Language basics

1PHPD

1PHPD – Language Basics

Course Objectives

By the end of the course, students should:

- Understand default operations
- Be able to write simple PHP code

Time:

- course: 3h
- exercises: 2h

Summary

1. Syntax
2. Variables
3. Mathematical Operations
4. Conditions

1. Syntax

1. Syntax

PHP syntax defines the set of rules for how to write instructions that will be executed by the PHP engine.

Understanding the basic syntax is crucial for writing PHP scripts.

When naming a PHP file, there are a few conventions and rules you should follow to ensure your files are well-organized, understandable, and compatible with web servers and PHP interpreters.

1. Syntax

Use Meaningful Names

Choose names that reflect the purpose or content of the file.

This practice makes it easier to identify what the file does at a glance, improving maintainability and readability of your codebase.

For example, if your file handles user registration, a name like register_user.php is descriptive and clear.

1. Syntax

Use Lowercase Letters

It's a common convention to use lowercase letters for file names in PHP and other web technologies.

This convention helps avoid confusion on case-sensitive operating systems, where File.php and file.php would be considered different files.

1. Syntax

Use Underscores for Separation

When your file name comprises multiple words, use underscores _ to separate them.

This approach enhances readability and is widely adopted in PHP community. For example, use user_profile.php instead of userprofile.php or UserProfile.php.

1. Syntax

Stick to the .php Extension

PHP files must have the .php extension.

This extension tells the server that the file should be processed by the PHP engine.

Even if your PHP file contains HTML code, as long as it includes PHP code or needs to be interpreted by PHP, it should use the .php extension.

1. Syntax

Avoid Using Spaces

Spaces in file names can cause issues, especially in URLs.

Use underscores _ instead of spaces. For example, instead of naming a file My File.php, name it my_file.php.

1. Syntax

Keep It Short and Simple

While descriptive names are helpful, excessively long file names can be cumbersome to work with.

Aim for a balance between descriptiveness and brevity.

1. Syntax

Avoid Special Characters

Special characters, such as ?, %, *, :, |, ", <, >, and \, can cause problems because they have specific meanings in file systems and URLs.

Stick to letters, numbers, underscores, and dashes to avoid issues.

1. Syntax

Here are a few examples of well-named PHP files:

- index.php - A common name for the main entry point of a website.
- contact_form.php - A file that handles the display or processing of a contact form.
- config.php - A configuration file that might include settings or connection details.
- login_process.php - A script that processes login data.

1. Syntax

Naming your PHP files with care is an important part of developing a clean, maintainable web application - like in all languages.

By following these conventions and best practices, you can ensure that your files are easily identifiable, accessible, and free from compatibility issues.

1. Syntax

In PHP, a "tag" refers to a special delimiter that encloses PHP code within a document, signaling to the PHP engine that it should start interpreting the code enclosed between these tags as PHP code.

These tags differentiate PHP code from other types of code in a file, such as HTML.

1. Syntax

The most common and universally supported PHP tags are

- the opening <?php tag
- the closing ?> tag

These are known as the standard PHP tags.

When the PHP interpreter encounters <?php, it knows that the following code, up until it reaches the ?> closing tag, should be executed as PHP.

1. Syntax

```php
<?php
// PHP code goes here
?>
```

1. Syntax

PHP also supports a shorter form of the opening tag, known as the short open tag, which is simply <?.

However, using the short open tag is discouraged because it requires the short_open_tag directive to be enabled in the php.ini file.

Additionally, it can lead to compatibility issues with XML parsers and is not as portable across different server configurations

1. Syntax

```
<?
// PHP code goes here
?>
```

1. Syntax

Another form of shorthand syntax is the echo short tag <?= which is used to directly output the expression that follows it.

This tag is equivalent to <?php echo.

The echo short tag is always available as of PHP 5.4.0, regardless of the short_open_tag setting

1. Syntax

```
<?= "Hello, World!" ?>
```

1. Syntax

The recommended practice is to use the full <?php opening tag for maximum compatibility and clarity, especially in shared environments or open-source projects where you cannot guarantee the server configuration.

