General Syntax and Data Types

Disclaimer: вы смотрите просто запись лекции, это **НЕ** специально подготовленный видеокурс!

PHP Tags

When PHP parses a file, it looks for opening and closing tags, which tell PHP to start and stop interpreting the code between them. Everything outside of a pair of opening and closing tags is ignored by the PHP parser.

```
<?php
// this is the opening tag

// PHP code here...

// this is the closing tag (NEVER USE IT!!!)
?>

<?= 'this is the equivalent of echo' ?>
<?php echo 'this approach is similar to the previous line' ?>
```

Why shouldn’t we ever use **?>** tag?!

With ?> tag

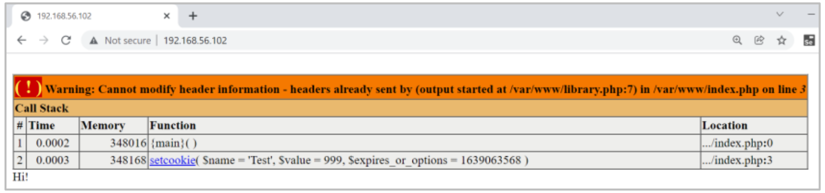
```php
<?php
function someCoolFunction(): void
{
}
?>

```

Empty line here ☹

library.php

```php
<?php
include 'library.php';
setcookie("Test", 999, time() + 3600);
echo "Hi!";
?>
```

index.php



```
192.168.56.102
Not secure | 192.168.56.102

( ! ) Warning: Cannot modify header information - headers already sent by (output started at /var/www/library.php:7) in /var/www/index.php on line 3
Call Stack
# Time    Memory  Function                                                                      Location
1 0.0002  348016  {main}( )                                                                     .../index.php:0
2 0.0003  348168  setcookie( $name = 'Test', $value = 999, $expires_or_options = 1639063568 )  .../index.php:3
Hi!
```

Without ?> tag

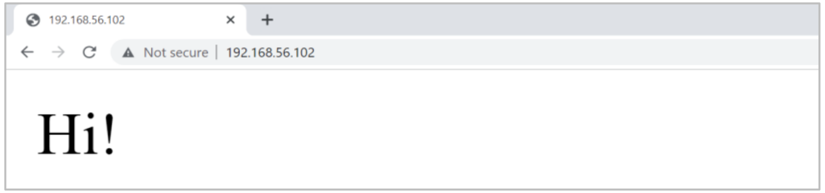
```php
<?php
function someCoolFunction(): void
{
}


```

Empty lines here ☺

library.php

```php
<?php
include 'library.php';
setcookie("Test", 999, time() + 3600);
echo "Hi!";
```

index.php



```
192.168.56.102
Not secure | 192.168.56.102

Hi!
```

PHP Comments

PHP supports C/C++ and Unix shell-style (Perl style) comments.

```
<?php

echo 'This is a test'; // This is a one-line C++ style comment
/* This is a multi line comment
   yet another line of comment */
echo 'This is yet another test';
echo 'One Final Test'; # This is a one-line shell-style comment
```

PHP Case Sensitivity

Most PHP syntax is case-sensitive (yes, there are some exceptions, but let’s assume that all PHP code is case-sensitive).

```
<?php

$var1 = 1;
$vAr1 = 1;
```

These are two different variables!

PHP Variables and Data Types

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

The variable name is case-sensitive.

A valid variable name starts with a letter or underscore, followed by any number of letters, numbers, or underscores.

```php
<?php

$someVariable = 1;
```

PHP Variables and Data Types

PHP allows a variable to change its type.

PHP (for now?) allows to use variables of different types in a single expression.

```php
<?php

$someVariable = 999;
$someVariable = true;
$otherVariable = "55 Test";
$result = $someVariable + $otherVariable; // It works :)
// Still it produces a warning and soon may raise an exception.
```

PHP Variables and Data Types

"Ordinary variables" do not require declaration in PHP, we only have to initialize them. Class properties require declaration.

```php
<?php

class SampleClass
{
    // Class properties, explicit declaration
    private static string $userName;
    private static ?string $userNickname; // May be null

    function doSomething() : void
    {
        $someVariable = 999; // "Ordinary variable", no explicit declaration
    }
}
```

Variable variables

This is dangerous! Do NOT do it until you absolutely have to, and you clearly understand all possible consequences!

```php
<?php

$initialVariable = 'Test';
$trickyVariable = 'initialVariable';
$evenMoreTrickyVariable = 'trickyVariable';

echo $initialVariable; // 'Test'
echo $$trickyVariable; // 'Test'
echo $$$evenMoreTrickyVariable; // 'Test'
```

PHP Data Types

Scalar	Compound	Special	“Pseudo”
boolean	array	resource	mixed
integer	object	null	number
float	iterable		
string	callable		
	enumeration		

For documentation only!

