IPL
escola superior
de tecnologia e gestão
instituto politécnico
de leiria

PHP

Introduction and Language

Marco Monteiro

Contributors

- Author(s):
  - Marco Monteiro (marco.monteiro@ipleiria.pt)
- Contributor(s):
  - Vitor Carreira (vitor.carreira@ipleiria.pt)
  - Fernando Silva(vitor.carreira@ipleiria.pt)

Summary

1. Server-Side Programming
2. Introduction to PHP
3. Syntax and basic concepts of the language
4. Types & Operators
5. Arrays
6. Control structures
7. Functions
8. Blocks and files
9. Bibliography

1 – SERVER-SIDE PROGRAMMING

Dynamic Web Page

- A dynamic web page is a kind of web page where content is prepared (fetched / created / aggregated) according to information available at the moment, context, user preferences or a combination of all.

Server-Side Programming

- Static pages (HTML)

```
http://site.com/docs/index.html
```

  - Web server loads the HTML file and sends it, **without any modification**, on the HTTP response to the client

- Dynamic Web Pages

```
http://site.com/app/prog.php
```

  - Web server loads the .php file, **processes it** (executes it), and sends the result in the HTTP response to the client

Server-Side Programming

- For the Web Client, static and dynamic Web Pages **are exactly the same**.
  - In both cases, the Web Server returns an HTML document.
  - The Web Client only has access to HTTP Response (with the HTML document). It completely ignores the instructions on the Web Server

Server-Side Programming

- Web Client (JavaScript Engine) executes Javascript
- Web Server **DOES NOT** execute, nor understands, Javascript code (1)
  - For the Web Server, Javascript Code is just plain text, like HTML or CSS.
  - It is a resource, as any other, that the server provides to the client

(1) *Some technologies run Javascript on the server (e.g. node.js), but this concept is out of scope for this course.*

2 – INTRODUCTION TO PHP

PHP

- Most commonly used programming language on Web Servers
- Open source
- Supported by several types of Web Servers: Apache; nginx, IIS, etc.
- Extensive development and publishing support
- Latest versions of PHP incorporate advanced programming concepts, such as Object Oriented Programming

PHP

- PHP as a language is:
  - Interpreted
  - *Weakly typed*
    - Variables can store any type of value
    - Operations between variables of any type are possible. Implicit conversions are made automatically
  - Dynamic
    - Language operations (such as datatype checking) are made dynamically at run-time. On a static language, these operations are made at compile time.
    - This means that PHP has no compile errors – just runtime errors

PHP

- PHP code can be embedded on HTML
  - Inside special "tags": **<?php** ... and ... **?>**

```
<!doctype html>
<html>
  <head>
    <meta charset="UTF-8">
    <title>Example Document</title>
  </head>
  <body>
    <p>
    <?php
        echo "Hello World!";
    ?>
    </p>
  </body>
</html>
```

PHP

- Resulting HTML from previous example
  - Web Client only accesses the resulting HTML. It does not care how the HTML code was created

```
<!doctype html>
<html>
  <head>
    <meta charset="UTF-8">
    <title>Example Document</title>
  </head>
  <body>
    <p>
    Hello World!
    </p>
  </body>
</html>
```

PHP

- PHP blocks can generate any type of content (including HTML elements)

```
<!doctype html>
<html>
    <head>
        <meta charset="UTF-8">
        <title>Doc</title>
    </head>
    <body>
        <?php
        echo "<div class='clsA'>";
        echo "Hello World!";
        echo "</div>";
        ?>
    </body>
</html>
```

PSR-1 (Basic Coding Standard):
- Files MUST **use only <?php** and **<?= tags**
- PHP code MUST use **only UTF-8 without BOM**.

PSR-2 (Coding Style Guide)
Code MUST use **4 spaces for indenting, not tabs**.

PHP

- Resulting HTML from previous example

```
<!doctype html>
<html>
    <head>
        <meta charset="UTF-8">
        <title>Documento de Exemplo</title>
    </head>
    <body>
        <div class='clsA'>Hello World!</div>
    </body>
</html>
```

