Always code as if the person who will maintain your code is a maniac serial killer that knows where you live.

Martin Golding

1 The basics

1.1 Introduction

PHP is a server-side scripting language. It doesn't require compilation. Instead it compiles at runtime. Although all ObjectOriented design patterns can be applied in the latest versions of PHP , the language itself is rather easy to learn.

PHP is open-source. You can download and install it from `www.php.net`. Most of the time, PHP is used in combination with a webserver (apache, ...) and a database (mySQL, postgreSQL, ...). By using PHP in this combination, PHP is a common language to create dynamic websites.

1.2 Quick example

Listing 1.1 a quick example

```php
<?php // php-code has to start with this tag

// comments can be added if they start with a double slash forward (//)
/*
Or start with a slash and an asterisk
And end with an asterisk and a slash
*/

// this is the declaration of a variable of datatype string with a value 'Jan'
$surname = 'Jan';

// every line in php needs to end with a semicolon (;)
$street = 'Herestraat 49';

// echo prints text
echo 'hello, my name is ';

// echo also prints the value of a variable
echo $surname;

// you can also use double quotes (") for a string.
// a \n adds a line break to the output.
echo ", \n";

// strings can be glued together (concatinated) with a dot (.)
echo 'I live in ' . $street;

// this is the ending tag
?>
```

1.3 Creating and uploading files

Choose a texteditor, enter the lines of code of the quick example and save the file with .php as extension, e.g. test.php. Now you have your code.

To execute this code, you need a PHP interpreter.

There are a lot of possibilities: installing a webserver with php enabled, installing the php commandline interface (CLI), uploading your php-file to a webserver, ...

At KHLeuven you can use your webspace at `webontwerp.khleuven.be`. It is a webserver that is PHP -enabled.

The connection details are explained in the slides you can find here:

`https://dynweb.webontwerp.khleuven.be/dynweb/slides/00ftp-settings.pdf`

After uploading the file to your webspace, you can test it.

Open a browser and go to the your website:

`http://<studentnr>.webontwerp.khleuven.be/test.php`

You'll see something like this:

Fig. 1.1 webpage, marked up in a browser

All looks fine, but there is supposed to be a line break. You'll see that there is no layout, nor line breaks. Although we added a `\n`. In fact, there is a line break, but the browser is rendering the generated code as if it were HTML.

Take a look at the source code (this will be your friend for a long time).

Fig. 1.2 Source code in a browser

Here you will see the line break.

Try to understand the next schema. It explains the fact that PHP is a server-side scripting language. When the PHP code is compiled at runtime (whenever you request the file test.php), the generated code is sent to the client's browser. The browser looks at this received source code, interpretes the mark-up and displays it as an HTML-page.

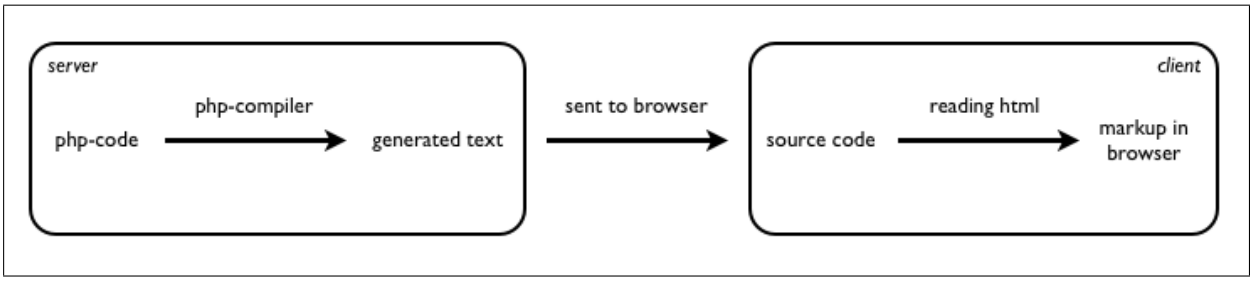

Fig. 1.3 simple display of process of generating dynamic webpage

In the upcoming lessons, the client-server model is explained in detail.