Not really a “data type”…

PHP Data Types: Boolean (bool)

A bool expresses a truth value. It can be either **true** or **false**.

```php
<?php

$someTrueVar = true;
$someFalseVar = false;
```

```php
<?php

var_dump((bool) "");        // bool(false)
var_dump((bool) "0");       // bool(false)
var_dump((bool) 1);         // bool(true)
var_dump((bool) -2);        // bool(true)
var_dump((bool) "foo");     // bool(true)
var_dump((bool) 2.3e5);     // bool(true)
var_dump((bool) array(12)); // bool(true)
var_dump((bool) array());   // bool(false)
var_dump((bool) "false");   // bool(true)
```

See “Data Types Detection and Conversion” later…

PHP Data Types: Integer (int)

An int is a number of the set ℤ = {..., -2, -1, 0, 1, 2, ...}.

```php
<?php

$integerVar = 1234;   // decimal number
$integerVar = 0123;   // octal number (equivalent to 83 decimal)
$integerVar = 0o123; // octal number (as of PHP 8.1.0)
$integerVar = 0x1A;   // hexadecimal number (equivalent to 26 decimal)
$integerVar = 0b11111111; // binary number (equivalent to 255 decimal)
$integerVar = 1_234_567;  // decimal number (as of PHP 7.4.0)
```

PHP Data Types: Float (float)

Floating point numbers are also known as “floats”, “doubles”, or “real numbers”.

```php
<?php

$floatVar = 1.234;
$floatVar = 1.2e3;
$floatVar = 7E-10;
$floatVar = 1_234.567; // as of PHP 7.4.0
```

PHP Data Types: String (string)

A string is series of characters, where a character is the same as a byte. This means that PHP only supports a 256-character set, and hence does not offer native Unicode support (yet?). *As of PHP 7.0.0, there are no particular restrictions regarding the length of a string on 64-bit builds.*

```php
<?php

$aSingleQuotedString = 'Some string';
$aDoubleQuotedString = "Some string";
```

We'll discuss the difference in a moment…

Please, use this syntax only. STOP using "heredoc" and "nowdoc" for the God's sake! Just STOP IT!

PHP Data Types: Single-quoted strings

Just a set of symbols.

No “internal processing”.

```
<?php

$someVariable = 999;
$aSingleQuotedString = 'Some \'string\' with \n and $someVariable';
var_dump($aSingleQuotedString);
// string(39) "Some 'string' with \n and $someVariable"
```

PHP Data Types: Double-quoted strings

A set of symbols with…

“internal processing”.

```
<?php

$someVariable = 999;
$aDoubleQuotedString = "Some \"string\" with \n and $someVariable";
var_dump($aDoubleQuotedString);
// string(28) "Some "string" with
// and 999"
```

PHP Data Types: Single-quoted strings and double-quoted strings

```php
<?php

// Just some cheat-sheet:
$aSingleQuotedString = 'Test';
$aDoubleQuotedString = "Test";
$aDoubleQuotedString = "Hotel \"Minsk\" and hotel 'Moscow'";
$aSingleQuotedString = 'Hotel "Minsk" and hotel \'Moscow\'';
$aDoubleQuotedString = "His age is $age";
$aDoubleQuotedString = "The value is {$values['val']}";
$aDoubleQuotedString = "Two\nlines";
```

PHP Data Types: String (string), accessing symbols (bytes)

In a string one may access a symbol (byte) by its number (starting from 0).

```php
<?php

$singleByteEncoding = 'Test';
$multiByteEncoding = 'Тест';

echo $singleByteEncoding[2]; // s
echo $multiByteEncoding[2]; // �

$singleByteEncoding[2] = 'z';
$multiByteEncoding[2] = 'z';

echo $singleByteEncoding; // Tezt
echo $multiByteEncoding; // Tz�ст
```

