <?php
2  $fn = './test.text';
3
4  $msg = is_readable($fn) ? $msg = 'File is readable'
5  : $msg = 'File is not readable';
6  echo $msg . '<br/>';
7
8  $msg = is_writable($fn) ? $msg = 'File is writable'
9  : $msg = 'File is not writable';
10
11 echo $msg . '<br/>';
12
13 $msg = is_executable($fn) ? $msg = 'File is executable'
14   : $msg = 'File is not executable';
15
16 echo $msg . '<br/>';
```

Besides those functions, PHP also provides the *fileperms()* function that returns an integer, which represents the permissions that are set on a particular file.

```
1 <?php
2 $fn = './test.txt';
3 $fp = fileperms($fn);
4
5 echo substr(sprintf('%o', $fp), -4); //0666
```

PHP changing file permissions

To change the file permissions, or mode, you use *chmod()* function. First, you need to pass the name of the file that you want to set permission. Second, you need to specify the desired permission. The *chmod()* function returns *true* if the permission was set successfully otherwise it returns *false*.

A file permission is represented by an octal number that contains three digits:

- The first digit specifies what the owner of the file can do with file.
- The second digit specifies what the owner group of the file can do with the file.
- The third digit specifies what everyone can do with the file.

The following table illustrates the value of each digit that represents the access permission for particular users (user, user group or everyone) :

Value	Permission
0	cannot read, write or execute
1	can only execute
2	can only write
3	can write and execute
4	can only read
5	can read and execute
6	can read and write
7	can read, write and execute

For example, to set a permission that only creator or owner can read and write a file, everyone else only can read the file, we use the following code:

```
1 <?php
2 $fn = './test.txt';
3 chmod($fn, 0644);
```

4 a. What is DOM

The **DOM** defines a standard for accessing documents: "The W3C Document Object Model (**DOM**) is a platform and language-neutral interface that allows programs and scripts to dynamically access and update the content, structure, and style of a document.

The Document Object Model (DOM) is a programming API for HTML and XML documents. It defines the logical structure of documents and the way a document is accessed and manipulated. In the DOM specification, the term "document" is used in the broad sense - increasingly, XML is being used as a way of representing many different kinds of information that may be stored in diverse systems, and much of this would traditionally be seen as data rather than as documents.

Nevertheless, XML presents this data as documents, and the DOM may be used to manage this data.

4 b. Compare and contrast DOM parser and SAX parser

SAX	DOM
Both SAX and DOM are used to parse the XML document. Both has advantages and disadvantages and can be used in our programming depending on the situation.	
Parses node by node	Stores the entire XML document into memory before processing
Doesn't store the XML in memory	Occupies more memory
We cant insert or delete a node	We can insert or delete nodes
Top to bottom traversing	Traverse in any direction.
SAX is an event based parser	DOM is a tree model parser
SAX is a Simple API for XML	Document Object Model (DOM) API
import javax.xml.parsers.*; import org.xml.sax.*; import org.xml.sax.helpers.*;	import javax.xml.parsers.*; import org.w3c.dom.*;
doesn't preserve comments	preserves comments
SAX generally runs a little faster than DOM	SAX generally runs a little faster than DOM
If we need to find a node and doesn't need to insert or delete we can go with SAX itself otherwise DOM provided we have more memory.	

5 a. List any two XML Tags with their attributes and values

XML elements can have attributes, just like HTML.
Attributes are designed to contain data related to a specific element.
XML Attributes Must be Quoted
Attribute values must always be quoted. Either single or double quotes can be used.
For a person's gender, the <person> element can be written like this:
<person gender="female">
or like this:
<person gender='female'>
If the attribute value itself contains double quotes you can use single quotes, like in this example:

<gangster name='George "Shotgun" Ziegler'>

or you can use character entities:
<gangster name="George "Shotgun" Ziegler">

A date attribute is used in the first example:

```
<note date="2008-01-10">
  <to>Tove</to>
  <from>Jani</from>
</note>
```

XML Attributes for Metadata

Sometimes ID references are assigned to elements. These IDs can be used to identify XML elements in much the same way as the id attribute in HTML. This example demonstrates this:

```
<messages>
  <note id="501">
    <to>Tove</to>
    <from>Jani</from>
    <heading>Reminder</heading>
    <body>Don't forget me this weekend!</body>
  </note>
  <note id="502">
    <to>Jani</to>
    <from>Tove</from>
    <heading>Re: Reminder</heading>
    <body>I will not</body>
  </note>
</messages>
```

Attributes are part of XML elements. An element can have multiple unique attributes. Attribute gives more information about XML elements. To be more precise, they define properties of elements. An XML attribute is always a name-value pair.

Syntax

An XML attribute has the following syntax −

```
<element-name attribute1 attribute2 >

....content..

< /element-name>
```

where *attribute1* and *attribute2* has the following form −

```
name = "value"
```

value has to be in double (" ") or single (' ') quotes. Here, *attribute1* and *attribute2* are unique attribute labels.

Attributes are used to add a unique label to an element, place the label in a category, add a Boolean flag, or otherwise associate it with some string of data. Following example demonstrates the use of attributes −

```
<?xml version = "1.0" encoding = "UTF-8"?>
<!DOCTYPE garden [
   <!ELEMENT garden (plants)*>
   <!ELEMENT plants (#PCDATA)>
   <!ATTLIST plants category CDATA #REQUIRED>
]>

