



Web Application Development : PHP Data Types, Operators, Functions and Control Structures

Week 2



Content of this Lecture

- Variables and constants
- Data types
- Expressions and operator precedence
- Functions and variable scope
- Control structures

2

Using Variables and Constants

- The values stored in computer memory are called **variables**
- The values, or data, contained in variables are classified into categories known as **data types**
- The name you assign to a variable is called an **identifier** and it
 - must begin with a dollar sign (\$)
 - can include letters (A to Z, a to z) and numbers (0 to 9) or an underscore (_) ... but cannot start with a number
 - cannot include spaces
 - is case sensitive (are \$firstName and \$FirstName referring to the same variable?)
 - should follow a consistent variable naming style (either \$votingAge or \$voting_age or ...)

3

Declaring, Initialising and Modifying Variables

- Specifying and creating a variable name is called **declaring the variable**
- Assigning a first value to a variable is called **initialising the variable**
- In PHP, you must declare and initialise a variable in the same statement:


```
$variable_name = value;
```
- You can change the variable's value at any point


```
$variable_name = new_value;
```

4

Displaying the Values of Variables

- To print the value of a variable, pass the variable name to the `echo` statement without enclosing it in quotation marks:

```
echo $votingAge;
```
- To print both text strings and variables, send them to the `echo` statement as individual arguments, separated by commas:

```
echo "<p>The legal voting age is ",  
    $votingAge, "</p>";
```
- Alternatively, by double/single quotation marks

```
echo "<p>The legal voting age is  
    $votingAge.</p>";
```
- Note, single quotation marks behave differently

```
echo '<p>The legal voting age is  
    $votingAge.</p>';
```

Text '\$votingAge' itself will be printed out.

5

Defining Constants

- A constant contains information that does not change during the course of program execution
- Constant names use all uppercase letters
- You should follow a consistent constant naming style
 - `PASSING_MARK` **or** `PASINGMARK`
- Constant names do not begin with a dollar sign (\$)
 - e.g., `MAX_ELEMENTS` **not** `$MAX_ELEMENTS`
- Use the `define()` function to create a constant
 - `define("CONSTANT_NAME", value);`
- The value you pass to the `define()` function can be a text string, number, or Boolean value
- PHP includes numerous predefined constants that you can use in your scripts

6

Working with Data Types

- A **data type** is the specific category of information that a variable contains
- Data types that can be assigned only a single value are called **primitive types**

Primitive PHP data types

Data Type	Description
Integer numbers	Positive or negative numbers with no decimal places
Floating-point numbers	Positive or negative numbers with decimal places or numbers written using exponential notation
Boolean	A logical value of true or false
String	Text such as "Hello World"
NULL	An empty value, also referred to as a NULL value

7

Working with Data Types (continued)

- **Strongly typed programming languages** require you to declare the data types of variables
 - **Static** or **strong typing** refers to data types that *do not* change after they have been declared
 - C is a strongly typed programming language
- **Loosely typed programming languages** do not require you to declare the data types of variables
 - **Dynamic** or **loose typing** refers to data types that can change after they have been declared
 - PHP is a loosely typed programming language

8

Numeric Data Types

PHP supports two numeric data types:

- An **integer** is a positive or negative number with no decimal places (-250, 2, 100, 10,000)
- A **floating-point** number is a number that contains decimal places or that is written in exponential notation (-6.16, 3.17, 2.7541)
 - **Exponential notation**, or **scientific notation**, is short for writing very large numbers or numbers with many decimal places (2.0e11)

9

Boolean Values

- A **Boolean value** is a value of true or false
- It decides which part of a program should execute and which part should compare data
- In PHP programming, you can only use true or false
- In other programming languages, you can use integers such as 1 = true, 0 = false

10

Working with Data Types (continued)

- The data type of a variable (identifiers) or constant depends on the data type of the value assigned to it

□ `$unitName = "Web Programming";`

□ `$lectureHours = 2;`

□ `$creditPoints = 12.5;`

□ `$isCoreUnit = TRUE;`

Hint: Give meaningful names

Did you notice any naming pattern?

- Are the following correct?

