

Chapter 1

Server Side Programming

Static vs Dynamic Pages & Client Side vs Server Side Scripting

- The most basic type of Web page is a completely static, text-based one, written entirely in HTML.
- The contents of the HTML file on the server are exactly the same as the source code of the page on the client.

```
<html>
```

```
<title>An Average Website</title>
```

```
<body bgcolor="#003399" text="#ffcc33">
```

```
<h1>An Average Website</h1>
```

```
<p>This is an average website. </p>
```

```
</html>
```

The above HTML code is static.

Static vs Dynamic Pages

- If the user reloads a static website, they would see the exact same content every time.
- Its content was written directly by an author, and when the user goes to the site, that code is downloaded into a browser and interpreted.
- Client-side technologies cannot do anything that requires connecting to a back end server.
- JavaScript cannot assemble a customized drop-down list on the fly from user preferences stored in a database
- If a change is needed in the list, the Web developer must go and edit the page by hand.
- This gap is filled by server-side programming.

Static vs Dynamic Pages

- What does dynamic web page mean?
- A basic distinction exists between static and dynamic Web pages
- But dynamic can mean almost anything beyond plain HTML.
- Web developers use the term to describe both client and server-side functions.
- On the client, it can mean
 - multimedia presentations
 - scrolling headlines
 - pages that update themselves automatically, or
 - elements that appear and disappear.
- On the server, the term generally denotes content assembled on the fly, at the time the page is requested.
- If you display the current date and time on a page, for example, the content will change and thus will be dynamic.

Static vs Dynamic Pages ...

Client-Side Technology	Main Use		Example Effects
Cascading Sheets, Dynamic HTML	Style	Formatting controlling size, color, placement, layout, timing of elements	pages: Overlapping, different colored/sized fonts Layers, exact positioning
Client-side scripting (JavaScript, VBScript)	Event	handling: controlling consequences of defined events	Link that changes color on mouseover Mortgage calculator
Java applets	Delivering	standalone applications	small Moving logo Crossword puzzle
Flash	animations	Animation	Short cartoon

Static vs Dynamic Pages

- In contrast to a static website, a dynamic website is one whose content is regenerated every time a user visits or reloads the site.
- Server-side web scripting is mostly about connecting Web sites to back end servers, such as databases.
- This enables the following types of two-way communication:
 - **Server to client:** Web pages can be assembled from back end-server output.
 - **Client to server:** Customer-entered information can be acted upon.
- Common examples of client-to-server interaction are online forms with some drop-down lists that the script assembles dynamically on the server.

Static vs Dynamic Pages

- There are two ways to generate dynamic content.
- One method is to use scripting languages.
- Popular server side scripting languages are:
 - PHP
 - Servlet
 - Active Server Pages
 - Coldfusion
- The other method is to use CGI(Common Gateway Interface).

Introduction to PHP

- PHP is the web development language written by and for Web developers.
- PHP stands for Hypertext Preprocessor.
- PHP is currently in its fifth major version called PHP 7.0.
- PHP is a server-side scripting language, which can be embedded in HTML or used as a standalone binary.
- Proprietary products in this niche are:
 - Microsoft's Active Server Pages (ASP)
 - Macromedia's ColdFusion, and
 - Sun's Java Server Pages (JSP).
- Over the past few years, PHP and server-side Java have gained momentum, while ASP has lost its share.

Introduction to PHP...

- PHP is a server-side scripting language, which means that the scripts are executed on the server, the computer where the Web site is located.
- This is different than JavaScript, another popular language for dynamic Web sites.
- JavaScript is executed by the browser, on the user's computer.
- Thus, JavaScript is a client-side language.
- Because PHP scripts execute on the server, PHP can dynamically create the HTML code that generates the Web page.
- This allows individual users to see customized Web pages.
- Web page visitors see the output from scripts, but not the scripts themselves.

Introduction to PHP...

- PHP is particularly strong in its ability to interact with databases.
- PHP supports pretty much every database you've ever heard of and some you haven't.
- PHP handles connecting to the database and communicating with it, avoiding the knowledge of technical details for connecting to a database.
- You tell PHP the name of the database and where it is, and PHP handles the details.
- It connects to the database, passes your instructions to the database, and returns the database response to you.
- Major databases currently supported by PHP include the following:
 - dBASE
 - Informix
 - Ingres
 - Microsoft SQL Server
 - mSQL
 - MySQL
 - Oracle
 - PostgreSQL
 - Sybase

Introduction to PHP...

- In addition, PHP supports ODBC (Open Database Connectivity).
- ODBC is a standard that allows you to communicate with even more databases, such as Access and IBM DB2.
- PHP works well for a database-driven Web site.
- PHP scripts in the Web site can store data in and retrieve data from any supported database.
- PHP also can interact with supported databases outside a Web environment.
- Database use is one of PHP's best features.

Software Requirements

- To work with PHP, you need to install a web server, PHP, and database server.
- The most commonly used web server is Apache(free)
- Similarly, for database, there are many options to use.
- The most popular database server for web pages is MySQL (free).
- PHP is also available freely on the internet.
- Software producers integrate the three software and offer them as bundle.
- Some of such bundles are:
 - Vertrigo(Apache, PHP, Mysql)
 - Wamp (Windows) (Apache, Mysql, PHP)
 - Xampp(Apache, Mysql, PHP, Perl)
 - Lamp (Linux)(Apache, PHP, Mysql)
- PHP files are executed through the web server only, not directly
<http://127.0.0.1/test.php>

Writing PHP

- You add PHP code to your web page by using tags, similar to other tags in the HTML file.

- The PHP code section is enclosed in PHP tags with the following form:

```
<?php
```

```
    PHP statements
```

```
?>
```

- For example, you can add the following PHP section to your HTML file.

```
<?php
```

```
echo "This line brought to you by PHP";
```

```
?>
```

- Web pages that contains PHP should be saved with .php extension.
- You can also add several PHP sections to a Web page.

Writing PHP...

- There are actually four different styles of PHP tags we can use.

- Short style

<?

```
echo "<p>Order processed. </p>";
```

?>

- This style of tag is the simplest and follows the style of an SGML.
- To use this tag, you either need to enable short tags in your config file (short_open_tag = On), or compile PHP with short tags enabled.

- XML style

<?php

```
echo "<p>Order processed. </p>";
```

?>

- This style of tag can be used with XML documents.
- Most commonly used tag in literatures

Writing PHP...

