Q2.a) Use HTML to make an image map with clickable regions on it.

Ans:

```
<html>
<body>

<p>Click on the sun or on one of the planets to watch it closer:</p>

<img src="planets.gif" width="145" height="126" alt="Planets" usemap="#planetmap" />

<map name="planetmap">
<area shape="rect" coords="0,0,82,126" alt="Sun" href="sun.htm" />
<area shape="circle" coords="90,58,3" alt="Mercury" href="mercur.htm" />
 <area shape="circle" coords="124,58,8" alt="Venus" href="venus.htm" />
</map>

</body>
</html>
```

b) What is a frameset? Write HTML code of a frameset to create a table of two rows and a single table.

Ans: <FRAMESET ...> defines the general layout of a web page that uses frames. <FRAMESET ...> is used in conjunction with <FRAME ...> and <NOFRAMES>.

<FRAMESET ...> creates a "table of documents" in which each rectangle (called a "frame") in the table holds a separate document. In its simplest use, <FRAMESET ...> states how many columns and/or rows will be in the "table". You must use either the COLS or the ROWS attributes or both. For example, this code creates a set of frames that is two columns wide and two rows deep:

```
<HTML>
<HEAD>
<TITLE>A Basic Example of Frames</TITLE>
</HEAD>

<FRAMESET ROWS="75%, *" COLS="*, 40%">
    <FRAME SRC="framea.html">
    <FRAME SRC="frameb.html">
    <FRAME SRC="framec.html">
    <FRAME SRC="framed.html">
</FRAMESET>

</HTML>
```

<FRAMESET ...> itself only define how many rows and columns of frames there will be. <FRAME ...> defines what files will actual go into those frames.

<FRAMESET ...> can be nested within another <FRAMESET ...> to create a "table within a table". By doing this you can create frames that are not strict grids like in the example above. This set of nested framesets creates the popular "title and sidebar" layout.

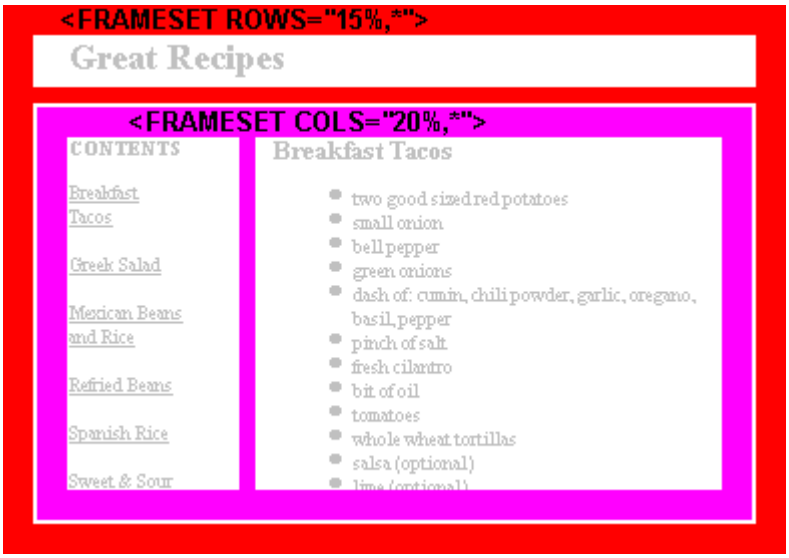
```
<HTML>
<HEAD>
<TITLE>Great Recipes</TITLE>
</HEAD>

<FRAMESET ROWS="15%,*">
    <FRAME SRC="recipetitlebar.html" NAME=TITLE SCROLLING=NO>

    <FRAMESET COLS="20%,*">
        <FRAME SRC="recipesidebar.html" NAME=SIDEBAR>
        <FRAME SRC="recipes.html" NAME=RECIPES>
    </FRAMESET>
</FRAMESET>

</HTML>
```

The first <FRAMESET ...> creates a "table" of two rows and only one column (because there is no COLS attribute). The first row in the frameset is filled in by the first <FRAME ...>. The row in the frameset is filled in not by a frame but by another <FRAMESET ...>. This *inner* frameset has two columns, which are filled in by two <FRAMESET ...>'s.



c) Write a HTML code to link to a mail message(assuming that the mail is already installed).

Ans:

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

```
<p>
This is an email link:
<a href="mailto:someone@example.com?Subject=Hello%20again">
Send Mail</a>
</p>

<p>
<b>Note:</b> Spaces between words should be replaced by %20 to ensure that the browser will
display the text properly.
</p>

</body>
</html>
```

Q3. (a) Using CSS, write a code to create a transparent box with text on a background image.

Ans:

```
<html>
<head>
<style type="text/css">
div.background
{
  width: 500px;
  height: 250px;
  background: url(klematis.jpg) repeat;
  border: 2px solid black;
}
div.transbox
{
  width: 400px;
  height: 180px;
  margin: 30px 50px;
  background-color: #ffffff;
  border: 1px solid black;
  opacity:0.6;
  filter:alpha(opacity=60); /* For IE8 and earlier */
}
div.transbox p
{
  margin: 30px 40px;
  font-weight: bold;
  color: #000000;
}
</style>
</head>
<body><div class="background">
<div class="transbox">
<p>This is some text that is placed in the transparent box.
This is some text that is placed in the transparent box.
```

This is some text that is placed in the transparent box.
This is some text that is placed in the transparent box.
This is some text that is placed in the transparent box.
</p>

</body>
</html>

b) Discuss the following CSS terms with examples:

i. CSS Border

Ans: (i) CSS Border Properties:

The CSS border properties allow you to specify the style and color of an element'sborder.

