Q.2 a. List out HTML LIST tags. Explain HTML LIST tags with example. (1+7)

Answer:

HTML provides three types of list: 1. unordered lists (bullet lists) 2. Ordered lists (numbered lists) and 3. Definition List

Unordered Lists

An unordered list is a list of items marked with bullets (typically small black circles). An unordered list starts with the `<ul>` tag. Each list item starts with the `<li>` tag.

Example:

```
<ul>
<li>Coffee</li>
<li>Milk</li>
</ul>
```

Result:

- Coffee
- Milk

Ordered Lists

An ordered list is also a list of items. The list items are marked with numbers. An ordered list starts with the `<ol>` tag. Each list item starts with the `<li>` tag.

Example

```
<ol style="list-style-type:decimal;">
<li>Item one</li>
 <li>Item two</li>
 <li>Watch, you can easily nest list items: This item has some sub-items</li>
 <ol style="list-style-type:lower-alpha;">
<li>Sub-item one</li>
<li>Sub-item two</li>
<li>Shall we do a 3rd nested list?</li>
<ol style="list-style-type:lower-roman;">
<li>OK</li>
<li>Your browser should automatically use different bullet styles for each
level.</li>
</ol></ol></ol>
```

Result:

1. Item one
2. Item two
3. Watch, you can easily nest list items: This item has some sub-items
    a. Sub-item one
    b. Sub-item two
    c. Shall we do a 3rd nested list?
        i. OK
        ii. Your browser should automatically use different bullet styles for each level.

Definition Lists

Definition lists consist of two parts: a term and a description . To mark up a definition list, you need three HTML elements; a container <dl>, a definition term <dt>, and a definition description <dd>.

Example:

```
<dl>
<dt>Cascading Style Sheets</dt>
<dd>Style sheets are used to provide
presentational suggestions for
documents marked up in HTML.
</dd>
</dl>
```

Result:

Cascading Style Sheets
    Style sheets are used to provide
    presentational suggestions for
documents marked up in HTML.

b. What purpose do the FORM Serves in HTML. Explain any six widgets used in FORM. (2+6)

Answer:

A form is an area that can contain form elements. Form elements are elements that allow the user to enter information (like text fields, textarea field s, drop-down menus, radio buttons, checkboxes, etc.) in a form. A form is defined with the <form> tag.

```
<form>
.
input elements
.
</form>
```

Input:

The most used form tag is the <input> tag. The type of input is specified with the type attribute. The most commonly used input types are explained below.

Text Fields:

Text fields are used when you want the user to type letters, numbers, etc. in a form.

```
<form> First name:<input type="text" name="firstname" /><br />
Last name:<input type="text" name="lastname" /> </form>
```

The width of the text field is 20 characters by default.

Password Field:

Password fields are used when user want to type letters, numbers, etc. And at the same time information typed by the user can't be displayed, they are masked with a character. In a form <input type="password" /> defines a password field: The characters in a password field are masked (shown as asterisks or circles).

```
<form> Password: <input type="password" name="pwd" />
</form>
```

How the HTML code above looks in a browser:

Password: ******

Radio Buttons:

Radio Buttons are used when you want the user to select one of a limited number of choices.

<form> <input type="radio" name="sex" value="male" /> Male
input type="radio" name="sex" value="female" /> Female </form>

Male○ Female○

Checkboxes:

Checkboxes are used when you want the user to select one or more options of a limited number of choices.

<form> I have a bike: <input type="checkbox" name="vehicle"
value="Bike" />
I have a car:<input type="checkbox" name="vehicle"
value="Car"/>
</form>

Submit:

A submit button is used to send form data to a server. The data is sent to the page specified in the form's action attribute. The file defined in the action attribute usually does something with the received input:

<input type="submit" value="Send form">

How it looks in a browser:

Send Without value the submit button look like this **Submit**

Reset:

A 'reset' button resets all controls in the form, clearing the user input and returning controls to their initial values.