Because we don't echo html-code yet, the source code is the place to look for the output of your PHP -script.

Exercise 1.1

- Make sure you have connected to your webspace via ftp.
- Create a small PHP -file with some code like the quick example.
- Name this code `1.1.php` and place it in a subfolder `exercises` so that it is accessible at
  `http://<studentnr>.webontwerp.khleuven.be/exercises/1.1.php`

1.4 Basic syntax

A very brief description of the syntax of the php-language:

- PHP code starts with `<?php` and ends with `?>`.
- Every line needs to end with a semicolon: `;`.
- You can add as much whitespace, tabs, breaks as you want.
- Variable names are case sensitive.
- Function names are case insensitive.
- You can add comments in two ways:

  `//` the rest of this line will be treated as comment

  `/*` text between will be treated as comment `*/`

1.5 Variables

- A variable contains some value (or is empty).
- A variable has a name and starts with a dollar sign `$`.
- A variable is automatically declared when you first use it, no explicit declaration needed. PHP is a loosely typed language which means you can implicitly create variables and automatically convert them to another datatype. On the opposite there is e.g. JAVA which is a strongly type language.
- A valid variable name starts with a letter or underscore, followed by any number of letters, numbers or underscores. Variable names are case sensitive.

  As a regular expression, it would be expressed as:

  `[a-zA-Z_][a-zA-Z0-9_]*`

  If you can't read the regular expression above, start studying regex. You'll need it. Try e.g. `www.regular-expressions.info`

Listing 1.2 variable names

```
// good variable varnames
$varname
$varname2
$_varNAME
$_231223

// bad variable names (find out why)
$123
$da-bomb
$I'm-empty
```

A last, important remark:

PHP is loosely typed: you can change a variable's data type at any time. (in contrast to e.g. JAVA which is strongly typed). This is useful is some cases, but also very error-prone.

Listing 1.3 datatypes change to whatever needed

```
<?php

$variable = 'I like birds'; // variable is a string
$variable = 0.21; // variable becomes an integer

?>
```

1.5.1 Strings

A string is a serie of characters. Specified by single quotes

```
$mystring = 'some value';
```

or double quotes

```
$mystring = "some value";
```

There are two main differences:

- You can use special characters like `\n` (new line) in a string with double quotes. (See the quick example)
- When you include variables in a double-quoted string, the value of the variable is substituted into the string.
  E.g. a variable `$user` holds a value `Jan`.
  If you your code contains `echo 'hello $user';` this would print `hello $user`.
  But if your code used double quotes `echo "hello $user";` it would print `hello Jan`.

When you want to use a `'` or `"`, you have to escape it by using a backslash `\`.

```
$mystring = 'That\'s just the way.';
$mystring = "She just said: \"hello\"";
```

You can concatinate two strings together using a dot. It is ok to mix double and single quoted strings or variables in one concatination.

```
$mybigstring = 'Come play with me. ' . $myotherstring . " it's fun.";
```

You'll see that you only have to escape the single quote when using single quotes to define your string and vice versa. Choosing the correct quoting can save you a lot of escaping.

1.5.2 String manipulations

There are a lot of built-in PHP functions for manipulating and validating strings.

We give some interesting functions to start with. Have a look at each of them on `www.php.net`.

- `trim()`
- `strlen()`
- `printf()`
- `strlen()`
- `substr()`
- `str_replace()`
- `strpos()`

More on `www.php.net/manual/en/ref.strings.php`

Exercise 1.2

- Have a look at the functions above and perhaps at some more at `www.php.net/manual/en/ref.strings.php`.
- Download the source file from `http://dynweb.webontwerp.khleuven.be/exercises/1.2-source.txt`
- Use string manipulations to achieve the result you can see at `http://dynweb.webontwerp.khleuven.be/exercises/1.2.php`. (Look at the source-code)
- Upload your file so that it is accessible at `http://<studentnr>.webontwerp.khleuven.be/exercises/1.2.php`

1.5.3 Numbers

Numbers in PHP are epxressed using familiar notation: a dot to seperate the decimal part. No comma.