1. Syntax

Comments in PHP are similar to those in other programming languages and can be used to explain sections of code or to temporarily disable code from executing.

PHP supports single-line and multi-line comments

1. Syntax

```
// This is a single-line comment

# This is also a single-line comment

/*
This is a multi-line comment
that spans over multiple
lines.
*/
```

1. Syntax

“echo” and “print” are used to output data to the screen.

“echo” can take multiple parameters (although not commonly used this way), whereas “print” can only take one argument and always returns 1, making it a bit different in terms of behavior.

1. Syntax

```php
echo "Hello, World!";
print "Hello, World!";
```

1. Syntax

As in C or Perl, PHP requires instructions to be terminated with a semicolon at the end of each statement.

The closing tag of a block of PHP code automatically implies a semicolon; you do not need to have a semicolon terminating the last line of a PHP block.

1. Syntax

The closing tag of a PHP block at the end of a file is optional, and in some cases omitting it is helpful when using include or require, so unwanted whitespace will not occur at the end of files, and you will still be able to add headers to the response later.

It is also handy if you use output buffering, and would not like to see added unwanted whitespace at the end of the parts generated by the included files.

1. Syntax

Embedding PHP code within an HTML document is a common practice in web development, enabling dynamic content generation.

This approach allows you to take advantage of PHP's server-side processing capabilities directly within an HTML file structure, making web pages more interactive and responsive to user input or other conditions.

1. Syntax

To embed PHP code in HTML, you use the PHP opening and closing tags <?php and ?>.

The PHP interpreter processes anything between these tags, while the rest of the document is treated as plain HTML.

1. Syntax

```
<!DOCTYPE html>
<html>
<head>
    <title>PHP in HTML Example</title>
</head>
<body>

    <h1>My First PHP Page</h1>

    <p>Today's date is <?php echo date('l, F jS, Y'); ?>.</p>

</body>
</html>
```

1. Syntax

You can freely mix PHP and HTML in the same file. PHP code blocks can be placed anywhere in the document.

In the next slide, we will see a loop over an “li” item, generating a dynamic list

1. Syntax

Example

```
<ul>
    <?php for ($i = 1; $i <= 5; $i++): ?>
        <li>Item <?php echo $i; ?></li>
    <?php endfor; ?>
</ul>
```

1. Syntax

You can also use PHP to dynamically generate HTML attributes.

1. Syntax

```
<div class="<?php echo $divClass; ?>"></div>
```

1. Syntax

For a PHP interpreter to process PHP code within an HTML file, the file typically needs to be named with a .php extension, not .html.

This instructs the server to pass the file through the PHP engine.

By embedding PHP code in HTML, you can create dynamic, interactive web pages that respond to user data, making your website more engaging and personalized.

Exercises

2. Variables

2. Variables

In PHP, a variable is used to store data that can be used and manipulated within your script.

Variables in PHP are represented by a dollar sign followed by the name of the variable.

The variable name is case-sensitive and must start with a letter or an underscore, followed by any number of letters, numbers, or underscores.

2. Variables

Before we see how variables work, it is important to note that variables are stored in the memory of the server, meaning that if the power is lost or if the server is restarted, all content stored in variables is lost.

Everything stored in variables would be flushed. Like in other languages, if you need to store data for a long time you will have to use the filesystem or an external database.

2. Variables

Basic Rules for PHP Variables

- Start with a $ (dollar sign): Every variable name starts with a dollar sign.
- A letter or underscore: After the dollar sign, a variable name must start with either a letter (a-z, A-Z) or an underscore (_). It cannot start with a number.

2. Variables

Basic Rules for PHP Variables

- Case-sensitive: Variable names are case-sensitive. This means $age and $Age are considered two different variables.
- Contains letters, numbers, and underscores: After the first letter or underscore, you can use any combination of letters, numbers, and underscores.

2. Variables

```php
$name = "John Doe";   // String
$age = 30;            // Integer
$height = 5.11;       // Float
$is_student = false; // Boolean
```

2. Variables

Opposed to language like C and C++, variables in PHP do not need to be declared before adding a value to them.

