COMP721 Web Development

AUT UNIVERSITY

Week 2: PHP
Part 1

PHP vs Java?

Why float is not precise?

Euclidian algorithm in PHP?

Note: we will introduce CSS and Bootstrap in week 4…

Misc

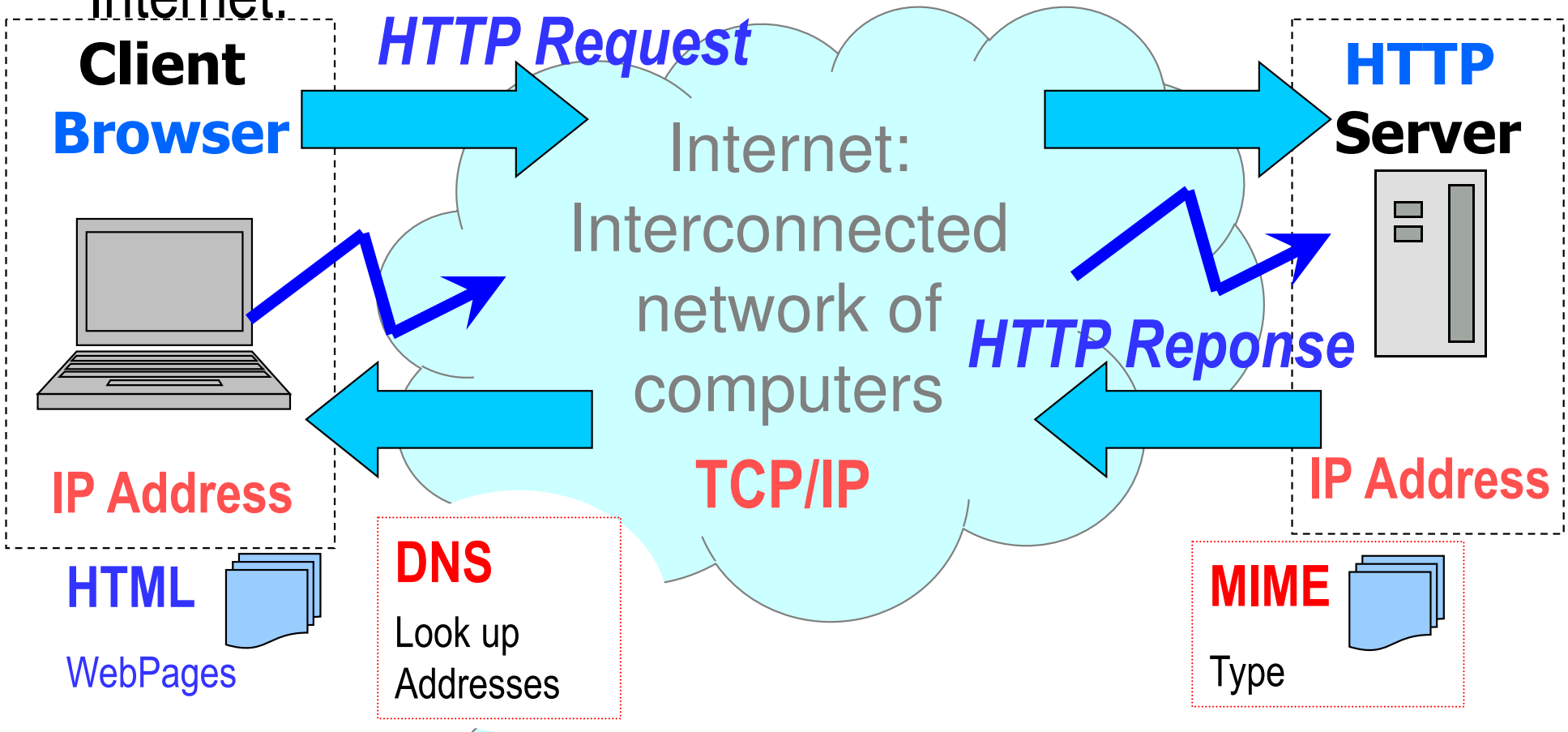
- Work on your labs and assignments at home
  - Xampp
    - https://www.apachefriends.org/index.html
    - Portable version (for Windows) on BB – resources section
    - XAMPP is the most popular PHP development environment
    - XAMPP is a completely free, easy to install Apache distribution containing MariaDB, PHP, and Perl. The XAMPP open source package has been setWork on your labs and assignments at home
- RASS might work for AUT cmslamp14 server …

Agenda

- Week 1 review
- Basic PHP scripts: get it running!
- PHP datatypes and operators
  - Why float/double is not precise? (converting decimal to binary)
- PHP functions and control structures
  - Euclidian Algorithm (GCD)

What is Web: Internet+HTTP+HTML+URI

- is a way of accessing information over the medium of the Internet.



- Uses the **HTTP Protocol**

Some experiements

- Using Telnet to access web servers
  - Telnet is used to send/receive TCP messages
  - http://telnet.browseas.com/telnet.php
  - What this experiment tells us?
  - Other ports of TCP services: https://www.webopedia.com/quick_ref/portnumbers.asp
- Online HTML/PHP editor/debugger
  - http://sandbox.onlinephpfunctions.com/

Student question: what's the difference between URI and URL?

PHP: a Recursive Acronym

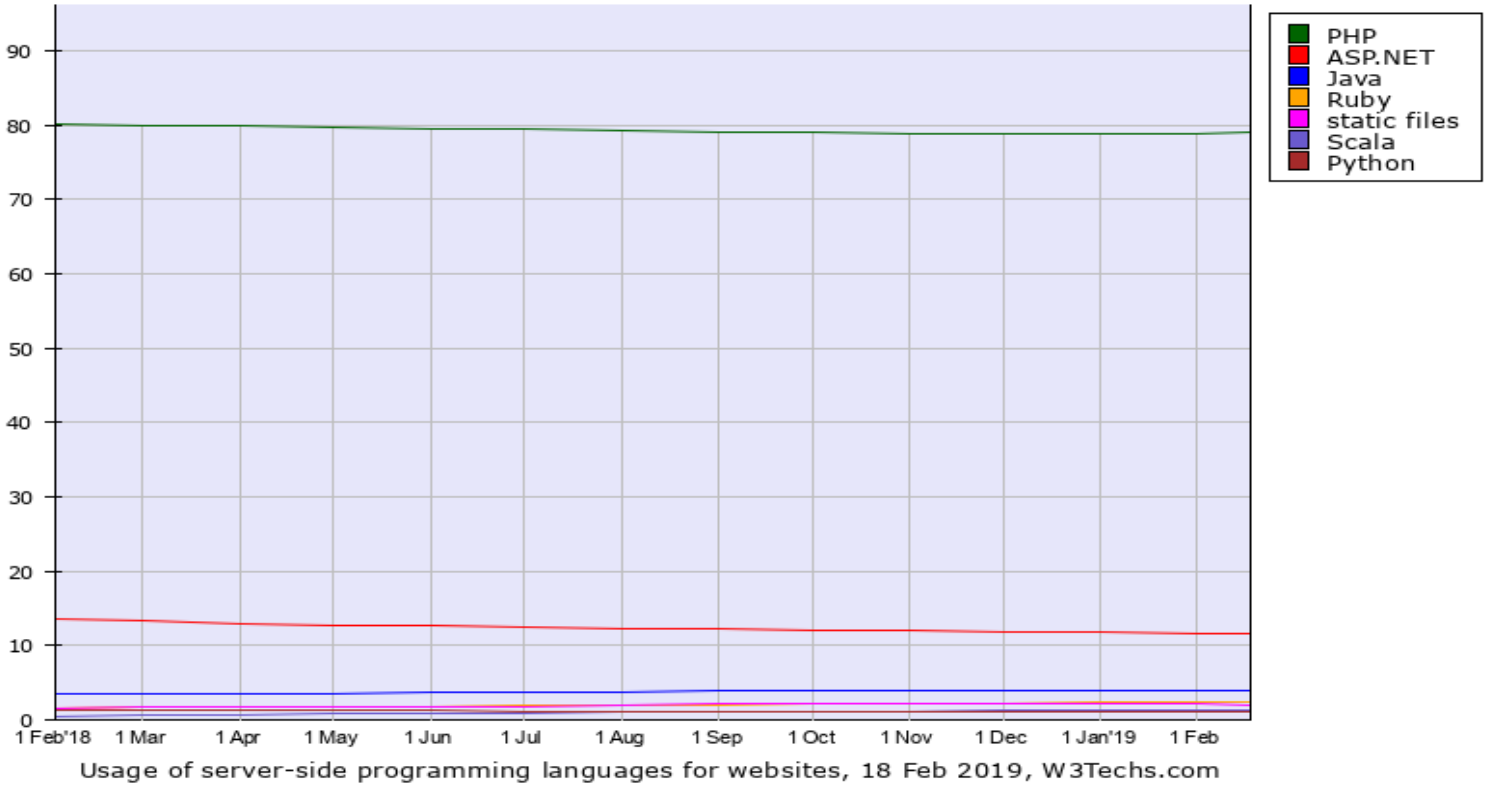
PHP: *PHP Hypertext Preprocessor*

GNU: GNU is Not Unix

Bing: Bing is not google

PIP: PIP Installs Packages

*Exercise: can you come up With a RA for **AUT**?*



A Web Development Environment

- A Web browser
  - e.g. Chrome, Firefox, Edge/IE, Safari
- A Web server
  - e.g. Apache
- The PHP software
- A database
  - e.g. MySQL, MariaDB

