BIT 2nd Year
Semester 3
IT 3505

Web Application Development II

Server Side Web Development (PHP & MySQL) –
Part 2
Duration: 30 hours

Instructional Objectives

- Install PHP in a windows environment
- Install PHP in Linux environment
- Explain basic features of PHP
- Articulate MVC architecture
- Differentiate available PHP frameworks
- Explain MVC
- Use web services with PHP
- Develop a web application with PHP

Sub Topics

1.1 Introduction to PHP (Ref 01 Pg:271-278)
1.2. Configuring the environment (Ref 01 Pg : 76 - 85)
1.3. PHP Syntax and Semantics
    1.3.1. Variables (Ref 01 Pg:281-287)
    1.3.2. Constants (Ref 01 pg:287 - 296)
    1.3.3. Conditional Statements (Ref 01 pg:320-335)
    1.3.4. Loops (Ref 01 Pg:335-346)
    1.3.5. Functions (Ref 01 Pg: 346-357)
1.4. Arrays and data processing with arrays (Ref 01 Pg: 296-307)
1.5. Form processing with PHP (Ref 02)
1.6. Session control and Cookies (Ref 01 Pg:437-446)
1.7. File system management (Ref 01 Pg: 366-389)
1.8. Email sending using PHP (Ref 03)
1.9. Object Orientation with PHP (Ref 01 pg :397-423)
1.10. Working with MySQL database (Ref 01 PG:515-528)
1.11. Introduction to PHP frameworks (Ref 5)
1.12. Fundamentals of MVC (Ref 6)
1.13. How to call web service using PHP (Ref 01 pg:541-553)

Data Types

- A Data type can be described as a collection of values and a set of operations over these values.
- The different primitive data types provided by PHP can be classified as
  - **Scalar Types**
  - **Compound Types**
  - **Special Types**

PHP Data Types

- PHP supports the following primitive scalar data types.
  - **Integer**
  - **Floating point (or Double)**
  - **String**
  - **Boolean**

Integer Data Type

- Any number in the set {….,-2,-1,0,1,2,……} is considered as an integer. The maximum and the minimum integer value that can be used in a program are platform dependent.

- The number of bytes allocated to store an integer and the maximum integer value that can be used in your platform can be determined by using the constants PHP_INIT_SIZE and PHP_INT_MAX

- If PHP encounters a number larger than PHP_INT_MAX, the number will be interpreted as a floating point number.

Example :

```
<?php
 echo PHP_INT_SIZE,"\n",PHP_INT_MAX;
?>
```

Integer Data Type .....

- An integer literal can be specified in decimal, octal, hexadecimal or binary.

BNF Definition of integers in PHP

<Integer> : [+-]?<decimal> | [+-]?<hexadecimal> | [+-]?<octal> | [+-]?<binary>

<decimal> : [1-9][0-9]* | 0

<hexadecimal> : 0[xX][0-9a-fA-F]+

<octal> : 0[0-7]+

<Binary> : 0b[01]+

Integer Data Type .....

Examples:

- **1234** // a positive integer in decimal form
- **-123** // a negative integer in decimal form
- **0123** // integer 83 in octal form
- **0x2b1** // integer 689 in hexadecimal form
- **0b01101** // integer 13 in binary form

Floating Point Data Type

- The numbers with decimal points are considered as floating point (double or real) numbers.
- The floating point numbers are represented internally by using IEEE floating point representation. Thus the representations are not exact. Therefor floating point numbers should not compare for equality.
- Floating point literals can be coded in a number of different ways.

Example :

-1.23

1.2e3

34.45E-12

Floating Point Data Type …

BNF Definition of floating point numbers in PHP

<Floating point numbers> : <LNUM> | <DNUM>| <EXPONENT_DNUM>

<LNUM> : [0-9]+

<DNUM> : [0-9]*. <LNUM> | <LNUM> . [0-9]*

<EXPONENT_DNUM> : [+-]?(<LNUM>|<DNUM>) [eE][+-]? <LNUM>

Arithmetic Operators

- The following operators can be applied on both integers and floating point numbers.

Operator	Result
Unary -	Negation of the number
Binary -	Subtraction
+	Addition
*	Multiplication
/	Division
%	Remainder

Arithmetic Operators

Example :

```
<?php
 echo -3,"\t",5 - 3,"\t",5.2*3.4,
        "\t",10/2,"\t",
        10/4,"\t",10%3,"\n";
?>
```

- Note that the division operator returns an integer when the two operands used are integers and numbers are evenly divisible.