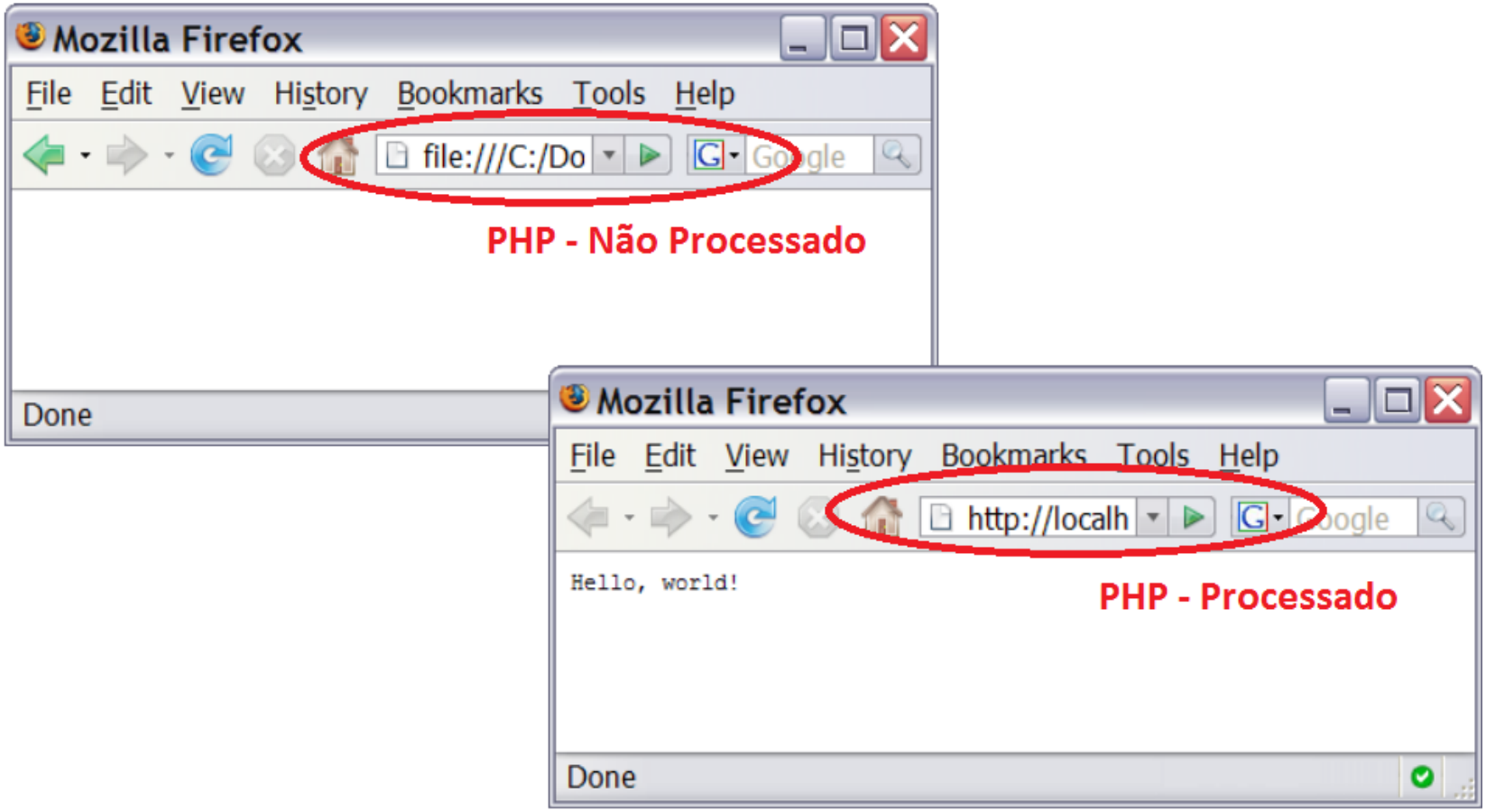
PHP

- **Attention**
- Do not open PHP files directly on the Browser. They are not processed!

URL:	file://C:/a/file.php	Invalid

- PHP files must be processed (executed) by an Web Server:

URL:	**http://localhost**/a/file.php	Valid
URL:	**http://www.mysite.com**/a/file.php	


Mozilla Firefox
File Edit View History Bookmarks Tools Help
file:///C:/Do
Google
PHP - Não Processado
Done
Mozilla Firefox
File Edit View History Bookmarks Tools Help
http://localh
Google
Hello, world!
PHP - Processado
Done

3 – SYNTAX AND BASIC CONCEPTS OF THE LANGUAGE

Basic Syntax

- Instructions end with **;**
- PHP is case insensitive – except variables and constants

PSR-2 (Coding Style Guide)
PHP keywords MUST be in lower case.

4 equivalent instructions:

```
echo "Hello World";
Echo "Hello World";
ECHO "Hello World";
ecHo "Hello World";
```

```
$abc = "Hello World";
$Abc = "Goodbye";
echo $abc;
echo $Abc;
```

2 distinct variables

- Comments
  - Up to the end of line: **//** and **#**
  - Multiple lines: **/* . . . */**

```
echo "Hello World"; // comment
echo "Hello World";  # comment
```

```
/* This is
a comment with
Multiple lines */
echo "Hello World";
echo "Hello World";
```

Variables

- Overtime, the same variable can store values of multiple types

```
$var = 10;
$var = "ten";
$var = 14.23;
```

- Variable names:
  - Always start by **$**
  - After $ it must have a letter or underscore, followed by letters, digits and underscores
  - Case sensitive
    - **$abc ≠ $Abc**

Valid Variable Names:

```
$_var        = 12;
$var32       = 12;
$_var_2_A    = 12;
```

Invalid Variable Names:

```
$3var        = 12;
$var abc     = 12;
$var-2A      = 12;
$a1?a!a.a    = 12;
```

Constants

- Constants are created by **define** function
- Constants are immutable
- Constant names:
  - Do not start by **$**
  - Case sensitive