<garden>
   <plants category = "flowers" />
   <plants category = "shrubs">
   </plants>
</garden>
```

Attributes are used to distinguish among elements of the same name, when you do not want to create a new element for every situation. Hence, the use of an attribute can add a little more detail in differentiating two or more similar elements.

In the above example, we have categorized the plants by including attribute category and assigning different values to each of the elements. Hence, we have two categories of *plants*, one *flowers* and other *color*. Thus, we have two plant elements with different attributes.

You can also observe that we have declared this attribute at the beginning of XML.

Attribute Types

Following table lists the type of attributes −

Attribute Type	Description
StringType	It takes any literal string as a value. CDATA is a StringType. CDATA is character data. This means, any string of non-markup characters is a legal part of the attribute.
TokenizedType	This is a more constrained type. The validity constraints noted in the grammar are applied after the attribute value is normalized. The TokenizedType attributes are given as − • **ID** − It is used to specify the element as unique. • **IDREF** − It is used to reference an ID that has been named for another element. • **IDREFS** − It is used to reference all IDs of an element. • **ENTITY** − It indicates that the attribute will represent an external entity in the document. • **ENTITIES** − It indicates that the attribute will represent external entities in the document. • **NMTOKEN** − It is similar to CDATA with restrictions on what data can be part of the attribute. • **NMTOKENS** − It is similar to CDATA with restrictions on what data can be part of the attribute.
EnumeratedType	This has a list of predefined values in its declaration. out of which, it must assign one value. There are two types of enumerated attribute − • **NotationType** − It declares that an element will be referenced to a NOTATION declared somewhere else in the XML document. • **Enumeration** − Enumeration allows you to define a specific list of values that the attribute value must match.

Element Attribute Rules

Following are the rules that need to be followed for attributes −

- An attribute name must not appear more than once in the same start-tag or empty-element tag.
- An attribute must be declared in the Document Type Definition (DTD) using an Attribute-List Declaration.
- Attribute values must not contain direct or indirect entity references to external entities.
- The replacement text of any entity referred to directly or indirectly in an attribute value must not contain a less than sign (<)

5 b. Collect the student details such as register number, name, subject and marks using forms and generate a DTD for this XML document. Display the collected information in the descending order of marks. Write XML source code for above.

6 a. What is JDBC ? What are the various drivers of JSP

JDBC stands for "Java DataBase Connectivity". It is an API which consists of a set of Java classes, interfaces and exceptions and a specification to which both JDBC driver vendors and JDBC developers adhere when developing applications.

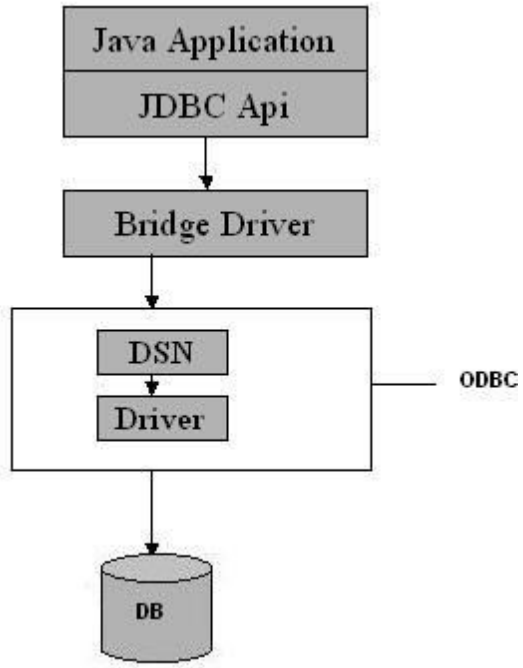
JDBC drivers
Javax.sql API
Database from JSP Page
Struts frame work
JDBC is a very popular data access standard. RDBMS (Relational Database Management Systems) or third-party vendors develop drivers which adhere to the JDBC specification. Other developers use these drivers to develop applications which access those databases e.g. you'll use ConnectorJ JDBC driver to access MySQL database. Since the drivers adhered to JDBC specification, the JDBC application developers can replace one driver for their application with another better one without having to rewrite their application. If they had used some proprietary API provided by some RDBMS vendor, they will not have been able to change the driver and/or database without having to rewrite the complete application.

JDBC Driver Types

JDBC drivers are divided into four types or levels. The **different types of jdbc drivers** are:
Type 1: JDBC-ODBC Bridge driver (Bridge) **Type 2:** Native-API/partly Java driver (Native) **Type 3:** AllJava/Net-protocol driver (Middleware) **Type 4:** All Java/Native-protocol driver (Pure)

Type 1 JDBC Driver
JDBC-ODBC Bridge driver

The Type 1 driver translates all JDBC calls into ODBC calls and sends them to the ODBC driver. ODBC is a generic API. The JDBC-ODBC Bridge driver is recommended only for experimental use or when no other alternative is available.



Type 1: JDBC-ODBC Bridge

Advantage

The JDBC-ODBC Bridge allows access to almost any database, since the database's ODBC drivers are already available.

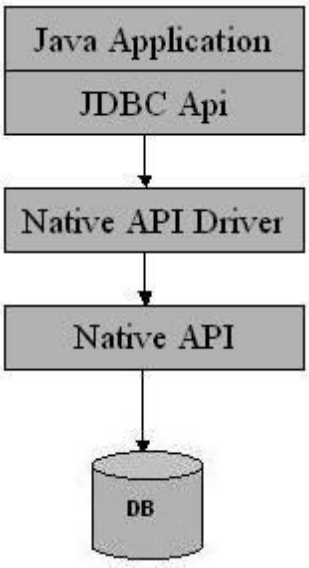
Disadvantages

1. Since the Bridge driver is not written fully in Java, Type 1 drivers are not portable. 2. A performance issue is seen as a JDBC call goes through the bridge to the ODBC driver, then to the database, and this applies even in the reverse process. They are the slowest of all driver types. 3. The client system requires the ODBC Installation to use the driver. 4. Not good for the Web.

Type 2 JDBC Driver

Native-API/partly Java driver

The distinctive characteristic of type 2 jdbc drivers are that Type 2 drivers convert JDBC calls into database-specific calls i.e. this driver is specific to a particular database. Some distinctive characteristic of type 2 jdbc drivers are shown below. Example: Oracle will have oracle native api.



Type 2: Native api/ Partly Java Driver

Advantage

The distinctive characteristic of type 2 jdbc drivers are that they are typically offer better performance than the JDBC-ODBC Bridge as the layers of communication (tiers) are less than that of Type 1 and also it uses Native api which is Database specific

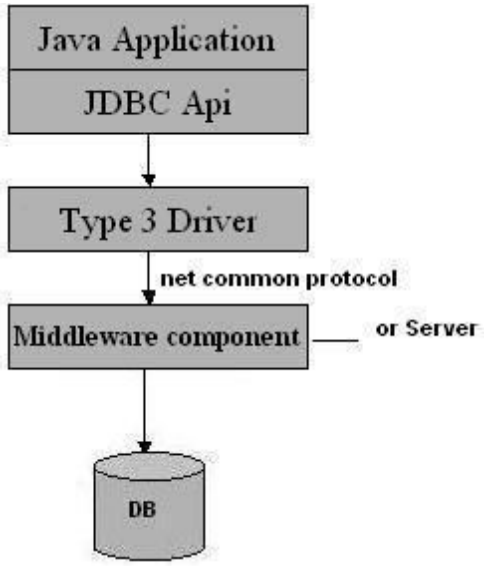
Disadvantage

1. Native API must be installed in the Client System and hence type 2 drivers cannot be used for the Internet. 2. Like Type 1 drivers, it's not written in Java Language which forms a portability issue. 3. If we change the Database we have to change the native api as it is specific to a database 4. Mostly obsolete now 5. Usually not thread safe.

Type 3 JDBC Driver

All Java/Net-protocol driver

Type 3 database requests are passed through the network to the middle-tier server. The middle-tier then translates the request to the database. If the middle-tier server can in turn use Type1, Type 2 or Type 4 drivers.



Type 3: All Java/ Net-Protocol Driver

Advantage

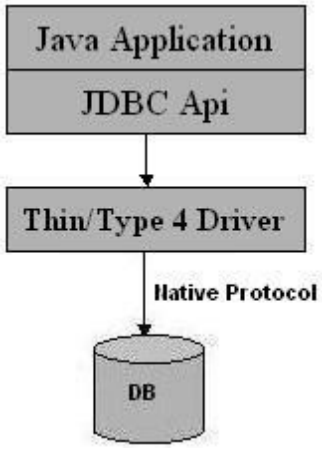
1. This driver is server-based, so there is no need for any vendor database library to be present on client machines. 2. This driver is fully written in Java and hence Portable. It is suitable for the web. 3. There are many opportunities to optimize portability, performance, and scalability. 4. The net protocol can be designed to make the client JDBC driver very small and fast to load. 5. The type 3 driver typically provides support for features such as caching (connections, query results, and so on), load balancing, and advanced system administration such as logging and auditing. 6. This driver is very flexible allows access to multiple databases using one driver. 7. They are the most efficient amongst all driver types. **Disadvantage**

It requires another server application to install and maintain. Traversing the recordset may take longer, since the data comes through the backend server.

Type 4 JDBC Driver

Native-protocol/all-Java driver

The Type 4 uses java networking libraries to communicate directly with the database server



Type 4: Native-protocol/all-Java driver

Advantage

1. The major benefit of using a type 4 jdbc drivers are that they are completely written in Java to achieve platform independence and eliminate deployment administration issues. It is most

suitable for the web. 2. Number of translation layers is very less i.e. type 4 JDBC drivers don't have to translate database requests to ODBC or a native connectivity interface or to pass the request on to another server, performance is typically quite good. 3. You don't need to install special software on the client or server. Further, these drivers can be downloaded dynamically.
Disadvantage With type 4 drivers, the user needs a different driver for each database

6 b. Explain about database connectivity using JDBC

ava Database Connectivity (JDBC) is an application programming interface (API) which allows the programmer to connect and interact with databases. It provides methods to query and update data in the database through update statements like SQL's CREATE, UPDATE, DELETE and INSERT and query statements such as SELECT. Additionally, JDBC can run stored procedures.

The JDBC API uses Java standard classes and interfaces to connect to databases. In order to use JDBC to connect Java applications to a specific database server, a JDBC driver that supports the JDBC API for that database server is required.

To establish a connection between the Java application and the database, JDBC follows certain steps:

1. Loading the driver: The driver provides a connection to the database.
2. Creating the connection: Once the driver is loaded, the next step is to create a connection. The connection object uses a URL in the specified format, which includes the machine name, port number and database name. It communicates with the database object.
3. Executing SQL statements: Requires an object for building the SQL statement.
4. Returning the resultset: Retrieves and manipulates the database queries. Records can be accessed from the first row to the last row of the database.

Java Program for JDBC Connection
Now we will start writing Java program to connect to EXPDB (our example database) through JDBC (Java database Connectivity) and perform INSERT and SELECT operations for demonstration. To illustrate this piece of work, we would take following steps, and finally collect all pieces of code to assemble the complete program.
1. Register JDBC Driver Class
Registering JDBC driver class with the DriverManager means loading JDBC driver class in memory. You can load JDBC driver class in two ways. One way is to load the JDBC driver class in Java program is as follows:

```
try
{
   // loads com.mysql.jdbc.Driver into memory
   Class.forName("com.mysql.jdbc.Driver");
}
catch (ClassNotFoundException cnf)
{
   System.out.println("Driver could not be loaded: " + cnf);
```

```
}
```

lternatively, you can load the JDBC driver class by setting jdbc.drivers property. Then at run time you specify the property with a command-line argument as follows:

```
C:\>java -D jdbc.drivers=com.mysql.jdbc.Driver <Program Name>
```

You can also set the system property within from Java code as follows:

```
System.setProperty("jdbc.drivers", "com.mysql.jdbc.Driver");
```

2. Connect to Database through JDBC Driver

In order to connect to example database EXPDB you need to open a database connection in Java program that you can do as follows by using JDBC driver:

```
//jdbc driver connection string, db username and password
private String connectionUrl = "jdbc:mysql://localhost:3306/EXPDB";
private String dbUser = "root";
private String dbPwd = "mysql";
private Connection conn;
try
{
   conn = DriverManager.getConnection(connectionUrl, dbUser, dbPwd);
}
catch (SQLException sqle)
{
   System.out.println("SQL Exception thrown: " + sqle);
}
```

JDBC Connection Basics

Java database connectivity (JDBC) is a standard application programming interface (API) specification that is implemented by different database vendors to allow Java programs to access their database management systems. The JDBC API consists of a set of interfaces and classes written in the Java programming language. Interestingly, JDBC is not an acronym and thus stands for nothing but popular as *Java Database Connectivity*. According to Sun (now merged in Oracle) JDBC is a trademark term and not an acronym. It was named to be redolent of ODBC.

JDBC implementation comes in form of a database driver for a particular DBMS vendor. A Java program that uses the JDBC API loads the specified driver for a particular DBMS before it actually connects to a database.

JDBC Connection - JDBC Design

JDBC --a trademark, comes bundled with Java SE as a set of APIs that facilitates Java programs to access data stored in a database management system, particularly in a Relational Database. At the time of designing JDBC (Java database connectivity), following goals were kept in mind:

- JDBC (Java database connectivity) should be an SQL level API; therefore, it should allow user to construct SQL statements and embed them into Java API calls to run those statements on a database. And then results from the database are returned as Java objects (ResultSets) and access-problems as exceptions.
- JDBC (Java database connectivity) should provide a driver manager to allow third party drivers to connect to specific databases, where database vendors could provide their own drivers to plug into the driver manager.
  For example MySQL provides its driver in form of a JAR archive (e.g., mysql-connector-java-***-bin.jar). This driver first should be loaded into memory in order to connect to a MySQL database, and then driver manager creates a connection with help of the loaded driver. Whole process will be explained step by step later in this article.
- JDBC (Java database connectivity) should be simple; there would then be a simple mechanism to register third party drivers with the driver manager. Finally, JDBC came into existence as a result of above goals, and two APIs were created. Application programmers use the JDBC API, and database vendors use the JDBC Driver API.

1. JDBC Connection - DataBase URL or String

Database URL for JDBC connection or JDBC connection string follows more or less similar syntax to that of ordinary URLs. It tells the protocol used to connect to database, subprotocol, location of database, port number on which database listens client requests, and database name. The example syntax may look like jdbc:mysql://localhost:3306/EXPDB. Aforementioned URL specifies a MySQL database named EXPDB running on localhost on port 3306.

2. JDBC Connection - Driver Class

As we have obtained the JDBC driver in form of a JAR file (mysql-connector-java-***-bin.jar) in which the driver for MySQL database is located. This driver needs to be registered in order to access EXPDB. Driver file name for MySQL is com.mysql.jdbc.Driver. This file has to be loaded into memory before you get connected to database, else you will result into java.sql.SQLException: No suitable driver exception.

3. JDBC Connection - Database User Name and Password

To get a JDBC connection to the database you would require the username and password, it is the same username and password which we used, while connecting to MySQL.

Java Program for JDBC Connection

Now we will start writing Java program to connect to EXPDB (our example database) through JDBC (Java database Connectivity) and perform INSERT and SELECT operations for demonstration. To illustrate this piece of work, we would take following steps, and finally collect all pieces of code to assemble the complete program.

1. Register JDBC Driver Class

Registering JDBC driver class with the DriverManager means loading JDBC driver class in memory. You can load JDBC driver class in two ways. One way is to load the JDBC driver class in Java program is as follows:

```
try
```

```
{
   // loads com.mysql.jdbc.Driver into memory
   Class.forName("com.mysql.jdbc.Driver");
}
catch (ClassNotFoundException cnf)
{
   System.out.println("Driver could not be loaded: " + cnf);
}
```

If you look at above piece of code you will see that Class.forName takes a string, fully qualified class name of the JDBC driver class file, as an argument and loads the corresponding class into memory. It throws a ClassNotFoundException if fails to locate the driver file. That's the reason it is surrounded by try block.
Alternatively, you can load the JDBC driver class by setting jdbc.drivers property. Then at run time you specify the property with a command-line argument as follows:

```
C:\>java -D jdbc.drivers=com.mysql.jdbc.Driver <Program Name>
```

You can also set the system property within from Java code as follows:

```
System.setProperty("jdbc.drivers", "com.mysql.jdbc.Driver");
```

2. Connect to Database through JDBC Driver

In order to connect to example database EXPDB you need to open a database connection in Java program that you can do as follows by using JDBC driver:

```
//jdbc driver connection string, db username and password
private String connectionUrl = "jdbc:mysql://localhost:3306/EXPDB";
private String dbUser = "root";
private String dbPwd = "mysql";
private Connection conn;
try
{
   conn = DriverManager.getConnection(connectionUrl, dbUser, dbPwd);
}
catch (SQLException sqle)
{
   System.out.println("SQL Exception thrown: " + sqle);
}
```

3. Execute SQL Statements through JDBC Connection

Now that you have a JDBC Connection object conn, you would like to execute SQL statements through the conn (JDBC connection) object. A connection in JDBC is a session with a specific database, where SQL statements are executed and results are returned within the context of a

connection. To execute SQL statements you would need a Statement object that you would acquire by invoking createStatement()method on conn as illustrated below.

```
try
{
   Statement stmt = conn.createStatement();
}
catch (SQLException sqle)
{
   System.out.println("SQL Exception thrown: " + sqle);
}
```

Next, form a query that you would like to execute on database. For an instance, we will select all records from EXPDB, our example database. Method executeQuery() will get you a ResultSet object that contains the query results. You can think a ResultSetobject as two dimensional array object, where each row of an array represents one record. And, all rows have identical number of columns; some columns may contain null values. It is all depend upon what is stored in database.

```
String queryString = "SELECT * FROM EXPTABLE"
try
{
   ResultSet rs = stmt.executeQuery(queryString);
}
catch (SQLException sqle)
{
   System.out.println("SQL Exception thrown: " + sqle);
}
```

Here again, method executeQuery() throws an SQLException, so it should either be surrounded by try block or further throw the exception. After getting a ResultSetobject rs you may like to process records for further use. Here, for illustration we would print them on screen.

```
System.out.println("ID \tNAME");
System.out.println("=============");
while(rs.next())
{
  System.out.print(rs.getInt("id") + "\t" + rs.getString("name"));
  System.out.println();
}
```

4. Close JDBC Connection

JDBC connection to database is a session; it has been mentioned earlier. As soon as you close the session you are no longer connected to database; therefore, you would not be able to perform any

operation on database. Closing connection must be the very last step when you are done with all database operations. You can close a connection to database as follows:

```
try
{
   if (conn != null)
   {
      conn.close();
      conn = null;
   }
}
catch (SQLException sqle)
{
   System.out.println("SQL Exception thrown: " + sqle);
}
```

JDBC Connection - Java Code

Let's assemble above JDBC code fragments explained from step *1. Registering JDBC Driver Class* to *4. Closing JDBC Connection* steps and get a working program.

```
/* JDBC_Connection_Demo.java */

import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.ResultSet;
import java.sql.SQLException;
import java.sql.Statement;

public class JDBC_Connection_Demo
{
 /* static block is executed when a class is loaded into memory
  * this block loads MySQL's JDBC driver
  */
 static
 {
  try
  {
   // loads com.mysql.jdbc.Driver into memory
   Class.forName("com.mysql.jdbc.Driver");
  }
  catch (ClassNotFoundException cnf)
  {
   System.out.println("Driver could not be loaded: " + cnf);
  }
 }
```

```
 public static void main(String[] args)
 {
  String connectionUrl = "jdbc:mysql://localhost:3306/EXPDB";
  String dbUser = "root";
  String dbPwd = "mysql";
  Connection conn;
  ResultSet rs;
  String queryString = "SELECT ID, NAME FROM EXPTABLE";

  try
  {
   conn = DriverManager.getConnection(connectionUrl, dbUser, dbPwd);
   Statement stmt = conn.createStatement();

   // INSERT A RECORD
   stmt.executeUpdate("INSERT INTO EXPTABLE (NAME) VALUES (\''TINU K\'')");

   // SELECT ALL RECORDS FROM EXPTABLE
   rs = stmt.executeQuery(queryString);

   System.out.println("ID \tNAME");
   System.out.println("============");
   while(rs.next())
   {
    System.out.print(rs.getInt("id") + ".\t" + rs.getString("name"));
    System.out.println();
   }
   if (conn != null)
   {
    conn.close();
    conn = null;
   }
  }
  catch (SQLException sqle)
  {
   System.out.println("SQL Exception thrown: " + sqle);
  }
 }
} //JDBC_Connection_Demo ends here
```

OUTPUT