PHP Data Types: String (string), numeric strings

A PHP string is considered numeric if it can be interpreted as an int or a float.

```
<?php
$x = 1 + "10.5";                // $x is float (11.5)
$x = 1 + "-1.3e3";              // $x is float (-1299)
$x = 1 + "bob-1.3e3";           // TypeError as of PHP 8.0.0, $x is integer (1) previously
$x = 1 + "bob3";                // TypeError as of PHP 8.0.0, $x is integer (1) previously

$x = 1 + "10 Small Pigs";       // $x is integer (11) and an E_WARNING is raised in PHP 8.0.0,
                                // E_NOTICE previously

$x = 4 + "10.2 Little Piggies"; // $x is float (14.2) and an E_WARNING is raised in PHP 8.0.0,
                                // E_NOTICE previously

$x = "10.0 pigs " + 1;          // $x is float (11) and an E_WARNING is raised in PHP 8.0.0,
                                // E_NOTICE previously

$x = "10.0 pigs " + 1.0;        // $x is float (11) and an E_WARNING is raised in PHP 8.0.0,
                                // E_NOTICE previously
```

PHP Data Types: Array (array)

An **array** in PHP is actually an **ordered map**.

A map is a type that associates values to keys. This type is optimized for several different uses; it can be treated as an array, list (vector), hash table (an implementation of a map), dictionary, collection, stack, queue, and more. As array values can be other arrays, trees and multidimensional arrays are also possible.

```php
<?php

$someArray = array(123, true, "Test");
```

PHP Data Types: Array (array), some important facts

Always dynamic.

There are no static arrays in PHP at all.

Dimension-dynamic.

Any dimension may be modified instantly.

Keys may be either **integers** or **strings**.

PHP Data Types: Array (array), some important facts

Always dynamic.

```php
<?php

// No need to declare any "length" or anything like that,
// just use an index (or key) and put an element into your array.
$someArray[500] = 'Good';
$someArray[999] = 'OK';
```

PHP Data Types: Array (array), some important facts

Dimension-dynamic.

```php
<?php

// No need to pre-declare new dimension (e.g. unlike in Java collections),
// just use an index (or key) and put an element into your array.
$someArray[500] = 'Good';
$someArray[999][12] = 'OK';
$someArray[5][21][77][0][16][56][21][1][1][1] = 'OK ;)';
```

PHP Data Types: Array (array), some important facts

Keys may be either **integers** or **strings**.

```
<?php

$someArray[500] = 'Good';
$someArray['name'] = 'John';
$someArray['surname'] = 'Smith';
```

“Scientifically speaking”, **integers** are ***indexes***, and **strings** are ***keys***, still there is no difference in PHP…

PHP Data Types: Array (array), some important facts

Keys may be either **integers** or **strings**.

```
<?php

$someArray[5] = 'Test';       // (int) 5
$someArray['7'] = 'Test';     // (int) 7
$someArray['09'] = 'Test';    // (string) '09'
$someArray[true] = 'Test';    // (int) 1
$someArray[false] = 'Test';   // (int) 0
$someArray[12.7] = 'Test';    // (int) 12
$someArray[null] = 'Test';    // (string) ""
```

Beware of type casting! (We shall discuss it soon.)

PHP Data Types: Array (array), declaration (initialization)

You may use either “array” syntax, or [ ] syntax:

```php
<?php

// These arrays are the same:
$someArrayOne = array('John', 'Smith');
$someArrayTwo[] = 'John';
$someArrayTwo[] = 'Smith';

// These arrays are the same:
$someArrayThree = array('name' => 'John', 'surname' => 'Smith');
$someArrayFour['name'] = 'John';
$someArrayFour['surname'] = 'Smith';

// These arrays are the same:
$someArrayFive = ['John', 'Smith'];
$someArraySix = ['name' => 'John', 'surname' => 'Smith'];
```

PHP Data Types: Array (array), declaration (initialization)

PHP may assign an index automatically:

```php
<?php

$someArrayOne = array(
    'one' => 'ONE',
    'two' => 'TWO',
    100   => 100,
    -100  => -100,
    'Test' // Key (index) is 101
);

$someArrayTwo = ['one' => 'ONE',
                 'two' => 'TWO',
                  100  => 100,
                 -100  => -100,
                 'Test'];  // Key (index) is 101
```

PHP Data Types: Array (array), declaration (initialization)

Beware of indexes auto-assignment in multidimensional arrays!

```php
<?php

// Beware of indexes auto-assignment in multidimensional arrays!
$multidimensionalArray[10][15] = 'A';
$multidimensionalArray[10][] = 'B'; // $multidimensionalArray[10][16] = 'B'
$multidimensionalArray[][] = 'C';    // $multidimensionalArray[11][0] = 'C'
```