□ `$unitCode = HIT3323;`

Note: HIT3323 may be a constant

□ `$creditPoints = 12.5cp;`

11

Working with Data Types (continued)

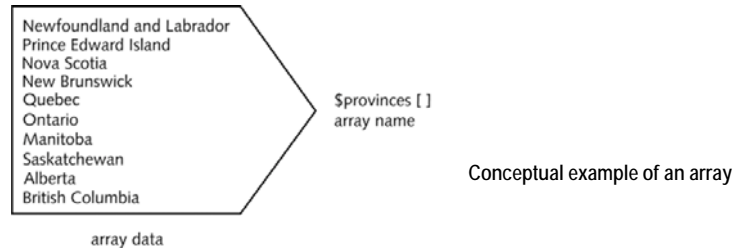
The PHP language supports:

- A **resource** data type – a special variable that holds a reference to an external resource such as a database or XML file
- **Reference** or **composite** data types, which contain multiple values or complex types of information
- Two reference data types: **arrays** and **objects**

12

Arrays

- An **array** contains a set of data represented by a single variable name



- Indexed arrays & associative arrays (to be introduced next week)
- An **element** refers to each piece of data that is stored within an array
- An **index** is an element's numeric position within the array (starts from 0 by default)
- An element is referenced by enclosing its index in brackets at the end of the array name, e.g., `$provinces[1]` // for the second element

13

Creating an Array

- The `array()` construct syntax is:

`$array_name = array(values);`

```
$provinces = array("Newfoundland and Labrador",
    "Prince Edward Island", "Nova Scotia",
    "New Brunswick", "Quebec", "Ontario", "Manitoba",
    "Saskatchewan", "Alberta", "British Columbia");
```

- Array name and brackets syntax is:

`$array_name[]`

```
$provinces[] = "Newfoundland and Labrador";
$provinces[] = "Prince Edward Island";
$provinces[] = "Nova Scotia"; $provinces[] = "New Brunswick";
$provinces[] = "Quebec"; $provinces[] = "Ontario";
$provinces[] = "Manitoba"; $provinces[] = "Saskatchewan";
$provinces[] = "Alberta"; $provinces[] = "British Columbia";
```

Note: In PHP, array elements can be of different data types

14

Accessing/Modifying Element Information

- Accessing an element

```
echo "Canada's largest province is  

    $provinces[4].</p>";
```
- Use the **count ()** function to find the total number of elements in an array

```
echo "<p>Canada has ",  

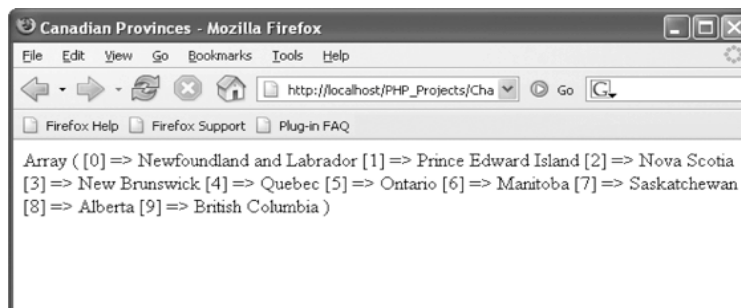
    count($provinces), " provinces.</p>";
```
- Modifying an element

```
$provinces[9] = "BC";
```

15

print_r () Function

- Use to print or return information about variables
- Most useful with arrays because they print the index and value of each element



Output of the `$provinces` array with the `print_r ()` function

16

Building Expressions

- An **expression** is a combination of **operands** and **operators** that are interpreted according to the particular **rules of precedence** and can be evaluated by the PHP scripting engine to produce a result
- **Operands** are literals, **constants**, **variables**, and **functions**
- A **literal** is a value such as a literal string or a number
- **Operators** are symbols (e.g. +, *) that are used in expressions to manipulate operands

17

Building Expressions (continued)

PHP Operator Types

Operator Type	Description
Array	Performs operations on arrays
Arithmetic	Performs mathematical calculations
Assignment	Assigns values to variables
Comparison	Compares operands and returns a Boolean value
Logical	Performs Boolean operations on Boolean operands
Special	Performs various tasks; these operators do not fit within other operator categories
String	Performs operations on strings