PSR-1 (Basic Coding Standard):
Constants MUST be declared in all upper case with underscore separators

PSR-2 (Coding Style Guide)
The PHP constants **true, false**, and **null** MUST be in lower case.

```
define("PI_VALUE", 3.14);
echo "value of pi = " . PI_VALUE;
```

- There are several predefined constants, such as :
  - true false null
  - More on: http://php.net/manual/en/reserved.constants.php

Write on the Web Page

- **echo**
  - One or more strings

```
$str = "word";
$v= 7;
echo "Anything";
echo $v;
echo "a", $v, "-", $str;
```

- **print**
  - Only one string

```
print "Anything";
print $v;
print ("Anything");
print ($v);
```

- **printf**
  - String with a specified format
  - Similar to: sprintf e vprintf

```
printf ("value a= %d", $v);
printf ("%s has %d letters",
          $str, $val);
```

Debug messages

- **var_dump**($var)
- **print_r**($var)

- They both write the value of $var, but with different format
- print_r can also return the information to a string (instead of writing it on the page)

```
var_dump($v);
```

```
print_r($v);
```

```
$str = print_r($v);
```

Debug messages

- Examples:

```
$a = 15;
$b = "texto";
$c = array(3,7,"abc");
    var_dump($a);
    var_dump($b);
    var_dump($c);
echo "<hr>";
    print_r($a);
echo "<br>";
    print_r($b);
echo "<br>";
    print_r($c);
```

```
int 15

string 'texto' (length=5)

array (size=3)
  0 => int 3
  1 => int 7
  2 => string 'abc' (length=3)
```

15
texto
Array ([0] => 3 [1] => 7 [2] => abc)

4 – TYPES & OPERATORS

Data types

- Implicit (automatic) data type conversion
  - Example: The add operator (+) always adds 2 numbers. Therefore, on the expression a + b, the values of a and b are always converted to numbers
  - If there is no direct conversion, PHP converts the value to the default number value (0 – zero)

```
$a = 16;
echo 'A' + "B";
echo $a + 'A ';
echo '2' + $a + " 7 " + 3;
```

```
0
16
28
```

Data types

- PHP also supports explicit data type conversion
  - Examples

```
$a = (int) "16";
$b = (int) 16.3;
$c = (int) "A";
$d = (bool) 1;
$e = (bool) "";
$f = (bool) 0;
$g = (float) "3.2";
```

```
$a = 16;
$b = 16;
$c = 0;
$d = true;
$e = false;
$f = false;
$g = 3.2;
```

Primitive Types

Type	Description	Conversion	Example			
Scalar						
string	Text	`(string)`	`"abc"`	`'abc'`	`'12'`	`'FALSE'`
integer	Integer number	`(int)`	`0`	`234`	`-54`	
float	Real number	`(float)`	`0.0`	`23.3`	`-3.4`	`3.98e12`
boolean	Logic values	`(bool)`	`true`	`false`	`True`	`FALSE`
Compound						
array	Set of values	`(array)`	`array(12, 3e4, "a", true, 2)`			
object	Class instance	`(object)`	`$obj = new cls();`			
Special						
resource	Reference to an external resource		`$fp=fopen("file", "w");`			
NULL	No value	`(unset)`	`NULL`	`Null`	`null`	

Strings

- Double quote / Single quote
  - Strings can be enclosed by single or double quotes

- Double quote "
  - Can include single quotes
  - Double quote character is \"

- Single quote '
  - Can include double quotes
  - Single quote character is \'

Valid Strings:

```
$a = "text";
$a = 'text';
$a = "<img src='a.png'>";
$a = '<img src="a.png">';
$a = "<img src=\"a.png\">";
$a = '<img src=\'a.png\'>';
```

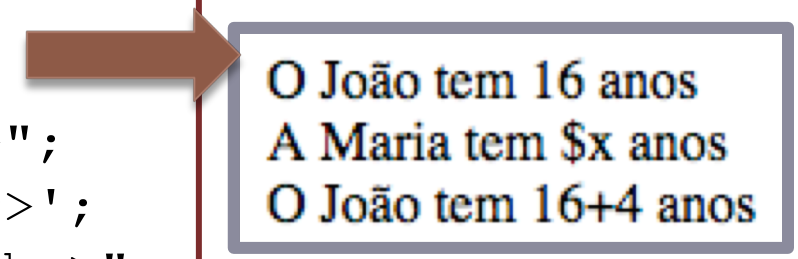
Invalid Strings:

```
$a = "text';
$a = 'text";
$a = "<img src="a.png">";
$a = '<img src='a.png'>';
```

Strings

- Double quote " – string is interpreted – this is called **string interpolation**
- Single quote ' – string is not interpreted

```
$x = 16;
$a = 4;
echo "O João tem $x anos <br>";
echo 'A Maria tem $x anos <br>';
echo "O João tem $x+$a anos <br>";
```