PHP automatically initializes the variable at the moment you first assign a value to it.

2. Variables

PHP is a dynamically typed language, which means you don't have to declare the type of a variable in advance.

The variable's type is determined by the context in which the variable is used. PHP automatically converts the variable to the correct data type, depending on its value.

2. Variables

```php
$x = 4;          // $x is an integer
$x = "four";     // Now $x is a string
```

2. Variables

PHP supports several data types, which are used to define the type of value that a variable can hold.

Understanding these data types is fundamental to PHP programming, as they affect how data is manipulated and operated on within your scripts.

PHP data types are broadly categorized into three groups: scalar types, compound types, and special types.

2. Variables

In the Scalar type we will find

- Integer
- Float
- String
- Boolean

2. Variables

Integer

An integer is a whole number without a decimal point, including positive, negative, and zero.

Integers can be specified in decimal (base 10), hexadecimal (base 16, prefixed with 0x), octal (base 8, prefixed with 0), and binary (base 2, prefixed with 0b) notation.

2. Variables

```php
$int1 = 42; // decimal
$int2 = 0xFF; // hexadecimal
$int3 = 052; // octal
$int4 = 0b101010; // binary
```

2. Variables

Float (double)

A floating-point number or a number with a decimal point or in exponential form.

Floats are used for representing fractional numbers.

2. Variables

```
$float = 10.5;
```

2. Variables

String

A sequence of characters where each character is the same as a byte.

This means that PHP supports a 256-character set, and hence can also be used to store binary data.

Strings can be specified using single quotes, double quotes, heredoc syntax, or nowdoc syntax.

2. Variables

```php
$string1 = 'This is a simple string';
$string2 = "This string includes $variables";
```

2. Variables

Boolean

A boolean expresses a truth value and can be either TRUE or FALSE.

Booleans are often used in conditional testing.

2. Variables

```php
$bool = true;
```

2. Variables

In the Compound type we will find

- Array
- Object

2. Variables

Array

An ordered map comprising values where each value can be accessed using its corresponding key.

Arrays in PHP are zero-indexed and can contain elements of different types.

2. Variables

```php
$array = array("foo", "bar", 42, 24);
```

2. Variables

Object

An instance of a class.
Objects are used to store both data and information on how to process that data (methods).

2. Variables

```php
class Car {
    function Car() {
        $this->model = "VW";
    }
}
$herbie = new Car();
```

2. Variables

In the Special type we will find

- Null
- Resources

2. Variables

NULL:

A special type that only has one value: NULL.
It is used to represent a variable with no value.

2. Variables

```php
$var = NULL;
```

2. Variables

Resource

A special variable, holding a reference to an external resource.

Resources are used to store references to functions and resources external to PHP, such as database connections.

2. Variables

```php
$file = fopen("log.txt", "r");
```

2. Variables

PHP supports variable variables, which means the name of a variable can be dynamic and stored in another variable.

In the next example, $$a creates a new variable named hello (the value of $a), and assigns it the value "world".

2. Variables

```php
$a = "hello";
$$a = "world";
echo "$a ${$a}"; // Outputs: hello world
echo "$a $hello"; // Outputs: hello world
```

2. Variables

In PHP, the location where variables are stored depends on their context and type, such as whether they're local variables, global variables, session variables, or others.

Understanding where these variables are stored helps in grasping how PHP manages data and memory.

2. Variables

Local variables are declared within a function and are stored in the stack.

The stack is a region of memory that stores temporary variables created by each function (including the main() function).

Each time a function is called, a new block is reserved on the top of the stack for that function's local variables.

2. Variables

When the function returns, its block of memory is freed.

This makes stack-allocated memory management very efficient but also means that local variables do not retain their values once the function exits.

2. Variables

Global variables are declared outside any function and can be accessed from anywhere in the script.

They are stored in the heap, a larger, more flexible region of memory.

The heap is used for dynamic memory allocation, which includes global variables and any data that needs to persist for the duration of the program's execution.