- A **binary operator** requires an operand before and after the operator
- A **unary operator** requires a *single* operand either before or after the operator

18

Arithmetic Operators

- **Arithmetic operators** are used in PHP to perform mathematical calculations

PHP arithmetic binary operators

Operator	Name	Description
+	Addition	Adds two operands
-	Subtraction	Subtracts one operand from another operand
*	Multiplication	Multiplies one operand by another operand
/	Division	Divides one operand by another operand
%	Modulus	Divides one operand by another operand and returns the remainder

e.g., the result of $15 / 6$ is 2.5,
and the result of $15 \% 6$ is 3

19

Arithmetic Unary Operators

- The increment (++) and decrement (--) unary operators can be used as prefix or postfix operators
- A **prefix operator** is placed before a variable
- A **postfix operator** is placed after a variable

PHP arithmetic unary operators

Operator	Name	Description
++	Increment	Increases an operand by a value of one
--	Decrement	Decreases an operand by a value of one

20

Prefix Operator vs. Postfix Operator

- Prefix increment operator

```
$ticketNum = 100;
$nextTicketNum = ++$ticketNum; // assigns 101
$nextTicketNum = ++$ticketNum; // assigns 102
```

- Postfix increment operator

```
$ticketNum = 100;
$curTicketNum = $ticketNum++; // assigns 100
$curTicketNum = $ticketNum++; // assigns 101
```

21

Assignment Operators

- **Assignment operators** are used for assigning a value to a variable:

```
$myFavoriteSuperHero = "Superman";
$myFavoriteSuperHero = "Batman";
```

- **Compound assignment operators** perform mathematical calculations on variables and literal values in an expression, and then assign a new value to the left operand

```
$x += $y;  same as  $x = $x + $y;
$x *= $y;  same as  $x = $x * $y;
```

22

Assignment Operators (continued)

PHP assignment operators

Operator	Name	Description
=	Assignment	Assigns the value of the right operand to the left operand
+=	Compound addition assignment	Combines the value of the right operand with the value of the left operand or adds the value of the right operand to the value of the left operand and assigns the new value to the left operand
-=	Compound subtraction assignment	Subtracts the value of the right operand from the value of the left operand and assigns the new value to the left operand
*=	Compound multiplication assignment	Multiplies the value of the right operand by the value of the left operand and assigns the new value to the left operand
/=	Compound division assignment	Divides the value of the left operand by the value of the right operand and assigns the new value to the left operand
%=	Compound modulus assignment	Divides the value of the left operand by the value of the right operand and assigns the remainder (modulus) to the left operand

Comparison Operators

- **Comparison operators** are used to compare two operands and determine how one operand compares to another
- A Boolean value of true or false is returned after two operands are compared
- The comparison operator compares values, whereas the assignment operator assigns values
- Comparison operators are used with **conditional statements** and **looping statements**

Comparison Operators (continued)

PHP comparison operators

Operator	Name	Description
==	Equal	Returns true if the operands are equal
===	Strict equal	Returns true if the operands are equal and of the same type
!= or <>	Not equal	Returns true if the operands are not equal
!==	Strict not equal	Returns true if the operands are not equal or not of the same type
>	Greater than	Returns true if the left operand is greater than the right operand
<	Less than	Returns true if the left operand is less than the right operand
>=	Greater than or equal to	Returns true if the left operand is greater than or equal to the right operand
<=	Less than or equal to	Returns true if the left operand is less than or equal to the right operand

25

Conditional Operators

- The **conditional operator** executes one of two expressions, based on the results of a conditional expression

- The syntax for the conditional operator is:

```
conditional expression
? expression1 : expression2;
```

- If the conditional expression evaluates to true, *expression1* executes; otherwise, *expression2* executes

```
$blackjackPlayer1 = 20;
($blackjackPlayer1 <= 21)
    ? $result = "Player 1 is still in the game."
    : $result = "Player 1 is out of the action.";
/* $result will be assigned "Player 1 is still in the game."
```

26

Logical Operators

- **Logical operators** are used for comparing two Boolean operands for equality
- A Boolean value of true or false is returned after two operands are compared