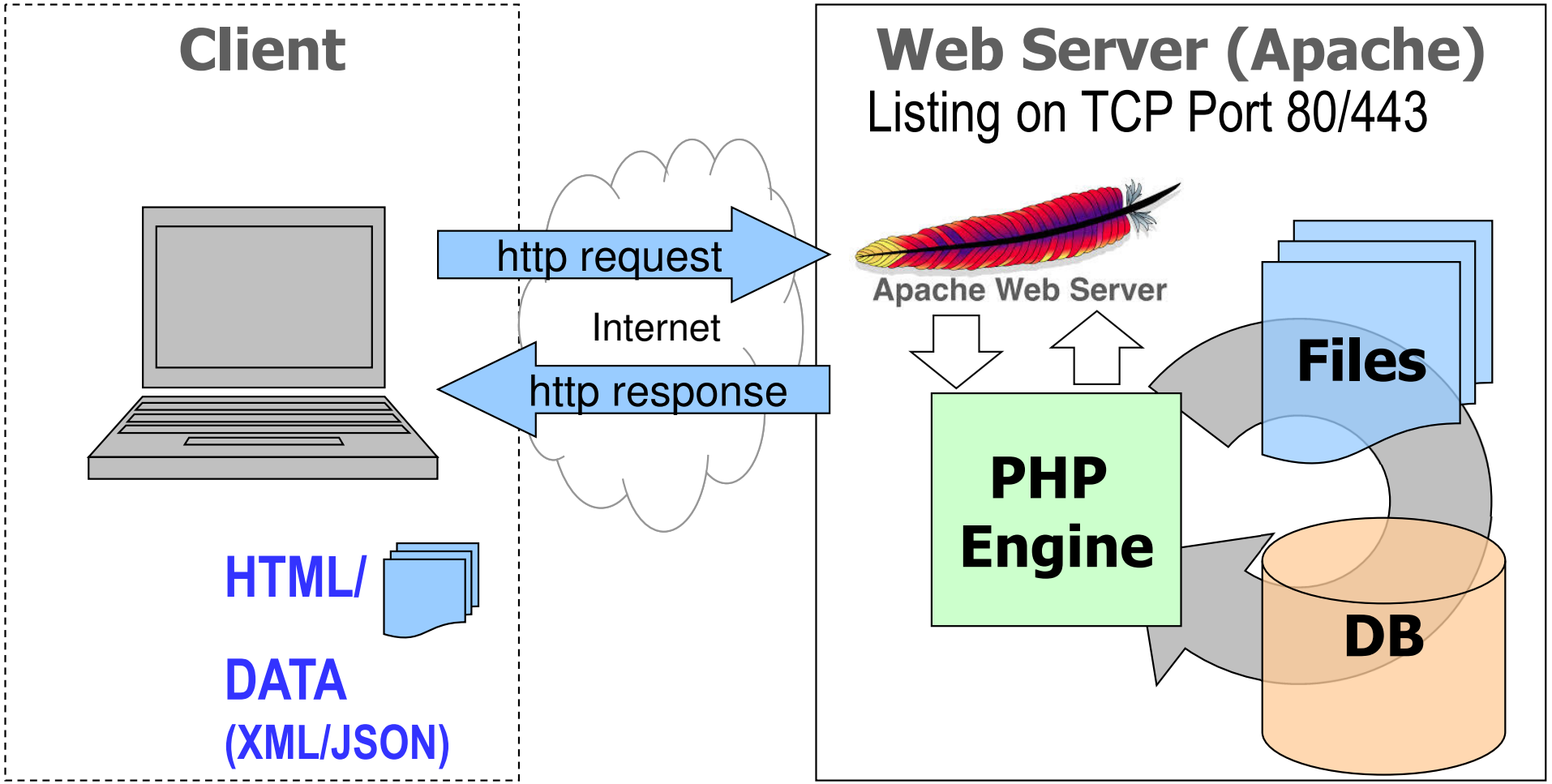
Alternative:

Server: Node.js

Database: NoSQL DB such as MongoDB

Embedded Scripting

- Apache/PHP/MySQL example



Creating Basic PHP Scripts

- **Embedded scripting languages** (JavaScript or PHP) refer to code that is embedded within a Web page (either an HTML or XHTML document)
- This code is typed directly into a Web page as a separate section
- A Web page document containing PHP code must have an extension of .php
- PHP code is never sent to a client's Web browser

Creating Basic PHP Scripts (continued)

- The Web page generated from the PHP code, and HTML elements found within the PHP file, is returned to the client
- A PHP file that does not contain any PHP code should have an **.html** extension
- .php is the default extension that most Web servers use to process PHP scripts

Creating PHP Code Blocks

- **Code declaration blocks** are separate sections within a Web page that are interpreted by the scripting engine
- There are four types of code declaration blocks:
  - **Standard PHP script delimiters**

    <?php statements; **?>**
  - (The `<script>` element)

    <script language =“php”> statements; **</script>**
  - (Short PHP script delimiters)

    <? statements; **?>**
  - (ASP-style script delimiters)

    <% statements; **%>**

Not portable, not recommended.

Some examples…

Standard PHP Script Delimiters

- A **delimiter** is a character or sequence of characters used to mark the beginning and end of a code segment
- The standard method of writing PHP code declaration blocks is to use the `<?php` and `?>` script delimiters
- The individual lines of code that make up a PHP script are called **statements**

```
<?php
    statements;
?>
```

`e.g. echo "Hello World!";` as a statement

Let's try http://sandbox.onlinephpfunctions.com/

Understanding Functions

- A **function** refers to a procedure that performs a specific task
- To execute a function, you must invoke, or **call**, it from somewhere in the script
- A **function call** is the function name followed by any data (**arguments**)
- The data in parentheses are **arguments/actual parameters**
- Sending data to a called function is called **passing arguments**

  e.g. `<?php pow(2,5);?>`

phpinfo()

- `<?php phpinfo(); ?>`
- Outputs a large amount of information about the current state of PHP. This includes information about PHP compilation options and extensions, the PHP version, server information and environment (if compiled as a module), the PHP environment, OS version information, paths, master and local values of configuration options, HTTP headers, and the PHP License.

Displaying Script Results

- To return to the client the results of any processing that occurs within a PHP code block, you must use an `echo` statement or the `print` statement
- The **`echo`** and **`print`** **statements** create new text on a Web page that is returned as a response to a client

Displaying Script Results (continued)

- The `echo` and `print` statements are language constructs of the PHP programming language, not functions.
- A **programming language construct** refers to a built-in feature of a programming language
- The `echo` and `print` statements are virtually identical except:
  - The `print` statement returns a value of 1 if it is successful
  - It returns a value of 0 if it is not successful

Displaying Script Results (continued)

- Use the `echo` and `print` statements to return the results of a PHP script within a Web page that is returned to a client
- A **text string**, or **literal string**, is text that is contained within double or single quotation marks
- To pass multiple arguments to the `echo` and `print` statements, separate them with commas like arguments passed to a function

```
<?php echo "Explore Africa,", "South America,", "and Australia!"; ?>
```

Note: It's not correct to put parentheses around multiple arguments as 'echo' is not a function!

Creating Multiple Code Declaration Blocks

For multiple script sections in a document, include a separate code declaration block for each section

```
...
</head>
<body>
<h1>Multiple Script Sections</h1>
<h2>First Script Section</h2>
<?php echo "<p>Output from the first script section.</p>";?>
<h2>Second Script Section</h2>
<?php echo "<p>Output from the second script section.</p>";?>
</body>
</html>
```

Let's try it on cmslamp14

Creating Multiple Code Declaration Blocks (continued)

PHP code declaration blocks execute on a Web server ***before*** the Web page is sent to a client

```
...
</head>
<body>
<h1>Multiple Script Sections</h1>
<h2>First Script Section</h2>
<p>Output from the first script section.</p>
<h2>Second Script Section</h2>
<p>Output from the second script section.</p>
</body>
</html>
```

Creating Multiple Code Declaration Blocks (continued)

Output of a document with two PHP script sections

Case Sensitivity in PHP

- Programming language constructs in PHP are mostly case **insensitive**

```
<?php
echo "<p>Explore <strong>Africa</strong>, <br />";
Echo "<strong>South America</strong>, <br />";
ECHO " and <strong>Australia</strong>!</p>";
?>
```

- However, ***code consistently***.
- Exceptions to case insensitivity: **variable** and **constant names** which ***are*** case sensitive and are studied later.