String Data Type

- A string comprises of a series of characters.
- The series of characters in a string literal are normally represented either in single quotes or in double quotes.

Example :

```
<?php
 echo "This is a string literal","\n";
 echo 'Another string literal';
?>
```

String Data Type

- A string comprises of a series of characters.
- The series of characters in a string literal are normally represented either in single quotes or in double quotes.

Example :

```
<?php
 echo "This is a string literal \n";
 echo 'Another string literal';
?>
```

Special character sequences

- Certain character sequences are given special meanings in PHP.

Character sequence	Special Meaning
\n	Linefeed
\t	Horizontal tab
\$	Dollar sign
\”	Double quote

Special character sequences

- The special character sequences retain their special meanings only when used inside double quotes.

Example :

```
<?php
 echo 'How the character sequence \n works';
 echo "\n As a line feed";
?>
```

String Operators

Operator	Result
.	String Concatenation

String Operators …..

Example :

```
<?php
 echo "abc" . "def";
?>
```

Boolean Data Type

- The Boolean data type consists of the two Boolean literals "**TRUE**" and "**FALSE**".
- The representation of the Boolean literals is not case sensitive.
- In PHP the following values are also considered as "FALSE"
  - **The integer 0.**
  - **The floating value 0.0**
  - **The empty string "" and ''**
  - **The string "0" and '0'**

Boolean (Logical) Operators

- The following operators can be applied on both integers and floating point numbers.

Operator	Result
and, &&	TRUE when both operands are TRUE
or, ||	TRUE when either operand is TRUE
xor	TRUE when either operand is TRUE, but not both
!	negation

Print() vs echo()

- echo and print both are language constructs, and can be used with or without parentheses:
  - **echo or echo()**
  - **print or print()**
- But , there are some differences :
  - **echo** - can output more than one strings
  - **print** - can output only one string, and returns always 1

```
<?php
    $txt1="Learn PHP";
    $txt2="BIT @ UCSC";
    $Modules=array("HTML5","CSS",
    "JS");

    echo $txt1;
    echo "<br>";
    echo "Study PHP at $txt2";
    echo "I also
    learn{$Modules[0]}";
?>
```

```
<?php
    $txt1="Learn PHP";
    $txt2="BIT @ UCSC";
    $Modules=array("HTML5","CSS",
    "JS");

    print $txt1;
    print "<br>";
    print "Study PHP at $txt2";
    print "I also
    learn{$Modules[0]}";
?>
```

echo command on Boolean values

All parameters of echo command should be of type string. Thus when printing Boolean literals

by using echo will cause Boolean values to be converted into strings prior to printing. When

converting Boolean literals to strings the True Boolean literal is converted to the string “1” and

the “FALSE” literal is converted to the empty string.

Example :

```
<?php
 echo "Result of false or true : ", false or true, "\n";
 echo "Result of false or false : ", false or false, "\n";
?>
```

Variables

- Variables should start by the dollar sign followed by the name of the variable.
- The variable name is case-sensitive.
- A valid variable name starts with a letter or underscore, followed by any number of letters, numbers, or underscores.
  - *$this* is a special variable, thus cannot be used for a different purpose.

Variables

- a, _a, a11, _1, _a1__ are all valid variable names.
- $a, $_a, $a11, $_1, $_a1__ are all valid variables.

The following variables are not valid

- _a does not start with the $ symbol
- $1a_b does not start with a letter or the _ symbols
- $a_# # is not a valid symbol to be used in a variable name

Variables

- The variable names are case-sensitive.
- The variables listed below are different variable:
  - $Name
  - $name
  - $NAME
  - $NAme

Assignment Operator

- PHP assignment operator is “=“.
- Syntax of the assignment operator:

  Variable = expression

Semantic of the assignment operator

- The right hand side expression to “=“ is computed and the result is assigned to the left hand side operand – assign by value
- In PHP assignment statement has a return value, which is the result of the expression on the right hand side.

