Web Database Application

CIT414 (2 Units)

Dr. B.L. Muhammad-Bello

Objectives

At the end of this unit, you should be able to:

- Describe the GET and POST form methods.
- Explain the functions of the $_GET, $_POST and $_REQUEST variables.
- Distinguish between constants and variables in PHP.
- Describe the types of operators supported by PHP.
- Describe conditional statements for control structures in PHP
- Describe the different ways of looping in PHP

HTML: QUICK REFRESHER

Inside a form -Fields

- Between the <FORM> and </FORM> tags you define the text and *fields* that make up the form.
- There are a variety of types of form fields:
  - text fields: **text, password, textarea**
  - radio buttons
  - checkboxes
  - select
  - buttons: user defined, submit, reset (clear)
  - hidden fields

Input Fields

- Fields that allow the user to type in a string value or upload data file as input.
- Each field is created using an **<INPUT>**tag with the attribute **TYPE**.
- The TYPE attribute is used to specify what kind of input is allowed: **TEXT**, **PASSWORD**, **FILE.**
- Every INPUT tag must have a **NAME** attribute.

Home Work

- Complete this refresher by studying the other Form Inputs:
  - Check Box input
  - Radio Buttons
  - Multiline Text Area
  - Select Option / Pull Down Menu
  - Image Buttons
  - Push Buttons (choice of submit buttons)
  - Input -> type = "file"

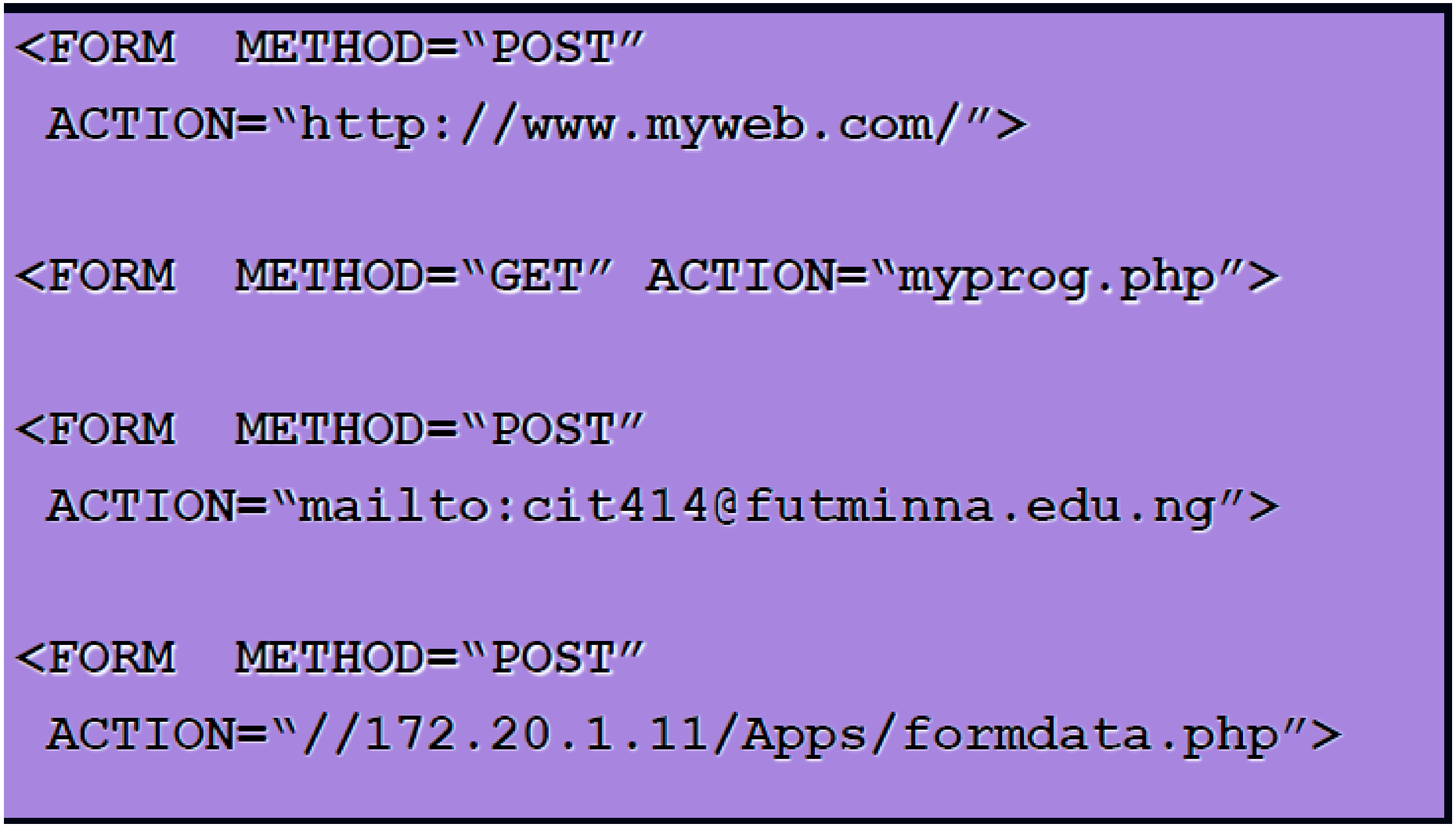
- Also study how the <Table> </Table> tag can be used/embedded within the <Form> </Form> Tags.