Recommend to use lower-case just as other PLs

Adding Comments to a PHP Script

- **Comments** are nonprinting lines placed in code such as:
  - The name of the script
  - Your name and the date you created the program
  - Notes to yourself
  - Instructions to future programmers who might need to modify your work
- Line comments hide a single line of code
  - Add // or # before the text
- Block comments hide multiple lines of code
  - Add /* to the first line of code
  - And */ after the last character in the code

Just like Java/C++

Adding Comments to a PHP Script (continued)

```
<?php
/*
This line is part of the block comment.
This line is also part of the block comment.
*/
echo "<h1>Comments Example</h1>"; // Line comments
// This line comment takes up an entire line.
# This is another way of creating a line comment.
/* This is another way of creating
a block comment. */
?>
```

PHP Data types and Operators

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

Naming Variables

- You must follow a consistent variable naming style
  - **`$votingAge`** ← **camelCase**
  - **`$voting_age`**
  - `$votingage`
  - `$VotingAge`
  - `$VOTING_AGE`

Variable names are case-sensitive!

- Are the two variable names below referring to the same variable (identifier)?
  - `$firstName`
  - `$FirstName`

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**
- It is not necessary to initialize variables in PHP however it is a very good practice.
- `$variable_name = value;`
- You can change the variable's value at any point

```
$variable_name = new_value;
```

Displaying the Values of Variables

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

```
$votingAge = 18;

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>";
```

Displaying the Values of Variables

- What if surrounded by double/single quotation marks

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

Content of $votingAge will be printed

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

Text '$votingAge' itself will be printed out.

Hint: If in doubt, separate with commas

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

Important PHP pre-defined variables

- $_GET
- $_POST
- Used to receive client data, will discuss in more detail in the next lecture

Defining Constants

- A constant contains information that does not change during the course of program execution
- Constant names do not begin with a dollar sign ($)
- 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 includes numerous predefined constants that you can use in your scripts

http://www.php.net/manual/en/reserved.constants.php

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

NULL means a variable has no value.

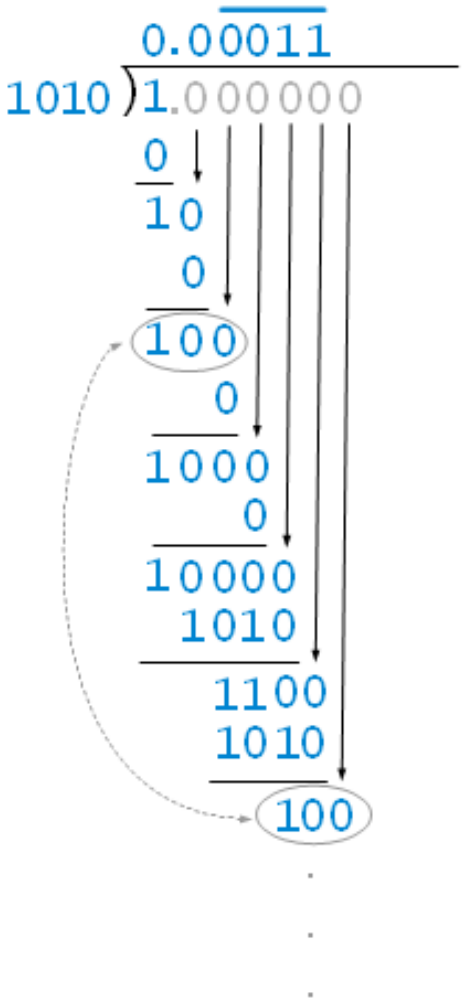
What data type shall we use for the 'balance' of a bank account?

Aside: converting decimal to binary

- Not a problem for integers: 10=>1010
- How about converting 0.1 to binary?

(infinite places…)

- The 24-bit binary representation of 0.10 actually corresponds to 0.099999904632568359375, which is off by 0.000000095367431640625.
- For PHP, we can use BCMath Arbitrary Precision Mathematics; or use the Money package



***Round-off Error incident**: During the Gulf War in 1991, a U.S. Patriot missile failed to intercept an Iraqi Scud missile, and 28 Americans were killed.*

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

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)

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/TRUE or false/FALSE
- In other programming languages, you can use integers such as 1 = true, 0 = false

true and false are case-insensitive

Working with Data Types (continued)

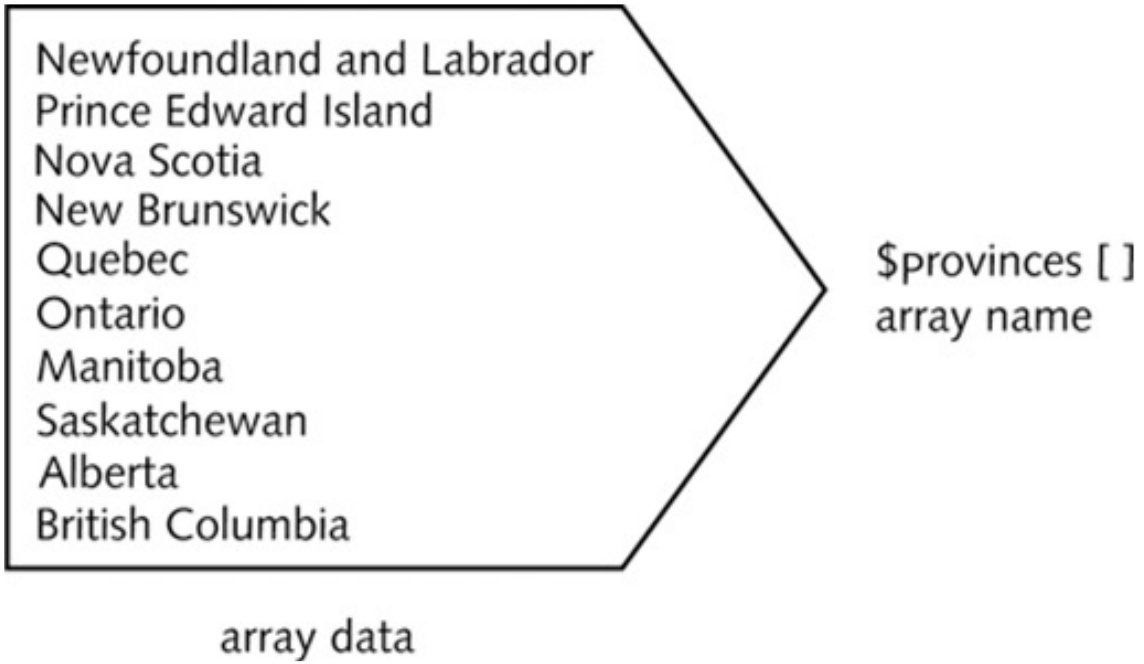
- The data type of a variable (identifiers) or constant depends on the data type of the value assigned to it
  - $courseName = “Web Development”;
  - $lectureHours = 2;
  - $hasLabs = TRUE;

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 connection or XML file
- **Reference** or **composite** data types, which contain multiple values or complex types of information
- Two reference data types: **arrays** and **objects**

Arrays

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



Conceptual example of an array

- Indexed arrays & associative arrays

Declaring and Initialising Indexed Arrays

- An **element** refers to each piece of data that is stored within an array
  - By default, it starts with the number zero (0)
- An **index** is an element's numeric position within the array
  - Referenced by enclosing its index in brackets at the end of the array name:
  - `$provinces[1]`

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"
        );
```

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

Accessing Element Information

- `echo "<p>Canada's smallest province is $provinces[1].<br />";`
- `echo "Canada's largest province is $provinces[4].</p>";`

Output of elements in the $provinces[ ] array

`count()` Function

- Use the **`count()`** function to find the total number of elements in an array

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

$territories = array("Nunavut", "Northwest
Territories", "Yukon Territory");

echo "<p>Canada has ",
count($provinces), " provinces and ",
count($territories), " territories.</p>";
```