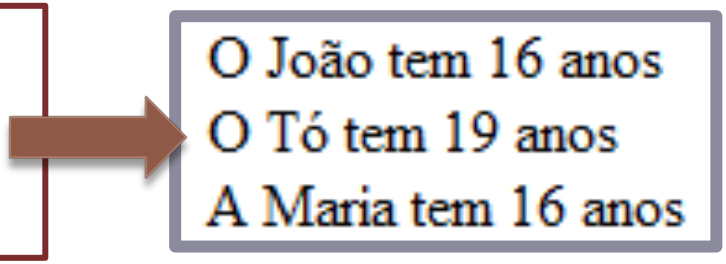



```
echo "valor de x=$x ";
echo "valor de x={$x} ";
echo "valor de um elemento do array $a[0] ";
echo "mais exemplos com arrays $a[chave] ";
echo "mais exemplos com arrays {$a['chave'][0]} ";
```

Strings

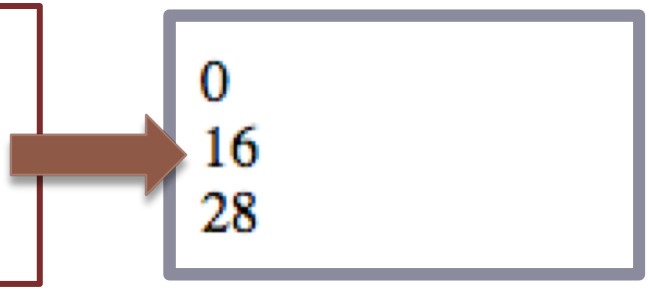
- String concatenation – **dot**
  - It is possible to concatenate variables of any type. PHP will convert implicitly all values to strings

```
$x = 16;
echo "O João tem $x anos" . "<br>";
echo "O Tó tem " . 19 . " anos <br>";
echo 'A Maria tem ' . $x . " anos";
```



- **Attention** – remember that add operator (+) does not concatenate strings

```
$a = 16;
echo 'A' + "B";
echo $a + 'A ';
echo '2' + $a + " 7 " + 3;
```



Boolean

- 2 possible values: **true** and **false**
  - Case insensitive: True; False; TRUE; FALSE; etc.
- The following values are converted implicitly to **false**:
  - **0**, **0.0** (attention, 0.00 or 0.000 are converted to true)
  - **""**, **"0"**, **NULL** (including variables destroyed with unset function)
  - **Empty array** (0 elements) and **Empty Object** (0 members)
- All other values are converted implicitly to **true**
- Implicit conversion from boolean to string:
  - **false** converts to empty string
  - **true** converts to the string **"1"**

Date/Time

- Current date and time: **`time()`**
- Date/time to string: date()
- String to date/time: strtotime ()

Convert strings to dates

```
$d1 = strtotime("now");
$d2 = strtotime("today");
$d3 = strtotime("tomorrow");
$d4 = strtotime("now + 24 hours");
$d5 = strtotime("last saturday");
$d5 = strtotime("8pm + 3 days");
$d6 = strtotime("2 weeks ago");
$d7 = strtotime("today 4am");
```

Date format

```
date("d/m/Y", time());
date("G:i:s", time());
```

Date arithmetic operations

```
$t = $data1 - $data2;
```

$timeSpan = Nº of seconds between the 2 dates

Date/Time

Some useful functions (http://php.net/manual/en/function.date.php)

Function	
`time`	Obtains current date and time
`mktime`	Obtains a date and time by specifying the year, month, day, hour, minute and second
`getdate`	Obtains an array with all information about a date/time (year, month, day, hour, minute, second, etc.)
`date`	Converts a date/time to a string with a specified format
`strtotime`	Converts a string to a date/time

Data type check

- **isset** – true if variable exists and is not null
  - Can be used when **variable does not exist**
- **is_null** – true if variable is null
  - Can only be applied to existing (declared) variables
- **empty** – true if variable is empty
  - Variable is empty when its value is: empty string, false, array(), NULL, "0", 0, unset variable
- **unset** – destroys a variable.
  - Also used to delete an element from an array

```
$a = 10;
unset($a);
```

```
$a[3] = 10;
unset($a[3]);
```

Data type check

- Examples:

isset

```
// isset($a) = false
$a = 10;
// isset($a) = true
unset($a)
// isset($a) = false
$a = null;
// isset($a) = false
$a = "";
// isset($a) = true
```

is_null

```
// is_null($a) = true
$a = 10;
// is_null($a)  = false
unset($a)
// is_null($a)  = true
$a = null;
// is_null($a)  = true
$a = "";
// is_null($a)  = false
```

empty

```
// empty($a) = true
$a = 10;
// empty ($a) = false
unset($a)
// empty ($a) = true
$a = null;
// empty ($a) = true
$a = "";
// empty ($a) = true
```

Note: If variable still doesn't exist, is_null and empty functions return true, isset function returns false