Listing 1.4 Numeric values

```
<?php

$mynumber = 132;
$mynumber = 11.2;
$mynumber = 0;
$mynumber = -5121;
$mynumber = -0.00;

?>
```

1.5.4 Arithmetic operators and functions

Arithmetic operators also are straight-forward.

Listing 1.5 arethmetic operators

```
<?php

echo 12 + 23;
echo 12 - 23;
echo 12 * 23;
echo 12 / 23;
echo 12 % 23; // this is a modulus division. If you don't know what it does, google it.

?>
```

If you combine arithmetic operators, the precedence of the operators follows mathematical guidelines. More info on `www.php.net/manual/en/language.operators.precedence.php`

You can overrule these precedence by using parentheses to group operators.

Find here ome interesting functions you can use with numeric values. Have a look at each of them on `www.php.net`.

- `is_numeric()`
- `is_int()`
- `pow()`
- `ceil()`
- `round()`
- `floor()`
- `dechex()`

More on www.php.net/manual/en/ref.math.php

Exercise 1.3

- Have a look at the functions above and perhaps at some more at www.php.net/manual/en/ref.math.php.
- Download the source file from http://dynweb.webontwerp.khleuven.be/exercises/1.3-source.txt
- Use Arithmetic operators and functions to achieve the result you can see at http://dynweb.webontwerp.khleuven.be/exercises/1.3.php. (Look at the source-code)
- Upload your file so that it is accessible at http://<studentnr>.webontwerp.khleuven.be/exercises/1.3.php

1.5.5 String and incrementing/decrementing operators

Have a look at www.php.net/manual/en/language.operators.string.php and www.php.net/manual/en/language.operators.increment.php.

Make sure you understand the difference between $a++ and ++$a.

1.5.6 Mixing operators, implicit dataconversion

Another important thing to know is the combination of strings and numeric values/operators. Look at the example below:

Listing 1.6 datatypes change to whatever needed

```
1 <?php
2
3 echo "thr"."ee";           //prints the string "three"
4 echo "twe" . "lve";        //prints the string "twelve"
5 echo 1 . 2;                //prints the string "12"
6 echo 1.2;                  //prints the number 1.2
7 echo 1+2;                  //prints the number 3
8
9 ?>
```

As you can see on line 5: if you use a string operator . between two numeric values, the numeric values are implicitly (and thus automatically) converted to strings because of the string operator. They are not mathematically added together, but concatinated.

Implicit conversion from string to numeric follows this rule:

- The string is read from left to right.
- The string is truncated from the first non-numeric character.
- When no numeric character is found, 0 is returned.

Examples:

- string `'123 hopla'` will be converted to the numeric value `123`
- string `'12.3 hopla'` will be converted to the numeric value `12.3`
- string `'12 hopla 3'` will be converted to the numeric value `12`
- string `'hopla'` will be converted to the numeric value `0`

If you mix operators in one expression, the operators are executed from left to right. If you want to make sure the arithmic operator is executed before the string operator, use brackets to prioritize the execution, as you can see in the example below:

Listing 1.7 datatypes change to whatever needed

```
<?php

echo 'this is number ' . 3 + 2;      //prints the number 2. Find out why!
echo 'this is number ' . (3 + 2);    //prints the string " this is number 5".

?>
```

1.5.7 If Then Else

The basic if-control structure is written like this:

Listing 1.8 if-then-else control structure

```
<?php

if ($a > $b) {
    echo "a is bigger than b";
} elseif ($a == $b) {
    echo "a is equal to b";
} else {
    echo "a is smaller than b";
}

?>
```

A shortlist of comparison operators:

- == : equal to (with conversion of datatype)
  - when datatypes are the same: conversion happens, see `http://be.php.net/types.comparisons` for a complete list of PHP type comparison tables.
  - use === to do a strict comparison. This means that the datatypes should be equal as well.
- >, <, >=, <=
  - string with string comparison: alphabetically, lowercase before uppercase
  - numeric with numeric comparison: mathematically
  - string with numeric comparison: mathematically, implicit conversion to force string comparison, use `strcmp()`

Listing 1.9 Common error, not using double equatation

```
<?php

if ($password = 'forgot it') {
    echo 'you are now logged in';
}

?>
```

Watch out for not using the double equatation ==. In the example above, a user is always logged in. Find out what exactly happens when you only use a single equatation =.