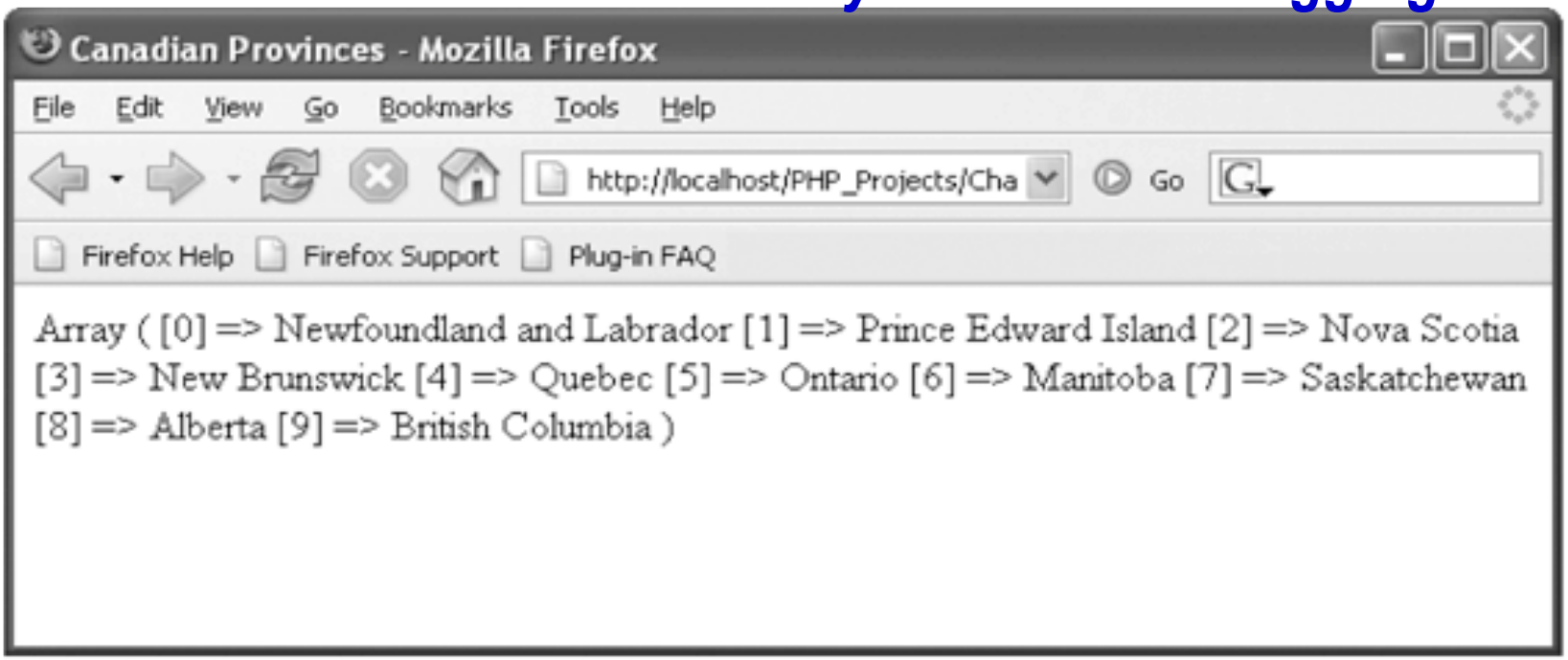
Output:

```
Canada has 10 provinces and 3 territories.
```

`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

Very useful for debugging



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

Modifying Elements

- Include the index for an individual element of the array:

```
$hospitalDepts = array(
    "Anesthesia",              // first element (0)
    "Molecular Biology",       // second element(1)
    "Neurology");              // third element (2)
```

To change the first array element in the `$hospitalDepts[]` array from “Anesthesia” to “Anesthesiology” use:

```
$hospitalDepts[0] = "Anesthesiology";
```

Group discussion: PHP vs Java

- Execution: Interpreter? Compiler?
- Embedded?
- Case sensitive?
- Variable name?
- Data type system?
- Equivalent PHP data type in Java?

PHP Expressions

Building Expressions

- An **expression** is a literal value or variable
  - that can be evaluated by the PHP scripting engine to produce a result
- **Operands** are variables and literals contained in an expression
- 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

Expressions are recursively defined so it can be represented as a tree!

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

Array operators: http://php.net/manual/en/language.operators.array.php

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

Arithmetic Operators

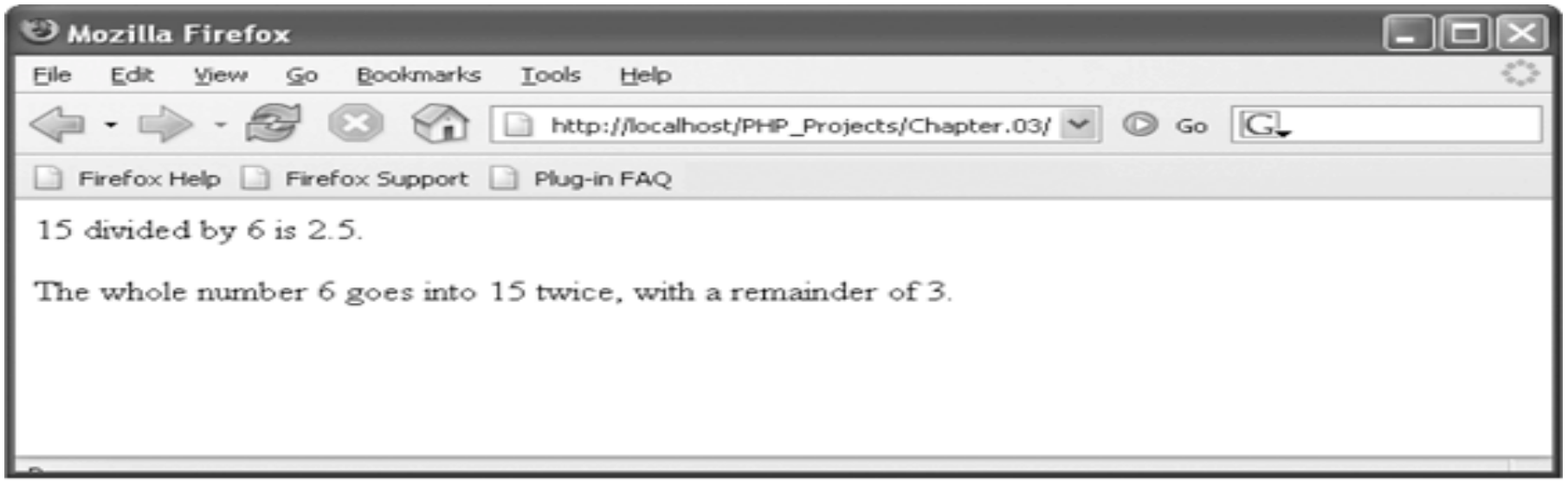
- **Arithmetic operators** are used in PHP to perform mathematical calculations

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

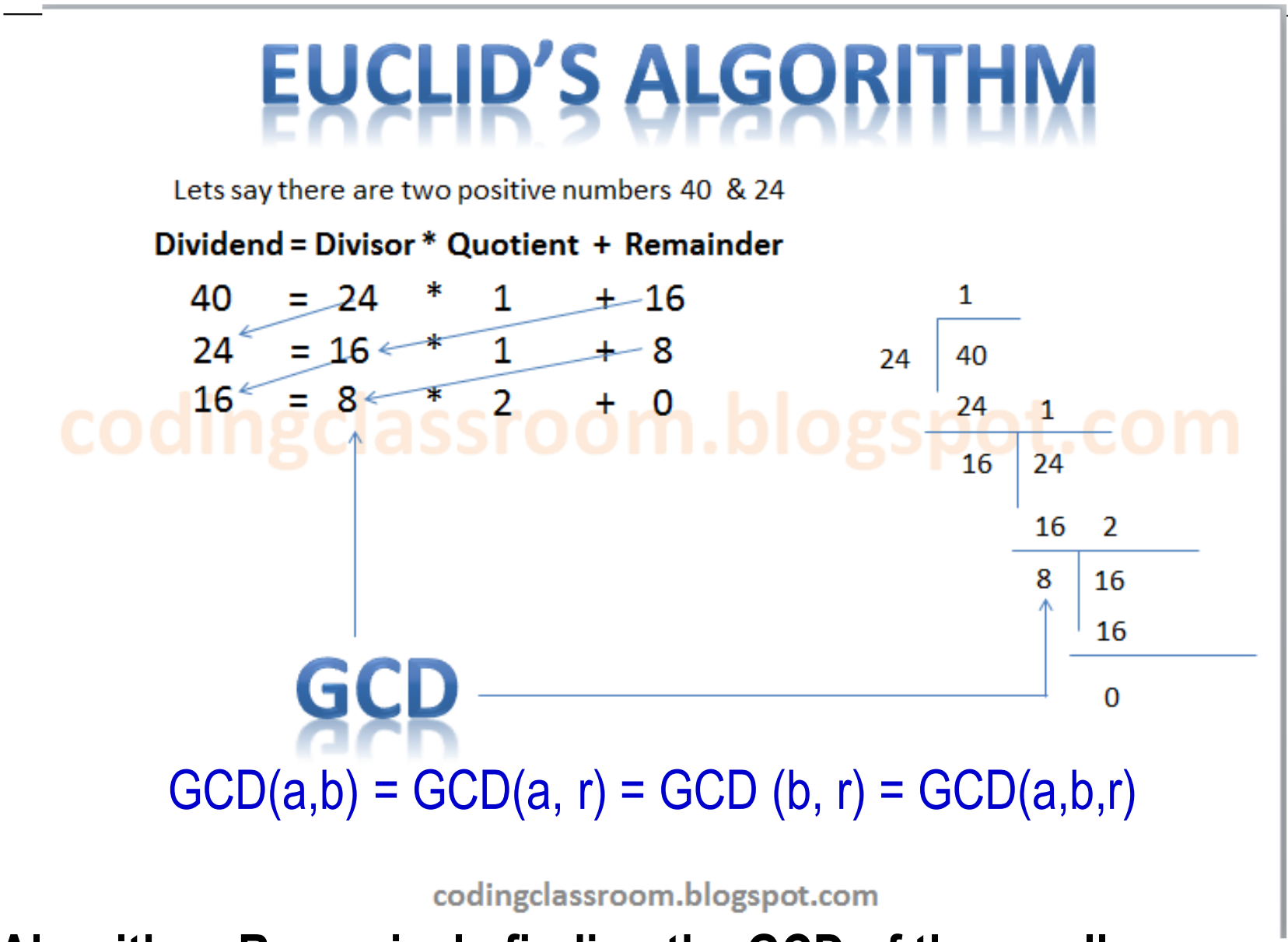
Arithmetic Operators (continued)

```
$divisionResult = 15 / 6;
$modulusResult = 15 % 6;
echo "<p>15 divided by 6 is $divisionResult.</p>"; //
  prints '2.5'
echo "The whole number 6 goes into 15 twice, with a
  remainder of $modulusResult.</p>"; // prints '3'
```