```
ID          NAME
============
```

1. ANUSHKA K
2. GARVITA K
3. CHIRU P

7 a. Develop a servlet that handles an HTTP POST request

Like doGet () method, the do Post () method is invoked by server through service () method to handle HTTP POST request. The doPost () method is used when large amount of data is required to be passed to the server which is not possible with the help of doGet () method.

In doGet () method, parameters are appended to the URL whereas, in do Post () method parameters are sent in separate line in the HTTP request body. The do Get ( ) method is mostly used when some information is to be retrieved from the server and the doPost () method is used when data is to be updated on server or data is to be submitted to the server. To understand the working of do Post () method, let us consider a sample program to define a servlet for handling the HTTP POST request.

A program to define a servlet for handling HTTP POST request

```
import java.io.*;
import java.util.*;
import javax.servlet.*;
public class ServletPostExample extends HttpServlet
{
    public void doPost(HttpServletRequest req, HttpServletResponse res) throws ServletException, IOException
  {
    PrintWriter out = res.getWriter();
    String login= req.getParameter("loginid");
    String password= req.getParameter("password");
    out.println("Your login ID is: ");
    out.println(login);
    out.println("Your password is: ");
    out.println(password);
    out.close();
  }
}
```

The corresponding HTML code for this servlet is as follows

```
<HTML>
      <BODY>
            <CENTER>
                  <FORM NAME="Form1" METHOD="post" ACTION="http://localhost:8080/ServletGetExample">
                  <B>Login ID</B> <INPUT TYPE="text" NAME="loginid" SIZE="30">
                  <P>
                    <B>Password</B>  <INPUT  TYPE="password"  NAME="password" SIZE="30">
```

```
                    </P>
                    <P>
                    <INPUT TYPE=submit VALUE="Submit".>
                    </P>
        </BODY>
</HTML>
```

This HTML code create as web page containing a form

Enter the required data and press the submit button on the web page. The browser will display the response generated dynamically by the corresponding servlet.

7 b. Discuss about session tracking in servlet with a suitable example

Session Tracking is a way to maintain state (data) of an user. It is also known assession management in servlet. Http protocol is a stateless so we need to maintain state using session tracking techniques. Each time user requests to the server, server treats the request as the new request.

Session Tracking in Servlets

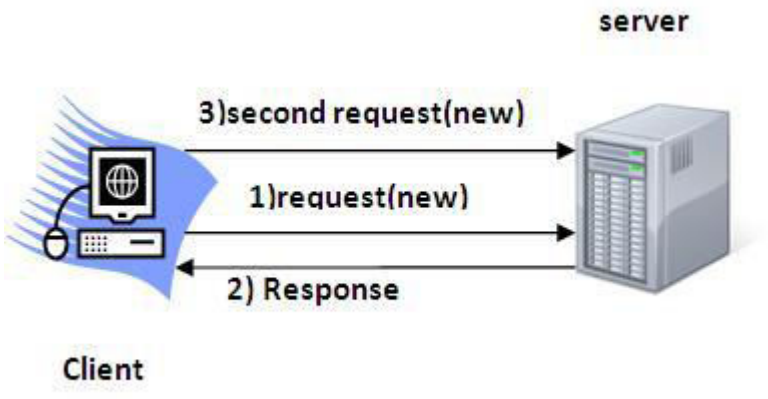
1. Session Tracking
2. Session Tracking Techniques

Session simply means a particular interval of time.

Session Tracking is a way to maintain state (data) of an user. It is also known as session management in servlet.

Http protocol is a stateless so we need to maintain state using session tracking techniques. Each time user requests to the server, server treats the request as the new request. So we need to maintain the state of an user to recognize to particular user.

HTTP is stateless that means each request is considered as the new request. It is shown in the figure given below:



Why use Session Tracking?

To recognize the user It is used to recognize the particular user.

Session Tracking Techniques

There are four techniques used in Session tracking:

1. Cookies
2. Hidden Form Field
3. URL Rewriting
4. HttpSession

Session Tracking Example

This example describes how to use the HttpSession object to find out the creation time and the last-accessed time for a session. We would associate a new session with the request if one does not already exist.

```
// Import required java libraries
import java.io.*;
import javax.servlet.*;
import javax.servlet.http.*;
import java.util.*;

// Extend HttpServlet class
public class SessionTrack extends HttpServlet {

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

```
throws ServletException, IOException {
        // Create a session object if it is already not  created.
HttpSession session = request.getSession(true);
       // Get session creation time.
Date createTime = new Date(session.getCreationTime());
        // Get last access time of this web page.
Date lastAccessTime = new Date(session.getLastAccessedTime());

String title = "Welcome Back to my website";
Integer visitCount = new Integer(0);
String visitCountKey = new String("visitCount");
String userIDKey = new String("userID");
String userID = new String("ABCD");
// Check if this is new comer on your web page.
if (session.isNew()) {
   title = "Welcome to my website";
   session.setAttribute(userIDKey, userID);
} else {
   visitCount = (Integer)session.getAttribute(visitCountKey);
   visitCount = visitCount + 1;
   userID = (String)session.getAttribute(userIDKey);
}
session.setAttribute(visitCountKey,  visitCount);
// Set response content type
response.setContentType("text/html");
PrintWriter out = response.getWriter();
String docType =
   "<!doctype html public \"-//w3c//dtd html 4.0 " +
   "transitional//en\">\n";
out.println(docType +
   "<html>\n" +
      "<head><title>" + title + "</title></head>\n" +
              "<body bgcolor = \"#f0f0f0\">\n" +
         "<h1 align = \"center\">" + title + "</h1>\n" +
         "<h2 align = \"center\">Session Infomation</h2>\n" +
         "<table border = \"1\" align = \"center\">\n" +

           "<tr bgcolor = \"#949494\">\n" +
             "  <th>Session info</th><th>value</th>
           </tr>\n" +

           "<tr>\n" +
             "  <td>id</td>\n" +
             "  <td>" + session.getId() + "</td>
           </tr>\n" +
```

```
            "<tr>\n" +
              "  <td>Creation Time</td>\n" +
              "  <td>" + createTime + "  </td>
            </tr>\n" +

            "<tr>\n" +
              "  <td>Time of Last Access</td>\n" +
              "  <td>" + lastAccessTime + "  </td>
            </tr>\n" +

            "<tr>\n" +
              "  <td>User ID</td>\n" +
              "  <td>" + userID + "  </td>
            </tr>\n" +

            "<tr>\n" +
              "  <td>Number of visits</td>\n" +
              "  <td>" + visitCount + "</td>
            </tr>\n" +
          "</table>\n" +
        "</body>
      </html>"
    );

  }

}
```

Compile the above servlet SessionTrack and create appropriate entry in web.xml file. Now running *http://localhost:8080/SessionTrack* would display the following result when you would run for the first time −

Welcome to my website

Session Infomation

Session info	value
id	0AE3EC93FF44E3C525B4351B77ABB2D5
Creation Time	Tue Jun 08 17:26:40 GMT+04:00 2010