<input type="reset" value="Reset form">

How it looks in a browser **Reset**

Button:

The <button> tag defines a push button. A button is marked up as follows: **Click**

<button type="button">Click Me!</button>

Inside a button element you can put content, like text or images. This is the difference between this element and buttons created with the input element.

Q.3 a. What is CSS? What are the benefits of using CSS? Explain the types of CSS with syntax. (1+2+5)

Answer:

CASCADING STYLE SHEETS (CSS):

- CSS lets us to assign several properties at once to
- all the elements
- It is a set of formatting instructions
- Common format for all web pages

- Saves Time
- Easy to Change
- Easy to apply
- Overrides earlier definitions
- Controls text in a way that HTML cannot.

There are three types CSS

1. Inline Style Sheets
2. Internal Style Sheets
3. External Style sheets

INLINE STYLE SHEETS:

Style information can be included inline i.e. within the tag of HTML elements. It can also be called as local styles. All style elements are declared with in the <BODY> tag.

Syntax:

STYLE = " declaration "

Declaration => property: value

Ex.

```
<BODY>
<P STYLE="font-family:Arial"> NRIIT </P>
<BODY>
```

INTERNAL STYLE SHEETS:

Internal styles are ideal for individual pages. Style information can be embedded directly with in thedocument by using the <STYLE> tag.

Syntax:

```
<STYLE>
SELECTOR{ declarator} ....
</STYLE>
```

SELECTOR can be any HTML element

Ex:

```
<HEAD>
<STYLE>
H1{color:red}
</STYLE>
</HEAD>
```

EXTERNAL STYLE SHEETS:

External style sheets are ideal for giving all the pages on our web site a common look. External stylesheet is separate web document which has style definitions, and has an extension ".css". This externalstyle sheet can be linked to any web documents via a URL.Style document:

```
<STYLE>
H2{color:blue}
```

```
H3{background : yellow}
</STYLE>
```

This document should be saved by giving extension as ".css" Linking of external document to our present web document can be done as

```
<HEAD>
<LINK REL=stylesheet TYPE="text/css" HREF="css file
path">
</HEAD>
```

b. Explain CSS Font Properties with Values. **(8)**

Answer:

Font

The font property can set the style, weight, variant, size, line height and font:

 font: italic bold normal small/1.4em Verdana, sans-serif;

The above would set the text of an element to an italic style a bold weight a normal variant a relative size a line height of 1.4em and the font to Verdana or another sans-serif typeface.

Font-Family

You can set what font will be displayed in an element with the font-family property.

There are 2 choices for values:

- family-name
- generic family

If you set a family name it is best to also add the generic family at the end. As this is a priortized list. So if the user does not have the specified font name it will use the same generic family. (see below)

 font-family: Verdana, sans-serif;

Font Size

You can set the size of the text used in an element by using the font-size property.

 font-size: value;

There are a lot of choices for values:

- xx-large
- x-large
- larger
- large
- medium
- small
- smaller
- x-small
- xx-small
- length
- % (percent)

Font Style

You can set the style of text in a element with the font-style property

 font-style: value;

Possible values are

- normal

- itailc
- oblique

Font Variant
You can set the variant of text within an element with the font-variant property
font-variant: value;
Possible values are
- normal
- small-caps

Font Weight
You can control the weight of text in an element with the font-weight property:
font-weight: value;
Possible values are
- lighter
- normal
- 100
- 200
- 300
- Bold
- Bolder

Q.4 a. Explain different types of operators with example in JavaScript. (8)
Answer:
There are six category of the operator in Javascript.
1) Arithmetic
2) Assignment
3) Comparisons
4) Logical Operators
5) Conditional Operators
6) Bitwise operators