Division and modulus expressions

Finding the Greatest Common Divisor (GCD)



Algorithm: Recursively finding the GCD of the smaller number and remainder until the remainder is 0

PHP code for Euclidian Algorithm

```
<?php
/*Returns the greatest common divisor of two integers
using the Euclidean algorithm.*/
function gcd( $a, $b ) {
        $large = $a > $b ? $a: $b;
        $small = $a > $b ? $b: $a;
        $remainder = $large % $small;
        return 0 == $remainder ? $small :       gcd(
$small, $remainder );
}
```

Proof it?

- Two integers a and b (assume a>b) have common divisor d (means):
  - a = m*d --- (1)
  - b = n*d --- (2)
  - a = q*b + r --- (3)
  - md = qnd + r --- plug 1 and 2 to 3
  - r = d(m - qn) --- which gives d is also a divisor of r

Time complexity?

the worst case is when a and b are consecutive Fibonacci numbers.

Worse case: $O(\log b)$

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

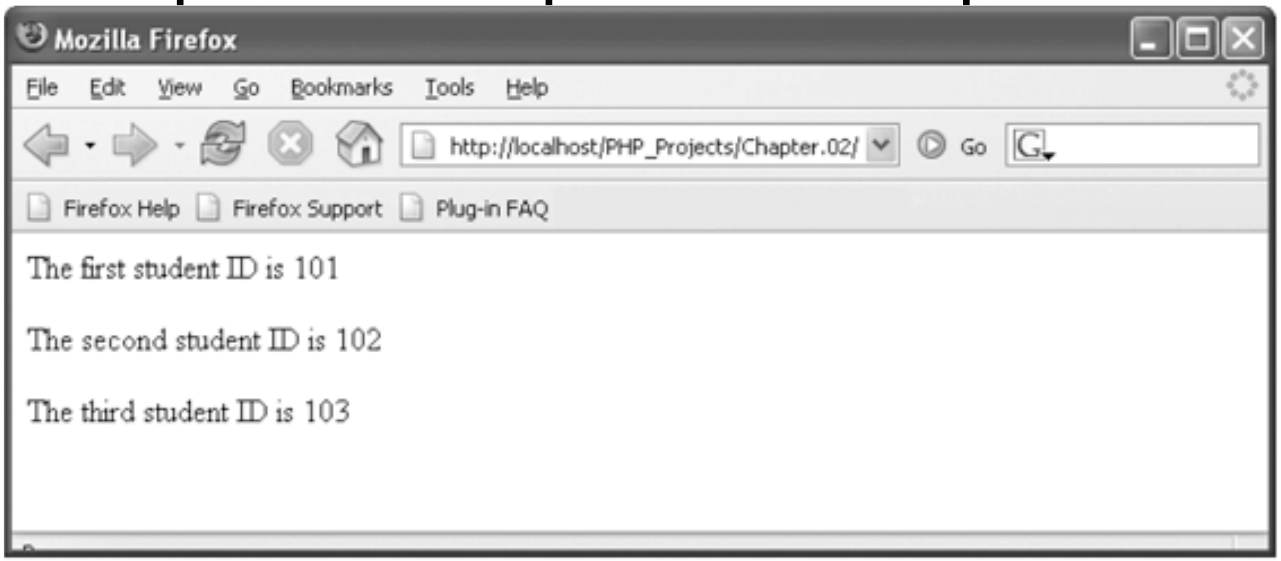
Operator	Name	Description
++	Increment	Increases an operand by a value of one
--	Decrement	Decreases an operand by a value of one

Arithmetic Unary Operators (continued)

```
$StudentID = 100;
$CurStudentID = ++$StudentID; // assigns '101'
echo "<p>The first student ID is ",
     $CurStudentID, "</p>";
$CurStudentID = ++$StudentID; // assigns '102'
echo "<p>The second student ID is ",
     $CurStudentID, "</p>";
$CurStudentID = ++$StudentID; // assigns '103'
echo "<p>The third student ID is ",
     $CurStudentID, "</p>";
```

prefix increment operator

Script that uses the prefix increment operator



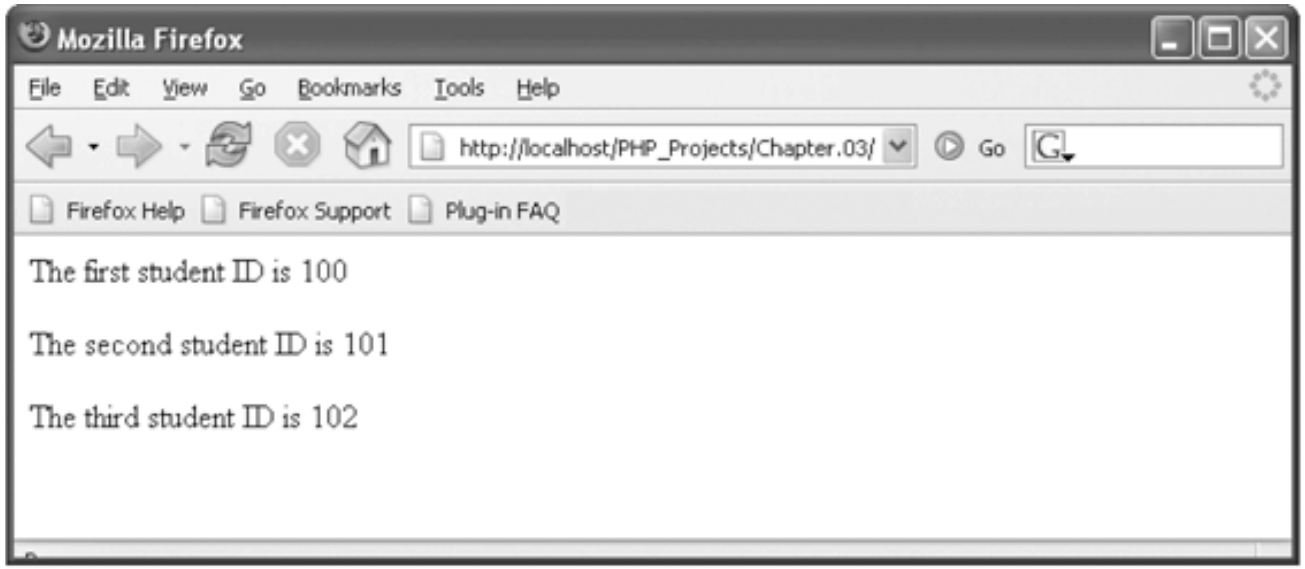
Output of the prefix version of the student ID script

Arithmetic Unary Operators (continued)

```
$StudentID = 100;
$CurStudentID = $StudentID++; // assigns '100'
echo "<p>The first student ID is ",
     $CurStudentID, "</p>";
$CurStudentID = $StudentID++; // assigns '101'
echo "<p>The second student ID is ",
     $CurStudentID, "</p>";
$CurStudentID = $StudentID++; // assigns '102'
echo "<p>The third student ID is ",
     $CurStudentID, "</p>";
```

postfix increment operator

Script that uses the postfix increment operator



Output of the postfix version of the student ID script

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

Assignment Operators (continued)

AUT UNIVERSITY

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

Assignment Operators (continued)

```
$x = 100;
$y = 200;
$x += $y;    same as  $x = $x + $y;    (300)

$x = 2;
$y = 6;
$x *= $y;    same as  $x = $x * $y;    (12)
```

Comparison and Conditional 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 and Conditional 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

Strict equal is something new in PHP

true== 1? 1.0== 1?

true===1? 1.0===1?