- SCRIPT style

```
<SCRIPT LANGUAGE="php">  
    echo "<p>Order processed. </p>";  
</SCRIPT>
```

- This style of tag is the longest and will be familiar if you've used JavaScript.

- ASP style

```
<%  
    echo "<p>Order processed. </p>";  
%>
```

- This style of tag is the same as used in Active Server Pages (ASP).
- It can be used if you have enabled the asp_tags configuration setting.

Writing PHP...

- Example: your first hello world PHP script

```
<html>
<head><title>Hello World Script</title></head>
<body>
<?php
    echo "<p>Hello World!</p>"
?>
</body>
</html>
```

- When the PHP section is processed, it is replaced with the output.
- In this case, the output is as follows:

```
<p>Hello World!</p>
```

Attaching PHP to Web Pages

- PHP file is attached to web pages by using the `<form>` tag.
- Since PHP is written to process data sent from client computers and form is the means to get that data, this is a logical place to attach the PHP to HTML.
- We use the action attribute to specify the PHP to execute when the web page is submitted.
`<form name="formname" method="submittingmethod"
action="phpfile.php">
form elements
</form>`
- Example: to attach a php called `saver.php` to an HTML page
`<form name="detail" method="post" action="saver.php">
form content
</form>`
- When this form is submitted, the web server looks for *saver.php* and executes it.
- The result from executed php is sent back to the client machine.

Output Statements

- The two most basic ways for displaying output in PHP are echo and print.
- Both can be used either with parentheses or without them.
- Function calls always have the name of the function first, followed by a parenthesized list of the arguments to the function.
- The general format of the echo statement is as follows:
`echo outputitem1, outputitem2, outputitem3, . . . ;`
`echo (output);`
- The parameterized version of echo does not accept multiple arguments.
- The general format of the print statement is as follows:
`print output;`
`print(output);`

Output Statements...

- Example: different ways of echo and print

echo 123; //output: 123

echo "Hello World!"; //output: Hello world!

echo ("Hello World!"); //output: Hello world!

echo "Hello","World!"; //output: Hello World!

echo Hello World!; //error, string should be in quotes

print ("Hello world!"); //output: Hello world!

print "Hello world!"; //output: Hello world!

- The command print is very similar to echo, with two important differences:
 - Unlike echo, print can accept only one argument.
 - Unlike echo, print returns a value, which represents whether the print statement succeeded.

Output Statements...

- The value returned by print will be 1 if the printing was successful and 0 if unsuccessful.
- It is rare that a syntactically correct print statement will fail.
- But in theory this return value provides a means to test, for example, if the user's browser has closed the connection.
- It is possible to embed HTML tags in echo or print statements.
- The browser will parse and interpret them like any tag included in HTML page and display the page accordingly.

Output Statements...

echo/print statement	PHP output	web page display
echo "Hello World!";	Hello World!	Hello World!
echo "Hello"; echo "World!";	HelloWorld!	HelloWorld!
echo "Hello\nWorld!";	Hello World!	Hello World!
echo "Hello World!";	Hello World	Hello World!
echo "<u><i>Hello world!</i></u>";	<u><i>Hello world! </i></u>	<u>Hello world!</u>

Comments

- PHP supports two types of comments:
 - single-line comment (short comment), and
 - multi-line comment (long comment).
- The format for multi-line comments is as follows:

```
/* comment text  
   more comment text  
*/
```
- Your comments can be as long or as short as you need.

Comments...

- PHP has a short comment format too.
- You can specify a single line comment by using the # or //
This is comment line 1
// This is comment line 2
- It is customary and useful to put a block of comments at the top of your script giving information about the script and an overview of what it does.

/*

name: hello.php

description: Displays "Hello World!" on a Web page.

written by: John Steve

created: 2/1/11

modified: 3/15/11

*/

1.2 Working with Variables

- In PHP, all variable names should start with a dollar sign (\$).
- This tells PHP that it is a variable name.
- To store information in a variable you use a single equal sign (=).

- For example:

```
$age = 21;
```

```
$price = 20.52;
```

```
$temperature = -5;
```

```
$name = "Clark Kent";
```

```
echo "Your age is $age";
```

1.2 Working with Variables...

- PHP has a total of eight types: integers, doubles, Booleans, strings, arrays, objects, NULL, and resources:
 - *Integers* are whole numbers, without a decimal point, like 495.
 - *Doubles* are floating-point numbers, like 3.14159 or 49.0.
 - *Booleans* have only two possible values: *TRUE* and *FALSE*.
 - *Strings* are sequences of characters, like “*PHP is very interesting*”.
 - *Arrays* are named and indexed collections of other values.
 - *Objects* are instances of programmer-defined classes.
 - *Resources* are special variables that hold references to resources external to PHP such as database connections.
 - *NULL* is a special type that only has one value: *NULL*.

1.2 Working with Variables...

- PHP is a loosely typed language.
- This means that a single variable may contain any type of data.
- This could be a number, a string of text, or some other kind of value, and may change types over its lifetime.
- Example:

```
$testvar = 3 + 4;
```

```
echo "The value is $testvar"; //output: The value is 7
```

```
$testvar = "three";
```

```
echo "The value is $testvar"; //output: The value is  
three
```

1.2 Working with Variables...

Displaying variable values

- You can display the value stored in a variable with print statement. You can output the value of a variable as in the following statements:

```
$today = "Sunday";  
print("The day today is $today");
```
- The output from the preceding statements is *"The day today is Sunday"*.

Removing Variables

- You can uncreate the variable by using this statement:

```
unset($age);
```
- After this statement, the variable \$age no longer exists.
- If you try to echo it, you get an "undefined variable" notice.
- You can unset more than one variable at once, as follows:

```
unset($age, $name, $address);
```

1.2 Working with Variables...

Checking variable content

- Sometimes you just need to know whether a variable exists or what type of data is in the variable.
- Here are some common ways to test variables:
 - `isset($varname)` - true if variable is set, even if nothing is stored in it.
 - `empty($varname)` - true if value is 0 or is a string with no characters in it or is not set.
- You can also test what type of data is in the variable.
- For example, to see if the value is an integer:
 - `is_int($number)` - the comparison is TRUE if the value in `$number` is an integer.