1) Arithmetic Operators
Arithmetic operators are : -, +, /, *, %
X + Y if x, y are numbers = num if x, y are in quotes "X" + "y" = xy
Modulus remainder when x is divided by y E.g. 5%2=1
X++ same as x=x+1
++X x+1=X
X-- x=x-1
--X x-1=x
-X reverse sign
++X adds one before any operation
X++ adds one after any operation
--X subtracts one before any operation
X-- subtracts one a
fter any operation
Screen Output
Test Value of a Value b
var a=100 b=a++ 101 100

var b= a++ b=++a 101 101
document.write (a); b=a-- 99 100 document.write (b); b=--a 99 99

2) Assignment Operators
Putting a value into a variable this usually the final operation performed in an expression by equating a variable to a literal value. The = sign is not the only assignment operator.
Assignment What it does
x = y Sets 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
x<<=y
x>>=Y

3) Comparison Operators
Comparison operators generate a TRUE or FALSE output based on a logical comparison of the operands on either side of the comparison. There are 6 kinds of comparison operators

== set to or equals (not to be confused with =)
!= not equals (return true if x is not equal to y)
< less than (return true if x is less than y)
> greater than
<= less than or equal to
<= greater than or equal to
In some languages there is no distinction made between = and ==
x == y return true if x and y are equal
Logical Comparison
x && y return true if x and y are equal
x || y return true if x or y are equal
!x return true if x is false

4) Logical Operators
var snow=true;
var day="Monday"
var month="11"
These are assignments not comparisons Operators&& logical and, returns true if both operands are true, otherwise false;
II logical or, returns true if either operand is true
! not, returns true if the operand is false and false if the operand is true

5) Conditional operators
Evaluate ot one of two different values based on a condition
(condition) ? val1: val2
val1 (true condition)
val2 (false condition)
e.g.
(day == "Saturday") ? "Weekend!" : "Not Saturday!"

6) Bitwise operators
Operate on bits of a variable i.e. 1s and Os, treats numbers as a 32 bit value
NOT ~
AND &
OR |
NOR ^
>>
>>>
<<

b. Explain any four JavaScript string functions with example. (8)

Answer:

There are many JavaScript string function (any four):

1.replace(regexp/substr, replacetext) Searches and replaces the regular expression (or sub string) portion (match) with the replaced text instead.

```
//replace(substr, replacetext)
var myString = 'JavaScript Example;
console.log(myString.replace(/JavaScript/i, "Test"));
//output: Test Example

//replace(regexp, replacetext)
var myString = '11 JavaScript Example ';
console.log(myString.replace(new RegExp( "11", "gi" ), "The"));
//output: The JavaScript Example
```

2. search(regexp) Tests for a match in a string. It returns the index of the match, or -1 if not found.

```
//search(regexp)
var intRegex = /[0-9 -()+]+$/;

var myNumber = '999';
var isInt = myNumber.search(intRegex);
console.log(isInt);
//output: 0

var myString = '999 Test Example’;
var isInt = myString.search(intRegex);
console.log(isInt);
//output: -1
```

3. indexOf(substr, [start]) Searches and (if found) returns the index number of the searched character or substring within the string. If not found, -1 is returned. “Start” is an optional argument specifying the position within string to begin the search. Default is 0.

```
//indexOf(char/substring)
var sentence="Hi, my name is XYZ!"
if (sentence.indexOf("Sam")!=-1)
```

```
        alert("XYZ is in there!")
charCodeAt(x) Returns the Unicode value of the character at position "x" within the
string.
        //charAt(position)
        var message="jquery4u"
        //alerts "q"
        alert(message.charAt(1))
```

Q.5 a. What is DHTML? Write advantage and disadvantages of DHTML. (1+3+3)

Answer:

Dynamic HyerText Markup Language (**DHTML**) is a combination of Web development technologies used to create dynamically changing websites. Web pages may include animation, dynamic menus and text effects. The technologies used include a combination of HTML, JavaScript or VB Script, CSS and the document object model (DOM).