Getting User Inputs: Form Elements

- Each HTML form contains the following:
  - <FORM> </FORM> tags
  - The <FORM> tag has two **required** attributes:
- **METHOD:** specifies the HTTP method used to send the request to the server (when the user submits the form).
- **ACTION:** specifies the URL the request is sent to.

- Forms send data to a server for further processing using the method and action specified.

<FORM> Tag Example



```
<FORM  METHOD="POST"
 ACTION="http://www.myweb.com/">

<FORM  METHOD="GET" ACTION="myprog.php">

<FORM  METHOD="POST"
 ACTION="mailto:cit414@futminna.edu.ng">

<FORM  METHOD="POST"
 ACTION="//172.20.1.11/Apps/formdata.php">
```

FORM Methods

- **GET:** any user input is submitted as part of the URL following a "?". in name-value pair.
  - **GET server_address?name-value pairs of form data**
- **POST:** any user input is submitted as the content of the request (after the HTTP headers).
- The PHP $_GET and $_POST variables are used to retrieve information from forms, like user input.
  - The $_GET variable is used to collect values from a form with method="get". While the $_POST variable is used to collect values from a form with method="post".
  - The $_GET variable is an array of variable names and values sent by the HTTP GET method.

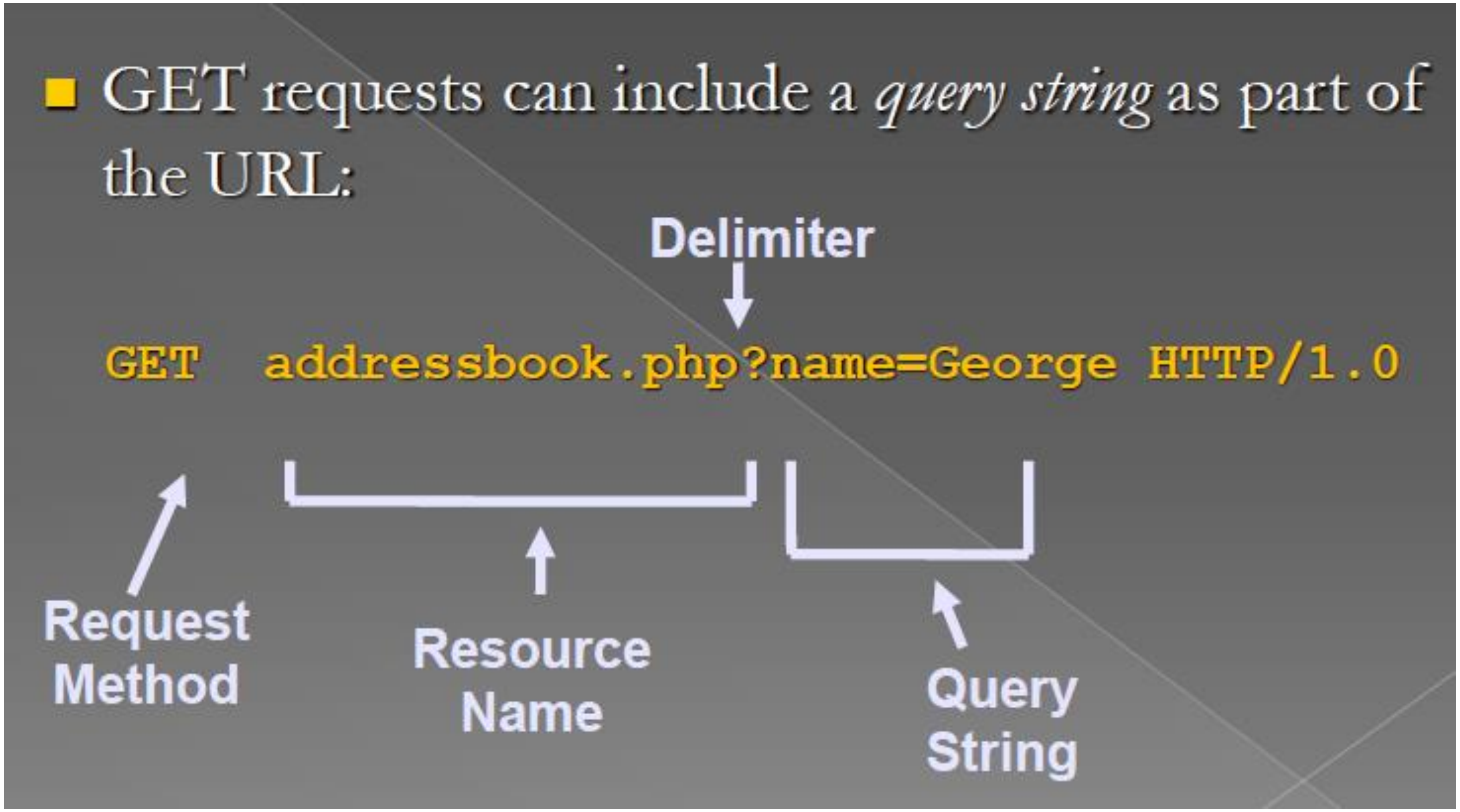
The $_GET Variable

- Example:

  -
    ```
    Welcome <?php echo $_GET["name"]; ?>.<br />
    You are <?php echo $_GET["age"]; ?> years old!
    ```
- When the get method is used, the $_GET variable is used to catch the form data. However, the names of the form fields will automatically be the ID keys in the $_GET array).
- When using the $_GET variable all variable names and values are displayed in the URL. Therefore, this method will not be suitable when sending passwords or other sensitive information.
- However, because the variables are displayed in the URL, it is possible to bookmark the page. This can be useful in some cases.

- **Note:** The HTTP GET method is not suitable on large variable values; the value cannot exceed 100 characters.

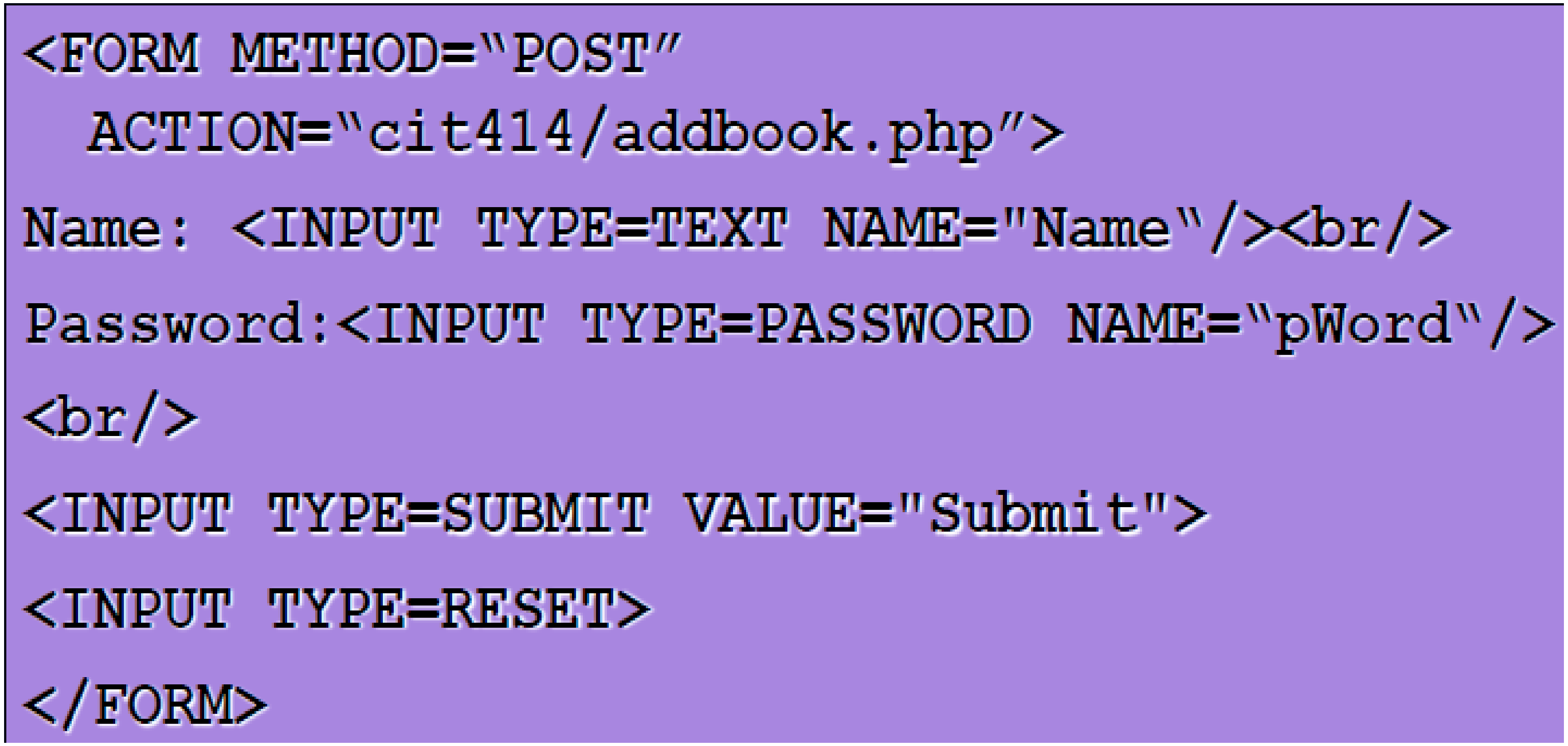
Request Method: Get



- GET requests can include a *query string* as part of the URL:

```
GET  addressbook.php?name=George HTTP/1.0
```

Example: A Complete Form



```
<FORM METHOD=“POST”
  ACTION=“cit414/addbook.php”>
Name: <INPUT TYPE=TEXT NAME="Name“/><br/>
Password:<INPUT TYPE=PASSWORD NAME=“pWord“/>
<br/>
<INPUT TYPE=SUBMIT VALUE="Submit">
<INPUT TYPE=RESET>
</FORM>
```

Form Submission to Server

- When the user clicks on the SUBMIT button within the form the following happens:
- Browser uses the FORM method and action attributes to construct a request.
- A query string is built using the (name,value) pairs from each form element.
- Query string is URL-encoded.

The $_REQUEST Variable

- The PHP $_REQUEST variable contains the contents of both $_GET, $_POST, and $_COOKIE.
- The PHP $_REQUEST variable can be used to get the result from form data sent with both the GET and POST methods.
- Example:
  - `Welcome <?php echo $_REQUEST["name"]; ?>.<br />`
  - `You are <?php echo $_REQUEST["age"]; ?> years old!`

Constants

- A constant is an identifier (name) for a simple value. As the name suggests, that value cannot change during the execution of the script (except for magic constants, which aren't actually constants).
- A constant is case-sensitive by default. By convention, constant identifiers are always **UPPERCASE**.
- Like superglobals, the scope of a constant is global. You can access constants anywhere in your script without regard to scope.
- You can define a constant by using the define()-function or by using the const keyword outside a class definition. Once a constant is defined, it can never be changed or undefined.
- Only scalar data (boolean, integer, float and string) can be contained in constants.
- You can get the value of a constant by simply specifying its name.
- Constants and (global) variables are in a different namespace. This implies that for example **TRUE** and *$TRUE* are generally different.

Distinguishing Constants & Variables

- These are the differences between constants and variables:
  - Constants do not have a dollar sign ($) before them;
  - Constants may only be defined using the define() function or const keyword, not by simple assignment;
  - Constants may be defined and accessed anywhere without regard to variable scoping rules;
  - Constants may not be redefined or undefined once they have been set; and
  - Constants may only evaluate to scalar values.

Defining Constants

- Example 1:

  -
    ```
    <?php
    define("CONSTANT", "Hello world.");
    echo CONSTANT; // outputs "Hello world."
    echo Constant; // outputs "Constant" and issue
    s a notice.
    ?>
    ```
- Example 2:

  -
    ```
    <?php
    // Works as of PHP 5.3.0
    const CONSTANT = 'Hello World';

    echo CONSTANT;
    ?>
    ```

Expressions

- In PHP, almost anything you write is an expression. The simplest yet most accurate way to define an expression is "anything that has a value".
- The most basic forms of expressions are constants and variables. When you type "*$a* = 5", you're assigning '5' into *$a*. '5', obviously, has the value 5, or in other words '5' is an expression with the value of 5 (in this case, '5' is an integer constant).
- After this assignment, you'd expect *$a*'s value to be 5 as well, so if you wrote *$b* = *$a*, you'd expect it to behave just as if you wrote *$b* = 5. In other words, *$a* is an expression with the value of 5 as well. If everything works right, this is exactly what will happen.
- Slightly more complex examples for expressions are functions.

Operators

- **Arithmetic Operators:** +, -, *,/ , %, ++, --
- **Assignment Operators:** =, +=, -=, *=, /=, %=
- **Comparison Operators:** ==, ===, !=, >, <, >=, <=
- **Logical Operators:** &&, ||, !, and, or, xor
- **String Operators:** . .=

Operator Precedence

- The precedence of an operator specifies how "tightly" it binds two expressions together.
- For example, in the expression *1 + 5 * 3*, the answer is *16* and not *18* because the multiplication ("*") operator has a higher precedence than the addition ("+") operator.
- Parentheses may be used to force precedence, if necessary. For instance: *(1 + 5) * 3* evaluates to *18*.
- If operator precedence is equal, left to right associativity is used.

Operator Precedence

Associativity	Operators
non-associative	++ -- < <= > >= <>
right	!
left	* / % + - .
left	&& ||
right	= += -= *= /= .= %= &= |= ^= <<= >>=
left	and or xor
non-associative	== != === !==

Examples

1.
```
<?php
$a = "Hello ";
$b = $a . "World!"; // now $b contains "Hello World!"?>
```
2.
```
<?php
$a = 3 * 3 % 5; // $a=4    ?>
```

3.
```
<?php
 $a = 1;
 $b = 2
 $a = $b += 3; // $a = 5, $b = 5
 ?>
```

4.
```
<?php
 $a = 3;
 $a += 5; // sets $a to 8, as if we had said: $a = $a + 5;
 $b = "Hello ";
 $b .= "There!"; // sets $b to "Hello There!", just like $
b = $b . "There!";
 ?>
```

Control Structures

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

The *If* construct

- The *if* construct is one of the most important features of many languages, PHP included. It allows for conditional execution of code fragments. PHP features an *if* structure that is similar to that of C:
  - `if (expr) statement`
- The expression is evaluated to its Boolean value. If expression evaluates to **TRUE**, PHP will execute statement, and if it evaluates to **FALSE** - it'll ignore it.
  - Example:
    ```
    <?php
    if ($a > $b)
      echo "a is bigger than b";
    ?>
    ```
  - If more than one statement to be executed, you enclose them within curly braces.

Conditional Statements

- Very often when you write code, you want to perform different actions for different decisions.
- You can use conditional statements in your code to do this.
  - **if...else statement** - use this statement if you want to execute a set of code when a condition is true and another if the condition is not true
  - **elseif statement** - is used with the if…else statement to execute a set of code if **one** of several condition are true

The If...Else Statement

- *else* extends an *if* statement to execute a statement in case the expression in the *if* statement evaluates to **FALSE**.
- Syntax:
```
if (condition) {
    Block of code to be executed if condition
    is TRUE;}
    else {
    Block of code to be executed if condition
    is FALSE;   }
```
- Example 1:
```
<?php
    if ($a > $b) {
    echo "a is greater than b"; }
    else {
    echo "a is NOT greater than b";    } ?>
```

elseif/else if

- It extends an *if* statement to execute a different statement in case the original *if* expression evaluates to **FALSE**. However, unlike *else*, it will execute that alternative expression only if the *elseif* conditional expression evaluates to **TRUE**.
- Example:

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

- There may be several *elseif*s within the same *if* statement. The first *elseif* expression (if any) that evaluates to **TRUE** would be executed.

Practice Exercise

```
<html><head></head><body>
<?php
$x="Bilkisu";
if ($x=="Bilkisu")
{
echo "Hello " . $x ." <br/>";
echo "Good morning<br/>";
}