Border Style
The border-style property specifies what kind of border to display.
None of the border properties will have ANY effect unless the border-style property is set!

border-style values:

none: Defines no border

dotted: Defines a dotted border

dashed: Defines a dashed border

solid: Defines a solid border

double: Defines two borders. The width of the two borders are the same as the border-width value

groove: Defines a 3D grooved border. The effect depends on the border-color value

ridge: Defines a 3D ridged border. The effect depends on the border-color value

inset: Defines a 3D inset border. The effect depends on the border-color value

outset: Defines a 3D outset border. The effect depends on the border-color value

Border Width

The border-width property is used to set the width of the border.

The width is set in pixels, or by using one of the three pre-defined values: thin, medium, or thick.

Note: The "border-width" property does not work if it is used alone. Use the "border-style" property to set the borders first.

Example

```
p.one
{
border-style:solid;
border-width:5px;
}
p.two
{
border-style:solid;
border-width:medium;
}
```

Border Color

The border-color property is used to set the color of the border. The color can be set by:

- name - specify a color name, like "red"
- RGB - specify a RGB value, like "rgb(255,0,0)"
- Hex - specify a hex value, like "#ff0000"

You can also set the border color to "transparent".

The "border-color" property does not work if it is used alone. Use the "border-style" property to set the borders first.

Example

```
p.one
{
border-style:solid;
border-color:red;
}
p.two
{
border-style:solid;
```

```
border-color:#98bf21;
}
```

Border - Individual sides

In CSS it is possible to specify different borders for different sides:

Example

```
p
{
border-top-style:dotted;
border-right-style:solid;
border-bottom-style:dotted;
border-left-style:solid;
}
```

The example above can also be set with a single property:

Example

```
border-style:dotted solid;
```

The border-style property can have from one to four values.

- border-style:dotted solid double dashed;
  - top border is dotted
  - right border is solid
  - bottom border is double
  - left border is dashed
- border-style:dotted solid double;
  - top border is dotted
  - right and left borders are solid
  - bottom border is double
- border-style:dotted solid;
  - top and bottom borders are dotted
  - right and left borders are solid
- border-style:dotted;
  - all four borders are dotted

The border-style property is used in the example above. However, it also works with border-width and border-color.

Border - Shorthand property

As you can see from the examples above, there are many properties to consider when dealing with borders.

To shorten the code, it is also possible to specify all the border properties in one property. This is called a shorthand property.

The shorthand property for the border properties is "border":

Example

```
border:5px solid red;
```

When using the border property, the order of the values are:

- border-width
- border-style
- border-color

It does not matter if one of the values above are missing (although, border-style is required), as long as the rest are in the specified order.

ii. Padding Properties

Ans. (ii) The CSS padding properties define the space between the element border and the element content.

Padding

The padding clears an area around the content (inside the border) of an element. The padding is affected by the background color of the element.

The top, right, bottom, and left padding can be changed independently using separate properties. A shorthand padding property can also be used, to change all paddings at once.

Possible Values

Value	Description
Length	Defines a fixed padding (in pixels, pt, em, etc.)
%	Defines a padding in % of the containing element

Padding - Individual sides In CSS, it is possible to specify different padding for different sides:

Example

```
padding-top:25px;
padding-bottom:25px;
padding-right:50px;
padding-left:50px;
```

Padding - Shorthand property

To shorten the code, it is possible to specify all the padding properties in one property. This is called a shorthand property.

The shorthand property for all the padding properties is "padding":

Example

```
padding:25px 50px;
```

The padding property can have from one to four values.

- padding:25px 50px 75px 100px;
  - top padding is 25px
  - right padding is 50px
  - bottom padding is 75px
  - left padding is 100px
- padding:25px 50px 75px;
  - top padding is 25px
  - right and left paddings are 50px
  - bottom padding is 75px
- padding:25px 50px;
  - top and bottom paddings are 25px
  - right and left paddings are 50px
- padding:25px;
- all four paddings are 25px

iv. CSS media types

Ans. (iv) Media Types allow you to specify how documents will be presented in different media. The document can be displayed differently on the screen, on the paper, with an aural browser, etc.

Media Types

Some CSS properties are only designed for a certain media. For example the "voice-family" property is designed for aural user agents. Some other properties can be used for different media types. For example, the "font-size" property can be used for both screen and print media, but perhaps with different values. A document usually needs a larger font-size on a screen than on paper, and sans-serif fonts are easier to read on the screen, while serif fonts are easier to read on paper.

The @media Rule

The @media rule allows different style rules for different media in the same style sheet.

The style in the example below tells the browser to display a 14 pixels Verdana font on the screen. But if the page is printed, it will be in a 10 pixels Times font. Notice that the font-weight is set to bold, both on screen and on paper:

```
<html>
<head>
<style>
@media screen
  {
  p.test {font-family:verdana,sans-serif;font-size:14px;}
  }
@media print
  {
  p.test {font-family:times,serif;font-size:10px;}
  }
@media screen,print
  {
  p.test {font-weight:bold;}
  }
</style>
</head>

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

Different Media Types

Note: The media type names are not case-sensitive.

Media Type	Description
all	Used for all media type devices
aural	Used for speech and sound synthesizers
braille	Used for braille tactile feedback devices
embossed	Used for paged braille printers
handheld	Used for small or handheld devices
print	Used for printers
projection	Used for projected presentations, like slides
screen	Used for computer screens
tty	Used for media using a fixed-pitch character grid, like teletypes and terminals
tv	Used for television-type devices

Q4 a) Write a code in Javascript which checks if a field has been left empty. If the field is blank, an alert box alerts a message, the function returns false, and the form will not be submitted.

Ans:

```
 function validateForm()
{
var x=document.forms["myForm"]["fname"].value;
if (x==null || x=="")
  {
  alert("First name must be filled out");
  return false;
  }
}
```

The function above could be called when a form is submitted:

```
<form name="myForm" action="demo_form.asp" onsubmit="return validateForm()"
method="post">
First name: <input type="text" name="fname">
<input type="submit" value="Submit">
</form>
```

b) Write a Javascript code to resize a window to a specified size.

Ans:

```
 <html>
<head>
<script type="text/javascript">
function resizeWindow()
{
top.resizeTo(500,300);
}
</script>
</head>

<body>
<form>
<input type="button" onclick="resizeWindow()" value="Resize window" />
</form>
<p><b>Note:</b> We have used the <b>top</b> element instead of the <b>window</b>
element, because we use frames. If you do not use frames, use the <b>window</b> element
instead.</p>
</body>

</html>
```

c) Explain the terms ‘Exception Handling’ with an appropriate example in Javascript.

Ans: The try...catch statement allows you to test a block of code for errors. The try block contains the code to be run, and the catch block contains the code to be executed if an error occurs.

Syntax

```
try
  {
  //Run some code here
  }
catch(err)
  {
  //Handle errors here
  }
```

Note that try...catch is written in lowercase letters. Using uppercase letters will generate a JavaScript error!

The example below is supposed to alert "Welcome guest!" when the button is clicked. However, there's a typo in the message() function. alert() is misspelled as adddlert(). A JavaScript error occurs. The catch block catches the error and executes a custom code to handle it. The code displays a custom error message informing the user what happened:

Example

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

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

</html>
```

The next example uses a confirm box to display a custom message telling users they can click OK to continue viewing the page or click Cancel to go to the homepage. If the confirm method

returns false, the user clicked Cancel, and the code redirects the user. If the confirm method returns true, the code does nothing:

Example

```
<html>
<head>
<script type="text/javascript">
var txt="";
function message()
{
try
  {
  adddlert("Welcome guest!");
  }
catch(err)
  {
  txt="There was an error on this page.\n\n";
  txt+="Click OK to continue viewing this page,\n";
  txt+="or Cancel to return to the home page.\n\n";
  if(!confirm(txt))
    {
    document.location.href="http://www.w3schools.com/";
    }
  }
}
</script>
</head>

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

</html>
```

The throw Statement

The throw statement can be used together with the try...catch statement, to create an exception for the error

Q5 a) Generate a multiplication table using nested loops in Javascript.

1	2	3	4	5	6	7	8	9
2	4	6	8	10	12	14	16	18
3	6	9	12	15	18	21	24	27
4	8	12	16	20	24	28	32	36
5	10	15	20	25	30	35	40	45
6	12	18	24	30	36	42	48	54
7	14	21	28	35	42	49	56	63
8	16	24	32	40	48	56	64	72
9	18	27	36	45	54	63	72	81
10	20	30	40	50	60	70	80	90

Ans:

```
<SCRIPT>

// Use single-spaced text for Multiplication table
document.write("<CENTER><BLOCKQUOTE><STRONG><PRE><FONT
COLOR='FF0000'">)

var i, j, total; // global variables

for (i = 1; i <= 10; i++) {
      for (j = 1; j < 10; j++) {
         total = i * j;
         total = " " + total //add space before each number

         // Add another space before single digits
         if (total < 10) total = " " + total;

         // Display result
         document.write(total);
       } // end inner j loop

    document.write("<BR>"); // end of line break
   } // end of i outer loop

document.write("</FONT></PRE></STRONG></BLOCKQUOTE></CENTER>");

</SCRIPT>

for (i = 1; i <= 10; i++) { // when i = 1
      for (j = 1; j < 10; j++) {

      when j = 1 => i * j = 1 * 1 = 1
      when j = 2 => i * j = 1 * 2 = 2
      when j = 3 => i * j = 1 * 3 = 3
      etc...
```

when j = 10 => i * j = 1 * 10 = 10
break out of the inner loop & go back to the outer loop

for (i = 1; i <= 10; i++) { // when i = 2
for (j = 1; j < 10; j++) {

when j = 1 => i * j = 2 * 1 = 2
when j = 2 => i * j = 2 * 2 = 4
when j = 3 => i * j = 2 * 3 = 6
etc...
when j = 10 => i * j = 2 * 10 = 20
break out of the inner loop & go back to the outer loop

etc...

for (i = 1; i <= 10; i++) { // when i = 10
for (j = 1; j < 10; j++) {

when j = 1 => i * j = 10 * 1 = 10
when j = 2 => i * j = 10 * 2 = 20
when j = 3 => i * j = 10 * 3 = 30
etc...
when j = 10 => i * j = 10 * 10 = 100

Finished at this point

b) How can we download multiple pages in a single download? Explain by using a DHTL and Javascript program.