PHP logical operators

Operator	Name	Description
&&, and	And	Returns true if both the left operand and right operand return a value of true; otherwise, it returns a value of false
, or	Or	Returns true if either the left operand or right operand returns a value of true; if neither operand returns a value of true the expression containing the Or () operator returns a value of false
!	Not	Returns true if an expression is false and returns false if an expression is true

27

Special Operators

PHP special operators

Operator	Description
new	Creates a new instance of a user-defined object type or a predefined PHP object type
[]	Accesses an element of an array
=>	Specifies the index or key of an array element
,	Separates arguments in a list
?:	Executes one of two expressions based on the results of a conditional expression
instanceof	Returns true if an object is of a specified object type
@	Suppresses any errors that might be generated by an expression to which it is prepended (or "placed before")
(int), (integer), (bool), (boolean), (double), (string), (array), (object)	Casts (or transforms) a variable of one data type into a variable of another data type

Note: These Special Operators are introduced throughout this unit as necessary

28

Type Casting

- Casting or type casting copies the value contained in a variable of one data type into a variable of another data type

- The PHP syntax for casting variables is:

```
$newVariable = (new_type) $oldVariable;
```

- (new_type) refers to the type-casting operator representing the type to which you want to cast the variable

```
$speedLimitMiles = "55 mph";  
$speedLimitKilometers = (int) $speedLimitMiles * 1.6;  
echo "$speedLimitMiles is equal to  
    $speedLimitKilometers kph";
```

29

gettype() function

Returns one of the following strings, depending on the data type
– *no guessing needed:*

- ☐ Boolean
- ☐ Integer
- ☐ Double
- ☐ String
- ☐ Array
- ☐ Object
- ☐ Resource
- ☐ NULL
- ☐ Unknown type

30

Understanding Operator Precedence

- **Operator precedence** refers to the order in which operations in an expression are evaluated
- **Associativity** is the order in which operators of equal precedence execute
- Associativity is evaluated on a left-to-right or a right-to-left basis
- What to do if not certain when you write code?

31

Operator Precedence in PHP

Operator precedence in PHP

Operators	Description	Associativity
<code>new</code>	New object—highest precedence	None
<code>[]</code>	Array elements	Right to left
<code>!</code>	Logical Not	Right to left
<code>++</code>	Increment	Right to left
<code>--</code>	Decrement	Right to left
<code>(int), (double), (string), (array), (object)</code>	Cast	Right to left
<code>@</code>	Suppress errors	Right to left
<code>* / %</code>	Multiplication/division/modulus	Left to right
<code>+ - .</code>	Addition/subtraction/string concatenation	Left to right
<code>< <= > >=</code>	Comparison	None
<code>== != <> === !==</code>	Equality	None
<code>&&</code>	Logical And	Left to right
<code> </code>	Logical Or	Left to right
<code>?:</code>	Conditional	Left to right
<code>= += -= *= /= %=</code>	Assignment	Right to left
<code>and</code>	Logical And	Left to right
<code>or</code>	Logical Or	Left to right
<code>,</code>	List separator—lowest precedence	Left to right

32

Defining Functions

- **Functions** are groups of statements that you can execute as a single unit
- **Function definitions** are the lines of code that make up a function

```
<?php
function name_of_function(parameters) {
    statements;
}
?>
```

- A **parameter** is a variable that is used within a function
- Parameters are placed within the parentheses that follow the function name. Functions do not have to contain parameters
- The set of curly braces (called **function braces**) contain the function statements, which do the actual work of the function

33

Defining and Calling Functions

- simplephp example in Week 1

```
function concatenatel($str1, $str2) {
    echo $str1." : ".$str2;
}
concatenatel($name, $pwd);
```

- Use **return statement** that returns a value to the statement that called the function

```
function concatenate2($str1, $str2) {
    return $str1." : ".$str2;
}
echo concatenate2($name, $pwd);
```

34

Understanding Variable Scope

- **Variable scope** is 'where in your program' a declared variable can be used
- A variable's scope can be either global or local
- A **global variable** is one that is declared outside a function and is available to all parts of your program
- A **local variable** is declared inside a function and is only available within the function in which it is declared