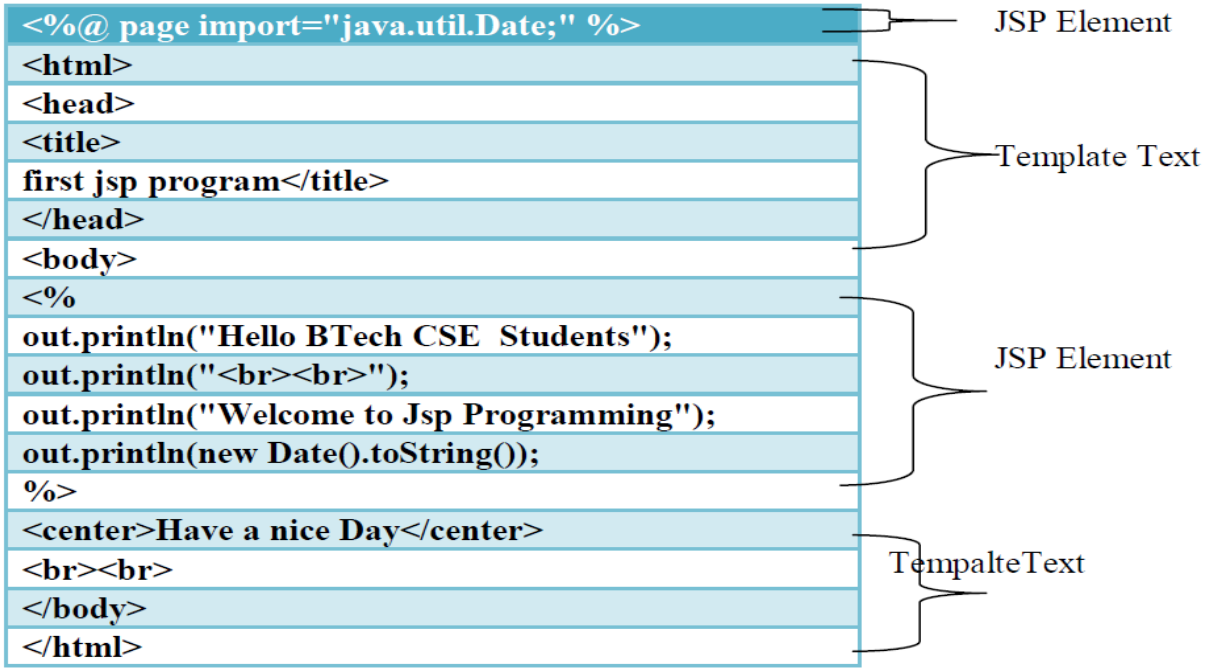
Time of Last Access	Tue Jun 08 17:26:40 GMT+04:00 2010
User ID	ABCD
Number of visits	0

8 a. Describe the anatomy of JSP page

ANATOMY OF JSP PAGE:

- JSP page is a simple web page which contains the **JSP elements** and **template text**.
- The template text can be scripting code such as Html, Xml or a simple plain text.
- Various Jsp elements can be action tags, custom tags, JSTL library elements. These JSP elements are responsible for generating dynamic contents.

Anatomy of JSP



When JSP request gets processed template text and JSP elements are merged together and sent to the browser as response.

A new JSP looks like this:

```
<%@ taglib uri="http://www.atg.com/taglibs/daf/dspjspTaglib1_0" prefix="dsp" %>
<dsp:page>
<html>
<head>
  <title>A Simple Test Page</title>
</head>
```

```
<body bgcolor="#ffffff">
  <h1>A Simple Test Page</h1>
</body>
</html>
</dsp:page>
```

8 b. Write a in brief about JSP Declarations

JSP Declaration Tag
The JSP declaration tag is used to declare fields and methods.
The code written inside the jsp declaration tag is placed outside the service() method of auto generated servlet.
So it doesn't get memory at each request.
Syntax of JSP declaration tag
The syntax of the declaration tag is as follows:

```
<%!  field or method declaration %>
```

xample of JSP declaration tag that declares field

In this example of JSP declaration tag, we are declaring the field and printing the value of the declared field using the jsp expression tag.

index.jsp

```
1. <html>
2. <body>
3. <%! int data=50; %>
4. <%= "Value of the variable is:"+data %>
5. </body>
6. </html>
```

Example of JSP declaration tag that declares method

In this example of JSP declaration tag, we are defining the method which returns the cube of given number and calling this method from the jsp expression tag. But we can also use jsp scriptlet tag to call the declared method.

index.jsp

```
1. <html>
2. <body>
3. <%!
4. int cube(int n){
5. return n*n*n*;
6. }
7. %>
8. <%= "Cube of 3 is:"+cube(3) %>
9. </body>
```

10. </html>

Example:

In this example, we are going to use the declaration tags

```
1. <%@ page language="java" contentType="text/html; charset=ISO-8859-1"
2.    pageEncoding="ISO-8859-1"%>
3. <!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN"
   "http://www.w3.org/TR/html4/loose.dtd">
4. <html>
5. <head>
6. <meta http-equiv="Content-Type" content="text/html; charset=ISO-8859-1">
7. <title>Guru Declaration Tag</title>
8. </head>
9. <body>
10. <%! int count =10; %>
11. <% out.println("The Number is " +count); %>
12. </body>
13. </html>
```

Explanation the code:

Code Line 10: Here we are using declaration tag for initializing a variable count to 10.

9 . Explain in detail of how to use java Beans in JSP pages with suitable examples

JSP - JavaBeans

A JavaBean is a specially constructed Java class written in the Java and coded according to the JavaBeans API specifications.

Following are the unique characteristics that distinguish a JavaBean from other Java classes −

- It provides a default, no-argument constructor.
- It should be serializable and that which can implement the Serializable interface.
- It may have a number of properties which can be read or written.
- It may have a number of "getter" and "setter" methods for the properties.

JavaBeans Properties

A JavaBean property is a named attribute that can be accessed by the user of the object. The attribute can be of any Java data type, including the classes that you define.

A JavaBean property may be read, write, read only, or write only. JavaBean properties are accessed through two methods in the JavaBean's implementation class −

S.No.	Method & Description
1	getPropertyName() For example, if property name is *firstName*, your method name would be getFirstName() to read that property. This method is called accessor.
2	setPropertyName() For example, if property name is *firstName*, your method name would be setFirstName() to write that property. This method is called mutator.

A read-only attribute will have only a getPropertyName() method, and a write-only attribute will have only a setPropertyName() method.

