Contents

1 PHP
   Motivation

2 Overview
   Features
   Applications

3 Types and Variables
   Types
   Integers and Floating-point numbers
   Booleans
   Strings
   Variables

4 Type juggling and Type casting

5 Further Reading
The Common Gateway Interface (CGI) is a standard method for web servers to use external applications, a CGI program, to dynamically generate web pages.

1. A web client generates a client request, for example, from a HTML form, and sends it to a web server.
2. The web server selects a CGI program to handle the request, converts the client request to a CGI request, executes the program.
3. The CGI program then processes the CGI request and the server passes the program’s response back to the client.
Disadvantages of CGI

• A distinction is made between static web pages and dynamic web pages created by external CGI programs

• Using CGI programs it is difficult to add ‘a little bit’ of dynamic content to a web page
  \( \sim \) can be alleviated to some extent by ‘packing’ big chunks of HTML markup into a few strings

• Use of an external program requires
  • starting a separate process every time an external program is requested
  • exchanging data between web server and external program
  \( \sim \) resource-intensive

If our main interest is the creation of dynamic web pages, then the programming language we use

• should integrate well with HTML
• should not require a web server to execute an external program
PHP

- PHP is (now) a recursive acronym for **PHP: Hypertext Preprocessor**
- Development started in 1994 by Rasmus Lerdorf
- Originally designed as a tool for tracking visitors at Lerdorf’s website
- Developed into full-featured, scripting language for server-side web programming
- Shares a lot of the syntax and features with other languages
- Easy-to-use interface to databases
- Free, open-source
- Probably the most widely used server-side web programming language
- Negatives: Inconsistent, muddled API; no scalar objects; compatibility problems between PHP 5.x and PHP 7.x (PHP 6 was never released)
PHP Processing

- **Server plug-ins** exist for various web servers
  → avoids the need to execute an external program
- **PHP code** is **embedded into HTML pages** using tags
  → static web pages can easily be turned into dynamic ones

PHP satisfies the criteria we had for a good web scripting language

Processing proceeds as follows:

1. The web server receives a **client request**
2. The web server recognizes that the client request is for a HTML document containing **PHP code**
3. The server executes the **PHP code**, substitutes output into the HTML document, the resulting page is then send to the client

As in the case of CGI programs, the client never sees the **PHP code**, only the HTML document that is produced
PHP: Applications

- Applications written using PHP
  - activeCollab – Project Collaboration Software
    http://www.activecollab.com/
  - Drupal – Content Management System (CMS)
    http://drupal.org/home
  - Magento – eCommerce platform
    http://www.magentocommerce.com/
  - MediaWiki – Wiki software
    http://www.mediawiki.org/wiki/MediaWiki
  - Moodle – Virtual Learning Environment (VLE)
    http://moodle.org/
  - Sugar – Customer Relationship Management (CRM) platform
    http://www.sugarcrm.com/crm/
  - WordPress – Blogging tool and CMS
    http://wordpress.org/
Overview

Applications

**PHP: Websites**

- Websites using PHP:
  - Delicious – social bookmarking  
  - Digg – social news website  
    [http://digg.com](http://digg.com)
  - Facebook – social networking  
    [http://www.facebook.com](http://www.facebook.com)
  - Flickr – photo sharing  
    [http://www.flickr.com](http://www.flickr.com)
  - Frienster – social gaming  
    [http://www.frienster.com](http://www.frienster.com)
  - SourceForge – web-based source code repository  
    [http://sourceforge.net/](http://sourceforge.net/)
  - Wikipedia – collaboratively built encyclopedia  
Overview Applications

PHP: Hello World!

```html
<html lang="en-GB">
<head><title>Hello World</title></head>
<body>
<h1>Our first PHP script</h1>
<?php
    print("<p><b>Hello World!</b></p>
);
?>
</body></html>
```

- **PHP code** is enclosed between `<?php` and `?>`
- File must be stored in a directory accessible by the web server, for example `~/public_html`, and be readable by the web server
- File name must have the extension `.php`, e.g. `hello_world.php`

Our first PHP script

Hello World!
PHP: Hello World!

Since version 4.3.0, PHP also has a command line interface

```
#!/usr/bin/php
<?php

/* Author: Ullrich Hustadt
   A "Hello World" PHP script. */

print ("Hello World!\n");
// A single-line comment

?>
```

Hello World!