PHP Data Types: Array (array), typical operations

Let’s see some typical operations with array elements:

```
<?php

// Add an element:
$someArray['name'] = 'John';
$someArray['surname'] = 'Smith';

// Iterate through an array:
foreach ($someArray as $key => $value) {
    echo 'Key = ' . $key . "\n";
    echo 'Value = ' . $value . "\n\n";
}

// Access an element:
echo $someArray['name'];

// Check if an element exists (by index):
if (isset($someArray['name'])) {
    echo $someArray['name'];
}

// Remove an element:
unset($someArray['name']);
```

PHP Data Types: Object (object)

In PHP an **object** is an instance of class.

```
<?php

class SampleClass
{


}

$someObject = new SampleClass();
```

We shall discuss classes and objects in details further in this course (in “OOP” part)…

PHP Data Types: Iterable (iterable)

Iterable is a pseudo-type. An **iterable** stands for an **array** or **object** implementing the Traversable interface. Both of these types are iterable using foreach and can be used with yield from within a generator.

```php
<?php

// The main idea is as follows:
if (is_iterable($something)) {
    foreach ($something as $key => $value) {
        echo 'Key = ' . $key . "\n";
        echo 'Value = ' . $value . "\n\n";
    }
}
```

PHP Data Types: Iterable (iterable)

Some more samples…

```php
<?php

// Yes, it's an iterable.
$someArray = [1, 5, 7];

// Yes, it's an iterable.
function justSomeFunction(): iterable {
    return [1, 5, 7];
}

// Yes, it's an iterable.
$someFunction = function()
{
    yield 1;
    yield 5;
    yield 7;
};
```

PHP Data Types: Iterable (iterable)

And even more samples…

```php
<?php

class ChattyArray implements Iterator // ... that extends Traversable
{
    private $someArray;

    public function __construct($givenArray)
    {
        $this->someArray = $givenArray;
    }

    #[ReturnTypeWillChange] function rewind()
    {
        return 'We are at the array beginning now, the value is ' . reset($this->someArray);
    }

    #[ReturnTypeWillChange] function current()
    {
        return 'We are at the same element now, the value is ' . current($this->someArray);
    }

    #[ReturnTypeWillChange] function key()
    {
        return 'We are at the same element now, the key is ' .  key($this->someArray);
    }

    #[ReturnTypeWillChange] function next()
    {
        return 'We are at the next element now, the value is ' . next($this->someArray);
    }

    #[ReturnTypeWillChange] function prev()
    {
        return 'We are at the previous element now, the value is ' .  prev($this->someArray);
    }

    function valid(): bool
    {
        return key($this->someArray) !== null;
    }
}

$chattyArray = new ChattyArray([2, 7, 8, 99]);
echo $chattyArray->next(); // We are at the next element now, the value is 7
```

PHP Data Types: Callable (callable)

In PHP a **callable** is something one may call (a function, a method).

```php
<?php

// Yes, it's a callable:
function someFunction()
{

}

// Yes, it's a callable:
$someAnotherFunction = function () {
};

class SomeClass
{
    // Yes, it's a callable:
    public function someMethod(): void
    {

    }
}
```

PHP Data Types: Enumeration (enum)

Not really a “data type”…

Enumerations are a restricting layer on top of classes and class constants, intended to provide a way to define a closed set of possible values for a type.

```
<?php

enum Season
{
    case Winter;
    case Spring;
    case Summer;
    case Autumn;
}

function doSomething(Season $season)
{
    // ...
}

doSomething(Season::Summer);
```

```
<?php

enum Color: string
{
    case DEFAULT = "\033[39m";
    case BLACK = "\033[30m";
    case RED = "\033[31m";
    case GREEN = "\033[32m";
    case YELLOW = "\033[33m";
    case BLUE = "\033[34m";
    case MAGENTA = "\033[35m";
    case CYAN = "\033[36m";
    case LIGHT_GRAY = "\033[37m";
    case DARK_GRAY = "\033[90m";
    case LIGHT_RED = "\033[91m";
    case LIGHT_GREEN = "\033[92m";
    case LIGHT_YELLOW = "\033[93m";
    case LIGHT_BLUE = "\033[94m";
    case LIGHT_MAGENTA = "\033[95m";
    case LIGHT_CYAN = "\033[96m";
    case WHITE = "\033[97m";
}
```

PHP Data Types: Resource (resource)

There is a tendency to replace resources with objects…

A resource is a special variable, holding a **reference to an external resource**. Resources are created and used by special functions.