Ans:

```
#!/usr/bin/perl
#=== VARIABLES ===#

$cal_prog="/usr/bin/cal";#cal program is usually /bin/cal or /usr/bin/cal

$inner="#D3D3D3"; #Inside table color

$width="WIDTH=\"30\"";#Inside table cell width

$align="ALIGN=\"RIGHT\""; #Table cell align

$color="COLOR=\"#003CBA\"";#The current date color

@MONTH=(Jan,Feb,Mar,Apr,May,Jun,Jul,Aug,Sep,Oct,Nov,Dec);

@DAY= (Sun,Mon,Tue,Wed,Thu,Fri,Sat);

#=== GET DATE & ADJUST ===#

(undef,undef,undef,$mday,$mon,$year,$wday,undef,undef) = localtime(time);
```

```
$year += 1900;#ADJUST YEAR FROM 103 TO 2003
if($mday<10) {$mday="0$mday";}#ADJUST DATE IF LESS THAN 10
$mo_num=$mon+1;
#=== START CALENDER TABLE ===#
print <<HEADER;
Content-type: text/html
<TABLE BORDER=1 BGCOLOR=$inner>
 <TH><FONT $color>$DAY[$wday]  $MONTH[$mon] $mday  $year</FONT></TH>
 <TR><TD>
 <TABLE BORDER=1 BGCOLOR=$inner>
<TR>
HEADER
for($cnt=0;$cnt<7;$cnt++) {
print "<TD $align $width>$DAY[$cnt]</TD>\n";
 }
print "</TR>";
#=== START DATES ON EACH DAY ===#
#THE 'if' below tosses out the first two lines and the last empty one
open(INCAL,"$cal_prog $mo_num $year | ") or die "Can't run $cal_prog: $!";
$cnt=1;
while($in_line = <INCAL>)
{
 if($cnt > 2 && $in_line ge " ")
 {
 chop $in_line;
 $sun=substr($in_line,0,2);
 $mon=substr($in_line,3,2);
 $tue=substr($in_line,6,2);
 $wed=substr($in_line,9,2);
```

```
 $thu=substr($in_line,12,2);
 $fri=substr($in_line,15,2);
 $sat=substr($in_line,18,2);
 print "<TR>\n";
 if($sun eq $mday) {print "<TD $align><FONT $color><B>$mday</B></FONT></TD>\n";}
 else{print "<TD $align>$sun</TD>\n";}
 if($mon eq $mday) {print "<TD $align><FONT $color><B>$mday</B></FONT></TD>\n";}
 else{print "<TD $align>$mon</TD>\n";}
 if($tue eq $mday) {print "<TD $align><FONT $color><B>$mday</B></FONT></TD>\n";}
 else{print "<TD $align>$tue</TD>\n";}
 if($wed eq $mday) {print "<TD $align><FONT $color><B>$mday</B></FONT></TD>\n";}
 else{print "<TD $align>$wed</TD>\n";}
 if($thu eq $mday) {print "<TD $align><FONT $color><B>$mday</B></FONT></TD>\n";}
 else{print "<TD $align>$thu</TD>\n";}
 if($fri eq $mday) {print "<TD $align><FONT $color><B>$mday</B></FONT></TD>\n";}
 else{print "<TD $align>$fri</TD>\n";}
 if($sat eq $mday) {print "<TD $align><FONT $color><B>$mday</B></FONT></TD>\n";}
 else{print "<TD $align>$sat</TD>\n";}
 print "</TR>\n";
 }
 $cnt++;
}
#=== END CALENDER TABLE ===#
print <<FOOTER;
 </TABLE>
 </TD></TR>
</TABLE>
FOOTER
close(INCAL);
```

```
exit(0);
```

Q6 b) Write short notes on the following:

i. PERL Control Structures

Ans: i. Control Structures

Control Structures provide a means of controlling the flow of a program. So far we've looked only at basic expressions, now we look at how to tie these expressions together in to a program that can iterate (loop) or make decisions (conditionals) based on the state of variables.

Perl contains the typical assortment of control structures including if-then-else conditionals, while and for loops. Perl also contains some new control structures you may not have seen before (new to a C/C++ programmer).

The if statement

The perl if statement looks much like a C if statement. Here is the general structure

```
if (expression) {
        stmt1;}
```

The value of expression determines whether or not the perl statement stmt1 is executed. If the value of expression is considered "TRUE" by perl, then stmt1 is executed, otherwise stmt1 is skipped.

perl TRUE and FALSE

In the C language any non-zero value is considered to be TRUE and the value 0 is considered FALSE. In perl things are a bit more complex, this is necessary since we often deal with strings (not just numbers). In perl, the following determine the TRUTH of an expression:

The numeric value 0 is considered FALSE, all other numeric values are considered TRUE.

The empty string "" is FALSE, any non-empty string is TRUE*.

*: There is an exception to this rule - the string "0" is FALSE

Formally, perl first converts a conditional value to a string and then if the result is either the empty string or the string "0" the expression is FALSE, otherwise it is TRUE. Note that the string "00" is considered TRUE!

The special perl value undef is considered false (formally this is because when converted to a string it has the value "").

Curly Brackets

In perl you can create a block of statements by enclosing a sequence of statements inside curly braces { }. In C/C++ a single statment inside a control strucuture (like an IF) does not require curly braces - in perl the curly braces are always required.

If Examples

We can now look at some example if statements:

```
# prompt and read in a value from stdin

printf("Enter a number greater than 0\n");

$num = <STDIN>;

# make sure they followed the directions

if ($num < 1) {

  print "Not a valid number dummy!\n";

}

# if the number is less than 50, call user a pessimist

if ($num < 50) {

  print "Your number is too small, you must not be a happy person\n";

}
```

Perl allows a sequence of statements inside the curly braces, this sequence can include multiple if statements (any kind of statement). Perl also supports an else continuation of the if. A more complex example:

```
# prompt and read in a value from stdin

printf("Enter a number between 1 and 100 inclusive\n");

$num = <STDIN>;

# make sure they followed the directions

if ($num >= 1) {

    if ($num <= 100) {
```

```
        # They followed directions - now evaluate thier personality
        if ($num < 50) {
            print "Your number is small - you think little of yourself\n";
        } else {
            print "Your number is large - you think too much of yourself\n";
        }
    } else {
        # they entered a number too large - call them a name
        print "Your number is too large dummy!\n";
    }
} else {
    # the number entered is too small - call them a name
    print "Your number is too small you are either a zero or too negative!\n";
}
```

A common pattern is a sequence of if-else statements. In perl you can combine an else with a following if with an elsif as in this example:

```
print "Enter a number between 1 and 100 inclusive\n";
$num = <STDIN>;
# 1-33 is too small
# 34-65 is just right
# 66-100 is too big
if ($num < 1) {
    print "Invalid\n";
} elsif ($num < 34) {
```

```
    print "Too small\n";
} elsif ($num < 66) {
    print "Just right\n";
} elsif ($num < 101) {
    print "Too large\n";
} else {
    print "Invalid\n";
}
```