Data type check

Value of variable ($var)	isset($var)	empty($var)	is_null($var)
“” (an empty string)	bool(true)	bool(true)	
” ” (space)	bool(true)		
FALSE	bool(true)	bool(true)	
TRUE	bool(true)		
array() (an empty array)	bool(true)	bool(true)	
NULL		bool(true)	bool(true)
“0” (0 as a string)	bool(true)	bool(true)	
0 (0 as an integer)	bool(true)	bool(true)	
0.0 (0 as a float)	bool(true)	bool(true)	
var $var; (a variable declared, but without a value)		bool(true)	bool(true)
NULL byte (“\ 0”)	bool(true)		

https://www.virendrachandak.com/techtalk/php-isset-vs-empty-vs-is_null/

isset

- Example of **isset** usage:
  - $id is assigned the query string parameter ‘id’ value, but if the parameter does not exists, it will assume -1 value. This guarantees that $id will always exist and have a meaningful value.

```
$id = -1;
if (isset($_GET['id']) {
    $id = $_GET['id']
}
```

?? - Null Coalescing Operator

- PHP 7 supports the Null Coalescing Operator
  - Binary operator that returns the first operand if it exists and is not null, or returns the second operator otherwise
  - Syntax: `value ?? Value_if_null`
- Example of **??** Usage (similar to previous isset example):
  - $id is assigned the query string parameter 'id' value, but if the parameter does not exists, it will assume -1 value. This guarantees that $id will always exist and have a meaningful value.

```
$id = $_GET['id'] ?? -1;
```

Data type check

Function	Checks if variable is		
is_float	is_double	is_real	A real number
is_int	is_integer	is_long	An integer number
is_numeric	A number (integer or real) or a numeric string (eg: " 23")		
is_string	A string		
is_bool	A Boolean		
is_scalar	Is a scalar type (integer, real, string or boolean)		
is_array	An array		
is_object	An objet		
is_resource	A resource		
is_callable	A function		

Function	
getype($var)	Gets the $var data type
settype($var, type)	Changes $var data type

Operators

Aritmethic	
`-$a`	Negative
`$a + $b`	Addition
`$a - $b`	Subtraction
`$a * $b`	Multiplication
`$a / $b`	Division
`$a % $b`	Remainder

Assignment	
`$a= 3`	Assignment
`$a+= $b`	$a = $a + $b
`$a-= $b`	$a = $a - $b
`$a*= $b`	$a = $a * $b
`$a/= $b`	$a = $a / $b

Strings	
`$a . $b`	Concatenation
`$a.= $b`	$a = $a . $b

Increment / decrement	
`$a++`	*Post-increment* – returns $a and then increments the value of $a
`++$a`	*Pre-increment* – increments the value of $a and then returns $a
`$a--`	*Post-decrement*– returns $a and then decrements the value of $a
`--$a`	*Pre-decrement* – decrements the value of $a and then returns $a

Operators

Logic	
`!$a`	Not (negation)
`$a && $b`	**AND** – true if both ($a and $b) are true
`$a || $b`	**OR** – true if one of them ($a or $b) is true

Comparison	
`$a == $b`	$a equals a $b (after $a and $b are converted to the same data type)
`$a === $b`	$a equals $b (value and type are equal)
`$a != $b`	$a different of $b (after $a and $b are converted to the same data type)
`$a <> $b`	
`$a !== $b`	$a different of $b (value or type are different)
`$a < $b`	$a less than $b
`$a <= $b`	$a less than or equal $b
`$a > $b`	$a more than $b
`$a >= $b`	$a more than or equal $b

5 – ARRAYS

Arrays

- In PHP all arrays have **variable size**
- Syntax- Examples:
  - Create an empty array

```
$arr= array();
```

```
$arr= [];
```

  - Create an array with some values

```
$arr= array("value1", "value2", "value3");
```

```
$arr= ["value1", "value2", "value3"];
```

  - Access the value of an element of the array

```
$v = $arr[0];      // $v = "value1"
```

  - Change the value of an element of the array

```
$arr[0] = "new value";
```

Arrays

- In PHP all arrays have are **associative arrays**
  - They have a **key** and a **value**
  - The key must be unique
  - If no key is defined, PHP creates an automatic numeric key that starts at zero

```
$array[key]= value;
```

```
$a= array("value1", "value2", "value3");
```

- Key can be defined with this syntax:

key => value

```
$a= array(0 => "value1",
          1 => "value2",
          2 => "value3");
```

Key	Value
0	"value1"
1	"value2"
2	"value3"

Arrays

- Keys and values can be of any data type

```
$a= array(
    "abc" => "value1",
    12 => 23.345,
    "x" => true,
    0 => 500);
```

```
$a= [
    "abc" => "value1",
    12 => 23.345,
    "x" => true,
    0 => 500
    ];
```

Key	Value
"abc"	"value1"
12	23.345
"x"	true
0	500

Arrays

- Add an element to an array
  - Using a key that still does not exist on the array:

```
$arr[100] = "new value"; // if key 100 does not exist
                         // a new element is added to the
                         // array with key =100
```

  - Without specifying the key:

```
$arr[]= "New value"; // A new element is added to the
                     // array (at the end)
                     // Key is automatic.
```