35

Understanding Variable Scope (Cont.)

```
<?php
    // all functions usually grouped together
    // in one location
    function testScope() {
        $localVariable = "<p>Local variable</p>";
        echo "<p>$localVariable</p>";
        // prints successfully
    }
    $globalVariable = "Global variable";
    testScope();
    echo "<p>$globalVariable</p>";
    echo "<p>$localVariable</p>"; // error message
?>
```

36

The `global` Keyword

- With many programming languages, global variables are automatically available to all parts of your program including functions.
- In PHP however, we need to use the `global` keyword to declare a global variable in a function where you would like to use it.

```
<?php
function testScope() {
    global $globalVariable;
    echo "<p>$globalVariable</p>";
}
$globalVariable = "Global variable";
testScope();
?>
```

- If no `"global"` above, an error message will be printed out.

37

Using Autoglobals

- PHP includes various predefined global arrays, called **autoglobals** or **superglobals**
- Autoglobals contain client, server, and environment information that you can use in your scripts
- Autoglobals are **associative arrays** – arrays whose elements are referred to with an alphanumeric key instead of an index number

See *Predefined Variables, Superglobals and examples:*
<http://php.net/manual/en/reserved.variables.php>

38

PHP Autoglobals

<i>Array</i>	<i>Description</i>
<code>\$GLOBALS</code>	An array of references to all variables that are defined with global scope
<code>\$_GET</code>	An array of values from the submitted form with the GET method
<code>\$_POST</code>	An array of values from the submitted form with the POST method
<code>\$_COOKIE</code>	An array of values passed to HTTP cookies
<code>\$_SESSION</code>	An array of session variables that are available to the current script
<code>\$_SERVER</code>	An array of information about this script's web server
<code>\$_FILES</code>	An array of information about uploaded files
<code>\$_ENV</code>	An array of environment information
<code>\$_REQUEST</code>	An array of all the elements found in the <code>\$_COOKIE</code> , <code>\$_GET</code> , and <code>\$_POST</code> array

39

Using Autoglobals (continued)

- Use the `global` keyword to declare a global variable within the scope of a function
- Use the `$GLOBALS` autoglobal to refer to the global version of a variable from inside a function
- `$_GET` is the default method for submitting a form
- `$_GET` and `$_POST` allow you to access the values of forms that are submitted to a PHP script
- `$_GET` appends form data as one long string to the URL specified by the action attribute
- `$_POST` sends form data as a transmission separate from the URL specified by the action attribute

40

Using Autoglobals (continued)

```
echo "This script was executed with the following  
server software: ", $_SERVER["SERVER_SOFTWARE"],  
"<br />";  
  
echo "This script was executed with the following  
server protocol: ", $_SERVER["SERVER_PROTOCOL"],  
"<br />";  
  
echo $_GET["name"];  
echo $_GET["address"];  
echo $_POST["name"];  
echo $_POST["address"];
```

41

Making Decisions

- **Decision making or flow control** is the process of determining the order in which statements execute in a program
- The special types of PHP statements used for making decisions are called **decision-making statements** or **decision-making structures**
- There are three types of decision-making statements
 - if statement
 - if...else statement
 - switch statement

42

if Statement

- Used to execute specific programming code if the evaluation of a conditional expression returns a value of **true**

```
if (conditional expression)
    statement; // or {statements}
```

- example

```
$exampleVar = 5;
if ($exampleVar == 5) { // Condition evaluates to 'TRUE'
    echo "<p>The condition evaluates to true.</p>";
    echo '<p>$exampleVar is equal to ',
        "$exampleVar.</p>";
    echo "<p>Each of these lines will be printed.</p>";
}
echo "<p>This statement always executes after if.</p>";
```

43

if...else Statement

- An if statement that includes an else clause. The else clause executes when the condition in an if...else statement evaluates to **false**

```
if (conditional expression)
    statement; // or {statements}
else
    statement; // or {statements}
```

- Example

```
$today = "Tuesday";
if ($today == "Monday")
    echo "<p>Today is Monday</p>";
else
    echo "<p>Today is not Monday</p>";
```

- Multiple if ...else statements can be used together:

```
if...elseif...else
```