2. Variables

However, because memory in the heap is managed manually (or semi-automatically in languages with garbage collection, like Java), it's generally slower to allocate and deallocate than stack memory.

2. Variables

Static variables are declared within a function, but unlike local variables, they do not lose their value when the function execution is completed.

Instead, their value is preserved between function calls.

Static variables are also stored in the heap, allowing them to retain their value throughout the program's execution.

Exercises

3. Mathematical Operation

3. Mathematical Operation

PHP provides a wide range of operators to perform mathematical operations, allowing you to manipulate numbers and perform calculations within your scripts.

Understanding these operators is fundamental to handling numerical data and performing tasks that require mathematical computations.

3. Mathematical Operation

Addition (+): Adds two operands.

```php
$result = 5 + 3; // $result is 8
```

3. Mathematical Operation

Subtraction (-): Subtracts the right operand from the left operand.

```
$result = 5 - 3; // $result is 2
```

3. Mathematical Operation

Multiplication (*): Multiplies two operands.

```
$result = 5 * 3; // $result is 15
```

3. Mathematical Operation

Division (/): Divides the left operand by the right operand.

```php
$result = 15 / 3; // $result is 5
```

3. Mathematical Operation

Modulus (%): Finds the remainder of the division of the left operand by the right operand.

```php
$result = 5 % 3; // $result is 2
```

3. Mathematical Operation

Exponentiation (**): Raises the left operand to the power of the right operand.

```
$result = 5 ** 3; // $result is 125
```

3. Mathematical Operation

Assignment operators are used to write a value to a variable.

PHP provides arithmetic assignment operators that combine arithmetic operations with assignment

3. Mathematical Operation

Addition assignment (+=): Adds the right operand to the left operand and assigns the result to the left operand.

```php
$a = 5;
$a += 3; // $a is now 8
```

3. Mathematical Operation

Subtraction assignment (-=): Subtracts the right operand from the left operand and assigns the result to the left operand.

```
$a = 5;
$a -= 3; // $a is now 2
```

3. Mathematical Operation

Multiplication assignment (*=): Multiplies the left and right operands and assigns the result to the left operand.

```
$a = 5;
$a *= 3; // $a is now 15
```

3. Mathematical Operation

Division assignment (/=): Divides the left operand by the right operand and assigns the result to the left operand.

```
$a = 15;
$a /= 3; // $a is now 5
```

3. Mathematical Operation

Modulus assignment (%=): Applies modulus operation and assigns the result to the left operand.

```
$a = 5;
$a %= 3; // $a is now 2
```

3. Mathematical Operation

Exponentiation assignment (**=): Raises the left operand to the power of the right operand and assigns the result to the left operand.

```
$a = 5;
$a **= 3; // $a is now 125
```

3. Mathematical Operation

Increment and decrement operators are used to increase or decrease a variable's value by one, respectively

3. Mathematical Operation

Increment (++): Increases the value of a variable by one.

```php
$a = 5;
$a++; // $a is now 6
```

3. Mathematical Operation

Decrement (--): Decreases the value of a variable by one.

```php
$a = 5;
$a--; // $a is now 4
```

3. Mathematical Operation

In addition to these operators, PHP provides a rich set of mathematical functions to perform more complex calculations, such as abs(), cos(), sin(), tan(), exp(), pow(), sqrt(), log(), and many others, which are part of the PHP Math extension.

3. Mathematical Operation

I

```php
echo sqrt(16); // Outputs: 4
echo pow(2, 3); // Outputs: 8
echo abs(-4.2); // Outputs: 4.2
```

Exercises

4. Conditions

4. Conditions

PHP conditions allow scripts to make decisions and execute different actions based on specific conditions.

These conditional statements are a fundamental part of PHP and any programming language, enabling dynamic and interactive web applications.