  - Using the **array_push** function:

```
array_push($arr, "New value 1", "New value 2");
      // Adds 2 new elements to the array (at the end)
      // Key is automatic.
```

Arrays

- Example that adds several elements to an array

```
$a= array(12 => 23.345,
          "x" => true,
          0 => 500);
```

Note: Automatic key -> Biggest integer + 1

```
$a[]= "new1";
```

```
$a[100]= "new2";
```

```
$a["b"]= "new3";
```

```
array_push($a, "new4", "new5");
```

Key	Value
12	23.345
"x"	true
0	500
13	"new1"
100	"new2"
"b"	"new3"
101	"new4"
102	"new5"

+ 1

Arrays

- Delete an element from an array
  - Using the **unset** function:

```
$a= array("abc" => "vaue1",
          12 => 23.345,
          "x" => true,
          0 => 500);
```

Key	Value
"abc"	"value1"
12	23.345
"x"	true
0	500

```
unset($a["x"]);
```

Key	Value
"abc"	"value1"
12	23.345
0	500

Arrays

- **Arrays of Arrays** (*"simulate"* multi-dimensional arrays)
  - Array elements of an array can be themselves arrays
  - Access uses this notation:
    `[key1][key2]`

```
$a= array(
      array("a"=>"value1",
            "b"=>"value2",
            "c"=>"value3"),
      array('x','y','z'),
      "Simple Value");
```

```
$v = $a[0]["b"]; // "value2"
$v = $a[1][2];   // "z"
$v = $a[2]   ;   // "Simple Value"
$v = $a[1];   // array('x','y','z')
```

Key	Value
0	Key / Value "a" / "value1" "b" / "value2" "c" / "value3"
1	Key / Value 0 / "x" 1 / "y" 2 / "z"
2	"Simple Value"

Arrays

Some useful functions (http://php.net/manual/en/ref.array.php)

Function	
`array`	Creates an array
`count`	Number of elements in the array
`is_array`	Checks if is an array
`in_array`	Checks if a value exists within an array
`array_key_exists`	Checks if a key is used by an array
`array_keys`	Returns an array with all the keys (just the keys – no values)
`array_values`	Returns an array with all the values (just the values– no keys)
`array_push`	Adds a value to the end of an array
`array_pop`	Deletes the last element of an array (and returns it)
`array_unshift`	Adds a value to the beginning of an array
`Array_shift`	Deletes the first element of an array (and returns it)

6 – CONTROL STRUCTURES

PSR for control structures

- According with **PSR-2 – Coding Style Guide**, the general style rules for control structures are as follows:
  - There MUST be one space after the control structure keyword
  - There MUST NOT be a space after the opening parenthesis
  - There MUST NOT be a space before the closing parenthesis
  - There MUST be one space between the closing parenthesis and the opening brace
  - Opening braces MUST go on the same line
  - The structure body MUST be indented once
  - The closing brace MUST be on the next line after the body
  - The body of each structure MUST be enclosed by braces.
    - This standardizes how the structures look, and reduces the likelihood of introducing errors as new lines get added to the body.

if


```php
$d=date("D");
if ($d=="Fri") {
  echo "See you monday!";
}
```

```php
$d=date("D");
if ($d=="Fri") {
  echo "See you monday!";
}
else {
  echo "Good morning!";
}
```

```
if (expression1)
    block
elseif (expression2)
    block
elseif (expression3)
    block
else
    block
```

Note:
else and elseif are optionals

PSR-2 (Coding Style Guide)
The keyword **elseif SHOULD be used instead of else if** so that all control keywords look like single words.

Ternary Operator

- Logic expression ? if_true : if_false
  - Examples:

```
date("D")=="Fri" ? print("Yeah!") : print("Oh!");
```

Similar to:

```
$d=date("D");
if ($d=="Fri") {
  print("Yeah!");
}
else {
  print("Oh!");
}
```

```
echo date("D")=="Fri" ? "Yeah!" : "Oh!";
```

switch

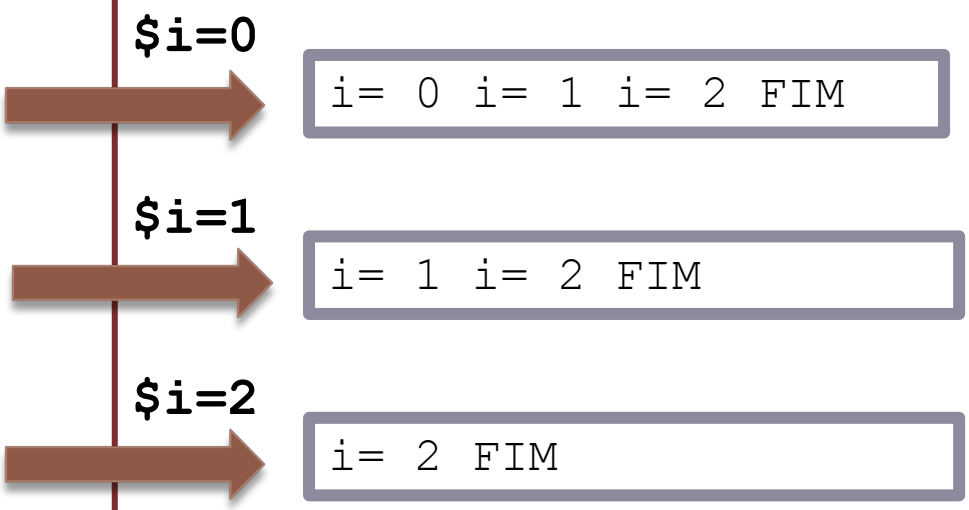

```
switch ($i) {
    case 0:
        echo "i is equal to 0";
        break;
    case 1:
        echo "Several instructions are possible";
        echo "i is equal to 1";
        break;
    default:
        echo "i is not equal to 0 or 1";
}
```