Advantages:-

1. Fast and Zippy: - dHTML loads content on fly. Your whole page does not loads but only the content part that needs to be altered, so saving the crucial time for the users and giving the snazzy look to the website.
2. Plug-ins, we don't need them:- dHTML uses most of the features already present in the browsers, so there is no need to download any sort of Plug-ins.
3. Great Utility:- The dynamic features possessed by dHTML are helping web designers to create Web pages that posses compact looks, downloads fast, have graphic effects, provides greater functionality and can hold much more text or content all at the same time.

Disadvantages:-

1. Costly Editing tools: - although dHTML provides great functionality but the editors' available for that in market are pretty expensive. Examples of dHTML editors are Dreamweaver and Fusion.
2. Long and Complex coding: - dHTML coding is long and complex. Only theexpert Javascript and HTML programmers can write them and edit them with good degree of functionality.
3. Browser Support problems: - dHTML suffers from browser support problems for different browsers. For example, a code written for Netscape might not work in Internet Explorer and Vice-Versa. The problem arises due to the different features of browsers.

b. Write a JavaScript code to display information in Status Bar. (9)

Answer:

```
<html>
<head>
   <title>Status Bar</title>
   <script type="text/javascript">
   <!--window.defaultStatus = "Welcome to the URL page."

     function changeStatus() {
       window.status = "Click me to go to home page."
```

```
        }

        function changeDefaultStatus() {
          window.defaultStatus=window.document.statusForm.
          messageList.options[window.document.statusForm.
          messageList.selectedIndex].text
        }
      //-->
      </script>
  </head>

  <body>
    <p>&#160;</p>
    <p>&#160;</p>
    <p align="center">
      <font size="7" color="#008040">
        <strong>http://www.abc.com</strong>
      </font>
    </p>
    <p align="center">
      <a href="http://www.abc.com"onmouseover="changeStatus();
      return true">Go...</a>
    </p>
    <p align=center>
      <font size="1">
        To change the default status bar message, select a
        message from the list below and click the Change button.
      </font>
    </p>
    <form name="statusForm" method="POST">

      <select name="messageList" size="1">
        <option selected>Welcome to the large URL page.</option>
        <option>A</option>
        <option>B</option>
        <option>C</option>
      </select>
      <input type="button" name="Change" value="Change"
         onclick="changeDefaultStatus()">
    </form>
  </body>
  </html>
```

Q.6 a. What are the fundamentals of arrays in Perl? Explain. (8)

Answer:

An array stores a ordered list of values. While a scalar variable can only store one value, an array can store many. Perl array names are prefixed with a @ sign. Here is an example:
my @colors = ("red","green","blue");

Each individual item (or element) of an array may be referred to by its index number. Array indices start with 0, so to access the first element of the array @colors, you use $colors[0]. Notice that when you're referring to a single element of an array, you prefix the name with $ instead of @. The $-sign again indicates that it's a single (scalar) value; the @-sign means you're talking about the entire array.
If you want to loop through an array, printing out all of the values, you could print each element one at a time:

```
my @colors = ("red","green","blue");
print "$colors[0]\n"; # prints "red"
print "$colors[1]\n"; # prints
"green" print "$colors[2]\n"; # prints "blue"
```

A much easier way to do this is to use a for each loop:

```
my @colors = ("red","green","blue");
foreach my $i (@colors) {
print "$i\n";
}
```

For each iteration of the for each loop, $i is set to an element of the @colors array. In this example, $i is "red" the first time through the loop. The braces {} define where the loop begins and ends, so for any code appearing between the braces, $i is set to the current loop iterator.

b. Explain the different categories of a variable in Perl with an example. (8)

Answer:

Categories of perl variable are:

i. Scalar Variable
ii. Array Variable
iii. Hash Variable

Scalars are variables that can store either numbers, strings, or references

- Numbers are stored in double format; integers are rarely used
- Numeric literals have the same form as in other common languages

Perl has two kinds of string literals, those delimited by double quotes and those delimited by single quotes