$d=date("D");
if($d=="Fri")
echo "Have a nice weekend! <br/>";
else
echo "Have a nice day! <br/>";
?>
</body></html>
```

Alternative syntax for control structures

- PHP offers an alternative syntax for some of its control structures; namely, *if*, *while*, *for*, *foreach*, and *switch*.
- In each case, the basic form of the alternate syntax is to change the opening brace to a colon (:) and the closing brace to *endif;*, *endwhile;*, *endfor;*, *endforeach;*, or *endswitch;*, respectively.
- Mixing syntaxes in the same control block is not supported.

PHP Looping

- Very often when you write code, you want the same block of code to run a number of times. You can use looping statements in your code to perform this.
- In PHP we have the following looping statements:
  - **while** - loops through a block of code if and as long as a specified condition is true
  - **do...while** - loops through a block of code once, and then repeats the loop as long as a special condition is true
  - **for** - loops through a block of code a specified number of times
  - **foreach** - loops through a block of code for each element in an array

The while Statement: Example

```
<?php
/* example 1 */

$i = 1;
while ($i <= 10) {
    echo $i++;  /* the printed value would be
                   $i before the increment
                   (post-increment) */
}

/* example 2 */

$i = 1;
while ($i <= 10):
    echo $i;
    $i++;
endwhile;
?>
```

do-while statement

- *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.
- The main difference from regular *while* loops is that the first iteration of a *do-while* loop is guaranteed to run (the truth expression is only checked at the end of the iteration)
- Example:

```
Example:<?php
          $i=0;
          do { $i++;
          echo "The number is " . $i . "<br
          />";
          } while ($i<5); ?>
```

The *for* Statement

- The for statement is the most advanced of the loops in PHP.
- In it's simplest form, the for statement is used when you know how many times you want to execute a statement or a list of statements.
  - **Syntax:** `for (expr1; expr2; expr3) statement`
  - **Alternate Syntax:**
    ```
    for (expr1; expr2; expr3):
        statement
        endfor;
    ```
- (*expr1*) is evaluated (executed) once unconditionally at the beginning of the loop.
- In the beginning of each iteration, *expr2* is evaluated. If it evaluates to **TRUE**, the loop continues and the nested statement(s) are executed. If it evaluates to **FALSE**, the execution of the loop ends.
- At the end of each iteration, *expr3* is evaluated (executed).
- Each of the expressions can be empty or contain multiple expressions separated by commas. In *expr2*, all expressions separated by a comma are evaluated but the result is taken from the last part. *expr2* being empty means the loop should be run indefinitely

The *for* Statement: Examples

```
<?php
/* example 1 */
for ($i = 1; $i <= 10; $i++) {    echo $i;
}

/* example 2 */
for ($i = 1; ; $i++) {    if ($i > 10) {
        break;
    }
    echo $i;
}

/* example 3 */
$i = 1;
for ( ; ; ) {    if ($i > 10) {
        break;
    }
    echo $i;
    $i++;
}

/* example 4 */
for ($i = 1, $j = 0; $i <= 10; $j += $i, print $i, $i++);
?>
```

The Switch Statement

- If you want to select one of many blocks of code to be executed, use the Switch statement.
- The switch statement is used to avoid long blocks of if..elseif..else code.
- Syntax: `switch (expression)`

```
{ case label1: code to be executed if expression
= label1;
break;
case label2: code to be executed if expression =
label2;
break;
default: code to be executed if expression is
different from both label1 and label2; }
```

switch structure: Example

-
  ```
  <?php
  if ($i == 0) {
      echo "i equals 0";
  } elseif ($i == 1) {
      echo "i equals 1";
  } elseif ($i == 2) {
      echo "i equals 2";
  }
  ?>
  ```

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

goto operator

- The *goto* operator can be used to jump to another section in the program.
- The target point is specified by a label followed by a colon, and the instruction is given as *goto* followed by the desired target label.
- This is not a full unrestricted *goto*. The target label must be within the same file and context, meaning that you cannot jump out of a function or method, nor can you jump into one.
- You also cannot jump into any sort of loop or switch structure. You may jump out of these, and a common use is to use a *goto* in place of a multi-level *break*.

Example

-
  ```
  <?php
  goto a;
  echo 'Foo';

  a:
  echo 'Bar';
  ?>
  ```
- **Note**: The *goto* operator is available as of PHP 5.3.

The foreach Statement

- The foreach statement is used to loop through arrays.
- For every loop, the value of the current array element is assigned to $value (and the array pointer is moved by one) - so on the next loop, you'll be looking at the next element.
- Syntax 1:`foreach (array_expression as $value) statement`
  Syntax 2: `foreach (array_expression as $key => $value) statement`
- Example:

```
<?php $arr=array("one", "two", "three");
foreach ($arr as $value)
{ echo "Value: " . $value . "<br />"; } ?>
```