PSR-2 (Coding Style Guide)
The case statement MUST be indented once from switch, and
the break keyword (or other terminating keyword) MUST be indented at the same level as the case body.

switch

- Execution flow goes to the case with the same value
- Break interrupts the execution flow, and it continues after the switch statement
- If there is no break instruction, all remaining instructions (after the case) are executed until a break is encountered

```
switch ($i) {
    case 0:
        echo " i= 0";
    case 1:
        echo " i= 1";
    case 2:
        echo " i= 2";
}
echo " FIM";
```



while / do-while


```
$i= 1;
while ($i <= 3) {
     echo $i;
     $i++;
}
```

```
$i= 1;
do {
     echo $i;
     $i++;
} while ($i <= 3);
```

Note: do – while statement will always execute once (at minimum)

for / foreach


```php
for ($i = 1; $i <= 3; $i++) {
    echo $i;
}
```

```php
$a =array(1, 2, 3, "a");
foreach ($a as $value) {
    echo $value;
}
```

```php
$a =array(1, 2, 3, "a");
foreach ($a as $key => $value) {
    echo $key . "= ". $value;
}
```

Note: the **foreach** statement is the preferable statement to transverse arrays (as compared to the for statement).

Why?
because PHP arrays may not have all keys as numeric consecutive numbers

break

- Interrupts execution flow of : switch; for; foreach; while; do-while.

```
$i = 1;
$a = 5;
while ($i < 100) {
    echo $i . " ";
    if ($i==$a)
        break;
    $i++;
}
```

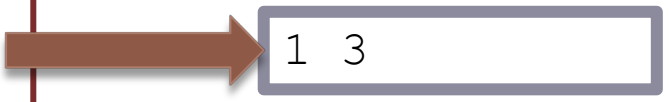

```
$a =array(1, 2, 3, "a");
$searchValue = 2;
foreach ($a as $value) {
    if ($ searchValue === $value){
      echo "Value Found";
      break;
    }
}
```

continue

- Ignores current cycle of for; foreach; while; do-while, but continues on the next cycle

```
for ($i = 1; $i <= 5; $i++) {
    if ($i==2)
        continue;
    echo $i . " ";
}
```


```
$a= array(1, "2", 3, "a", "5");
foreach ($a as $value) {
    if (is_string($value))
      continue;
    echo $value . " ";
}
```



Note: Ignores strings

7 – FUNCTIONS

Functions

- Invoke (call) functions
  - Examples:

```
func();
```

```
func($p1, 45);
```

```
$a = func();
```

```
$a = func($p1, 45);
```

  - *"Chained"* calls:

```
$a = "Abc";
var_dump(array($a, strtoupper($a)));
```

```
array (size=2)
  0 => string 'Abc' (length=3)
  1 => string 'ABC' (length=3)
```

PSR-2 (Coding Style Guide)

- When making a method or function call, there MUST NOT be a space between the method or function name and the opening parenthesis, there MUST NOT be a space after the opening parenthesis, and there MUST NOT be a space before the closing parenthesis.
- In the argument list, there MUST NOT be a space before each comma, and there MUST be one space after each comma.

Functions

- Function definition:
  - Examples:

```
function print_info()
{
    echo "<div>...</div>";
}
```

```
// Call the function
print_info();
```

```
function add($x , $y)
{
     $total=$x+$y;
     return $total;
}
```

```
// Call the function
$a = add(2,3);
```

Functions

- Function definition:
  - Parameters with default values
  - Parameter may be omitted on function call

```
function increment($value, $inc = 1)
{
    return $value + $inc;
}

$a= 10;
$r= increment($a, 2);    // Value of $a = 12
$r= increment($a);       // Value of $a = 11
```

Functions

- Local and Global variables
  - To access a global variable inside a function, it must be declared as global, otherwise, a new local variable with the same name will be created.

```
$globalCounter = 0;
function increment($inc)
{
    global $globalCounter;
    $globalCounter += $inc;
}

increment(2);
```

8 – BLOCKS AND FILES

PHP Blocks

- One PHP file can include multiple PHP blocks
- **<?php . . . ?>**
- **<?=expression ?>**
  - The same as: <?php **echo** expression; ?>
- **<?php . . .**

