Creating PHP Pages

Chapter 7

PHP Decisions Making

Basic Decision Making

- **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**

Basic Decision Making

- Decision making involves evaluating Boolean expressions (true / false)
- If($catishungry) { /* feed cat */ }
- “TRUE” and “FALSE” are reserved words
- Initialise as $valid = false;
- Compare with ==
- AND and OR for combinations
  - E.g. if($catishungry AND $havefood) {/* feed cat*/}

if **Statement**

- Used to execute specific programming code if the evaluation of a conditional expression returns a value of TRUE
- The syntax for a simple `if` statement is: **`if`** `(conditional expression) statement;`
- Contains three parts:
  - the keyword `if`
  - a conditional expression enclosed within parentheses
  - the executable statements

if **Statement**

- A command block is a group of statements contained within a set of braces
- Each command block must have an opening brace ({) and a closing brace (})
- Simple example :

```
if ($a < $b) {
print “$a is less than $b”;
}
else {
print “$b is less than $a”;
}
```

if .. else **Statement**

- An `if` statement that includes an `else` clause is called an `if...else` statement
- An `else` clause executes when the condition in an `if...else` statement evaluates to `FALSE`
- The syntax for an `if...else` statement is :

```
if (conditional expression)
statement;
else
statement;
```

if .. else **Statement**

- An `if` statement can be constructed without the `else` clause
- The `else` clause can only be used with an if statement

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

Nested if **and** if .. else **Statement**

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

```
if ($SalesTotal >= 50)
   if ($SalesTotal <= 100)
         echo " <p>The sales total is
between 50 and 100, inclusive.</p> ";
```

switch **Statement**

- Control program flow by executing a specific set of statements depending on the value of an expression
- Compare 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 **Statement**

- Consist of the following components :
  - The `switch` keyword
  - An expression
  - An opening brace
  - One or more `case` labels
  - The executable statements
  - The `break` keyword
  - A `default` label
  - A closing brace

switch **Statement**

- The syntax for the `switch` statement is :

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

switch **Statement**

- A `case` label consists of:
  - The keyword case
  - A literal value or variable name
  - 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

switch **Statement**

- The default label contains statements that execute when the value returned by the `switch` statement expression does not match a `case` label
- A `default` label consists of the keyword `default` followed by a colon (`:`)

Loop Statement

- 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

while **Statement**

- Tests the condition prior to executing the series of statements at each iteration of the loop
- The syntax for the `while` statement is:

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

- As long as the conditional expression evaluates to `TRUE`, the statement or command block that follows executes repeatedly

while **Statement**

- 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
- In an infinite loop, a loop statement never ends because its conditional expression is never `FALSE`

```
$Count = 1;
while ($Count <= 10) {

}
```

do .. while **Statement**

- Test the condition after executing a series of statements then repeats the execution as long as a given conditional expression evaluates to `TRUE`
- The syntax for the `do...while` statement is :

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

do .. while **Statement**

- `do...while` statements always execute once, before a conditional expression is evaluated

```
$Count = 2;
do {
    echo " <p>The count is equal to
    $Count</p> ";
    ++$Count;
    } while ($Count < 2);
```

for **Statement**

- Combine the initialize, conditional evaluation, and update portions of a loop into a single statement
- Repeat a statement or a series of statements as long as a given conditional expression evaluates to `TRUE`
- If the conditional expression evaluates to `TRUE`, the `for` statement executes and continues to execute repeatedly until the conditional expression evaluates to `FALSE`

for **Statement**

- Combine the initialize, conditional evaluation, and update portions of a loop into a single statement
- Repeat a statement or a series of statements as long as a given conditional expression evaluates to `TRUE`
- If the conditional expression evaluates to `TRUE`, the `for` statement executes and continues to execute repeatedly until the conditional expression evaluates to `FALSE`

foreach **Statement**

- Used to iterate or loop through the elements in an array
- Do not require a counter; instead, you specify an array expression within a set of parentheses following the `foreach` keyword
- The syntax for the `foreach` statement is :