Assignment Operator

```
<?php
$a =-47;
$_a = “abc”;
$a11 = 2.1e3;
$_1 = ‘xyz’;
$_a1__ = true;
print("$a,$_a,$a11,$_1,$_a1__")
?>
```

Assignment Operator - Examples

```
<?php
$a = 4;
$b = ($c = $a+6) + 4;
echo "$a,$b,$c";
?>
```

Expressions

- Anything that has a value is termed as an expression. This means that any expression in php should have a value.
- The following are examples for expressions
  - 5
  - "abc"
  - 5 + 45
  - 3 + 3.45e2
  - $a = 5

- In the last expression, the value of the expression is 5

Structure of a PHP script

- A PHP script is typically consists of a sequence of statements(instructions).
- Each statement, except the last one, must be terminated with a semicolon.
- PHP is a free format language, thus spaces can be used freely to separate different components in a statement.

Structure of a PHP script …

```
<?php
   $a =-47;
   $_a = "abc";
   $a11 = 3 +
         3.45e2;
   echo("$a,".
      "$_a,".
      "$a11");
?>
```

Operator Precedence and Associativity

Operator	Associativity	Type
**	right	Arithmetic operators
!	right	Logical not
*, /, %	left	Arithmetic operators
+, -, .	left	Arithmetic and String
&&	left	Logical
||	left	Logical
=	right	Assignment
and	left	Logical
xor	left	Logical
or	left	Logical

- Computation of the value of an expression is primarily governed by the precedence and associativity assigned for each operator in the expression.

- Precedence (from highest to lowest) of some of the commonly used operators are listed in the table.

Operator Precedence - Examples

- Precedence of operators is used when evaluating an expression involving operators of different precedence levels.

- 2 ** 3 + 3 * 2
- This expression is evaluated as below
- ((2 ** 3) + (3 * 2)) =(8 + (3 * 2)) = (8+6) = 14

Operator Associativity - Examples

- Associativity of operators is used when evaluating an expression involving operators of the same precedence levels.

2 * 3 * 3 * 2

This expression is evaluated as below

(((2 * 3) * 3) * 2) = ((6 * 3) * 2) = (18 * 2) = 36

* is left associative

Operator Associativity - Examples

- Associativity of operators is used when evaluating an expression involving operators of the same precedence levels.

2 ** 3 ** 2

This expression is evaluated as below

(2 ** (3 ** 2)) = (2** 9) = 512

** is right associative

Operator Precedence and Associativity - Examples

- Consider the following expression:

2 + 3 *4 – 3**2 + 7 * 2

This expression is evaluated as below

2 + 3 *4 – 3**2 + 7 * 2 = 2 + 3 *4 – 9 + 7 * 2 = 2 + 12 – 9 + 7 * 2 = 2 + 12 – 9 + 14

= 14 – 9 + 14 = 5 + 14 = 19

Operator Precedence and Associativity – Changing the default order of computation

The default order of computation can be changed by using parenthesis.

Example

(2 + 3) *4 – 3**2 + 7 * 2

This expression is evaluated as below

5 *4 – 3**2 + 7 * 2 = 5 *4 – 9 + 7 * 2 = 20 – 9 + 7 * 2 = 20 – 9 + 14 = 11 + 14 = 25

PHP Control Structures

Control Structures

- A PHP script is a sequence of statements.
- A Statement typically ends with a semicolon.
- Statements can be grouped into a statement-group by encapsulating a sequence statements with curly braces.
  - A statement-group is also treated as a statement. Thus a statement-group can appear anywhere a statement can appear.
- Control structures define the order of execution of statements in a program.

Conditional Execution : if Structure

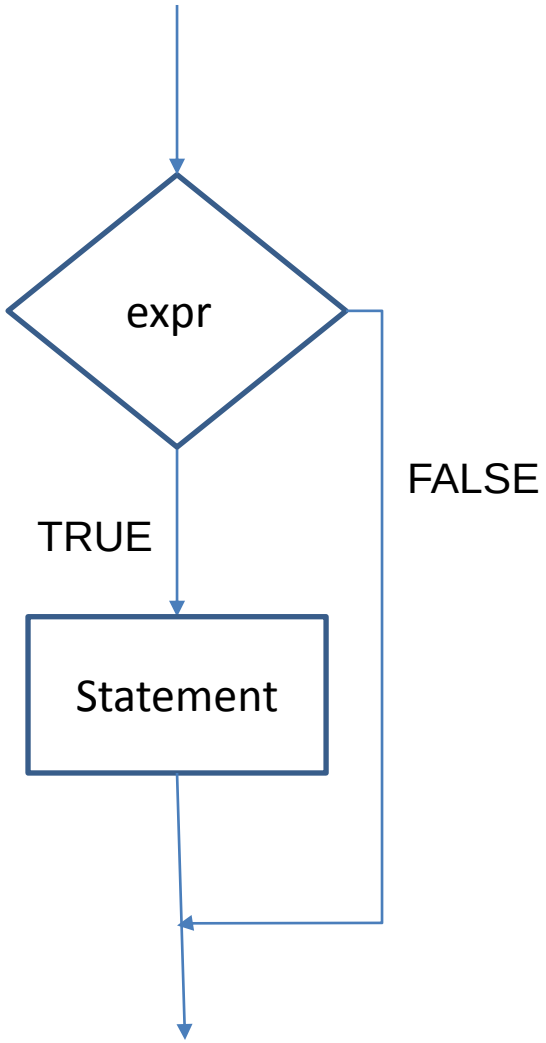
Syntax :