The unless Control Structure

Ever want to leave off the if part and just keep the else part of an if-else? Perl has just what you need - it's called unless. The statments inside an unless are not executed if the conditional expression is TRUE:

```
# we could do it this way
if ($cookie{$student} eq "chocolatechip") {
    # do nothing - this student is OK
} else {
    print "You fail!\n";
}
# or we could use an unless
unless ($cookie{$student} eq "chocolatechip") {
   print "You fail!\n";
}
```

The while statement

A perl while is used to create iteration (loops). The general form looks like this:

```
while (expression) {
```

```
  stmt1;
  ...  # could be more statments here
}
```

This is like an if except that stmt1 (and any others inside the braces) is executed over and over as long as expression is TRUE.

```
print "Enter a number greater than 0\n";
$num = <STDIN>;
$tries = 1;
# keep doing until the user gets it right!
while ($num <= 0) {
   print "Tough day? (try again)\n";
   print "Enter a number greater than 0\n";
  $num = <STDIN>;
   # keep track of how many times it takes to get it right
   $tries = $tries + 1;
}
print "You did that in $tries tries\n";
```

The until loop

If you would prefer to have the statements inside the loop repeated until some condition is FALSE, you can use an until:

```
print "Enter a number greater than 0\n";
$num = <STDIN>;
$tries = 1;
#keep doing until the user gets it right!
until ($num > 0) {
```

```
    print "Tough day? (try again)\n";

    print "Enter a number greater than 0\n";

    $num = <STDIN>;

    # keep track of how many times it takes to get it right

    $tries = $tries + 1;

}

print "You did that in $tries tries\n";
```

Looping over input

Perl programs often execute a sequence of statements for each line of input read in. The <STDIN> input operator returns the next line of input available each time it is called, and returns the special perl value undef when the end of file has been found. We can use <STDIN> as the condition used to control a loop:

```
# read in lines from stdin one at a time, and print them back out.

while ($line = <STDIN>) {

    print $line;

}

# This code simply prints out whatever it gets from STDIN
```

Logical Operators - OR and AND

Perl supports C style logical operators that can be used to build complex conditionals. The || operator is a binary operator that is TRUE if either operand is TRUE. So the expression ($num < 0) || ($num > 100) is true if either $num < 0 or if $num > 100. Perl also supports the && operator (logical AND) that is true only when both operands are true. Some examples:

```
print "Enter a number between 1 and 100 inclusive\n";

$num = <STDIN;>;

# if num is less than 1 or greater than 100 - not valid input.

if ( ( $num < 1) || ( $num > 100 ) ) {

    print "Dummy - can't you follow directions!?!\n";
```

```
} else {
    print "Excellent - you would make a great physican\n";
}
# if num is between 33 and 66 this is a great user!
if ( ($num>33) && ($num<66) ) {
    print "You are quite a user, best I've seen today!\n";
}
```

It is often useful to realize that the initializer and increment statments can do anything at all! Here is another example:

```
# print out a list of names, one per line
for ( @names = ("Bill","Mary","John","Sue") ;  @names != 0 ;  shift( @names ) {
   print "Name is $names[0]\n";
}
```

The foreach statement

The perl foreach is another iteration control structure. A scalar variable takes on each of the values from an array before a block of statements is executed. Here is the general structure:

```
foreach $var (@an_array) {
   stmt1;
   ...    # could be more statements here
}
```

The first time the body of the loop is executed the variable $var has the value $an_array[0], the next time it has the value $an_array[1] and so on through the array. An example:

```
# define an array
@names = ("Bill", "Nancy", "John", "Susan");
# print out each name on it's own line
```

```
foreach $i (@names) {
   print "Name is $i\n";
}
# This code should output:
#   Name is Bill
#   Name is Nancy
#   Name is John
#   Name is Susan
```

A foreach doesn't need to have $var specified, so it can look like this:

```
foreach (@an_array) {
   stmt1;
   ...   # could be more statements here
}
```

If no scalar variable is present in a foreach then perl uses the default scalar variable $_. The example could now become:

```
# define an array
@names = ("Bill", "Nancy", "John", "Susan");


# print out each name on it's own line
foreach (@names) {
   print "Name is $_\n";
}
```

ii. Arrays in PERL.

Ans: We can define Perl arrays as having the following general characteristics:

- they are ordered lists of scalars;
- the variable name starts with the **@** character;
- the first element of the list has the index 0, the next has the index 1, and so on;
- the number of array elements is unlimited;
- an element of the array variable starts with the $ character followed by the variable name and the array element index included in square brackets;

There is an example of a Perl array variable below:

```
@family = ("father","mother","son","daughter");
```

In this instance through @family we understand the entire array and if we want to get the first element of the @family array, we use the notation $family[0]. The strings "father", "mother", "son" and "daughter" are the elements of the @family array.

Before seeing what we can do with the Perl arrays, let's speak a bit about context, which is an important feature of Perl Language. There are two most important Perl contexts: scalar and list. The context modifies the operators and functions behavior depending on the type of expression we have to deal with: list or scalar. Please look at these examples shown below:

```
@v1 = @family;
($v2)  =  @family;
$v3  = @family;
$v4 = ("father", "mother", "son", "daughter");
```

In the first example we have list context on the left, after assignment @v1 and @family variables will have the same content, i. e. the list ("father", "mother", "son", "daughter").

In the second example, we have also list context on the left, but the list on the left has only one element, so the scalar variable $v2 will be assigned with the first element of the @family array, i.e. "father".

In the third example, we have scalar context on the left, so after assignment $v3 it will contain the number of @family array elements (i.e. 4) – because @family was evaluated in a scalar context.

And finally, in the fourth example we have scalar context on the left, so the variable $v4 will be assigned step by step with the @family elements and at the end it will contain the last value in the list, i.e. the string "daughter".

Q7 a) Write a program in CGI to implement multiline text fields and input. Implement scrolls also. Explain the tags used and their attributes also.