- Single-quoted literals cannot include escape sequences
- Double-quoted literals can include them
- In both cases, the delimiting quote can be embedded by preceding it with a backslash
- If you want a string literal with single-quote characteristics, but don‘t want delimit it with single quotes, use qx, where x is a new delimiter
- For double quotes, use qq
- If the new delimiter is a parenthesis, a brace, a bracket, or a pointed bracket, the right delimiter must be the other member of the pair
- A null string can be '' or "" Scalar type is specified by preceding the name with a $
- Name must begin with a letter; any number of letters, digits, or underscore characters can follow
- Names are case sensitive
- By convention, names of variables use only lowercase letters
- Names embedded in double-quoted string literals are interpolated e.g., If the

value of $salary is 47500, the value of "Jack makes $salary dollars per year" is "Jack makes 47500 dollars per year"
- Variables are implicitly declared
- A scalar variable that has not been assigned a value has the value undef (numeric value is 0; string value is the null string)
- Perl has many implicit variables, the most common
- of which is $_ (Look at perldoc perlvar)

An array stores a ordered list of values. While a scalar variable can only store one value, an array can store many. Perl array names are prefixed with a @-sign.
Here is an example: my @colors = ("red","green","blue");
Each individual item (or element) of an array may be referred to by its index number. Array indices start with 0, so to access the first element of the array @colors, you use $colors[0]. Notice that when you're referring to a single element of an array, you prefix the name with $ instead of @. The $-sign again indicates that it's a single (scalar) value; the @-sign means you're talking about the entire array. If you want to loop through an array, printing out all of the values, you could print each element one at a time:
A hash is a special kind of array — an associative array, or paired list of elements. Each pair consists of a string key and a data value.
Perl hash names are prefixed with a percent sign (%). Here's how they're defined: Hash Name key value

```
my %colors = ( "red",
"#ff0000", "green",
"#00ff00", "blue",
"#0000ff", "black",
"#000000", "white",
"#ffffff" );
```

This particular example creates a hash named %colors which stores the RGB HEX values for the named colors. The color names are the hash keys; the hex codes are the hash values. Remember that there's more than one way to do things in Perl, and here's the other way to define the same hash: Hash Name key value

```
my %colors = ( red =>
"#ff0000", green =>
"#00ff00",
blue => "#0000ff",
black =>
"#000000", white
=> "#ffffff" );
```

The => operator automatically quotes the left side of the argument, so enclosing quotes around the key names are not needed. To refer to the individual elements of the hash, you'll do:
$colors{'red'}
Here, "red" is the key, and $colors{'red'} is the value associated with that key. In this case, the value is "#ff0000".
You don't usually need the enclosing quotes around the value, either; $colors{red} also works if the key name doesn't contain characters that are also

Perl operators (things like +, -, =, * and /). To print out all the values in a hash, you can use a foreach loop:

```
foreach my $color (keys %colors) {
print "$colors{$color}=$color\n";
}
```

This example uses the keys function, which returns a list of the keys of the named hash. One drawback is that keys %hashname will return the keys in unpredictable order — in this example, keys %colors could return ("red", "blue", "green", "black", "white") or ("red", "white", "green", "black", "blue") or any combination thereof. If you want to print out the hash in exact order, you have to specify the keys in the for each loop:

```
foreach my $color ("red","green","blue","black","white") {
print "$colors{$color}=$color\n";
}
```

Q.7 a. Explain CGI.pm module with the help of an example. (8)

Answer:

The **Common Gateway Interface** (**CGI**) is a standard that defines how webserver software can delegate the generation of webpages to a console application. Such applications are known as CGI scripts; they can be written in any programming language, although scripting languages are often used. In simple words the CGI provides an interface between the webservers and the clients.

The task of a webserver is to respond to requests for webpages issued by clients (usually web browsers) by analyzing the content of the request (which is mostly in its URL), determining an appropriate document to send in response, and returning it to the client.