44

Nested `if` and `if...else` Statements

- When one decision-making statement is contained within another decision-making statement, they are referred to as nested **decision-making structures**
- Properly nest!

```
if ($_GET["SalesTotal"] > 50)
    if ($_GET["SalesTotal"] < 100)
        echo "<p>The sales total is between 50 and 100.</p>";
    else
        echo "<p>The sales total is no less than 100.</p>";
else
    echo "<p>The sales total is no more than 50.</p>";
```

45

`switch` Statement

- Controls program flow by executing a specific set of statements depending on the value of an expression
- Compares the value of an expression to a value contained within a special statement called a **case label**
- A **case label** is a specific value that contains one or more statements that execute if the value of the case label matches the value of the switch statement's expression

```
switch (expression) {
    case label:
        statement(s);
        break;
    case label:
        statement(s);
        break;
    ...
    default:
        statement(s);
}
```

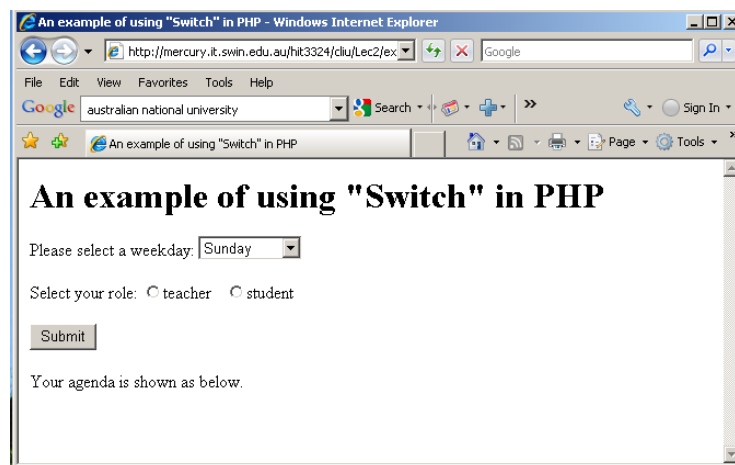
46

switch Statement (continued)

- A `case` label consists of:
 - ☐ The keyword `case`
 - ☐ A literal value or variable name (e.g. "Boston", 75, \$Var)
 - ☐ A colon
- A `case` label can be followed by a single statement or multiple statements
- Multiple statements for a `case` label do not need to be enclosed within a command block
- The **default** label contains statements that execute when the value returned by the `switch` statement expression does not match a `case` label

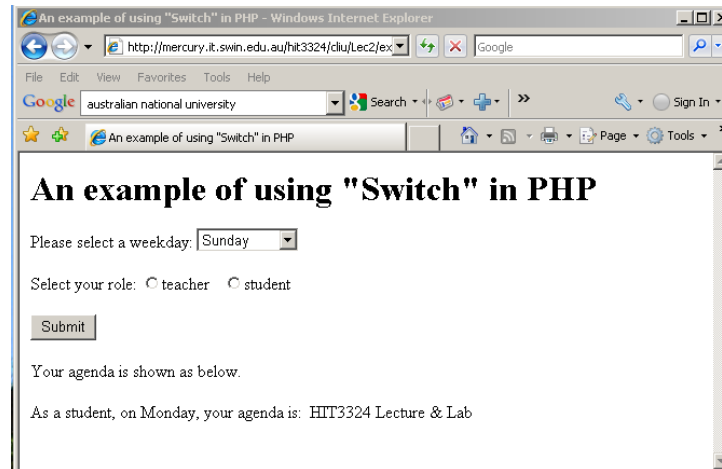
47

Example – What Should I Do Today?



48

Your Agenda is ...