if (*expr*)
  statement

The *expr* must be evaluated to a Boolean value(TRUE or FALSE).

Semantics:

If the value of *expr* is **TRUE** the *statement* is evaluated, else it is ignored.



if Structure : Examples

```
<?php
    $a = 4;
    $b = 3;
    if($a > $b)
        echo "$a is larger then $b";
?>
```

if Structure : Examples

A statement-group can appear anywhere a statement can appear.

```
<?php
   $a = 4;
   $b = 3;
   if($a > $b){
      echo "$a is larger then $b\n";
      $c = $a - $b;
      echo "$a is larger than $b by $c";
   }
?>
```

if Structure : Examples

If structures can be nested.

```
<?php
   $mark1 = 10;
   $mark2 = 80;
   $mark3 = 90;
   if((($mark1+$mark2+$mark3)/3) > 50){
      echo "Pass\n";
      if ($mark1 < 50)
         echo "mark1 needs improvement\n";
      if ($mark2 < 50)
         echo "mark2 needs improvement\n";
      if ($mark3 < 50)
         echo "mark2 needs improvement\n";
   }
?>
```

Conditional Execution : if/else Structure

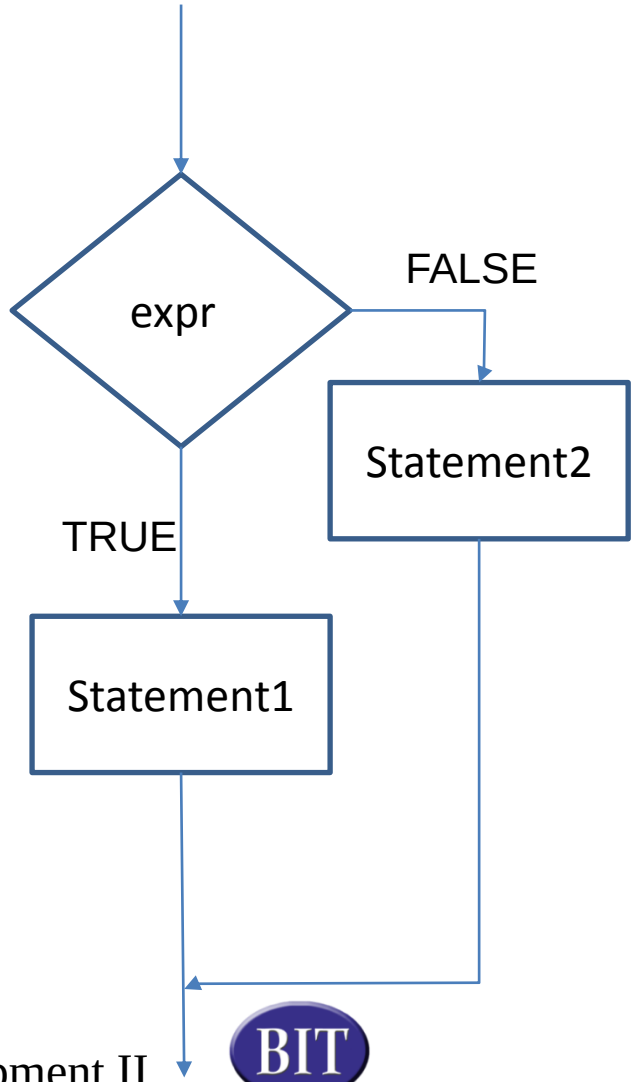
Syntax :

If (*expr*)
  statement1
else
  statement2

The *expr* must be evaluated to a Boolean value(TRUE or FALSE).