1.2 Working with Variables...

- Some other tests provided by PHP are as follows:
 - `is_integer($var)`: same as `is_int($var)`
 - `is_array($var2)`: Checks to see if `$var2` is an array
 - `is_float($number)`: Checks if `$number` is a floating point number
 - `is_null($var1)`: Checks to see if `$var1` is equal to 0
 - `is_numeric($string)`: Checks to see if `$string` is a numeric string
 - `is_string($string)`: Checks to see if `$string` is a string
 - `is_bool($var)`: finds out whether a variable is a boolean
- You can test for a negative, as well, by using not operator (!) in front of the expression.
- For example, the following statement returns TRUE if the variable does not exist at all:
 - `!isset($varname)`

1.2 Working with Variables...

```
<?php
    $a = "test";
    $b = "anothertest";
    echo isset($a); // TRUE
    echo isset($a, $b); //TRUE
    unset ($a);
    echo isset($a, $b); //FALSE
    $foo = NULL;
    print isset($foo); // FALSE
    $var = 0;
    if (empty($var)) // evaluates TRUE
        echo '$var is either 0 or not set at all';
?>
```

1.2 Working with Variables...

Type casting

- We often work with multiple data types at once.
- Converting one data type to another one is a common task in programming.
- Type conversion or typecasting refers to changing an variable of one data type into another.
- There are two types of conversion.
 - Implicit and
 - Explicit
- Implicit type conversion is an automatic type conversion by the compiler.
- Example:
`echo "45" + 12; //output: 57`
`echo 12 + 12.4; //output: 24.4`

1.2 Working with Variables...

- Explicit conversion happens, when we use the cast constructs.
- There are two ways to do this:
 - Using bool settype (mixed var, string type) function
 - Using cast methods
- The cast methods are:
 - (int), (integer) - cast to integer
 - (bool), (boolean) - cast to boolean
 - (float), (double), (real) - cast to float
 - (string) - cast to string
 - (array) - cast to array
 - (object) - cast to object

1.2 Working with Variables...

- Settype sets the type of a variable.
- The syntax is:
bool **settype** (mixed var, string type)
- Possibles values of *type* are:
 - "boolean" (or, since PHP 4.2.0, "bool")
 - "integer" (or, since PHP 4.2.0, "int")
 - "float" (since PHP 4.2.0, for older versions use "double")
 - "string"
 - "array"
 - "object"
 - "null" (since PHP 4.2.0)

1.2 Working with Variables...

Example:

```
$foo = "321.456number";  
settype($foo, "float");  
print("<br>Float: $foo");    //output: Float: 321.456  
$foo = "321.456number";  
settype($foo, "integer");  
print("<br>Integer: $foo");  //output: Integer: 321
```

```
$foo = 321.456;  
settype($foo, "string");  
print("<br>String: $foo");    //output: String: 321.456
```

```
$foo = "10.7S";           // $foo is string  
$bar = (float) $foo;      // $bar is float  
$val = (int) $foo;        //$val is int
```

1.2 Working with Variables...

Creating Constants

- We use constants to define values that do not change like PI in maths(3.14..), number of days in a week(7), number of minutes in an hour(60), etc.
- Constants are set by using the define statement.
- The general format is as follows:
`define("constantname", "constantvalue");`
- For example, to set a constant with the weather, use the following statement:
`define("PI","3.141");`
- This statement creates a constant called PI and sets its value to "3.141".

1.2 Working with Variables...

- unlike variables, constant names do not begin with a dollar sign (\$).
- By convention, constants are given names that are all uppercase
- so you can see easily that they're constants.
- However, PHP accepts lowercase letters without complaint.
- You can store either a string or a number in a constant.
- Example:
`define ("INTEREST",0.01);`
- Constants should not be given names that are keywords for PHP.
- Keywords are words that have meaning for PHP, such as echo, and they can't be used as constants because PHP treats them as the PHP feature of the same name.

1.2 Working with Variables...

Displaying constants

- You can determine the value of a constant by using print as follows:

```
print(INTEREST);
```

- You can also use a constant in an echo statement:

```
echo INTEREST;
```

- When you echo a constant, you can't enclose it in quotes.
- If you do, it echoes the constant name rather than the value.
- You can build more complicated output statements by using commas, as in the following example:

```
echo "The Canadian exchange rate is $", INTEREST;
```

1.2 Working with Variables...

```
$test_val = 5.5466;  
settype($test_val, "integer");  
echo $test_val;  
output: 5
```

```
$num = 33;  
(boolean) $num;  
echo $num;  
Output: 33
```

```
echo gettype("4");  
Output: String
```

1.3 Using Operators

- PHP supports many operators.

Mathematical Operators

- Arithmetic operators are straightforward—they are just the normal mathematical operators.

Operator	Description
----------	-------------

+	Adds two numbers together.
---	----------------------------

-	Subtracts the second number from the first number.
---	--

*	Multiplies two numbers together.
---	----------------------------------

/	Divides the first number by the second number.
---	--

%	Finds the remainder when the first number is divided by the second number.
---	--

1.3 Using Operators...

- Example: a program that performs all mathematical operation on two numbers

```
$a = 10;
```

```
$b = 20;
```

```
$c = $a + $b; //result: 30
```

```
$c = $a - $b; //result: -10
```

```
$c = $a * $b; //result: 200
```

```
$c = $a / $b; //result: 0.5
```

```
$c = $a % $b; //result: 10
```

1.3 Using Operators...

- You should note that arithmetic operators are usually applied to integers or doubles.
- If you apply them to strings, PHP will try and convert the string to a number.
- If it contains an “e” or an “E”, it will be converted to a double; otherwise it will be converted to an int.
- PHP will look for digits at the start of the string and use those as the value
- If there are none, the value of the string will be zero.

1.3 Using Operators...

Pre- and Post-Increment and Decrement operators

operator	description	example	meaning
++	postfix increment	<code>\$x++</code>	<code>\$x = \$x+1</code>
--	postfix decrement	<code>\$x--</code>	<code>\$x = \$x-1</code>
++	prefix increment	<code>++\$x</code>	<code>\$x = \$x+1</code>
--	prefix decrement	<code>--\$x</code>	<code>\$x = \$x-1</code>

Example: a program that performs postfix and prefix operations

```
$a = 4;  
echo "<br>", ++$a;  
$b = 10;  
echo "<br>", $b++;  
echo "<br>", $b;
```

1.3 Using Operators...

- All the increment operators have two effects—they increment and assign a value.

```
$a=4;
```

```
echo ++$a;
```

- The second line uses the prefix increment operator.
- First, \$a incremented by 1, and second, returns the incremented value.
- In this case, \$a is incremented to 5 and then the value 5 is returned and printed.