49

Implement the Interface?

```
<HTML xmlns="http://www.w3.org/1999/xhtml">
<head> <title>An example of using "Switch"</title></head>
<body>
<H1>An example of using "Switch" in PHP</H1>
<form> Please select a weekday:
  <select name="weekday" >
    <option value="Sunday">Sunday</option>
    <option value="Monday">Monday</option>
    <option value="Tuesday">Tuesday</option>
    <option value="Wednesday">Wednesday</option>
    <option value="Thursday">Thursday</option>
    <option value="Friday">Friday</option>
    <option value="Saturday">Saturday</option>
  </select> <br/><br/>
  Select your role:
  <input type="radio" name="role" value="teacher">teacher &nbsp;
  <input type="radio" name="role" value="student">student
  <br/><br/>
  <input type="submit" value="Submit" />
</form>
<p> Your agenda is shown as below.</p>
</body>
```

50

Deliver the Agenda

```
<?php
if(isset($_GET['weekday']) && isset($_GET['role'])){
    $weekday = $_GET['weekday']; $role = $_GET['role'];
    echo "As a $role, on $weekday, your agenda is: &nbsp;";
    if($role == 'student'){ // a student's agenda
        switch($weekday) {
            case 'Monday': echo 'HIT3324 Lecture & Lab'; break;
            case 'Wednesday': echo 'HIT3324 Lab'; break;
            default: echo 'Study harder';
        } // end of switch
    }
} else { // a teacher's agenda
    switch($weekday) {
        case 'Monday': echo 'HIT3324 Lecture & Lab'; break;
        case 'Wednesday': echo 'HIT3324 Lab'; break;
        case 'Tuesday':
        case 'Thursday':
        case 'Friday': echo 'Research'; break;
        default: echo 'Have a rest';
    } // end of switch
} // end of inner if
} // end of outer if
?>
</HTML>
```

51

Repeating Code

- A **loop statement** is a control structure that repeatedly executes a statement or a series of statements while a specific condition is true or until a specific condition becomes true
- There are four types of loop statements:
 - while statements
 - do...while statements
 - for statements
 - foreach statements

52

while Statement

- Repeats a statement or a series of statements as long as a given conditional expression evaluates to true

```
while (conditional expression) {
    statement(s);
}
```

- Each repetition of a looping statement is called an **iteration**
- A **while** statement keeps repeating until its conditional expression evaluates to false
- A **counter** is a variable that increments or decrements with each iteration of a loop statement

53

Counter in while Statement

- **Using an increment operator**

```
$count = 1;
while ($count <= 5) {
    echo "$count<br />";
    $count++;
} // print 1,2,3,4,5 in 5 lines
```

- **Using a decrement operator**

```
$count = 10;
while ($count > 0) {
    echo "$count<br />";
    $count--;
} // print 10,9,8,7,6,5,4,3,2,1 in 10 lines
```

- **Using the assignment operator *=**

```
$count = 1;
while ($count <= 100) {
    echo "$count<br />";
    $count *= 2;
} // print 1,2,4,8,16,32,64 in 7 lines
```

54

Be Careful with Infinite Loop

- In an **infinite loop**, a loop statement never ends because its conditional expression is never false

```
$count = 1;
while ($count <= 10) {
    echo "The number is $count";
}
```

- Properly use `continue` and `break` statements

```
$count = 1;
while (true) {
    echo "The number is $count";
    if ($count > 10) break;
    $count++;
    if ($count <= 5) echo "print the number again $count";
    else continue;
}
```

55

do...while Statement

- Executes a statement or statements once, then repeats the execution as long as a given conditional expression evaluates to true

```
do {
    statement(s);
} while (conditional expression);
```

- example

```
$daysOfWeek = array("Monday", "Tuesday", "Wednesday",
    "Thursday", "Friday", "Saturday", "Sunday");
$count = 0;
do {
    echo $daysOfWeek[$count], "<br />";
    $count++;
} while ($count < 7); // print "Monday" to "Sunday" in 7 lines
```

56

for Statement

- Used for repeating a statement or a series of statements as long as a given conditional expression evaluates to true
- Can also include code that initialises a counter and changes its value with each iteration

```
for (counter declaration and initialisation;
      condition; update statement) {
    statement(s);
}
```

- Example

```
$fastFoods = array("pizza", "burgers", "french fries",
                  "tacos", "fried chicken");

for ($count = 0; $count < 5; $count++) {
    echo $fastFoods[$count], "<br />";
} // print "pizza" to "fried chicken" in 5 lines
```

