PHP Basic Syntax

Data Types, Variables, Operators, Expressions

greenwich.edu.vn

UNIVERSITY of GREENWICH

Alliance with FPT Education

Table of Contents

PHP INTRODUCTION

What is PHP?

- PHP (PHP Hypertext Preprocessor) is server-side scripting language used for creating dynamic web content
  - First introduced in 1995 as module for Apache
  - Free and open-source, written in C
  - Can be deployed on almost any operating system
  - Provides interaction with Databases (CRUDs)
  - Can be embedded in HTML

PHP – Example

```
<html>
<head>
<title>My first PHP code!</title>
</head>
<?php
   $myName = 'Yordan';
   echo 'My name is ' . $myName;
?>
<body>
</body>
</html>
```

Code is enclosed with `<?php … ?>` tags

Mixing PHP and HTML

- PHP is designed to mix HTML and PHP code:

```
<?php $name = "John"; ?>
<?php if ($name == "John") { ?>
    <p>Hello John!</p>
<?php } else { ?>
    <p>You are not John.</p>
<?php } ?>
```

  - This is similar to writing echo "Hello John!";
  - Very useful for long texts

UNIVERSITY of GREENWICH

Alliance with FPT Education

DATA TYPES IN PHP

What Is a Data Type?

- A data type:
  - Is a domain of values of similar characteristics
  - Defines the type of information stored in the computer memory (in a variable)
- PHP supports eight types:
  - Scalar: Boolean, Integer, Floating point, String
  - Compound: Array, Object
  - resource and NULL
- PHP is a dynamically typed language

PHP Data Types

- PHP is a **dynamically typed** language
  - The variable types are not explicitly defined
  - The type of a variable can be changed at runtime
- Variables in PHP are declared with the symbol **$**

```
$count = 5;                   // variable holds an integer value
$count = 'hello’;             // the same variable now holds a string
$name = 'Svetlin Nakov’;     // variable holds a string
$mark = 5.25;                // mark holds a floating-point number
```

Integer Numbers

- Integer types represent whole numbers
- The size of an integer is platform-dependent
  - 32-bit: **-2147483647** to **2147483647**
  - 64-bit: **-9223372036854775807** to **9223372036854775807**
  - *Note: some 64-bit builds have used 32-bit integers, particularly older Windows builds of PHP*
  - Too large values for integer type are automatically turned into a floating-point number with exponent

```
$maxInteger = 9223372036854775807;
echo gettype($maxInteger); // integer
$maxInteger += 1;
echo gettype($maxInteger); // double
```

Floating-Point Numbers

- **Floating-point types represent real numbers, e.g. 5.63**
- **In PHP the floating-point numbers are 64-bit**
  - **Stored in the IEEE 754 format**
  - **Have range from** `-1.79e+308` to `1.79e+308`
  - **Have precision of roughly 14 digits**
- **Can behave abnormally in the calcula**
  - E.g. 0.1 + 0.2 = 0.30000000000000004

Numbers Conversion

- Convert to float number

```
$variable = 5;                           // Or:
$floatVar = floatval($variable); $floatVar = (float)$variable;
```

- Convert to integer number

```
$variable = 5.24245;                    // Or:
$varInt = intval($variable);            $varInt = (int)$variable;
```

- Convert string to integer

```
$num = "3.14";
$int = (int)$num;
$float = (float)$num;
```

The Boolean Data Type

- Has two possible values: true and false
  - Values are case-insensitive (True, true, TRUE & False, false, FALSE)
- Is useful in logical expressions
- Returns "1" or "null"
- Example of Boolean variables:

```
echo(true); // 1
echo(false); // (nothing)
var_dump(true); // bool(true)
var_dump(false); // bool(false)
```

The String Data Type

- The string data type represents a sequence of characters
- Strings are enclosed in quotes:
    - Best practices suggest using the most appropriate quotes

```
$string = 'Welcome to PHP';
$string = "'Welcome to PHP'";
```

- Strings can be concatenated (joined together)
    – Using the . (dot) operator

```
$name = ‘Viet' . ' ' . ‘Nam'; // Viet Nam
```

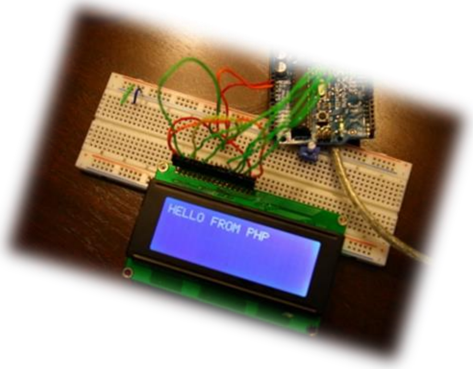
Variable Interpolation

- Single-quoted strings do not interpolate variables:

```
$name = 'Fred';
echo 'Hello, $name'; // Hello, $name
```