- However, if the ++ is after the \$a, we are using the postfix increment operator.
- This has a different effect.

```
$a=4;
```

```
echo $a++;
```

- In this case, first, the value of \$a is returned and printed, and second, it is incremented.
- The value that is printed is 4.
- However, the value of \$a after this statement is executed is 5.

1.3 Using Operators...

Operator	description	example	meaning
<code>+=</code>	add and assign	<code>\$x += 5</code>	<code>\$x = \$x + 5</code>
<code>-=</code>	subtract and assign	<code>\$x -= 5</code>	<code>\$x = \$x - 5</code>
<code>/=</code>	divide and assign	<code>\$x /= 5</code>	<code>\$x = \$x / 5</code>
<code>*=</code>	multiply and assign	<code>\$x *= 5</code>	<code>\$x = \$x * 5</code>
<code>%=</code>	modulate and assign	<code>\$x %= 5</code>	<code>\$x = \$x % 5</code>
<code>.=</code>	concatenate assign	and <code>\$x .= " test"</code>	<code>\$x = \$x." test"</code>

1.3 Using Operators...

Comparison operators

- The comparison operators are used to compare two values.

- Expressions using these operators return either true or false

operator	name	description
==	equal	True if its arguments are equal to each other, false otherwise
!=	not equal	False if its arguments are equal to each other, true otherwise
<	less than	True if the left-hand argument is less than its right-hand argument, but false otherwise
>	greater than	True if the left-hand argument is greater than its right-hand argument, but false otherwise
<=	less or equal	True if the left-hand argument is less than its right-hand argument or equal to it, but false otherwise
>=	greater or equal	True if the left-hand argument is greater than its right-hand argument or equal to it, but false otherwise
===	identical	True if its arguments are equal to each other and of the same type, but false otherwise

1.3 Using Operators...

□ Example:

```
<?php
    $var = "30";
    $num = 30;
    if($var == $num)
        print "They are equal.<br>";
    if($var === $num)
        print "They are equal and same data type.<br>";
    $var = (int) "30";
    if($var === $num)
        print "2. They are equal and same data type.<br>";
?>
```

Output:

They are equal

2. They are equal and same data type.

1.3 Using Operators...

Logical Operators

- The logical operators are used to combine the results of logical conditions.
- For example, we might be interested in a case where the value of a variable is between 0 and 100.
- We would need to test the conditions `$a >= 0` and `$a <= 100`, using the AND operator, as follows:

```
if($a >= 0 && $a <=100)
```

1.3 Using Operators...

operator	description
and	Is true if and only if both of its arguments are true
or	Is true if either (or both) of its arguments are true.
xor	Is true if either (but not both) of its arguments are true
!	Is true if its single argument (to the right) is false and false if its argument is true
&&	Same as and, but binds to its arguments more tightly
	Same as or but binds to its arguments more tightly

1.4 Conditional Statements

1.4.1 *if* Statements

- We can use an if statement to make a decision.
- You should give the if statement a condition that is evaluated to true or false.
- If the condition is true, the associated block of code will be executed.
- Conditions in if statements must be surrounded by brackets ().
- The syntax of if statement:

```
if(condition) {  
  statements;  
}
```
- Example: an if statement that checks if a variable value is less than 50 (student failed course)

```
if($mark<50){  
    print("You have to improve your mark");  
}
```

1.4 Conditional Statements...

if..else statements

- An else statement allows you to define an alternative action to be taken when the condition in an if statement is false.
- This allows to take one action when the condition is true and another action when the condition is false.

- The syntax of if statement:

```
if(condition) {  
statements;  
}  
else {  
statements;  
}
```

1.4 Conditional Statements...

□ Example:

```
if($mark<50) {  
    print("You failed this course");  
}  
else{  
    print("You have passed the course");  
}
```

1.4 Conditional Statements...

else if Statements

- For many of the decisions we make, there are more than two options.
- We can create a sequence of many options using the elseif statement.
- By providing a sequence of conditions, the program can check each until it finds one that is true.
- Example: a program that checks if a variable value is between 1 and 5 and display message accordingly

```
if ($day == 5)
    print("Five golden rings<BR>");
elseif ($day == 4)
    print("Four calling birds<BR>");
elseif ($day == 3)
    print("Three French hens<BR>");
elseif ($day == 2)
    print("Two turtledoves<BR>");
elseif ($day == 1)
    print("A partridge in a pear tree<BR>");
```

1.4 Conditional Statements...

1.4.2 switch statements

- The switch statement works in a similar way to the if statement, but allows the condition to take more than two values.
- In an if statement, the condition can be either true or false.
- In a switch statement, the condition can take any number of different values, as long as it evaluates to a simple type (integer, string, or double).
- You need to provide a case statement to handle each value you want to react to and, optionally, a default case to handle any that you do not provide a specific case statement for.

1.4 Conditional Statements...

```
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  
}
```

- The expression used in a switch statement is often just a variable.
- Within the switch statement's block of code, you find a number of case statements.
- Each of these cases tests a value against the result of the switch expression.
- If the case is equivalent to the expression result, the code after the case statement is executed.

1.4 Conditional Statements...

- The break statement ends the execution of the switch statement altogether.
- If the break statement is left out, the next case statement is executed.
- If the optional default statement is reached, its code is executed.
- It is important to include a break statement at the end of any code that will be executed as part of a case statement.
- Without a break statement, the program flow will continue to the next case statement and ultimately to the default statement.
- In most cases, this will result in unexpected

Example: switch statement that checks the mood of the person

```
<html>
<head>
<title>Listing 5.4</title>
</head>
<body>
<?php
$mood = "happy";
switch ($mood)
{
    case "happy":
        echo "Hooray, I'm in a good mood";
        break;
    case "sad":
        echo "Awww. Don't be down!";
        break;
    default:
        print "Neither happy nor sad but $mood";
        break;
}
?>
</body>
</html>
```