57

foreach Statement

- Used to iterate or loop through the elements in an array
- Does not require a counter; instead, you specify an array expression within the pair of parentheses following the `foreach` keyword

```
foreach ($array_name as $variable_name) {
    statements;
}
```

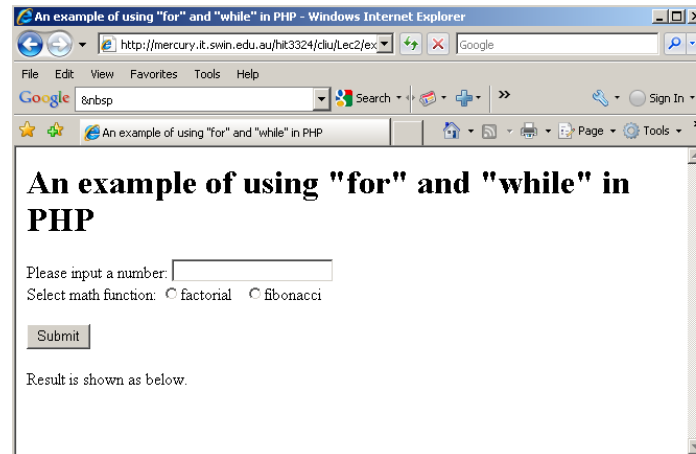
- example

```
$daysOfWeek = array("Monday", "Tuesday", "Wednesday",
                    "Thursday", "Friday", "Saturday", "Sunday");

foreach ($daysOfWeek as $day) {
    echo "<p>$day</p>";
}
```

58

Example – Factorial and Fibonacci



An example of using "for" and "while" in PHP

Please input a number:

Select math function: ☐ factorial ☐ fibonacci

Result is shown as below.

59

Factorial



An example of using "for" and "while" in PHP

Please input a number:

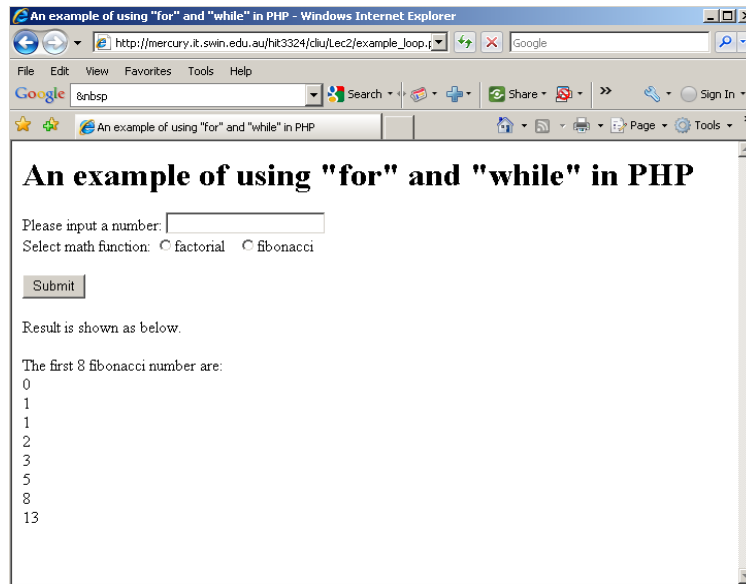
Select math function: ☐ factorial ☐ fibonacci

Result is shown as below.

The result of factorial(8) is : 40320

60

Fibonacci



61

Working on Factorial

```
function factorial($number)
{
    if($number > 170)
        return 0; // this function can only calculate
                  // the factorial of a number below 171

    $i = 1;
    $result = 1;
    while ($i <= $number){
        $result = $result * $i;
        $i++;
    }
    return $result;
}
```

62

Working on Fibonacci

```
function fibonacci($number)
{
    $first = 0; $second = 1;
    if($number >= 2) {
        echo "The first $number fibonacci number are: <br/>";
        echo $first.'<br/>';
        echo $second.'<br/>';
        for($i=1; $i<=$number-2; $i++){
            $temp = $first + $second;
            $first = $second;
            $second = $temp;
            echo $temp.'<br/>';
        }
    }
    elseif ($number==1)
        echo "The first fibonacci number is 0: <br/>";
    else
        echo 'Wrong input number';
}
```

63