- Double-quoted string interpolate variables:

```
$who = 'Svetlin';
$where = 'here';
echo "$who was $where"; // Svetlin was here
```



- Curly braces ensures the correct variable is interpolated – best

```
$n = 12;
echo "You are the {$n}th person";
```

Array Type

- An array holds a group of values, which you can identify by position or identifying name
  - Arrays with number identifiers (with zero being the first position)

```
$students[0] = "Dean";
$students[1] = "Vladislav";
$students = array("Dean", "Vladislav");
```

  - Associative arrays with string identifiers

```
$students['Dean'] = 6;
$students['Vladislav'] = 5;
$students = array('Dean' => 6, 'Vladislav' => 5);
```

Object Type

- A class is a definition of a structure
  - Contains properties (variables) and methods (functions)
- Once a class is defined, any number of objects can be made from it with the **new** keyword
- Object's properties/methods can be accessed with the **->** construct

```
class Person {
   public $name;
   function name($newname) {
      $this->name = $newname;
   }
}
```

```
$svetlin = new Person;
$svetlin->name('Svetlin');
echo "Hello, {$svetlin->name}\n";
```

Resource Type

- Special variable, holding a reference to an external resource
  - – E.g. opened file, database connection, image canvas area
- Resources are created and used by special functions.
- Resource with no more references to it is detected automatically
  - – It is freed by the garbage collector
- `is_resource()` function checks whether a value is a resource

```
$res = database_connect();  // database connect function
database_query($res);
$res = "boo"; // database connection automatically closed because $res is
redefined
```

UNIVERSITY of GREENWICH

Alliance with FPT Education

NULL VALUES

What is NULL in PHP?

Null Value

- In PHP there is a special value **NULL**
  - Undefined means that a variable is declared but not initialized
  - Null means that an object exists and is empty (has no value)
  - All variables can be reset to null with **unset()**

```
<?php
$variable; // variable is undefined
$variable = 4; // variable has value 4
$variable = NULL; // variable has no value
unset($variable); // variable is undefined
?>
```

Checking the Type of a Variable

- The variable type can be checked with **`gettype()`**
- Or just print it with **`var_dump()`**
  - Great for checking the type and value of a given variable in the code

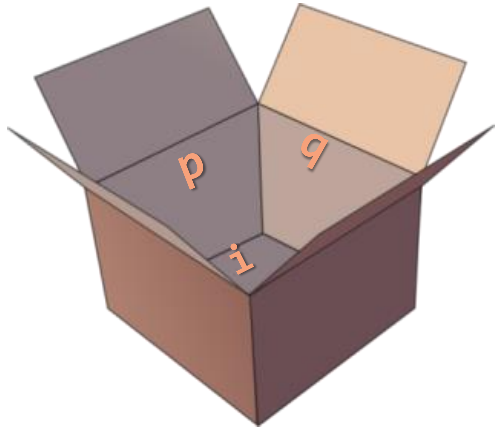
```
$boolVariable = true;
gettype($boolVariable); // boolean

$intVariable = 123;
gettype($intVariable); // integer

$stringVariable = "SoftUni";
gettype($stringVariable); // string
```

```
$b = true;
var_dump($b); // boolean(true)

$i = 123;
var_dump($i); // integer(123)

$s = "PHP";
var_dump($s); // string("PHP")
```

DECLARING AND USING VARIABLES

What Is a Variable?

- A variable is a:
  - Placeholder of information that can be changed at run-time
  - A piece of computer memory holding some value
- Variables allow you to:
  - Store information
  - Retrieve the stored information
  - Change the stored information



Variable Characteristics

- A variable has:
  - Name
  - Type (of stored data)
  - Value
- Example:

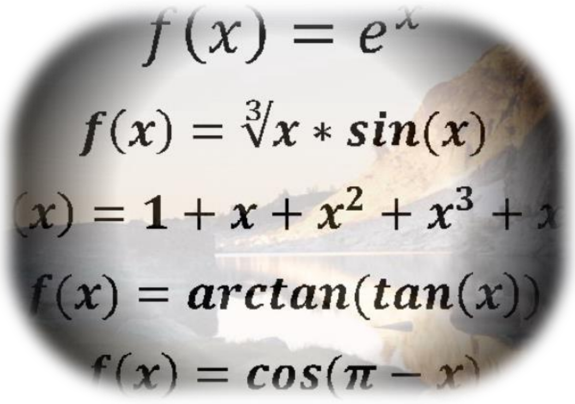
```
$counter = 5;
```

  - Name: **$counter**
  - Type: **integer**
  - Value: **5**

Declaring Variables

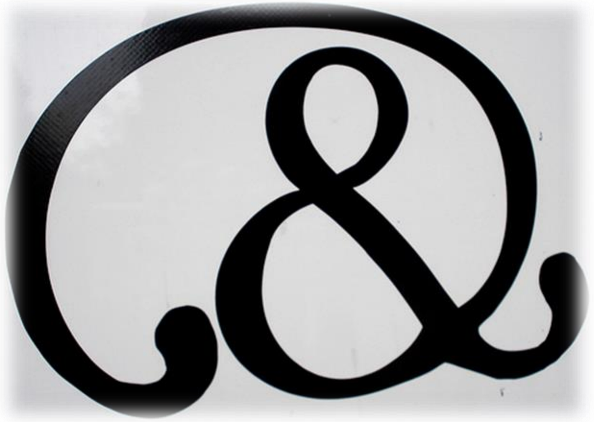
- When declaring a variable we:
  - Specify its name (called identifier)
  - The type is inferred by the value
  - Give it an initial value
- Examples:

```
$height = 200;
$str = 'Hello';
$isPositive = true;
```



Identifiers

- Identifiers may consist of:
  - Letters , digits [**0-9**], underscore '_'
  - Cannot start with a digit
  - Cannot be a PHP keyword
- Identifiers in PHP are case-sensitive
- Identifiers should have a descriptive name
  - Only Latin letters
- Variables and functions names: we recommend **camelCase**



Identifiers – Examples

- Examples of correct identifiers:

```
$New = 2; // Here N is capital, so it's not a PHP keyword
$_2Pac = 2; // This identifier begins with _

$поздрав = 'Hello'; // Unicode symbols used
$greeting = 'Hello'; // This is more appropriate

$n = 100; // Undescriptive
$number_of_clients = 100; // Descriptive

// Overdescriptive identifier:
$numberOfPrivateClientOfTheFirm = 100;
```

- Examples of incorrect identifiers:

```
$2Pac = 2; // Cannot begin with a digit
function new() { return 5; } // new is a keyword
```

Assigning Values

- The **=** operator is used to assign a value to a vari
  - Assignment operation has

```
// Assign a value (literal) to a variable
$firstValue = 5;

// Using an already declared variable:
$secondValue = $firstValue;

// Cascading assignment
$thirdValue = $newValue = 3;
```

Variable Variables

- Reference the value of a variable whose name is stored in another variable by prefacing the variable reference

```
// variable variables example
$variable = "first";
$$variable = "second";
echo $variable; // first
echo $first; // second
echo $$variable; // second
```

- After the second statement executes, the variable **$first** has the value "**second**"

Variables in PHP

- A variable in PHP can be:
  - undefined

    ```
    echo($asfd); // Error
    ```
  - NULL

    ```
    $p = null; echo($p); // nothing is printed
    ```
  - Has type

    ```
    $localVar = 5; echo($localVar); // 5
    ```
- Example: In this code **secondVar** is **undefined**:

```
$firstVar = 10;
echo($firstVar); // 10
echo($secondVar); // Undefined variable: secondVar
```

VARIABLE SCOPE

Local, Global, Static

Local Scope

- Local scope: a variable declared in a function is local to that function
  - Visible only to code in that function
  - Not accessible outside of the function
  - Variables defined outside a function (called global variables) are not accessible directly

```
function updateCounter() {
    $counter++;
}
$counter = 10;
updateCounter();
echo $counter; // 10
```

Global Scope

- Variables declared outside a function are global
  - Can be accessed from any part of the program
  - Use the global keyword inside the function to access global variables
  - Cumbersome way to update the global variable is to use PHP's **`$GLOBALS`**: **`$GLOBALS[counter]`**
  - **WARNING!** Avoid using global variables

```
function updateCounter() {
   global $counter;
   $counter++; // or $GLOBALS['counter']++;
}
$counter = 10;
updateCounter();
echo $counter; // 11
```

Static Variables

- Static variables retain their values between calls to a function
  - Visible only within the function where defined
  - Declare a variable static with the static keyword

```
function updateCounter() {
   static $counter = 0;
   $counter++;
   echo "Static counter: {$counter}\n";
}
$counter = 10;
updateCounter();
updateCounter();
echo "Global counter: {$counter}\n";
```

Output:

```
Static counter: 1
Static counter: 2
Global counter: 10
```

PHP Constants

- In PHP constants are defined with the **`define`** function

```
<?php
define("CONSTANT", "Hello world.");
echo CONSTANT; // outputs "Hello world."
define("GREETING", "Hello you.", true); // not recommended
echo GREETING; // outputs "Hello you."
echo Greeting; // outputs "Hello you."
?>
```

  – Constant values cannot be changed
  – Doesn't start with **`$`**
  – Can hold any scalar value

OPERATORS IN PHP

Arithmetic, Logical, Comparison, Assignment, …

What is an Operator?