JavaBeans Example

Consider a student class with few properties −

```
package com.tutorialspoint;

public class StudentsBean implements java.io.Serializable {
   private String firstName = null;
   private String lastName = null;
   private int age = 0;

   public StudentsBean() {
   }
   public String getFirstName(){
      return firstName;
   }
   public String getLastName(){
      return lastName;
   }
   public int getAge(){
      return age;
   }
   public void setFirstName(String firstName){
      this.firstName = firstName;
   }
   public void setLastName(String lastName){
```

```
    this.lastName = lastName;
  }
  public void setAge(Integer age){
    this.age = age;
  }
}
```

Accessing JavaBeans

The useBean action declares a JavaBean for use in a JSP. Once declared, the bean becomes a scripting variable that can be accessed by both scripting elements and other custom tags used in the JSP. The full syntax for the useBean tag is as follows −

```
<jsp:useBean id = "bean's name" scope = "bean's scope" typeSpec/>
```

Here values for the scope attribute can be a page, request, session or application based on your requirement. The value of the id attribute may be any value as a long as it is a unique name among other useBean declarations in the same JSP.

Following example shows how to use the useBean action −

```
<html>
  <head>
    <title>useBean Example</title>
  </head>

  <body>
    <jsp:useBean id = "date" class = "java.util.Date" />
    <p>The date/time is <%= date %>
  </body>
</html>
```

You will receive the following result − −

```
The date/time is Thu Sep 30 11:18:11 GST 2010
```

Accessing JavaBeans Properties

Along with <jsp:useBean...> action, you can use the <jsp:getProperty/>action to access the get methods and the <jsp:setProperty/> action to access the set methods. Here is the full syntax −

```
<jsp:useBean id = "id" class = "bean's class" scope = "bean's scope">
  <jsp:setProperty name = "bean's id" property = "property name"
    value = "value"/>
  <jsp:getProperty name = "bean's id" property = "property name"/>
  .....
</jsp:useBean>
```

The name attribute references the id of a JavaBean previously introduced to the JSP by the useBean action. The property attribute is the name of the getor the set methods that should be invoked.

Following example shows how to access the data using the above syntax −

```
<html>
   <head>
      <title>get and set properties Example</title>
   </head>

   <body>
      <jsp:useBean id = "students" class = "com.tutorialspoint.StudentsBean">
         <jsp:setProperty name = "students" property = "firstName" value = "Zara"/>
         <jsp:setProperty name = "students" property = "lastName" value = "Ali"/>
         <jsp:setProperty name = "students" property = "age" value = "10"/>
      </jsp:useBean>

      <p>Student First Name:
         <jsp:getProperty name = "students" property = "firstName"/>
      </p>

      <p>Student Last Name:
         <jsp:getProperty name = "students" property = "lastName"/>
      </p>

      <p>Student Age:
         <jsp:getProperty name = "students" property = "age"/>
      </p>

   </body>
</html>
```

Let us make the StudentsBean.class available in CLASSPATH. Access the above JSP. the following result will be displayed −

```
Student First Name: Zara

Student Last Name: Ali

Student Age: 10
```

10 a. What is need of scripting languages in Web Technologies.

A **scripting** or **script language** is a **programming language** that supports **scripts**: programs written for a special run-time environment that automate the execution of tasks that could alternatively be executed one-by-one by a human operator.**Scripting languages** are often interpreted (rather than compiled).

Characteristics

Typically scripting languages are intended to be very fast to learn and write in, either as short source code files or interactively in a read–eval–print loop (REPL, language shell).[3] This generally implies relatively simple syntax and semantics; typically a "script" (code written in the scripting language) is executed from start to finish, as a "script", with no explicit entry point.
For example, it is uncommon to characterise Java as a scripting language because of its lengthy syntax and rules about which classes exist in which files, and it is not directly possible to execute Java interactively, because source files can only contain definitions that must be invoked externally by a host application or application launcher.

```
public class HelloWorld {
  public void printHelloWorld() {
    System.out.println("Hello World");
  }
}
```

This piece of code intended to print "Hello World" does nothing as *main()* is *not declared* in HelloWorld class.
In contrast, Python allows definition of some functions in a single file, or to avoid functions altogether and use imperative programming style, or even use it interactively.

```
print ("Hello World")
```

This one line of python code prints "Hello World"; no *declarative* statement like *main()* is required here.
A scripting language is usually interpreted from source code or bytecode.[4] By contrast, the software environment the scripts are written for is typically written in a compiled language and distributed in machine codeform.
Scripting languages may be designed for use by end users of a program—end-user development—or may be only for internal use by developers, so they can write portions of the program in the scripting language. Scripting languages typically use abstraction, a form of information hiding, to spare users the details of internal variable types, data storage, and memory management.
Scripts are often created or modified by the person executing them,[5] but they are also often distributed, such as when large portions of games are written in a scripting language.

Top 15 programming languages in 2015

1. JavaScript 2. Java 3. Python 4. CSS 5.PHP 6.Ruby 7.C++ 8. C 9.Shell 10.C# 11.Perl

10 b. Write a program java script to convert temperature from Celsius to Fahrenheit and vice versa or height from centimeters to inches and vice-versa.

```
<!DOCTYPE html>
<html>
<body>
<h2>JavaScript Celcius to Fahrenhet</h2>
<p>Insert a number into one of the input fields below:</p>
<p><input id="c" onkeyup="convert('C')"> degrees Celsius</p>
<p><input id="f" onkeyup="convert('F')"> degrees Fahrenheit</p>
<p>Note that the <b>Math.round()</b> method is used, so that the result will be returned as an integer.</p>
<script>
function convert(degree) {
   var x;
   if (degree == "C") {
      x = document.getElementById("c").value * 9 / 5 + 32;
      document.getElementById("f").value = Math.round(x);
   } else {
      x = (document.getElementById("f").value -32) * 5 / 9;
      document.getElementById("c").value = Math.round(x);
   }
}
</script>
</body>
</html>
```

OUTPUT:

JavaScript Celcius to Fahrenhet

Insert a number into one of the input fields below:

45 degrees Celsius

113 degrees Fahrenheit

Note that the Math.round() method is used, so that the result will be returned as an integer.