Semantics:

If the value of *expr* is **TRUE** the *statement1* is evaluated, else statement2 is evaluated.



if/else Structure : Example

```
<?php
    $marks = 20;
    if($marks > 50)
        echo "Pass\n";
    else
        echo "Fail\n";

?>
```

Nesting if/else Structures : if/elseif/..../else

- If/else structures can be nested by using *if/elseif/else* structure. Any number of *elseif* statements can be included in between *if* and *else* parts.

```
<?php
   $mark = 70;
   if($mark > 70)
      echo "Excellent\n";
   elseif ($mark > 50)
     echo "good\n";
   elseif ($mark > 30)
     echo "pass\n";
   else
     echo "fail\n";
?>
```

Multi-way Branching : switch structure

- The *switch* statement can be considered as an aggregation of a series of IF statements on the same expression.

Syntax:

```
switch (expr){
        case val1 : statement1;
                    break;
        case val2 : statement2;
                    break;

……..
        case valn : statementn;
                    break;
       default : statement;
                    break;
}
```

Semantics:

The *expr* is evaluated first and its value is compared with val_1, val_2,.. , val_n in that order until a match found. If a match found PHP continue to execute the statements associated with the corresponding case statement and the following statements until the end of the *switch* block, or the first *break* statement is encountered.

Multi-way Branching : switch structure ….

- Examples :

```
<?php

switch ($i) {
  case 0 :
        echo “The
value is 0”;
        break;
 case 2 :
        echo “The
value is 2”;
        break;
 case 5 :
        echo “The
value is 5”;
        break;
```

```
<?php

switch ($i) {
  case 0 :
        echo
“statement 1”;
 case 2 :
        echo
“statement 2”;
 case 5 :
        echo
“statement 3”;
}
?>
```

```
<?php

switch ($i) {
  case 0 :
  case 1 :
        echo
“statement 1”;
 case 2 :
 case 3 :
 case 4 :
        echo
“statement 2”;
 case 5 :
        echo
“statement 3”;
```

Looping through statements : while construct

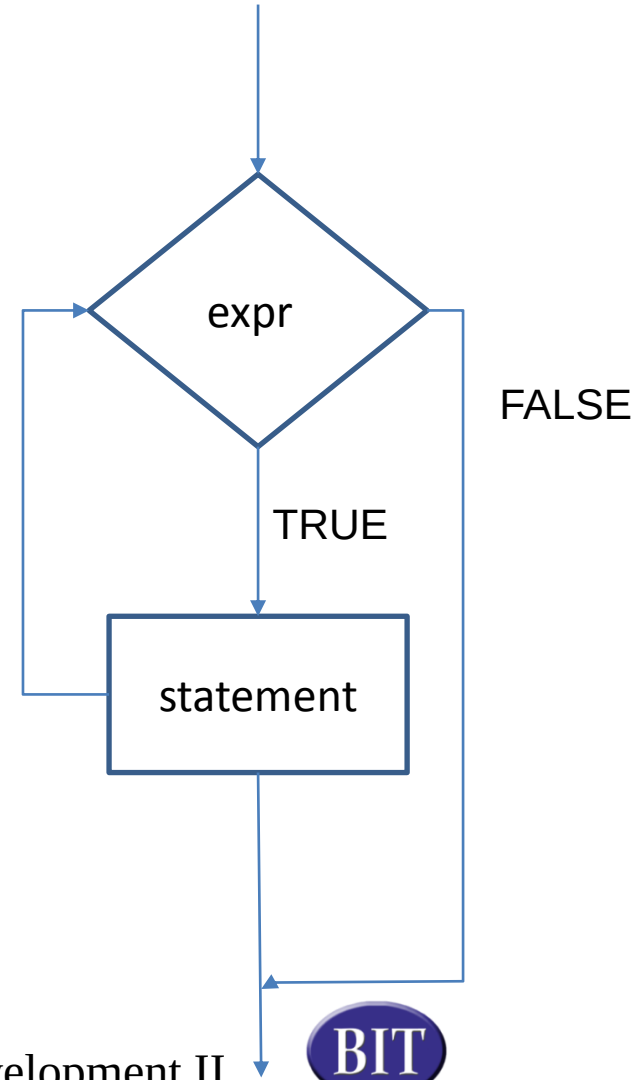
- *while* construct can be used to loop through a statement or a group of statements.