- Operator is an operation performed over data at runtime
  - Takes one or more arguments (operands)
  - Produces a new value
- Operators have precedence
  - Precedence defines which will be evaluated first
- Operators are used to build expressions
  - Expressions are sequences of operators and operands that are evaluated to a single value

Categories of Operators in PHP

Category	Operators
Arithmetic	`+ - * / % ++ --`
Logical	`&& || ! xor`
Binary	`& | ^ ~ << >>`
Comparison	`== != < > <= >= === !==`
Assignment	`= += -= *= /= %= &= |= ^= <<= >>=`
String concatenation	`.`
Other	`-> [] () ?: new`

Operators Precedence

Precedence	Operators
Highest	( )
	++ -- (postfix)
	++ -- (prefix) + - (unary) !
	* / %
	+ -
	<< >>
	< > <= >=
	== !=
Lower	&

Operators Precedence (2)

Precedence	Operators
Higher	^
	|
	&&
	||
	?:
Lowest	= *= /= %= += -= <<= >>= &= ^= |=

- Parenthesis operator always has the highest precedence
- Operator precedence and associativity !== order of evaluation
- Note: prefer using parentheses, even when it seems stupid to do so

Arithmetic Operators

- Arithmetic operators **+**, **-**, *****, **/** are the same as in math
- The division operator **/** returns number
  - Division / **0** returns **false** and "Division by zero" warning
- Remainder operator **%** returns the remainder from division
  - E.g. 5 % 3 → 2
- The operator **++** / **--** increments / decrement a variable
  - Prefix **++** vs. postfix **++**

Logical Operators

- Logical operators take boolean operands and return boolean result
- Operator **!** turns **true** to **false** and **false** to **true**
- Behavior of the operators **&&**, **||** and **xor** (**1** == **true**, **0** == **false**):

Operation	||	||	||	||	&&	&&	&&	&&	xor	xor	xor	xor
Operand1	0	0	1	1	0	0	1	1	0	0	1	1
Operand2	0	1	0	1	0	1	0	1	0	1	0	1
Result	0	1	1	1	0	0	0	1	0	1	1	0

Bitwise Operators

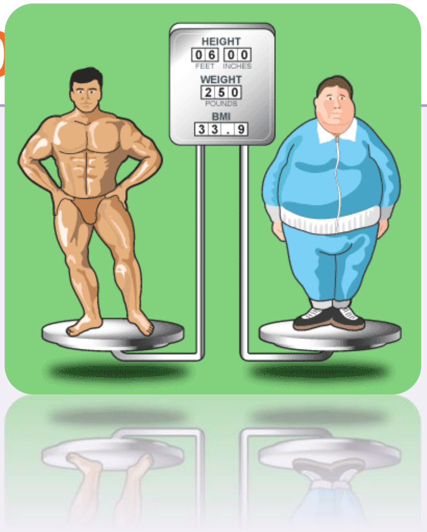
- Bitwise operator **~** turns all **0** to **1** and all **1** to **0**
  - Like **!** for boolean expressions but works bit by bit
- The operators **|**, **&** and **^** behave like logical **||**, **&&** and **xor**
- The **<<** and **>>** move the bits (left or right)
- Behavior of the operators **|**, **&** and **^**:

Operation	|	|	|	|	&	&	&	&	^	^	^	^
Operand1	0	0	1	1	0	0	1	1	0	0	1	1
Operand2	0	1	0	1	0	1	0	1	0	1	0	1
Result	0	1	1	1	0	0	0	1	0	1	1	0

Comparison Operators

- Comparison operators are used to compare variables
  - ==, <, >, >=, <=, !=, ===, !==
- The == means "equal after type conversion"
- The === means "equal and of the same typ

```
$a = 5;
$b = 4;
var_dump($a >= $b); // bool(true)
var_dump($a != $b); // bool(true)
var_dump($a == $b); // bool(false)
var_dump($a == "5"); // bool(true)
var_dump($a === "5"); // bool(false)
```