Height from centimeters to inches

Description: Take the headache out of figuring out how many inches 176 cm is, or feet 30 inches is, with this extremely handy conversion script.

```
<TABLE cellSpacing=0 cellPadding=0 width=250 border=0>

<FORM name=cminch method="post">
      <TBODY>
      <TR>
        <TD>
          <FONT face="Arial Narrow"><INPUT
          onblur=cmconvert() value=2.54 name=cm> </FONT><FONT face="Arial Narrow"
```

```
size=-1>cm</FONT></TD>
</TR>
<TR>
  <TD>
    <FONT face="Arial Narrow"><INPUT
    onblur=feetconvert() value=0 name=feet> </FONT><FONT face=Arial
    size=-1>feet</FONT></TD>
</TR>
<TR>
  <TD>
    <FONT face="Arial Narrow"><INPUT
    onblur=inchconvert() value=1 name=inch> </FONT><FONT face=Arial
size=-1>inch</FONT></TD>
</TR>
<TR>
  <TD width="50%">
     <p align="center"><input type="button" value="Submit" name="B3"></p>
  </TD>
</TR></TBODY></FORM></TABLE>
<p align="left">
<FONT face="Arial Narrow" size="2">Note: 1 Inch = 2.54 cm, 1 Feet = 30.48 cm, 1
Feet = 12 Inch</FONT>
<BR>

<SCRIPT language=JavaScript>
/*
cm/inch/feet converter - credit must stay intact for use
By Ada Shimar ada@chalktv.com
For this script and more,
Visit http://javascriptkit.com
*/
function roundit(which){
return Math.round(which*100)/100
}
function cmconvert(){
with (document.cminch){
feet.value = roundit(cm.value/30.84);
inch.value = roundit(cm.value/2.54);
}}
function inchconvert(){
with (document.cminch){
cm.value = roundit(inch.value*2.54);
feet.value=roundit(inch.value/12);
}}

function feetconvert(){
```

```
with (document.cminch){
cm.value=roundit(feet.value*30.48);
inch.value=roundit(feet.value*12);
}}
</SCRIPT>
<p align="center">This free script provided by<br />
<a href="http://javascriptkit.com">JavaScript
Kit</a></p>
```

Example: OUTPUT

2.54 cm

0 feet

1 inch

Note: 1 Inch = 2.54 cm, 1 Feet = 30.48 cm, 1 Feet = 12 Inch

11 Write about the following with reference to javascript with an example:

a. Functions

b. Form Validation

a. FUNCTIONS

A function is a piece of code that performs a specific task. The function will be executed by an event or by a call to that function. We can call a function from anywhere within the page (or even from other pages if the function is embedded in an external **.js** file). JavaScript has a lot of builtin functions.

Defining functions:

JavaScript function definition consists of the **function** keyword, followed by the name of the function.

A list of arguments to the function are enclosed in parentheses and separated by commas. The statements within the function are enclosed in curly braces { }.

Syntax:

function functionname(var1,var2,...,varX)
{
some code

Parameter Passing:

Not every function accepts parameters. When a function receives a value as a parameter, that value is given a name and can be accessed using that name in the function. The names of

parameters are taken from the function definition and are applied in the order in which parameters are passed in.

• Primitive data types are passed by value in JavaScript. This means that a copy is made of a variable when it is passed to a function, so any manipulation of a parameter holding primitive data in the body of the function leaves the value of the original variable untouched.

• Unlike primitive data types, composite types such as arrays and objects are passed by reference rather than value.

Examining the function call:

In JavaScript parameters are passed as arrays. Every function has two properties that can be used to find information about the parameters:

***functionname*.arguments**

This is an array of parameters that have been passed

***functionname*.arguments.length**

This is the number of parameters that have been passed into the function

Example:

```
<html>
<body>
<script language="javascript">
function fun(a,b){
var msg = fun.arguments[0]+".."+fun.arguments[1]; //referring a,b values
alert(msg);
}
fun(10,20); //function call
fun("abc","vit"); //function call
</script>
</body>
</html>
```

Returning values

The **return** statement is used to specify the value that is returned from the function. So functions that are going to return a value must use the **return** statement.

Example:

```
function prod(a,b)
{
```

x=a*b; return x;
}

Scoping Rules:

Programming languages usually impose rules, called scoping, which determine how a variable can be accessed. JavaScript is no exception. In JavaScript variables can be either ***local*** or ***global.***

global

Global scoping means that a variable is available to all parts of the program. Such variables are declared outside of any function.

local

Local variables are declared inside a function. They can only be used by that function.

`

Calling a Function

To invoke a function somewhere later in the script, you would simply need to write the name of that function as shown in the following code.

```
<html>
  <head>
      <script type="text/javascript">
      function sayHello()
      {
        document.write ("Hello there!");
      }
    </script>
      </head>
  <body>
    <p>Click the following button to call the function</p>
        <form>
      <input type="button" onclick="sayHello()" value="Say Hello">
    </form>
        <p>Use different text in write method and then try...</p>
  </body>
</html>
```

b. Form Validation:

1. JavaScript form validation
2. Example of JavaScript validation
3. JavaScript email validation

It is important to validate the form submitted by the user because it can have inappropriate values. So validation is must.

The JavaScript provides you the facility the validate the form on the client side so processing will be fast than server-side validation. So, most of the web developers prefer JavaScript form validation.

Through JavaScript, we can validate name, password, email, date, mobile number etc fields.

Form validation normally used to occur at the server, after the client had entered all the necessary data and then pressed the Submit button. If the data entered by a client was incorrect or was simply missing, the server would have to send all the data back to the client and request that the form be resubmitted with correct information. This was really a lengthy process which used to put a lot of burden on the server.

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

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

Example

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

```
<html>

  <head>
    <title>Form Validation</title>

    <script type="text/javascript">
      <!--
        // Form validation code will come here.
      //-->
    </script>

  </head>

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

        <tr>
```

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

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

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

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

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

      </table>
    </form>

  </body>
</html>
```

Basic Form Validation

First let us see how to do a basic form validation. In the above form, we are calling **validate()** to validate data when **onsubmit** event is occurring. The following code shows the implementation of this validate() function.

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

      if( document.myForm.Name.value == "" )
      {
        alert( "Please provide your name!" );
        document.myForm.Name.focus() ;
        return false;
      }

      if( document.myForm.EMail.value == "" )
      {
        alert( "Please provide your Email!" );
        document.myForm.EMail.focus() ;
        return false;
      }

      if( document.myForm.Zip.value == "" ||
      isNaN( document.myForm.Zip.value ) ||
      document.myForm.Zip.value.length != 5 )
      {
        alert( "Please provide a zip in the format #####." );
        document.myForm.Zip.focus() ;
        return false;
      }

      if( document.myForm.Country.value == "-1" )
      {
        alert( "Please provide your country!" );
        return false;
      }
      return( true );
    }
  //-->
</script>
```

Data Format Validation

Now we will see how we can validate our entered form data before submitting it to the web server.

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

Example

Try the following code for email validation.

```
<script type="text/javascript">
  <!--
    function validateEmail()
    {
      var emailID = document.myForm.EMail.value;
      atpos = emailID.indexOf("@");
      dotpos = emailID.lastIndexOf(".");
      if (atpos < 1 || ( dotpos - atpos < 2 ))
      {
        alert("Please enter correct email ID")
        document.myForm.EMail.focus() ;
        return false;
      }
      return( true );
    }
  //-->

</script>
```