1.5 Using Loops

- Loop statements are designed to enable you to achieve repetitive tasks.
- A loop will continue to operate until a condition is achieved, or you explicitly choose to exit the loop.
- For example, a loop that echoes the names of all the files in a directory needs to repeat until it runs out of files.
- There are three types of loops:
- *for loop*: Sets up a counter; repeats a block of statements until the counter reaches a specified number
- *while loop*: Sets up a condition; checks the condition, and if it's true, repeats a block of statements until the condition becomes false
- *do..while loop*: Sets up a condition; executes a block of statements; checks the condition; if the condition is true, repeats the block of statements until the condition becomes false

1.5 Using Loops...

1.5.1 for loop

- The most basic for loops are based on a counter.
- You set the beginning value for the counter, set the ending value, and set how the counter is incremented/decremented.
- The general format is as follows:

```
for (startingvalue; endingcondition; increment)
{
    block of statements;
}
```

- Example:

```
for ($i = 1; $i <= 3; $i++) {
    echo "$i. Hello World!<br>";
}
```

- You can nest for loops inside of for loops.
- Suppose you want to print out the times tables from 1 to 9.

$1 \times 1 = 1$

...

$1 \times 9 = 9$

$2 \times 1 = 2$

...

$2 \times 9 = 18$

...

$9 \times 1 = 9$

...

$9 \times 9 = 81$

- You can use the following nested loop statements:

```
for($i=1; $i<=9; $i++)  
{  
  for($j=1; $j<=9; $j++)  
  {  
    $result = $i * $j;  
    echo "$i x $j = $result\n";  
  }  
}
```

1.5 Using Loops...

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

```
<html >
<head>
<title>Multiplication Table</title>
</head>
<body>
<h2>Multiplication Table</h2>
<?php
    echo "<table border=\"1\">";
    for ($row=1; $row<=9; $row++)
    {
        echo "<tr>";
        for ($col=1; $col<=9; $col++){
            $x=$col * $row;
            echo "<td>$x</td>";
        }
        echo "</tr>";
    }
    echo "</table>";
?>
</body>
</html>
```

1.5 Using Loops...

1.5.2 while loop

- One of PHP looping construct is while, which has the following syntax:

```
while (condition)
{
    statement
}
```

- The while loop evaluates the condition expression as a Boolean
- If it is true, it executes statement and then starts again.
- If the condition is false, the while loop terminates.
- The body of a while loop may not execute even once, as in:

```
while (FALSE)
    print("This will never print.<BR>");
```

1.5 Using Loops...

- Or it may execute forever, as in this code snippet:

```
while (TRUE)  
print("All work and no play makes Jack a dull boy.<BR>");
```
- or it may execute a predictable number of times, as in:

```
$count = 1;  
while ($count <= 10)  
{  
  print("count is $count<BR>");  
  $count = $count + 1;  
}
```
- which will print exactly 10 lines.

1.5 Using Loops...

1.5.3 do..while loop

- Like a while loop, a do..while loop continues repeating as long as certain conditions are true.
- Unlike while loops, the conditions are tested at the bottom of the loop.
- If the condition is true, the loop repeats.
- When the condition is not true, the loop stops.
- The general format for a do..while loop is as follows:

```
do  
{  
    block of statements  
} while (condition);
```

1.5 Using Loops...

- Example: a loop that displays numbers from 1 to 10

```
$count = 1;  
do  
{  
    print("count is $count<BR>");  
    $count = $count + 1;  
}while ($count <= 10);
```

Avoiding infinite loops

- You can easily set up loops that never stops i.e. which repeats forever.
- These are called infinite loops and it is usually a mistake in the programming.
- This is very stressful on your Web server, and renders the Web page in question unusable.

1.5 Using Loops...

- Example: infinite loop

```
$i = 1;  
while($i < 10)  
{  
    print("i is $i");  
}
```

- output:

```
i is 1  
i is 1  
...
```

- The variable `$i` is not incremented in the body of the while loop.
- As a result, the value of `$i` is always 1 making the while condition, `$i<10`, always true.

1.5 Using Loops...

Breaking out of a loop

- Sometimes you want your script to break out of a loop.
- PHP provides two statements for this purpose:
 - **break**: breaks completely out of a loop and continues with the script statements after the loop.
 - **continue**: stops current iteration and goes back to condition check. If condition check is true, it will go to the next iteration.
- The break and continue statements are usually used in conditional statements.
- In particular, break is used most often in switch statements.

1.5 Using Loops...

Example: break statement

```
$counter = 0;
while ( $counter < 5 ) {
    $counter++;
    if ( $counter == 3 ) {
        echo "break\n";
        break;
    }
    echo "Last line in loop: counter=$counter\n";
}
echo "First line after loop\n\n";
```

The output of this statement is the following:

Last line in loop: counter=1

Last line in loop: counter=2

break

First line after loop

1.5 Using Loops...

Exercise

1. Write a while loop that displays prime numbers between 1 and 100
2. Write a for loop that displays the squared and cubed value of numbers between 0 and 100.
3. Write a do while loop that calculates the factorial of a number.
4. Write a loop that gets the factors of a given number.

1.6 Arrays

- An array is a variable that stores a set or sequence of values.
- One array can have many elements.
- Each element can hold a single value, such as text or numbers, or another array.
- An array containing other arrays is known as a multidimensional array.
- A scalar variable is a named location in which to store a single value
- Similarly, an array is a named place to store a set of values, thereby allowing you to group common scalars.
- The values stored in an array are called the array elements.
- Each array element has an associated index (also called a key) that is used to access the element.
- Arrays in most programming languages have numerical indexes that typically start from zero or one.
- PHP supports this type of array.

1.6 Arrays...

- In addition to numerical indexes, PHP also supports associative arrays.
- Associative arrays can have almost anything as the array indices, but typically use strings.

Creating Arrays

- There are different ways to create an array in a PHP script:
 - by assigning a value into one (and thereby implicitly creating it),
 - by using the array() construct, or
 - by calling a function that happens to return an array as its value.
- The simplest way to create an array is to act as though a variable is already an array and assign a value into it.
- Example: the following code will create an array called \$products by assigning value:

```
$products[0] = "Tires";  
$products[1] = "Oil";  
$products[2] = "Spark Plugs";  
$products[3] = "battery";  
$products[4] = "jar";
```

1.6 Arrays...

- If `$products` array does not already exist, the first line will create a new array with just one element.
- The subsequent lines add values to the array.
- The other way to create an array is via the `array()` construct
- This creates a new array from the specification of its elements and associated keys.
- In its simplest version, `array()` is called with no arguments, which creates a new empty array.
- In its next simplest version, `array()` takes a comma-separated list of elements to be stored, without any specification of keys.
- Elements are stored in the array in the order specified and assigned integer keys beginning with zero.
- For example, the statement:
`$fruit = array("apple", "orange", "banana", "pear");`

1.6 Arrays...

- This causes the variable \$fruit to be assigned to an array with four string elements('apple', 'banana', 'orange', 'pear'), with the indexes 0, 1, 2, and 3, respectively.
- The array will remember the order in which the elements were stored.