4. Conditions

The if statement executes some code if one condition is true.

```
if ($condition) {
    // code to be executed if condition is true
}
```

4. Conditions

The if...else statement executes some code if a condition is true and another code if the condition is false.

```
if ($condition) {
    // code to be executed if condition is true
} else {
    // code to be executed if condition is false
}
```

4. Conditions

The if...elseif...else statement executes different codes for more than two conditions.

```
if ($condition1) {
    // code to be executed if condition1 is true
} elseif ($condition2) {
    // code to be executed if the condition1 is false and condition2 is true
} else {
    // code to be executed if both condition1 and condition2 are false
}
```

4. Conditions

The switch statement is used to perform different actions based on different conditions.

It's a more efficient way to write a series of if...elseif...else statements when dealing with variable comparisons.

4. Conditions

```php
switch ($variable) {
    case value1:
        // code to be executed if $variable == value1
        break;
    case value2:
        // code to be executed if $variable == value2
        break;
    ...
    default:
        // code to be executed if $variable is different from all labels
}
```

4. Conditions

The ternary operator is a shorthand for the if...else statement, useful for simple conditions.

```php
// Syntax: condition ? exprIfTrue : exprIfFalse;

echo $condition ? "Condition is true" : "Condition is false";
```

4. Conditions

PHP 7 introduced the null coalescing operator (??) as a shorthand for common usage of ternary operations with isset().

It returns its first operand if it exists and is not NULL; otherwise, it returns its second operand.

```php
// Equivalent to: isset($a) ? $a : 'default';
echo $a ?? 'default';
```

4. Conditions

Conditional statements are crucial for controlling the flow of execution in a PHP script, allowing for more complex, dynamic, and responsive web applications.

They enable developers to write code that can adapt to different inputs or situations, making the web experience more interactive and tailored to the user's context.

4. Conditions

PHP conditional statements often involve comparison and logical operators to evaluate conditions and make decisions based on those evaluations.

The operators are split in two mains categories

- comparison operators
- logical operators

Let’s first see the comparison operators.

4. Conditions

Equal (==): Checks if the value of two operands are equal or not. If yes, then the condition becomes true.

```
if ($a == $b)
```

4. Conditions

Identical (===): Checks if the value and the type of two operands are equal or not. If yes, then the condition becomes true.

```
if ($a === $b)
```

4. Conditions

Not Equal (!= or <>): Checks if the value of two operands are equal or not. If values are not equal, then the condition becomes true.

```
if ($a != $b)
```

4. Conditions

Not Identical (!==): Checks if the value and the type of two operands are not equal. If either the value or the type is different, then the condition becomes true.

```php
if ($a !== $b)
```

4. Conditions

Greater Than (>): Checks if the value of the left operand is greater than the value of the right operand. If yes, then the condition becomes true.

```
if ($a > $b)
```

4. Conditions

Less Than (<): Checks if the value of the left operand is less than the value of the right operand. If yes, then the condition becomes true.

```
if ($a < $b)
```

4. Conditions

Greater Than or Equal To (>=): Checks if the value of the left operand is greater than or equal to the value of the right operand. If yes, then the condition becomes true.

```
if ($a >= $b)
```

4. Conditions

Less Than or Equal To (<=): Checks if the value of the left operand is less than or equal to the value of the right operand. If yes, then the condition becomes true.

```
if ($a <= $b)
```

4. Conditions

And now the list of logical operators. They can also be mixed with comparison operators and the normal priority applies.

4. Conditions

And (&& or and): The condition becomes true if both the operands are true.

```
if ($a && $b)
```

4. Conditions

Or (|| or or): The condition becomes true if either of the operands is true.

```
if ($a || $b)
```

4. Conditions

Not (!): The condition becomes true if the operand is false.

```
if (!$a)
```

4. Conditions

XOR (xor): The condition becomes true if either of the operands is true, but not both.

```
if ($a xor $b)
```

4. Conditions

If using PHP to control conditional display on the HTML side, it can be used.

PHP can control the display of HTML elements based on certain conditions.

4. Conditions

```
<?php if ($loggedIn): ?>
    <h2>Welcome back, user!</h2>
<?php else: ?>
    <h2>Please log in.</h2>
<?php endif; ?>
```

Exercises