Ans:

```
<FORM ACTION="/cgi-bin/program.pl" METHOD="POST">
```

```
<TEXTAREA ROWS=10 COLS=40 NAME="comments">
</TEXTAREA>
```

This creates a scrolled text field with 10 rows and 40 columns. (10 rows and 40 columns designates only the visible text area; the text area will scroll if the user types further).

Notice that you need both the beginning <TEXTAREA> and the ending </TEXTAREA> tags. You can enter default information between these tags.

```
<TEXTAREA ROWS=10 COLS=40 NAME="comments_2">
This is some default information.
Some more...
And some more...
</TEXTAREA>
</FORM>
```

You have to remember that newlines (or carriage returns) are not ignored in this field--unlike HTML. In the preceding example, the three separate lines will be displayed just as you typed them.

b) Implement reading cookies through CGI and HTML. Explain the code.

Ans:

```
 #!/usr/bin/perl -wT
use CGI qw(:standard);
use CGI::Carp qw(warningsToBrowser fatalsToBrowser);
use strict;

print header();
print start_html("Cookie");
print h2("Welcome!");

if (my $cookie = cookie('mycookie')) {
   print "Your cookie is $cookie.<br>";
} else {
   print "You don't have a cookie named `mycookie'.<br>";
}

print end_html;
```

c) Give a short note on uploading files on web with PERL.

Ans:

```
 if ($g_restrict_by_ext == 1)
    {
       my $file=$q::upload_file;
       my @ta=split('\.',$file);
```

```
        my $sz=scalar(@ta);
        if ($sz > 1)
        {
            my $ext=$ta[$sz-1];
            if (! grep(/$ext/i,@g_allowed_ext))
            {
                &printError("You are not allowed to upload this file");
                return;
            }

        }
        else
        {
            &printError("You are not allowed to upload this file");
             return;
        }
    }

    # now upload file
    &uploadFile();

    if ($g_debug == 1)
    {
        my @all=$query->param;
        my $name;
        foreach $name (@all)
        {
            print "$name ->", $query->param($name),"<br>\n";
        }
    }
}
```

Q8 a) Give a short note on PHP-XML Parser Functions.

Ans: PHP XML Parser Introduction

The XML functions lets you parse, but not validate, XML documents.

XML is a data format for standardized structured document exchange. More information on XML can be found in our XML Tutorial.

This extension uses the Expat XML parser.

Expat is an event-based parser, it views an XML document as a series of events. When an event occurs, it calls a specified function to handle it.

Expat is a non-validating parser, and ignores any DTDs linked to a document. However, if the document is not well formed it will end with an error message.

Because it is an event-based, non validating parser, Expat is fast and well suited for web applications.

The XML parser functions lets you create XML parsers and define handlers for XML events.

PHP XML Parser Functions

PHP: indicates the earliest version of PHP that supports the function.

Function	Description	PHP
utf8_decode()	Decodes an UTF-8 string to ISO-8859-1	3
utf8_encode()	Encodes an ISO-8859-1 string to UTF-8	3
xml_error_string()	Gets an error string from the XML parser	3
xml_get_current_byte_index()	Gets the current byte index from the XML parser	3
xml_get_current_column_number()	Gets the current column number from the XML parser	3
xml_get_current_line_number()	Gets the current line number from the XML parser	3
xml_get_error_code()	Gets an error code from the XML parser	3
xml_parse()	Parses an XML document	3
xml_parse_into_struct()	Parse XML data into an array	3
xml_parser_create_ns()	Create an XML parser with namespace support	4
xml_parser_create()	Create an XML parser	3
xml_parser_free()	Free an XML parser	3
xml_parser_get_option()	Get options from an XML parser	3
xml_parser_set_option()	Set options in an XML parser	3
xml_set_character_data_handler()	Set handler function for character data	3
xml_set_default_handler()	Set default handler function	3
xml_set_element_handler()	Set handler function for start and end element of elements	3
xml_set_end_namespace_decl_handler()	Set handler function for the end of namespace declarations	4
xml_set_external_entity_ref_handler()	Set handler function for external entities	3
xml_set_notation_decl_handler()	Set handler function for notation declarations	3
xml_set_object()	Use XML Parser within an object	4
xml_set_processing_instruction_handler()	Set handler function for processing instruction	3
xml_set_start_namespace_decl_handler()	Set handler function for the start of namespace declarations	4
xml_set_unparsed_entity_decl_handler()	Set handler function for unparsed entity declarations	3

b) Using PHP, how can we create a database connection, then a result-set, and finally display the data in an HTML table.

Ans:

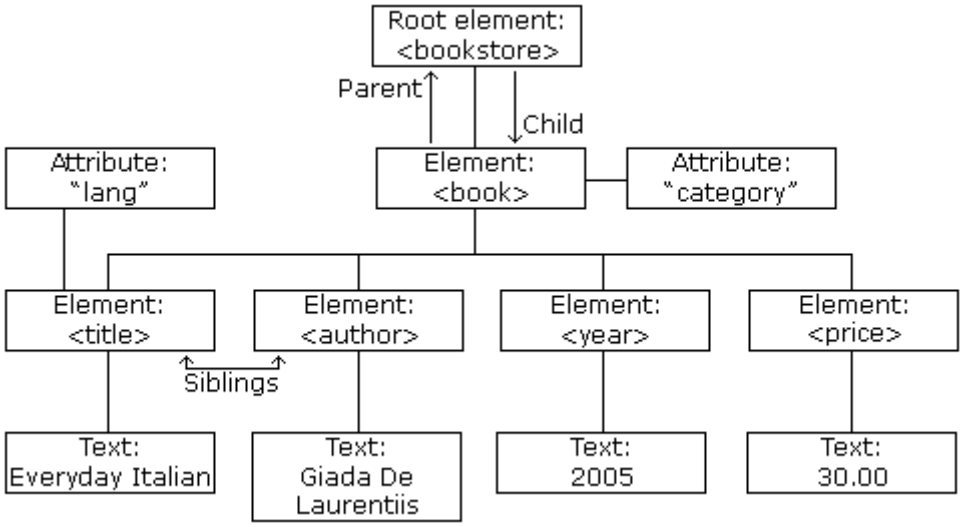
```
<html>
<body>
<?php
$conn=odbc_connect('northwind',",");
if (!$conn)
  {exit("Connection Failed: " . $conn);}
$sql="SELECT * FROM customers";
$rs=odbc_exec($conn,$sql);
if (!$rs)
  {exit("Error in SQL");}
echo "<table><tr>";
echo "<th>Companyname</th>";
echo "<th>Contactname</th></tr>";
while (odbc_fetch_row($rs))
  {
  $compname=odbc_result($rs,"CompanyName");
  $conname=odbc_result($rs,"ContactName");
  echo "<tr><td>$compname</td>";
  echo "<td>$conname</td></tr>";
  }
odbc_close($conn);
echo "</table>";
?>
</body>
</html>
```