Comparison and Conditional Operators

(continued)

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

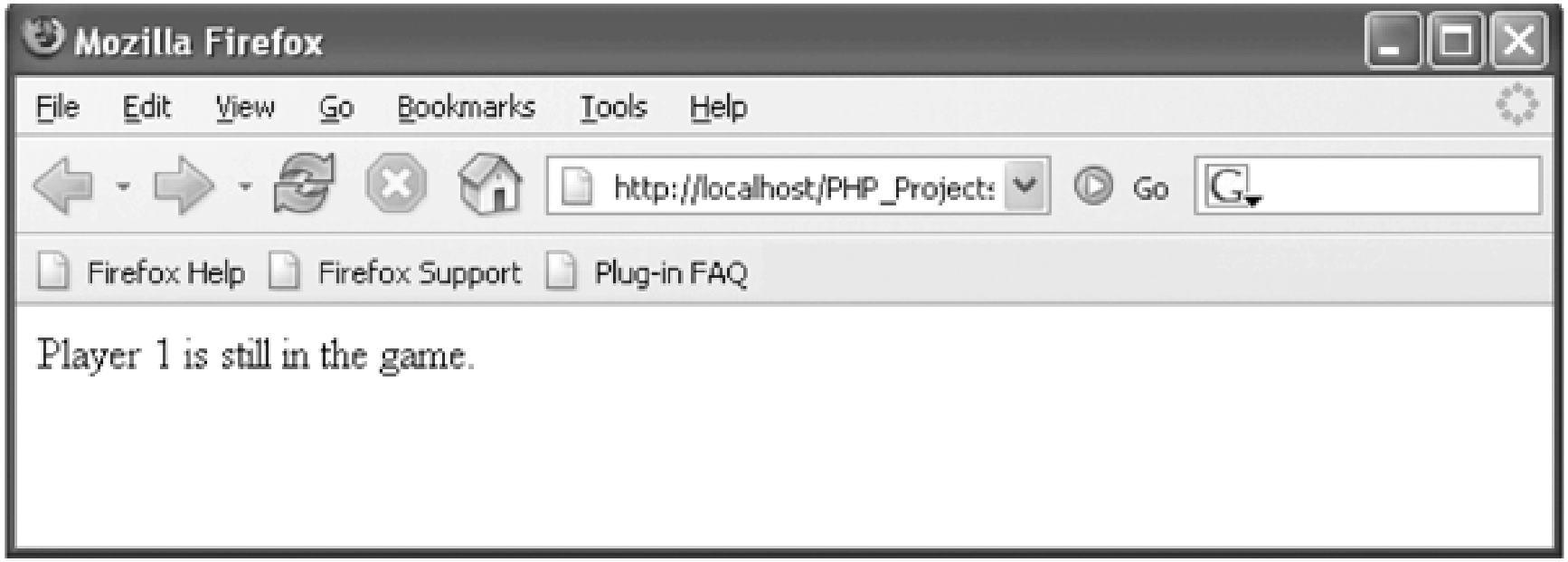
```
conditional expression
      ? expression1 :  expression2;
```

- If the conditional expression evaluates to true, expression1 executes
- If the conditional expression evaluates to false, expression2 executes

Comparison and Conditional Operators

(continued)

```
$blackjackPlayer1 = 20;
($blackjackPlayer1 <= 21)
    ? $result = "Player 1 is still in the game.“
    : $result = "Player 1 is out of the action.";
echo "<p>", $result, "</p>";
```



Output of a script with a conditional operator

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

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

```
$myArray = array(
    0 => 'Big',
    1 => 'Small',
    2 => 'Up',
    3 => 'Down');
```

***Note**: These Special Operators are introduced throughout this course as necessary*

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";
```

`gettype()` function

Returns one of the following strings, depending on the data type

- *no guessing needed:*
  - Boolean
  - Integer
  - Double (double float)
  - String
  - Array
  - Object
  - Resource
  - NULL
  - Unknown type

Unknown type

- E.g., whenever a resource pointer is closed, the variable holding the handle will point to an unknown resource

```
$f = fopen('somefile.txt','r');
echo gettype($f); // resource
fclose($f);
echo gettype($f); // unknown
```

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?

Understanding Operator Precedence

(continued)

Operator precedence in PHP

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

Exercise

- Read the following php program and write the generated HTML document:

```
<html ><head> <title>Using variables, arrays and operators</title></head>
  <body>
    <h1>Web Development </h1>
    <?php
          $marks = array (85, 85, 95);
          $marks[1] = 90;
          $ave = ($marks[0] + $marks[1] + $marks[2]) / 3;
          ($ave >= 50)
                    ? $status = "PASSED"
                    : $status = "FAILED" ;
          echo " <p>The average score is $ave. You $status</p> ";
  ?></body></html>
```

PHP functions and control structures

(very similar to Java/C++, so we just discuss it briefly)

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
- The syntax for defining a function is:

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

Defining Functions (continued)

- Functions, like all PHP code, must be contained within `<?php ... ?>` tags
- 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

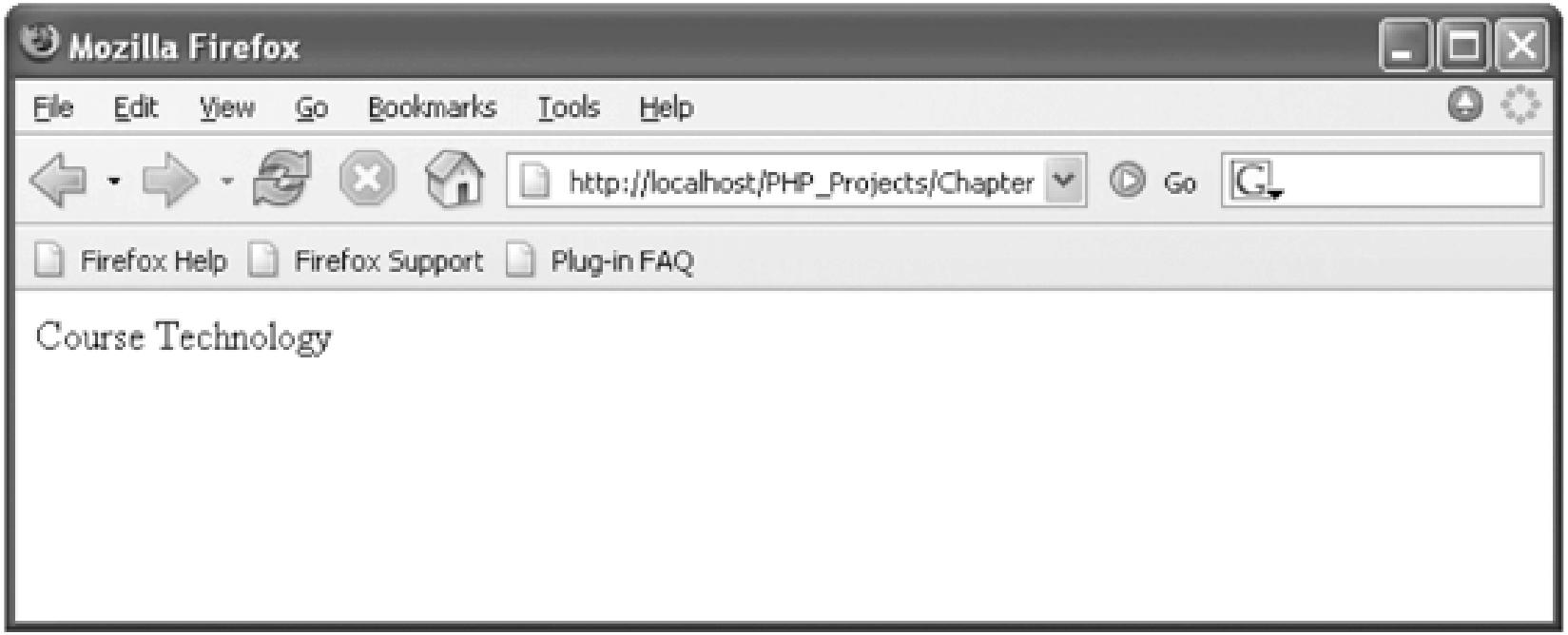
Defining Functions (continued)

- **Function statements** do the actual work of the function and must be contained within the function braces