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

foreach **Statement**

- Example :

```
$DaysOfWeek = array(("Monday", "Tuesday",
"Wednesday", "Thursday", "Friday",
"Saturday", "Sunday");
foreach ($DaysOfWeek as $Day) {
echo "<p>$Day</p>";
}
```

for **Statement**

- Combine the initialize, conditional evaluation, and update portions of a loop into a single statement
- Repeat a statement or a series of statements as long as a given conditional expression evaluates to `TRUE`
- If the conditional expression evaluates to `TRUE`, the `for` statement executes and continues to execute repeatedly until the conditional expression evaluates to `FALSE`

Creating PHP Pages

Chapter 6

PHP Variables, Constants and Operators

PHP Variables

- Variables are not statically typed
- Integers can become floats can become strings
- Variable types include :
  - Boolean
  - Integer
  - Float
  - String
  - Array
  - Object
  - Resource
  - NULL

PHP Variables

- Assigned by value

  ```
  $foo = "Bob"; $bar = $foo;
  ```

- Assigned by reference, this links vars

  ```
  $bar = &$foo;
  ```

- Some are preassigned, server and env vars

  For example, there are PHP vars, eg.

  `PHP_SELF`, `HTTP_GET_VARS`

PHP Constants

- Constants are special variables that cannot be changed
- Constant names do not begin with a dollar sign ($)
- Use them for named items that will not change
- Constant names use all uppercase letters
- 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 Operators

- Standard Arithmetic operators

  `+, -, *, / and % (modulus)`

- String concatenation with a period (.)

  `$car = “SEAT” . “ Altea”;`

  `echo $car; would output “SEAT Altea”`

- Basic Boolean comparison with “==”

  `Using only = will overwrite a variable value`

  `Less than < and greater than >`

  `<= and >= as above but include equality`

PHP Operators

- Assignment (=) and combined assignment

```
$a = 3;
$a += 5; // sets $a to 8;
$b = "Hello ";
$b .= "There!"; // sets $b to "Hello There!";
```

- Bitwise (&, |, ^, ~, <<, >>)

```
$a ^ $b(Xor: Bits that are set in $a or $b but not both are set.)
~ $a (Not: Bits that are set in $a are not set, and vice versa.)
```

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` Boolean values
- In other programming languages, you can use integers such as 1 = `TRUE`, 0 = `FALSE`

Creating PHP Pages

Chapter 8

PHP Functions

PHP Functions

- Functions are groups of statements that you can execute as a single unit
- PHP functions need to be defined with keyword function
- It can have zero or more values (parameters)
- Functions may or may not return values
- If a function need to return value, the last statement of the function should be return
  `return value;`
- The set of curly braces (called function braces) contain the function statements

PHP Functions

- Function declaration in PHP

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

for e.g.

```
function sayHello() {
echo("<b>hello<b><br />");
}
```

PHP Functions

- Parameter less function

```
<?php
function hello()
{
echo “hi”;
}
?>
```

- This can be called as <?php hello(); ?> in the program

PHP Functions

- Parameterized function

```
<?php
function greet($name)
{
echo "Hello " . $name;
}
?>
```

- This can be called <?php greet('You');?>
  This gives an output Hello You

PHP Functions

- Assigning functions to the variables

  for e.g

```
$first = "first_func";
to invoke the function first_func() through
the variable
$first();
```

- When an argument is to be passed by reference, an ampersand (&) is placed before the parameter name

  for e.g.

```
first_func(&$first_ref);
```

Returning Values

- A return statement returns a value to the statement that called the function
- Not all functions return values

```
function averageNumbers($a, $b, $c) {
$SumOfNumbers = $a + $b + $c;
$Result = $SumOfNumbers / 3;
return $Result;
}
```

Returning Values

- You can pass a function parameter by value or by reference
- A function parameter that is passed by value is a local copy of the variable.
- A function parameter that is passed by reference is a reference to the original variable.

global **Keyword**

- In PHP, you must declare a global variable with the `global` keyword inside a function definition to make the variable available within the scope of that function
- Example :

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