Q 9 a) Exemplify DOM to present an XML document as a tree-structure.

Ans: The XML DOM views an XML document as a tree-structure. The tree structure is called a node-tree.

All nodes can be accessed through the tree. Their contents can be modified or deleted, and new elements can be created.

The node tree shows the set of nodes, and the connections between them. The tree starts at the root node and branches out to the text nodes at the lowest level of the tree:

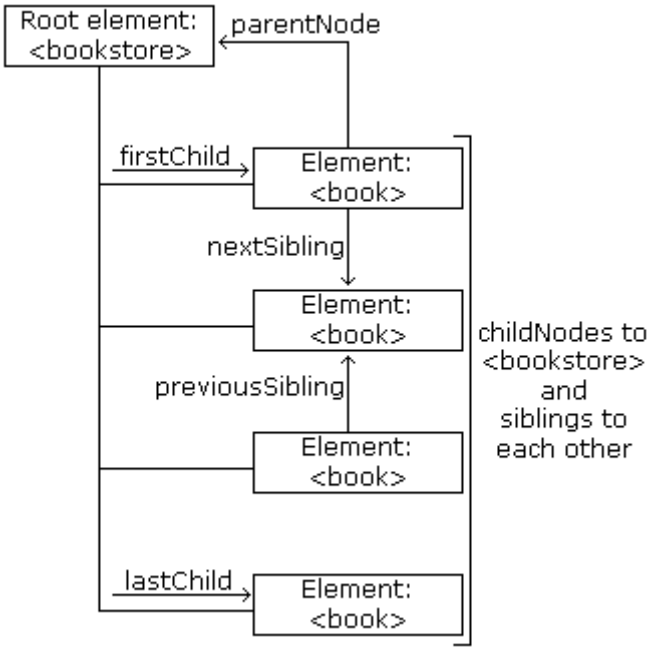


The nodes in the node tree have a hierarchical relationship to each other.

The terms parent, child, and sibling are used to describe the relationships. Parent nodes have children. Children on the same level are called siblings (brothers or sisters).

In a node tree, the top node is called the root
Every node, except the root, has exactly one parent node
A node can have any number of children
A leaf is a node with no children
Siblings are nodes with the same parent

Because the XML data is structured in a tree form, it can be traversed without knowing the exact structure of the tree and without knowing the type of data contained within.



First Child - Last Child
Look at the following XML fragment:

```
<bookstore>
  <book category="cooking">
    <title lang="en">Everyday Italian</title>
    <author>Giada De Laurentiis</author>
    <year>2005</year>
    <price>30.00</price>
  </book>
</bookstore>
```

In the XML above, the <title> element is the first child of the <book> element, and the <price> element is the last child of the <book> element.

Furthermore, the <book> element is the parent node of the <title>, <author>, <year>, and <price> elements

b) Give short notes on:

i. XML Namespaces
ii. PHP printf formatting code

Ans: An Exception With PHP 5 came a new object oriented way of dealing with errors.

Exception handling is used to change the normal flow of the code execution if a specified error (exceptional) condition occurs. This condition is called an exception.

This is what normally happens when an exception is triggered:

- The current code state is saved
- The code execution will switch to a predefined (custom) exception handler function
- Depending on the situation, the handler may then resume the execution from the saved code state, terminate the script execution or continue the script from a different location in the code

We will show different error handling methods:

- Basic use of Exceptions
- Creating a custom exception handler
- Multiple exceptions
- Re-throwing an exception
- Setting a top level exception handler

Note: Exceptions should only be used with error conditions, and should not be used to jump to another place in the code at a specified point

Basic Use of Exceptions

When an exception is thrown, the code following it will not be executed, and PHP will try to find the matching "catch" block.

If an exception is not caught, a fatal error will be issued with an "Uncaught Exception" message.

Lets try to throw an exception without catching it:

```
<?php
//create function with an exception
function checkNum($number)
  {
  if($number>1)
    {
    throw new Exception("Value must be 1 or below");
    }
  return true;
  }
```

```
//trigger exception
checkNum(2);
?>
```

The code above will get an error like this:

```
Fatal error: Uncaught exception 'Exception'
with message 'Value must be 1 or below' in C:\webfolder\test.php:6
Stack trace: #0 C:\webfolder\test.php(12):
checkNum(28) #1 {main} thrown in C:\webfolder\test.php on line 6
```

Try, throw and catch

To avoid the error from the example above, we need to create the proper code to handle an exception.