If the request identifies a file on disk, the server can just return the file's contents. Alternatively, the document's content can be composed on the fly. One way of doing this is to let a console application compute the document's contents, and tell the web server to use that console application. CGI specifies which information is communicated between the webserver and such a console application, and how.

The webserver software will invoke the console application as a command. CGI defines how information about the request (such as the URL) is passed to the command in the form of arguments and environment variables. The application is supposed to write the output document to standard output; CGI defines how it can pass back extra information about the output (such as the MIME type, which defines the type of document being returned) by prepending it with headers.

CGI.pm is a large and widely used Perl module for programming Common Gateway Interface (CGI) web applications, providing a consistent API for receiving user input and producing HTML or XHTML output. The module is written and maintained by Lincoln D. Stein.

A Sample CGI Page

Here is a simple CGI page, written in Perl using CGI.pm (in object oriented style):

```
#!/usr/bin/perl -w
```

```
#
use strict;
use warnings;
use CGI;
my $cgi = CGI->new();
print $cgi->header('text/html');
print $cgi->start_html('A Simple CGI Page'),
$cgi->h1('A Simple CGI Page'),
$cgi->start_form,
'Name: ',
$cgi->textfield('name'), $cgi->br,
'Age: ',
$cgi->textfield('age'), $cgi->p,
$cgi->submit('Submit!'),
$cgi->end_form, $cgi->p,
$cgi->hr;
if ( $cgi->param('name') ) {
print 'Your name is ', $cgi->param('name'), $cgi->br;
}
if ( $cgi->param('age') ) {
print 'You are ', $cgi->param('age'), ' years old.';
}
print $cgi-
>end_html;
```

This would print a very simple webform, asking for your name and age, and after having been submitted, redisplaying the form with the name and age displayed below it. This sample makes use of CGI.pm's object-oriented abilities;
it can also be done by calling functions directly, without the $cgi->.
Note: in many examples $q, short for query, is used to store a CGI object. As the above example illustrates, this might be very misleading.Here is another script that produces the same output using CGI.pm's procedural interface:

```
#!/usr/bin/perl use strict;
use warnings;
use CGI ':standard';
print header,
start_html('A Simple CGI Page'), h1('A Simple CGI Page'),
start_form,
'Name: ', textfield('name'), br,
'Age: ', textfield('age'), p, submit('Submit!'), end_form, p,
hr;
print 'Your name is ', param('name'), br if param 'name';
print 'You are ', param('age'), ' years old.' if param 'age';
print end_html;
```

b. Describe the steps to create Cookies in terms of CGI. (8)

Answer:

Using CGI.pm, creating cookies that you can store on your client's computers is

very simple. Listing 1 below shows the steps necessary to create a cookie from a Perl program.
In the first step, you just create a CGI object, just as you normally would when using the CGI.pm module.
In the second step, you'll define the cookie properties by calling the cookie method. In this example, the name of the cookie is MY_COOKIE, while the value of the cookie is "BEST_COOKIE=chocolatechip". You create a cookie just like this, by specifying the name and value of the cookie when using the cookie method:

```
$cookie = $query->cookie(-name=>'MY_COOKIE',
   -value=>'BEST_COOKIE=chocolatechip',
   -expires=>'+4h',
   -path=>'/');
```

In your applications you'll probably want to create cookies with names like "SITE-ID" or "USER-ID", and then store a value in the cookie like "12345", or perhaps use something more complicated like "UID12345:NITEMS3:AGE=30". As you can see from this example, what you put in your cookie is entirely up to you.
As you can see from the documentation in the code of Listing 1, the rest of the parameters (besides the name and value) for the $query->cookie method are optional, but in many cases you'll want to set these values to gain more control over your cookies.
Once you've defined your cookie in step two of the process, just add the cookie information to the header of your HTML page in step three with this statement:

```
print $query->header(-cookie=>$cookie);
```

This statement prints the header information to your visitor's browser, and includes the information about the cookie you just created. If the browser will allow cookies to be saved to your visitors computer, the cookie information will be saved to their disk drive for later retrieval.