```
function printCompanyName($company1,
    $company2, $company3) {
    echo "<p>$company1</p>";
    echo "<p>$company2</p>";
    echo "<p>$company3</p>";
}
```

Calling Functions

```
function printCompanyName($companyName) {
        echo "<p>$companyName</p>";
}
printCompanyName("Course Technology");
```



Output of a call to a custom function

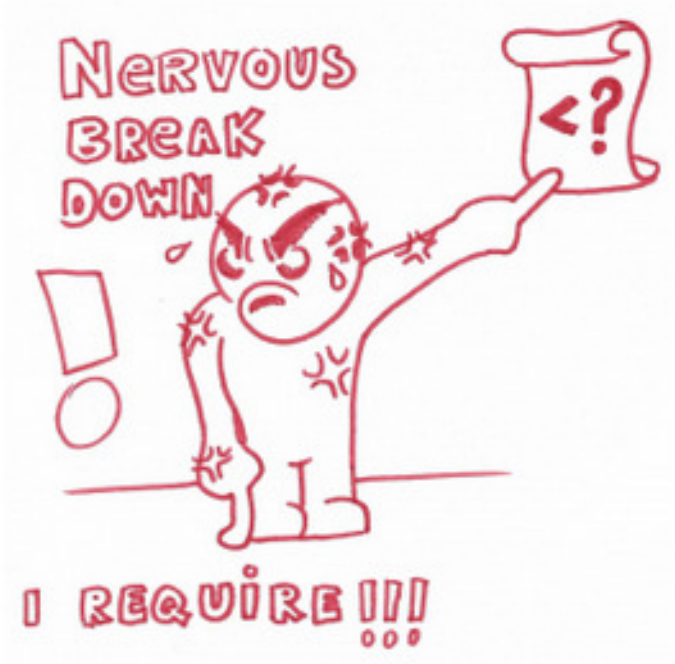
Returning Values

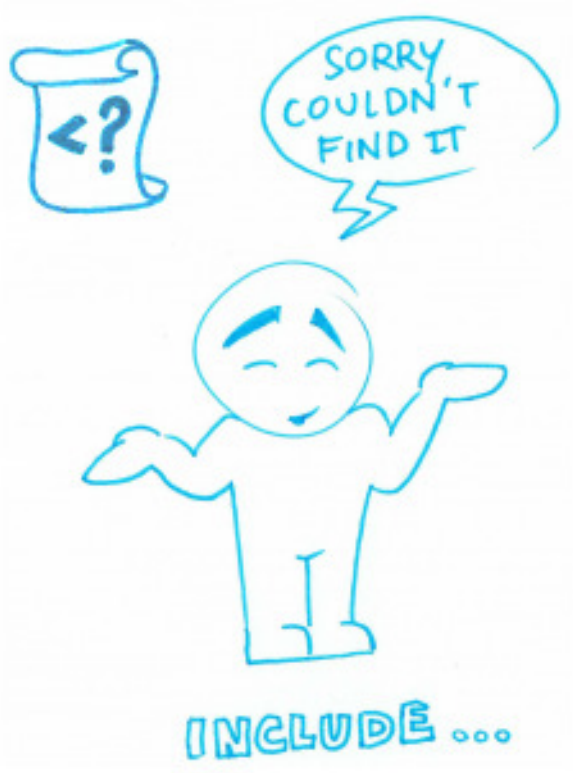
- A **return statement** is a statement that returns a value to the statement that called the function
- A function does not necessarily have to return a value

```
function averageNumbers($a, $b, $c) {
    $sum = $a + $b + $c;
    $result = $sum / 3;
    Return $result;
}
```

Include/require functions defined in external files

- <?php require('/var/www/public_html/config.php'); ?>





When Mr. Php Script "requires" *he makes a nervous break down and* **crashes**, *if not found.*

In the other hand, when **Mr. Php Script** *is asked to* **"include"** *the file, he just apologizes for inconvinience with a little "***Warning***" not found.*

Source: http://c.onfigs.com/php/difference-require-vs-include-php/

Require/include once

- <?php require_once('/var/www/public_html/config.php'); ?>
- The require_once statement is identical to require except PHP will check if the file has already been included, and if so, not include (require) it again.

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**

`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 (continued)

- 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 **}**

```
$exampleVar = 5;
if ($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>";
```

`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 (continued)

- 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>";
```

Note: Single statement within `if` … `else`, therefore no braces needed

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**

```
if ($_GET["SalesTotal"] > 50)
    if ($_GET["SalesTotal"] < 100)
        echo "<p>The sales total is between 50 and 100.</p>";
```

`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` Statement (continued)

- Consists of the following components:
  - The `switch` keyword
  - An expression
  - An opening brace
  - A `case` label
  - The executable statements
  - The `break` keyword
  - A default label
  - A closing brace

`switch` Statement (continued)

- The syntax for the `switch` statement is:

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

`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
- A `default` label consists of the keyword `default` followed by a colon

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

`while` Statement

- Repeats a statement or a series of statements as long as a given conditional expression evaluates to true
- The syntax for the `while` statement is:

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

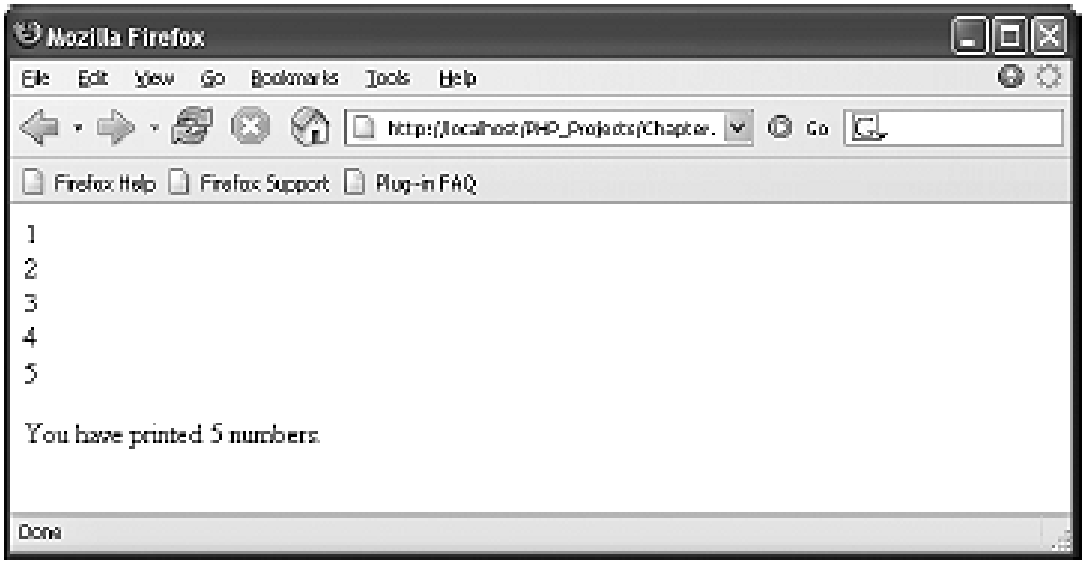
- As long as the conditional expression evaluates to true, the statement or command block that follows executes repeatedly

`while` Statement (continued)

- 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

`while` Statement (continued)

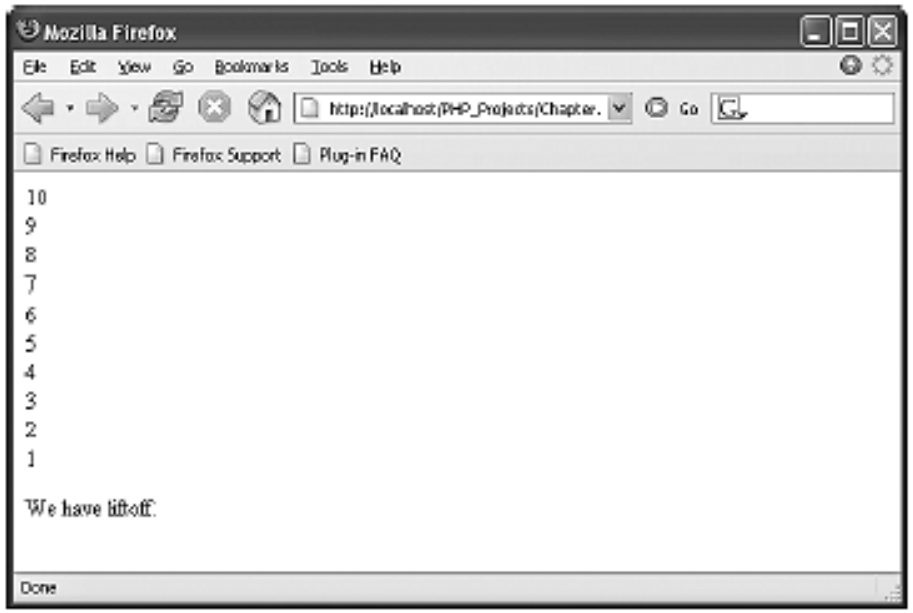
```
$count = 1;
while ($count <= 5) {
      echo "$count<br />";
      $count++;
}
echo "<p>You have printed 5 numbers.</p>";
```