Associative Arrays

- PHP also supports associative arrays.
- In an associative array, we can associate any key or index we want with each value.
- An associative array can be viewed as a list of key/value pairs, stored as follows:

```
$arrayname['key1'] = value1;
```

```
$arrayname['key2'] = value2;
```

```
$arrayname['key3'] = value3;
```

1.6 Arrays...

- Initializing an Associative Array

```
$person = array("name" => "Bob", "occupation" =>  
"engineer", "age" => 30, "special power" => "x-ray vision");
```

- The four keys in the \$person array are called *name*, *occupation*, *age*, and *special power*.
- The associated values are *Bob*, *engineer*, *30*, and *x-ray vision*.
- You can reference specific elements of an associative array using the specific key.
- Example: to access associative array element

```
echo $person['occupation']; //output: engineer
```

1.6 Arrays...

Viewing arrays

- You can see the structure and values of an array by using one of two statements
 - `var_dump()`
 - `print_r()`
- The `print_r()` statement gives somewhat less information.
- To display the `$products` array, use the following statement:

```
print_r($products);
```

- This `print_r` statement provides the following output:

```
Array (  
    [0] => Tires  
    [1] => Oil  
    [2] => Spark Plugs  
    [3] => battery  
    [4] => jar
```

1.6 Arrays...

- To get more information, use the following statement:

```
var_dump($products);
```

- This statement gives the following output:

Array(

0 => string 'Tires' (*length=5*)

1 => string 'Oil' (*length=3*)

2 => string 'Spark Plugs' (*length=11*)

3 => string 'battery' (*length=7*)

4 => string 'jar' (*length=3*)

)

1.6 Arrays...

Walking through Array

- It is possible to traverse through an array in different ways.
- One is by using loop together with array index.

```
for($i=0; $i<4; $i++)  
    print("\n$product[$i]);
```

- output:

Tires

Oil

Spark Plugs

battery

Jar

1.6 Arrays...

Using foreach to walk through an array

- You can use foreach to walk through an array one value at a time and execute a block of statements by using each value in the array.
- The general format is as follows:

```
foreach ( $arrayname as $keyname => $valuename)
{
    block of statements;
}
```

- In this format, you need to fill in the following information:
 - arrayname: The name of the array you are walking through.
 - keyname: The name of the variable where you want to store the key. If you leave out \$keyname =>, only the value is stored into \$valuename.
 - valuename: The name of the variable where you want to store the value.

1.6 Arrays...

- Example: foreach loop

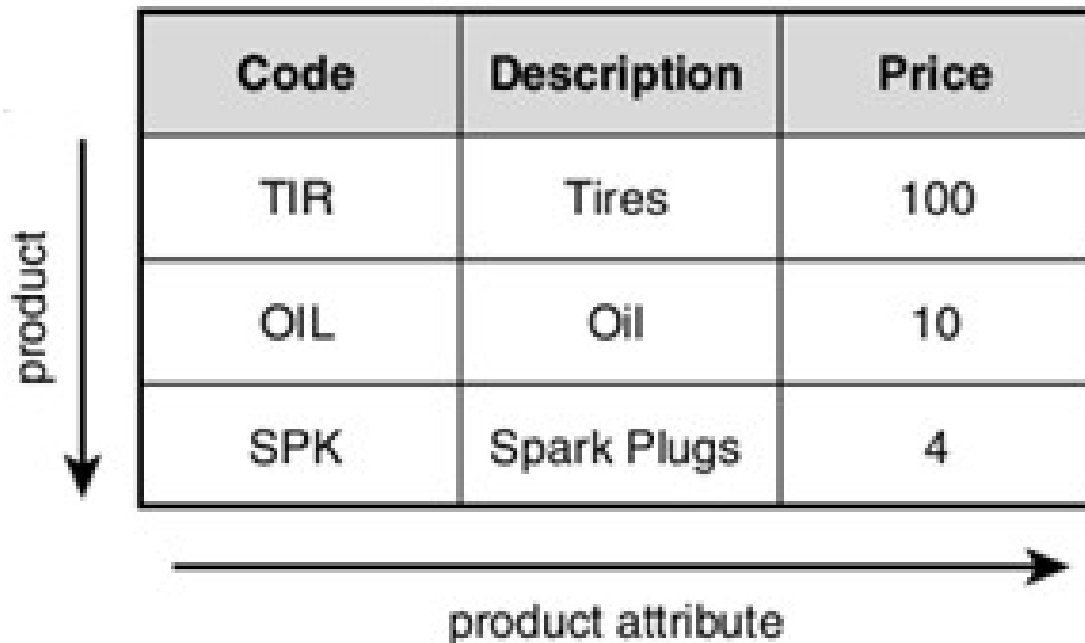
```
$population = array( "Oro"=>25000, "Amh"=>20000,  
    "Tig"=>5000);  
foreach($population as $state => $pop)  
{  
    $pop = number_format($pop);  
    echo "$state: $pop.<br>";  
}
```

Multidimensional Arrays

- Arrays do not have to be a simple list of keys and values—each location in the array can hold another array.
- This way, we can create a two-dimensional array.
- You can think of a two dimensional array as a matrix, or grid, with width and height or rows and columns.