Files, DBMS connections, images, SQL query results and so on…

```php
<?php

// $fileResource will be of a 'resource' type
$fileResource = fopen($_SERVER['PHP_SELF']); // Get a resource.

while (!feof($fileResource)) {
    echo fgets($fileResource); // Use a resource.
}
fclose($fileResource); // Release a resource.
```

There are only three types of operation available for a resource: **get** a resource, **use** a resource, **release** a resource.

PHP Data Types: Null (null)

The special **null** value represents a variable with no value: **null** is the only possible value of type **null**.

A variable is considered to be null if:

1. it has been assigned the constant null.
2. it has not been set to any value yet.
3. it has been unset().

PHP Data Types: Null (null)

This is how it looks like:

```php
<?php

// 1. it has been assigned the constant null:
$someVariable = 1;
$someVariable = null;

if (is_null($someVariable)) {
    echo '$someVariable is null.';
}

// 2. it has not been set to any value yet:
if (is_null($someAnotherVariable)) {
    echo '$someAnotherVariable is null.';
}

// 3. it has been unset():
$someYetAnotherVariable = 1;
unset($someYetAnotherVariable);

if (is_null($someYetAnotherVariable)) {
    echo '$someYetAnotherVariable is null.';
}
```

PHP Data Types: Mixed (mixed)

Mixed is equivalent to the union type object|resource|array|string|int|float|bool|null. Available as of PHP 8.0.0.

```php
<?php

function printSomething(mixed $something): void
{
    echo 'This is something: ' . $something . "\n";
}

printSomething(1);
printSomething(10.15);
printSomething(true);
printSomething('Test');
```

PHP Data Types: Number (number)

Number (used in documentation only!) is equivalent to the union type int|float.

Most likely you'll see **int|float** here in any up-to-date documentation.

```php
<?php

/**
 * @param number $number
 * @return void
 */
function printNumber(int|float $number): void
{
    echo 'This is number: ' . $number . "\n";
}

printNumber(1);
printNumber(10.15);
```

PHP Data Types: quick recap

Scalar	**Compound**	**Special**	**“Pseudo”**
boolean	array	resource	mixed
integer	object	null	number
float	iterable		
string	callable		
	enumeration		

For documentation only!

Not really a “data type”…

PHP Constants

A constant is an identifier (name) for a simple value. As the name suggests, **that value cannot change during the execution of the script** (except for magic constants, which aren't actually constants ☺). Constants are case-sensitive. By convention, constant identifiers are always uppercase.

```php
<?php

// Define a constant:
define('SOME_CONSTANT', 'ABC');

// Check if a constant is defined:
if (defined('SOME_CONSTANT')) {
    // Use a constant:
    echo SOME_CONSTANT;
}

// 'Magic constant' sample:
echo 'This is line number' . __LINE__;

// Predefined sample:
echo 'We are using PHP version ' . PHP_VERSION;
```

PHP Constants, “magic constants”

`__LINE__`	The current line number of the file.
`__FILE__`	The full path and filename of the file with symlinks resolved.
`__DIR__`	The directory of the file.
`__FUNCTION__`	The function name, or {closure} for anonymous functions.
`__CLASS__`	The class name (including the namespace if it was declared).
`__TRAIT__`	The trait name (including the namespace if it was declared).
`__METHOD__`	The class method name.
`__NAMESPACE__`	The name of the current namespace.
`ClassName::class`	The fully qualified class name.

Useful for logging, debugging, complex frameworks linking.

PHP Constants, useful pre-defined constants

See https://www.php.net/manual/en/reserved.constants.php:

```php
<?php

echo PHP_VERSION . "\n";                // 8.1.0
echo PHP_MAJOR_VERSION . "\n";          // 8
echo PHP_MINOR_VERSION  . "\n";         // 1
echo PHP_MAXPATHLEN . "\n";             // 2048
echo PHP_OS . "\n";                     // WINNT
echo PHP_OS_FAMILY . "\n";              // Windows
echo PHP_SAPI . "\n";                   // cli
echo PHP_EOL . "\n";                    // \r\n
echo PHP_INT_MAX . "\n";                // 9223372036854775807
echo PHP_INT_MIN  . "\n";               // -9223372036854775808
// ... and so on.
```

General Syntax and Data Types

Disclaimer: вы смотрите просто запись лекции, это **НЕ** специально подготовленный видеокурс!