Proper exception code should include:

1. Try - A function using an exception should be in a "try" block. If the exception does not trigger, the code will continue as normal. However if the exception triggers, an exception is "thrown"
2. Throw - This is how you trigger an exception. Each "throw" must have at least one "catch"
3. Catch - A "catch" block retrieves an exception and creates an object containing the exception information

Lets try to trigger an exception with valid code:

```
<?php
//create function with an exception
function checkNum($number)
  {
  if($number>1)
    {
    throw new Exception("Value must be 1 or below");
    }
  return true;
  }

//trigger exception in a "try" block
try
  {
  checkNum(2);
  //If the exception is thrown, this text will not be shown
```

```
  echo 'If you see this, the number is 1 or below';
  }

//catch exception
catch(Exception $e)
  {
  echo 'Message: ' .$e->getMessage();
  }
?>
```

The code above will get an error like this:

Message: Value must be 1 or below

The code above throws an exception and catches it:

1. The checkNum () function is created. It checks if a number is greater than 1. If it is, an exception is thrown
2. The checkNum () function is called in a "try" block
3. The exception within the checkNum () function is thrown
4. The "catch" block retrives the exception and creates an object ($e) containing the exception information
5. The error message from the exception is echoed by calling $e->getMessage () from the exception object

However, one way to get around the "every throw must have a catch" rule is to set a top-level exception handler to handle errors that slip through.

Creating a custom exception handler is quite simple. We simply create a special class with functions that can be called when an exception occurs in PHP. The class must be an extension of the exception class.

The custom exception class inherits the properties from PHP's exception class and you can add custom functions to it.

Lets create an exception class:

```
<?php
class customException extends Exception
  {
  public function errorMessage()
    {
    //error message
    $errorMsg = 'Error on line '.$this->getLine().' in '.$this->getFile()
```

```
   .': <b>'.$this->getMessage().'</b> is not a valid E-Mail address';
 return $errorMsg;
}
}

$email = "someone@example...com";

try
  {
  //check if
  if(filter_var($email, FILTER_VALIDATE_EMAIL) === FALSE)
    {
    //throw exception if email is not valid
    throw new customException($email);
    }
  }

catch (customException $e)
  {
  //display custom message
  echo $e->errorMessage();
  }
/>
```

The new class is a copy of the old exception class with an addition of the errorMessage() function. Since it is a copy of the old class, and it inherits the properties and methods from the old class, we can use the exception class methods like getLine() and getFile() and getMessage().

The code above throws an exception and catches it with a custom exception class:

1. The customException() class is created as an extension of the old exception class. This way it inherits all methods and properties from the old exception class
2. The errorMessage() function is created. This function returns an error message if an e-mail address is invalid
3. The $email variable is set to a string that is not a valid e-mail address
4. The "try" block is executed and an exception is thrown since the e-mail address is invalid
5. The "catch" block catches the exception and displays the error message

Multiple Exceptions

It is possible for a script to use multiple exceptions to check for multiple conditions.

It is possible to use several if..else blocks, a switch, or nest multiple exceptions. These exceptions can use different exception classes and return different error messages:

```
<?php
class customException extends Exception
{
public function errorMessage()
{
//error message
$errorMsg = 'Error on line '.$this->getLine().' in '.$this->getFile()
.': <b>'.$this->getMessage().'</b> is not a valid E-Mail address';
return $errorMsg;
}
}

$email = "someone@example.com";

try
  {
  //check if
  if(filter_var($email, FILTER_VALIDATE_EMAIL) === FALSE)
    {
    //throw exception if email is not valid
    throw new customException($email);
    }
  //check for "example" in mail address
  if(strpos($email, "example") !== FALSE)
    {
    throw new Exception("$email is an example e-mail");
    }
  }

catch (customException $e)
  {
  echo $e->errorMessage();
  }

catch(Exception $e)
  {
  echo $e->getMessage();
  }
?>
```

Example explained:

The code above tests two conditions and throws an exception if any of the conditions are not met:

1. The customException() class is created as an extension of the old exception class. This way it inherits all methods and properties from the old exception class
2. The errorMessage() function is created. This function returns an error message if an e-mail address is invalid
3. The $email variable is set to a string that is a valid e-mail address, but contains the string "example"
4. The "try" block is executed and an exception is not thrown on the first condition
5. The second condition triggers an exception since the e-mail contains the string "example"
6. The "catch" block catches the exception and displays the correct error message

If there was no customException catch, only the base exception catch, the exception would be handled there

Re-throwing Exceptions

Sometimes, when an exception is thrown, you may wish to handle it differently than the standard way. It is possible to throw an exception a second time within a "catch" block.

A script should hide system errors from users. System errors may be important for the coder, but is of no interest to the user. To make things easier for the user you can re-throw the exception with a user friendly message:

```
<?php
class customException extends Exception
  {
  public function errorMessage()
    {
    //error message
    $errorMsg = $this->getMessage().' is not a valid E-Mail address.';
    return $errorMsg;
    }
  }

$email = "someone@example.com";

try
  {
  try
    {
    //check for "example" in mail address
    if(strpos($email, "example") !== FALSE)
      {
      //throw exception if email is not valid
      throw new Exception($email);
```

```
    }
   }
  catch(Exception $e)
   {
   //re-throw exception
   throw new customException($email);
   }
  }

catch (customException $e)
  {
  //display custom message
  echo $e->errorMessage();
  }
?>
```

The code above tests if the email-address contains the string "example" in it, if it does, the exception is re-thrown:

1. The customException() class is created as an extension of the old exception class. This way it inherits all methods and properties from the old exception class
2. The errorMessage() function is created. This function returns an error message if an e-mail address is invalid
3. The $email variable is set to a string that is a valid e-mail address, but contains the string "example"
4. The "try" block contains another "try" block to make it possible to re-throw the exception
5. The exception is triggered since the e-mail contains the string "example"
6. The "catch" block catches the exception and re-throws a "customException"
7. The "customException" is caught and displays an error message

If the exception is not caught in its current "try" block, it will search for a catch block on "higher levels".

Also refer to Page No. 456 of Textbook

TEXTBOOK

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