1.6 Arrays...

- If we want to store more than one piece of data about products, we could use a two-dimensional array



Code	Description	Price
TIR	Tires	100
OIL	Oil	10
SPK	Spark Plugs	4

1.6 Arrays...

- Using array() construct:

```
$products = array( array( "TIR", "Tires", 100 ),  
                  array( "OIL", "Oil", 10 ),  
                  array( "SPK", "Spark Plugs", 4 ) );
```

- Using assigning values:

```
$product[0][0] = "TIR";  
$product[0][1] = "Tires";  
$product[0][2] = "100";  
$product[1][0] = "OIL";  
$product[1][1] = "Oil";  
$product[1][2] = "10";  
$product[2][0] = "SPK";  
$product[2][1] = "Spark Plugs";  
$product[2][2] = "4";
```

1.6 Arrays...

Array Functions

Sorting Arrays

- It is often useful to sort data stored in an array.
- Sorting a one-dimensional array it into order is quite easy.
- The following code results in the array being sorted into ascending alphabetical order:

```
$products = array( "Tires", "Oil", "Spark Plugs" );  
sort($products);
```

Output:

Oil, Spark Plugs, Tires.

- Note that the sort function is case sensitive.
- All capital letters come before all lowercase letters.
- So 'A' is less than 'Z', but 'Z' is less than 'a'.

1.6 Arrays...

- Example:

```
$streets[0] = "Elm St.";
```

```
$streets[1] = "Oak Dr.";
```

```
$streets[2] = "7th Ave.";
```

- You enter the following sort statement:

```
sort($streets);
```

- Now the array becomes as follows:

```
$streets[0] = "7th Ave.";
```

```
$streets[1] = "Elm St.";
```

```
$streets[2] = "Oak Dr.";
```

1.6 Arrays...

- If we are using an associative array to store items and their prices, we need to use different kinds of sort functions to keep keys and values together as they are sorted.
- In this case, we use either `asort()` or `ksort()`.
- The function `asort()` orders the array according to the value of each element.
- If the sorting should be done by key instead of value, we use `ksort()`.
- The following code creates an associative array and then sorts the array into ascending price order.

```
$prices = array( "Tires"=>100, "Oil"=>10, "Spark Plugs"=>4 );  
asort($prices);  
$prices = array( "Tires"=>100, "Oil"=>10, "Spark Plugs"=>4 );  
ksort($prices);
```

1.6 Arrays...

- Example:

```
$capitals[1] = "Sacramento";
```

```
$capitals[2] = "Austin";
```

```
$capitals[3] = "Salem";
```

- After an `asort()` statement, the new array would be as follows:

```
$capitals[2] = "Austin"
```

```
$capitals[1] = "Sacramento"
```

```
$capitals[3] = "Salem"
```

1.6 Arrays...

Sort Statement	What It Does
<code>sort(\$arrayname)</code>	Sorts by value; assigns new numbers as index.
<code>rsort(\$arrayname)</code>	Sorts by value in reverse order; assigns new numbers as the keys.
<code>asort(\$arrayname)</code>	Sorts by value; keeps the same key/index.
<code>arsort(\$arrayname)</code>	Sorts by value in reverse order; keeps the same key/index.
<code>krsort(\$arrayname)</code>	Sorts by key.
<code>ksort(\$arrayname)</code>	Sorts by key in reverse order.
<code>natsort(\$arrayname)</code>	Sorts mixed string/number values in natural order. For example, given an array with values day1, day5, day11, day2, it sorts into the following order: day1, day2, day5, day11. The previous sort functions sort the array into this order: day1, day11, day2, day5.

1.6 Arrays...

Determining Size of Array

- You can find out the size of your array by using either the count statement or a sizeof statement.
- The format for these statements is as follows:
`$n = count($arrayname);`
`$n = sizeof($arrayname);`
- After either of these statements, \$n will contain the number of elements in the array.

Built-in Arrays

- PHP has several built-in arrays that you can use when writing PHP scripts.
- Different types of information are stored in built-in arrays.
- For example, information about your server (such as headers, paths, and script locations) is stored in an array called `$_SERVER`

<i>Array</i>	<i>Description</i>
\$GLOBALS	Contains all the global variables. For example, if you use the statement, \$testvar = 1, you can then access the variable as \$GLOBALS ['testvar'].
\$_POST[]	Contains all the variables contained in a form if the form uses method="post".
\$HTTP_POST_VARS	Same as \$_POST.
\$_GET	Contains all the variables passed from a previous page as part of the URL. This includes variables passed in a form using method="get".
\$HTTP_GET_VARS	Same as \$_GET.
\$_COOKIE[]	Contains all the cookie variables.
\$HTTP_COOKIE_VARS	Same as \$_COOKIE.
\$_SESSION	Contains all the session variables.
\$HTTP_SESSION_VARS	Same as \$_SESSION.
\$_REQUEST	Contains all the variables together that are in \$_POST, \$_GET, and \$_SESSION.
\$_FILES	Contains the names of files that have been uploaded.
\$HTTP_FILES_VARS	Same as \$_FILES.
\$_SERVER	Contains information about your server. Because your Web server provides the information, the information that's available depends on what server you're using.
\$HTTP_SERVER_VARS	Same as \$_SERVER.
\$_ENV	Contains information provided by your operating system, such as the operating system name, the system drive, and the path to your temp directory. This info varies depending on your operating system
\$HTTP_ENV_VARS	Same as \$_ENV.

1.6 Arrays...

- The `$_SERVER` and `$_ENV` arrays contain different information, depending on the server and operating system you're using.
- You can see what information is in the arrays for your particular server and operating system by using the following statements:

```
foreach($_SERVER as $key => $value)  
    echo "Key=$key, Value=$value\n";
```

The output includes such lines as the following:

Key=HTTP_HOST, Value=127.0.0.1

Key=HTTP_ACCEPT, Value=text/html,application/xhtml+xml,
application/xml;q=0.9,*/*;q=0.8

Key=HTTP_ACCEPT_LANGUAGE, Value=en-us,en;q=0.5

Key=HTTP_ACCEPT_ENCODING, Value=gzip, deflate

Key=HTTP_ACCEPT_CHARSET, Value=ISO-8859-1,utf-8;q=0.7,*;q=0.7

Key=HTTP_CONNECTION, Value=keep-alive

Key=WINDIR, Value=C:\Windows

Key=SERVER_SIGNATURE, Value=

Key=SERVER_SOFTWARE, Value=Apache/2.2.17 (Win32) PHP/5.3.6

Key=SERVER_ADDR, Value=127.0.0.1

Key=SERVER_PORT, Value=80

Key=REMOTE_ADDR, Value=127.0.0.1

Key=DOCUMENT_ROOT, Value=C:/Program Files (x86)/VertrigoServ/www

Key=SERVER_ADMIN, Value=lata@jal.com

Key=SCRIPT_FILENAME, Value=C:/Program Files (x86)/VertrigoServ/www/environment.php

Key=REMOTE_PORT, Value=2005

Key=GATEWAY_INTERFACE, Value=CGI/1.1

Key=SERVER_PROTOCOL, Value=HTTP/1.1

Key=REQUEST_METHOD, Value=GET

Key=QUERY_STRING, Value=

Key=SCRIPT_NAME, Value=/environment.php

Key=PHP_SELF, Value=/environment.php

1.6 Arrays...

- The `PHP_SELF` element shows the file that contains the script that is currently running.
- You can see the information in the `$_ENV` array by using the `phpinfo()` statement with a 16 to specify the environmental variables, as follows:

```
phpinfo(16);
```

1.7 Functions in PHP

- Applications often perform the same task at different points in the script or in different scripts.
- This is when functions come in handy.
- A function is a group of PHP statements that perform a specific task.
- You can use the function wherever you need to perform the task.