- It is not required to close the last PHP block of the file
- This type of block (without closing delimiter) is recommended for files with PHP code only (no HTML)

No closing delimiter?>

```
<?php
    function f1(){
        . . .
    }
    function f2(){
        . . .
    }
```

PHP Blocks

- HTML inside PHP block { . . . }
  - For any PHP statement that accepts blocks {...}
    if, while; for; foreach; etc...

```
<?php
for ($i = 1; $i <= 10; $i++) { ?>
   <div>
      <p>
         HTML text . . .
        <?= $i ?>
      </p>
   </div>
<?php } ?>
```

Opens block

Closes block

PHP Blocks

- HTML inside PHP block { . . . }
  - Alternative syntax for **if** blocks

```
<?php if ($tipoUser == "adm"): ?>
   <div>
      <h1>Administration Zone</h1>
      <p>
           . . .
      </p>
   </div>
<?php endif; ?>
```

Opens block

Closes block

PHP Blocks

- HTML inside PHP block { . . . }
  - Alternative syntax for **if** and **switch** blocks

```
<?php if (exp): ?>
  HTML
<?php elseif (exp): ?>
  HTML
<?php else: ?>
  HTML
<?php endif; ?>
```

```
<?php switch (exp): ?>
  HTML
<?php case valor1: ?>
  HTML
<?php case valor2: ?>
  HTML
<?php default: ?>
  HTML
<?php endswitch; ?>
```

PHP Blocks

- HTML inside PHP block { . . . }
  - Alternative syntax for **while**, **for** and **foreach** blocks

```
<?php while (exp): ?>
  HTML
<?php endwhile; ?>
```

```
<?php for (start; end; repetition): ?>
  HTML
<?php endfor; ?>
```

```
<?php foreach ($array as $value): ?>
  HTML
<?php endforeach; ?>
```

```
<?php foreach ($array as $key => $value): ?>
  HTML
<?php endforeach; ?>
```

Files

- To reuse code from other (*"external"*) files:
  - include
  - include_once
  - require
  - require_once

```
include("file");
```

```
include_once("../file");
```

```
require("./folder/file");
```

```
require_once("folder/file");
```

- External files are included by copying content *inplace*
- External files may include PHP or HTML
- On external files, PHP blocks (<?php ... ?>) still have to be created explicitly

Files

- **include** versus **require**
  - If error occurs (for example, file does not exist):
  - **include** – triggers a warning, but execution continues
  - **require** – throws an error and execution is terminated
- **include_once** and **require_once**
  - File is included only once - if file has been included before, it will not be included again
  - Recommended for PHP file with functions / classes
- **include** e **require**
  - File is included multiple times - once for every include or require

Files

- Example – require_once

```
<?php
require_once("functions.inc");
?>
<html>
    <head>
    . . .
    </head>
    <body>
    . . .
        <?php
        . . .
        if (isOdd($a)) {
            . . .
        }
        ?>
    . . .
```

Ficheiro: functions.inc

```
<?php
function isOdd($number)
{
    return $number & 1;
    // 0 = even, 1 = Odd
}
```

Files

- Example – require

File: page.php

```
<?php require ("top.inc"); ?>

<div id="main">
   . . .
</div>

<?php require ("bottom.inc"); ?>
```

File: top.inc

```
<html>
  <head>
    <title>...</title>
    ...
  </head>
  <body>
    <header>
     ...
    </header>
```

File: bottom.inc

```
   <footer>
     ...
    </footer>
  </body>
</html>
```

Files

- Resulting page from previous example:

top.inc

page.php

bottom.inc

```
<html>
  <head>
    <title>...</title>
    ...
  </head>
  <body>
    <header>
      ...
    </header>
<div id="main">
    . . .
</div>
    <footer>
      ...
    </footer>
  </body>
</html>
```

PSR-1 Notes on side effects

PSR-1 (Basic Coding Standard):

A file SHOULD declare new symbols (classes, functions, constants, etc.) and cause no other side effects, or it SHOULD execute logic with side effects, **but SHOULD NOT do both**.

- “Side effects" means execution of logic not directly related to declaring classes, functions, constants, etc., merely from including the file
- "Side effects" include but are not limited to:
  - Generating output, explicit use of require or include, connecting to external services, modifying ini settings, emitting errors or exceptions, modifying global or static variables, reading from or writing to a file, and so on

9 – REFERENCES

References

- Official (PHP)
  - http://www.php.net/
  - http://php.net/manual/en/
  - http://php.net/manual/pt_BR/
- Other documentation:
  - Luke Welling and Laura Thomson, “PHP and MySQL Web Development (4th Edition)”, Addison-Wesley 2009
  - http://www.w3schools.com/php/default.asp
  - http://commons.oreilly.com/wiki/index.php/PHP_Cookbook
  - http://www.tuxradar.com/practicalphp
  - http://www.tizag.com/phpT/
- PHP PSR Standards
  - https://www.php-fig.org/psr/psr-1/
  - https://www.php-fig.org/psr/psr-2/