1.5.8 Logical Operators

- AND : `and` and &&
- OR : `or` and `||`
- NOT : `!`
- XOR : `xor`

Although you can use `and` and && (`or` and `||`) for the same purpose, they are not the same. They operate at different precedences. Best practice is to use && and `||`. If you want to know what happens when you mix them: read it here: `www.php.net/manual/en/language.operators.logical.php`.

1.5.9 True vs False

When using control structures and logical operators like `AND`, `OR` or `NOT`, it is important to know what happens when an implicit dataconversion happens to the datatype boolean.

Strings

All strings are converted to `TRUE`, except

- an empty string (the same as a string with `NULL` as value)
- a string containing `'0'`

Numeric

All numeric values are converted to `TRUE`, except

- an integer `0`
- a float `'0.00'`

Constant

All constants are converted to `TRUE`, except

- a constant with the value `FALSE`

Listing 1.10 Constants and their declaration

```php
<?php

// declaration of a constant
// it is good practice to use capitals in the name of the constant.
define('PI', 3.1415);

// constants don't have a leading $
echo PI;
?>
```

More on constants at `www.php.net/manual/en/function.constant.php`.

Arrays

All arrays are converted to `TRUE`, except

- an empty array

More on arrays at `www.php.net/manual/en/book.array.php` and in upcoming lessons.

Exercise 1.4

- Download the source file from `http://dynweb.webontwerp.khleuven.be/exercises/1.4-source.txt`
- Think out for every line if the variable will be converted to `TRUE` or `FALSE`. Test it.
- Remove the lines which return false and save your file.
- Upload your file so that it is accessible at `http://<studentnr>.webontwerp.khleuven.be/exercises/1.4.php`

1.5.10 Looping

PHP , like any other programming language, has a number of control structures for looping:

Listing 1.11 while control structure

```php
<?php

// $i has to be defined before the loop
$found = false;

// define the target
define('TARGET_TO_FIND', 33);

// $i is checked every time before the loop is executed
while ($found === false) {
        if ($i == TARGET_TO_FIND) {
                $found = true;
        }

    echo $i++;
}

?>
```

Listing 1.12 do-while control structure

```php
<?php

// $i has to be defined before the loop
$i = 1;

// $i is checked at the end of each iteration
do {
    echo $i++;
} while ($i <= 10)

?>
```

Listing 1.13 for-loop control structure

```php
<?php

// $i is defined once at the start of the loop
// $i is checked at the beginning of each loop
// $i is automatically incremented at the end of each loop
for ($i = 1; $i <= 10; $i++) {
    echo $i;
}

?>
```

If you want to stop the loop on a certain condition, you can use `break`.

`break` ends execution of the current for, foreach, while, do-while or switch structure. More info on `www.php.net/manual/en/control-structures.break.php`

Exercise 1.5

- Download the source file from
  `http://dynweb.webontwerp.khleuven.be/exercises/1.5-source.txt`
- Use a looping structure and string/numeric manipulations to produce the exact result as you can see at
  `http://dynweb.webontwerp.khleuven.be/exercises/1.5.php`.
  (Look at the source-code)
- upload your file so that it is accessible at
  `http://<studentnr>.webontwerp.khleuven.be/exercises/1.5.php`

Alternative notation for control structures

It can be interesting, e.g. when using php as a template language, to use the alternative syntax. Read it on `www.php.net/manual/en/control-structures.alternative-syntax.php`.