Syntax :

```
while(expr)
  statement
```

Semantics:

Execute the *statement* repeatedly as long as the *expr* evaluates to *TRUE*.



while construct : Example

```
<?php
    $i = 10;
    while ($i>0){
        echo "$i\n";
        $i = $i-1;
    }
?>
```

Looping through statements : do while construct

- *do while* loops are very similar to *while* loops, except the truth expression is checked at the end of each iteration instead of in the beginning.

Syntax :

```
do
   statement
while (expr)
```

do while construct : Example

```
<?php
   $i = 10;
   do {
      echo "$i\n";
      $i = $i-1;
   } while ($i>0)
?>
```

Infinite Loops

- An infinite loop is one in which the while condition never evaluates to the Boolean value “FALSE”. Consequently, the loop iterates forever.
- In certain applications infinite loops are necessary.

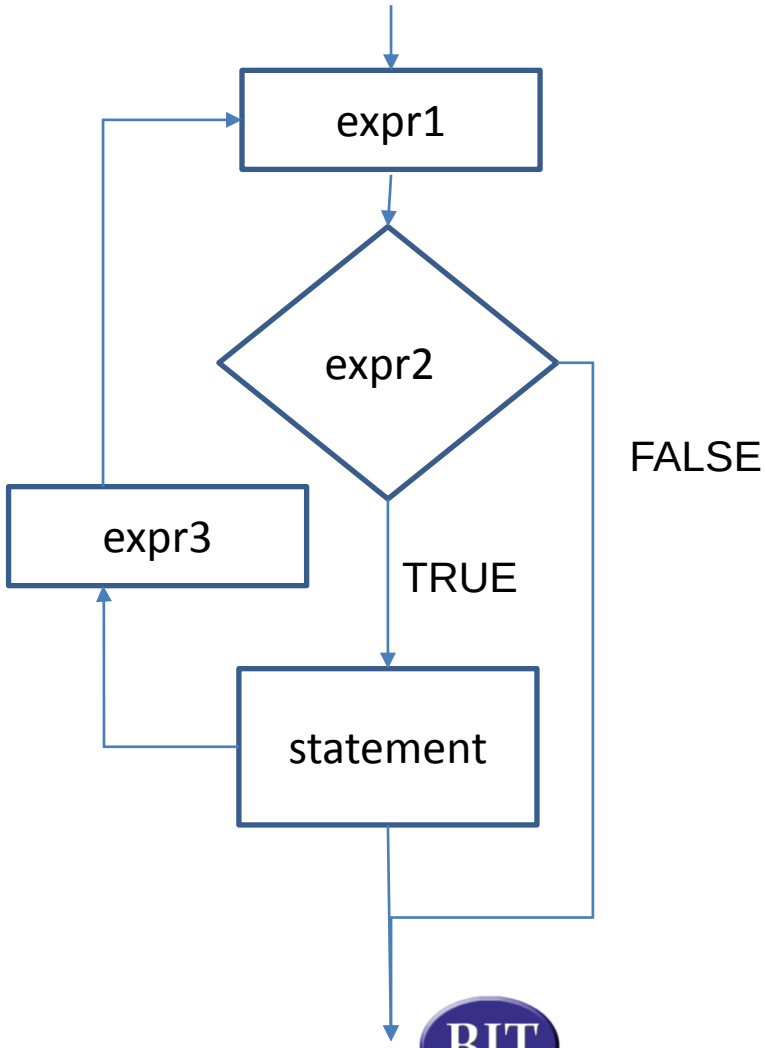
Looping through statements : for construct

Syntax :

for(expr1;expr2;expr3)
  statement

Each of the expressions (*expr1, expr2, expr3*) can be empty or contain multiple expressions separated by commas.

When *expr2* is empty, the Boolean value “TRUE” is assumed.



for construct - Example

```
<?php
    for($i = 10; $i>0; $i = $i-1)
        echo "$i\n";
?>
```

break and continue commands

- The command *break* can be used to terminate an iteration. After executing the *break* command, PHP terminates the current iteration and continues to execute the statements immediately after the iteration block.

Example :

```
<?php
    $i = 10;
    while ($i>0){
        echo "$i\n";
        $i = $i-1;
        if ($i  == 5){
            break;
        }
    }
?>
```

break and continue commands …..

- *continue* is used within a looping structures to skip the rest of the current loop iteration and to continue with the beginning of the next iteration.

Example :

```
<?php
    $i = 10;
    while ($i>0){
        if ($i  == 5){
            continue;
        }
        echo "$i\n";
        $i = $i-1;
}
?>
```