Output of a `while` statement using an increment operator

`while` Statement (continued)

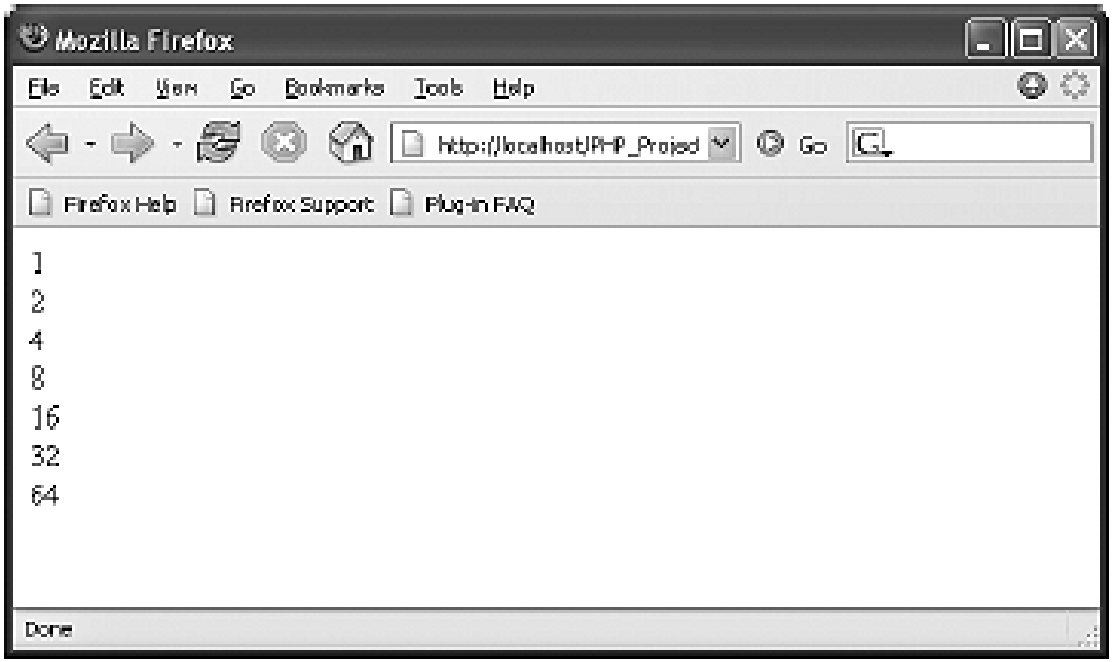
```
$count = 10;
while ($count > 0) {
      echo "$count<br />";
      $count--;
}
echo "<p>We have liftoff.</p>";
```



Output of a `while` statement using a decrement operator

`while` Statement (continued)

```
$count = 1;
while ($count <= 100) {
        echo "$count<br />";
        $count *= 2;
}
```



Output of a `while` statement using the assignment operator *=

`while` Statement (continued)

- 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";
}
```

- The `continue` statement

`do...while` Statement

- Executes a statement or statements once, 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);
```

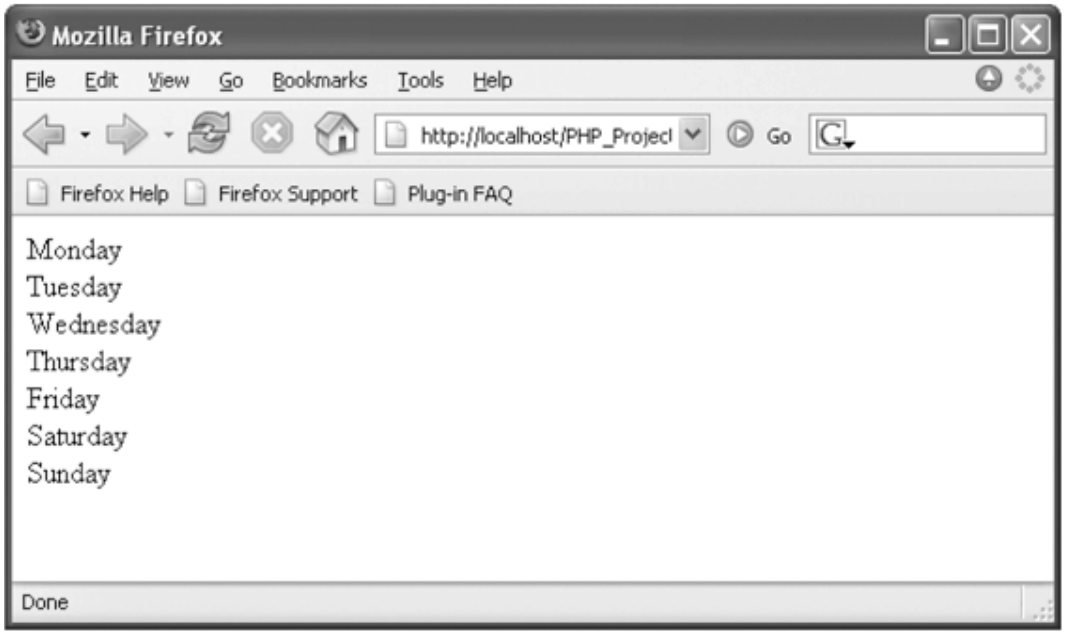
`do...while` Statement (continued)

- `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);
```

`do...while` Statement (continued)

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



Output of days of week script in Web browser

`for` Statement

- Used for repeating a statement or a series of statements as long as a given conditional expression evaluates to true
- If a conditional expression within the `for` statement evaluates to true, the `for` statement executes and continues to execute repeatedly until the conditional expression evaluates to false

`for` Statement (continued)

- Can also include code that initialises a counter and changes its value with each iteration
- The syntax of the `for` statement is:

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

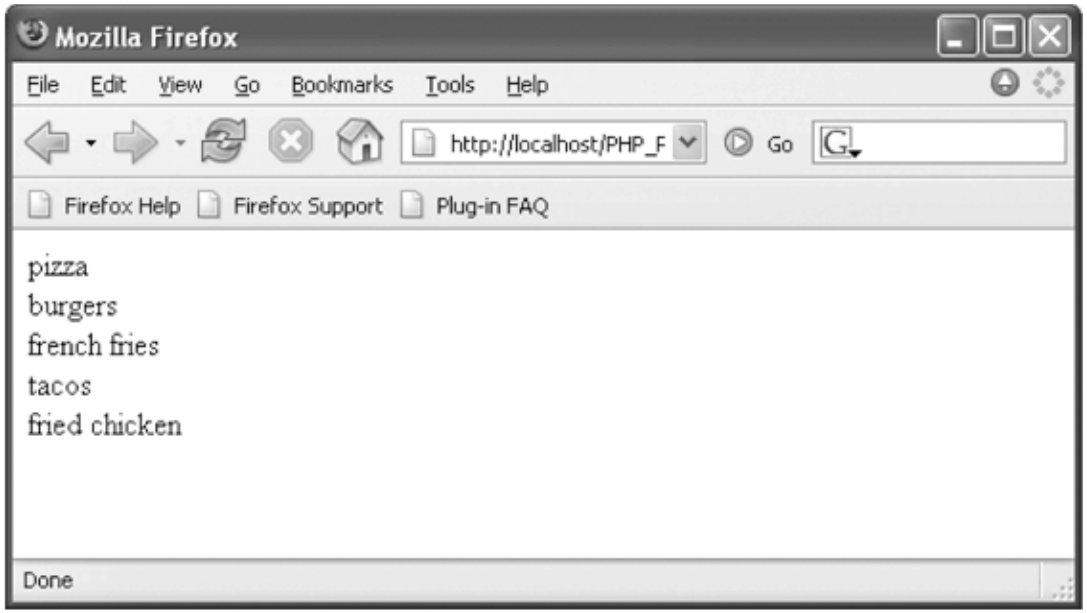
`for` Statement (continued)

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

for ($count = 0; $count < 5; $count++) {

      echo $fastFoods[$count], "<br />";

}
```



Output of fast-foods script

`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 a set of parentheses following the `foreach` keyword
- The syntax for the `foreach` statement is:

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