Assignment Operators

- Assignment operators are used to assign a value to a variable
  - =, +=, -=, |=, ...
- Assignment operators examples:

```
$x = 6;
$y = 4;
echo($y *= 2); // 8
$z = $y = 3; // $y = 3; $z = 3;
echo($z); // 3
echo($x |= 1); // 7
echo($x += 3); // 10
echo($x /= 2); // 5
```

Other Operators

- String concatenation operator **.** is used to concatenate strings
- If the second operand is not a string, it is converted to string automatically
- Member access operator **->** is used to access object members
- Square brackets **[]** are used with arrays to access element by index
- Parentheses **()** are used to override the default operator precedence

```
$output = "The number is : ";
$number = 5;
echo($output . $number);
// The number is : 5
```

Other Operators (2)

- Ternary operator ?: has the form:

```
$b ? $x : $y
```

  - If **b** is **true** then the result is **x,** else the result is **y**
- The **new** operator is used to create new objects
- **this** operator references the current context
- Spaceship operator in PHP 7 <=>

```
echo 1 <=> 1; // 0
echo 1 <=> 2; // -1
echo 2 <=> 1; // 1
```

Other Operators (3)

- **Null coalescing operator in PHP 7 ??**

```
$username = $_GET['user'] ?? 'nobody';

// This is equivalent to:
$username = isset($_GET['user']) ? $_GET['user'] : 'nobody;
// Chaining
$username = $_GET['user'] ?? $_POST['user'] ?? 'nobody';
```

Other Operators (4)

```
$a = 6;
$b = 4;
echo($a > $b ? "a > b" : "b >= a"); // a > b
echo "SoftUni" == "SoftUni" ? "Equal" : "Not Equal";
$c = $b = 3; // b = 3; followed by c = 3;
echo($c); // 3
echo(($a + $b) / 2); // 4.5
echo(gettype($a)); // integer
echo(gettype([])); // array
```

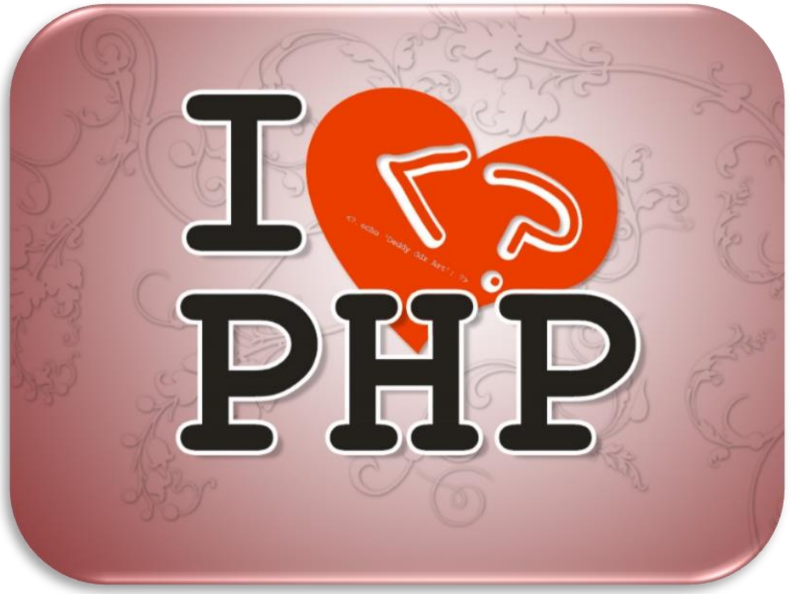
EXPRESSIONS

Expressions

- Expressions are:
  - Sequences of operators, literals and variables that are evaluated to some value
- Examples:

```
$r = (150 - 20) / 2 + 5; // r = 70

// Expression for calculation of circle area
$surface = pi() * $r * $r;

// Expression for calculation of circle perimeter
$perimeter = 2 * pi() * $r;
```

ACCESSING FORM FIELDS FROM PHP



Reading and Writing Form Data

Accessing Forms Fields

- You can access the form fields by their name property
- HMTL:

```
<form action="TakePostRequest.php" method="get">
    Name: <input type="text" name="name"><br>
    E-mail: <input type="text" name="email"><br>
    <input type="submit">
</form>
```

- PHP:

```
Welcome <?php echo htmlspecialchars($_GET["name"]) ?><br>
Your email is: <?php echo htmlspecialshars($_GET["email"]) ?>
```

Summary

- PHP dynamic data types
  - number, string, boolean, null, array, object
- Operators are similar to C#, Java and C++
- Expressions are as in C#, Java and C++
- Form fields can be accessed by `$_GET['key']` and `$_POST['key']`