Q.8 a. List out the data types that PHP supports. Explain any four data types in detail. (4+4)

Answer:

PHP supports the following data types:

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

Variables can store data of different types, and different types of data can do different things. For example, you can add variables whose values are numbers (1 + 2), but adding variables whose values are characters (a + b) doesn't make

much sense.
You can store the following simple types of data in PHP variables:

- **Integer:** A whole number (no fractions), such as –43, 0, 1, 27, or 5438. The range of integers that is allowed varies, depending on your operating system, but in general, you can usually use any number from –2 billion up to +2 billion.
- **Floating point number:** A number (usually not a whole number) that includes decimal places, such as 5.24 or 123.456789. This is often called a *real number* or a *float*.
- **Character string:** A series of single characters, such as **hello**. There is no practical limit on the length of a string.
- **Boolean:** A TRUE or FALSE value.

which determines the flow of execution.
Boolean data types represent two possible states — TRUE or FALSE. Boolean values are used mainly to compare conditions for use in conditional statements. For example, PHP evaluates an expression, such as **$a > $b**, and the outcome is either TRUE or FALSE.
PHP considers the following values FALSE :

- The string FALSE (can be upper- or lowercase)
- The integer 0
- The float 0.0
- An empty string
- The one-character string 0
- The constant NULL

Any other values in a Boolean variable are considered TRUE. If you echo a Boolean variable, the value FALSE displays as a blank string; the value TRUE echoes as a 1. Functions often return a Boolean variable that you can test to see whether the function succeeded or failed.

b. Explain the control structures in PHP with examples. (8)

Answer:

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**.

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";
}
?>
```

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";
}
?>
```

Q.9 a. What is the purpose of DTD? What are four possible keywords in DTD declaration? Write their format. (1+2+5)

Answer:

A DTD is a set of structural rules called declarations. These rules specify a set of elements, along with how and where they can appear in a document

- Purpose: provide a standard form for a collection of XML documents a define a markup language for them.
- DTD provides entity definition.
- With DTD, application development would be simpler.
- Not all XML documents have or need a DTD

<!keyword … >
There are four possible declaration keywords:
ELEMENT, ATTLIST, ENTITY, and NOTATION
<!ELEMENT element_name(list of child names)>
<!ATTLIST element_name attribute_name attribute_type [default _value]> More than one attribute
< !ATTLIST element_name attribute_name1 attribute_type default_value_1 attribute_name 2 attribute_type default_value_2 …>
<!ENTITY [%] entity_name "entity_value">

b. Mention the advantages of XML schema over DTD's. **(8)**

Answer:

XML schemas is similar to DTD i.e. schemas are used to define the structure of the document
DTDs had several disadvantages:

- The syntax of the DTD was un-related to XML, therefore they cannot be analysed with an XML processor
- It was very difficult for the programmers to deal with 2 different types of syntaxes
- DTDs does not support the datatype of content of the tag. All of them are specified as text Schemas themselves are written with the use of a collection of tags, or a vocabulary, from a namespace that is, in effect, a schema of schemas. The name of this namespace is abc.com/XMLSchema”
- Every schema has schema as its root element. This namespace specification appears as follows:
xmlns:xsd = “abc.com/XMLSchema”
- The name of the namespace defined by a schema must be specified with the targetNamespace attribute of the schema element.
targetNamespace = “abc.com/ planeSchema”
- If the elements and attributes that are not defined directly in the schema element are to be included in the target namespace, schema’s elementFormDefault must be set to qualified, as follows:
elementFormDefault = “qualified”
- The default namespace, which is the source of the unprefixed names in the schema, is given with another xmlns specification, but this time without the prefix:
xmlns = “abc.com/ planeSchema”

<u>TEXT BOOK</u>

I. Web Programming – Building Internet Applications, Chris Bates, Third Edition, Wiley Student Edition, 2006