Defining Functions

- You can create a function by putting the code into a function block.
- The general format is as follows:

```
function functionname($argument1, $argument2, ....)
{
    block of statements;
    return value;
}
```

1.7 Functions in PHP...

- Example:

```
function addNumbers($a, $b)
{
    $sum = $a + $b;
    return $sum
}
```

Calling a function

- The following line is the simplest possible call to a function:
`functionName();`
- This calls a function called `functionName` that does not require parameters.
- This line of code ignores any value that might be returned by this function.

1.7 Functions in PHP...

- Most functions do require one or more parameters.
- We pass parameters by placing the data or the name of a variable holding the data inside parentheses after the function name.
- A call to a function with a parameter resembles the following:
`function_name(parameter);`
- Example: other possible calls
`function_name(2);`
`function_name(7.993);`
`function_name($variable);`
- In the last line, `$variable` might be any type of PHP variable, including an array.
- A parameter can be any type of data

1.7 Functions in PHP...

- You can call functions by passing multiple values to the function by putting the values between the parentheses as follows:

```
functionname(value1,value2,...);
```

- Example: to call the above addNumbers function:

```
$result = addNumbers(30,20);
```

Pass by Reference Versus Pass by Value

- If we want to write a function called increment() that allows us to increment a value, we might be tempted to try writing it as follows:

```
function increment($value, $amount)
{
    $value = $value + $amount;
}
```

1.7 Functions in PHP...

- This code will be of no use.
- The output from the following test code will be “10”.

```
$value = 10;  
increment ($value, 1);  
echo $value;
```
- The contents of \$value have not changed because of the scope rules.
- This code creates a variable called \$value which contains 10 & calls the function increment().
- The variable \$value in the function is created when the function is called.
- One is added to it, so the value of \$value is 11 inside the function, until the function ends.
- In calling code, the variable \$value is a different variable, with global scope, and therefore unchanged.

1.7 Functions in PHP...

- The normal way that function parameters are called is called pass by value.
- When you pass a parameter, a new variable is created which contains the value passed in.
- It is a copy of the original.
- You are free to modify this value in any way, but the value of the original variable outside the function remains unchanged.
- The better approach is to use pass by reference.
- When a parameter is passed by reference, rather than creating a new variable, the function receives a reference to the original variable.
- This variable rather than having a value of its own, it merely refers to the original.
- Any modifications made to the reference also affect the original.

1.7 Functions in PHP...

- We specify a pass by reference by placing an ampersand (&) before the parameter name in the function's definition.
- No change is required in the function call.
- The preceding increment() example can be modified to have one parameter passed by reference, and it will work correctly.

```
function increment(&$value, $amount = 1)
{
    $value = $value + $amount;
}
$value = 10;
increment ($value, 1);
echo $value; //prints 11
```

1.7 Functions in PHP...

Variable Scope

- A variable declared within a function remains local to that function.
- It will not be available outside the function or within other functions.

- PHP has fairly simple rules:
 - Variables declared inside a function are in scope from the statement in which they are declared to the closing brace at the end of the function. This is called function scope. These variables are called local variables.
 - Variables declared outside of functions are in scope of the statement in which they are declared to the end of the file, but not inside functions. This is called global scope. These variables are called global variables.
 - Using `require()` and `include()` statements does not affect scope. If the statement is used within a function, function scope applies. If it is not inside a function, global scope applies.
 - The keyword `global` can be used to manually specify that a variable defined or used within a function will have global scope.

1.7 Functions in PHP...

- Example: local variable

```
<?php
$num=50;
function test()
{
    $testvariable = "this is a test variable";
    echo "Num is ", $num; //output: Num is
}
echo "test variable: $testvariable"; //output: test variable:
echo "Num is ", $num; //output: Num is 50
?>
```

- The value of the variable `$testvariable` is not printed.
- This is because no such variable exists outside the `test()` function.
- Similarly, a variable declared outside a function will not automatically be available within it.

1.7 Functions in PHP...

- If you want a variable created within a function to be global, we can use the keyword `global` as follows:

```
function fn()
{
    global $var;
    $var = "contents";
    echo "inside the function, \$var = ".$var."<br>";
}
fn();
echo "outside the function, \$var = ".$var."<br>";
```

- output:

inside the function, \$var = contents

outside the function, \$var = contents

1.7 Functions in PHP...

- In PHP global variables must be declared global inside a function if they are going to be used in that function.

```
<?php
$a = 1;
$b = 2;
function Sum() {
    global $a, $b;
    $b = $a + $b;
}
Sum();
echo $b;
?>
```

- The above script will output "3".
- By declaring \$a and \$b global within the function, all references to either variable will refer to the global version.

1.7 Functions in PHP...

- A second way to access variables from the global scope is to use the special PHP-defined `$GLOBALS` array.
- The previous example can be rewritten as:

```
<?php
$a = 1;
$b = 2;
function Sum()
{
    $GLOBALS["b"] = $GLOBALS["a"] + $GLOBALS["b"];
}
Sum();
echo $b;
?>
```

1.7 Functions in PHP...

□ Exercise

1. Write a function that accepts a number and returns the squared and cubed value of the number.
2. Function that identify the largest number of the given number.
3. That displays number between 150 and 200 that are divisible ten.
4. Write a function that accepts a number as parameter and then calculates the factorial of the number and return it.
5. Write a function that accepts price of a commodity and then calculates VAT tax(15%) and return it.
6. Write a function that calculates compound interest on principal.