- **PHP code** still needs to be enclosed between `<?php` and `?>`
- Code must be stored in an executable file
- File name does not need to have any particular format

～ PHP can be used to write CGI programs
～ PHP can be used as a **scripting language** outside a web programming context
PHP: Hello World!

```html
<!DOCTYPE html>
<html lang="en-GB">
<head><title>Hello World</title></head>
<body><h1>Our first PHP script</h1>
<?php
    print("<p><b>Hello World!</b></p>
");
?>
</body></html>
```

- Can also ‘executed’ using
  
  `php filename`

- File does not need to executable, only readable for the user

Output:

```html
<!DOCTYPE html>
<html lang="en-GB">
<head><title>Hello World</title></head>
<body><h1>Our first PHP script</h1>
<p><b>Hello World!</b></p>
</body></html>
```
PHP Scripts

- **PHP scripts** are typically embedded into HTML documents and are enclosed between `<?php` and `?>` tags.

- A **PHP script** consists of one or more **statements** and **comments**; there is no need for a main function (or classes).
  - **Statements** end in a semi-colon.
  - Whitespace before and in between statements is irrelevant.
    (This does **not** mean its irrelevant to someone reading your code.)
  - **One-line comments** start with `//` or `#` and run to the end of the line or `?>`.
  - **Multi-line comments** are enclosed in `/*` and `*/`. 
PHP has eight datatypes

- **Four primitive types:**
  - `bool` – booleans
  - `int` – integers
  - `float` – floating-point numbers
  - `string` – strings

- **Two compound types:**
  - `array` – arrays
  - `object` – objects

- **Two special types:**
  - `resource`
  - `NULL`

- Integers, floating-point numbers, and booleans do not differ significantly from the corresponding JavaScript types
- Strings differ from those in JavaScript
Integers and Floating-point numbers

- PHP distinguishes between
  - integer numbers 0 2012 -40 1263978
  - floating-point numbers 1.25 256.0 -12e19 2.4e-10

- PHP supports a wide range of pre-defined mathematical functions
  - `abs(number)` absolute value
  - `ceil(number)` round fractions up
  - `floor(number)` round fractions down
  - `round(number [,prec,mode])` round fractions
  - `log(number [,base])` logarithm
  - `rand(min,max)` generate an integer random number
  - `sqrt(number)` square root

- PHP provides pre-defined number constants including
  - `M_PI` 3.14159265358979323846
  - `NAN` ‘not a number’
  - `INF` ‘infinity’
Integers and Floating-point numbers: NAN and INF

The constants **NAN** and **INF** are used as return values for some applications of mathematical functions that do not return a number

- \(\log(0)\) returns \(-\text{INF}\) (negative ‘infinity’)
- \(\sqrt{-1}\) returns **NAN** (‘not a number’)

In PHP 5

- \(1/0\) returns **FALSE** and produces a **PHP warning**
- \(0/0\) returns **FALSE** and produces a **PHP warning**

and execution of the script continues!

In PHP 7

- \(1/0\) returns **INF** and produces a **PHP warning**
- \(0/0\) returns **NAN** and produces a **PHP warning**

and execution of the script continues!
Booleans

- PHP has a boolean datatype with constants `TRUE` and `FALSE` (case insensitive)
- PHP offers the same short-circuit boolean operators as Java and JavaScript:
  
  ```
  && (conjunction)  || (disjunction)  ! (negation)
  ```

- Alternatively, `and` and `or` can be used instead of `&&` and `||`, respectively
- However, `not` is not a PHP operator

- The truth tables for these operators are the same as for JavaScript
- Remember that `&&` and `||` are not commutative, that is,
  
  - `(A && B)` is not the same as `(B && A)`
  - `(A || B)` is not the same as `(B || A)`
Type conversion to boolean

When converting to boolean, the following values are considered FALSE:

- the boolean FALSE
- the integer 0 (zero)
- the float 0.0 (zero)
- the string '0' (but not 0.0 nor '00')
- the empty string ''
- an array with zero elements
- an object with zero member variables (PHP 4 only)
- the special type NULL (including unset variables)
- SimpleXML objects created from empty tags

Every other value is considered TRUE (including any resource)

When converting a boolean to a string,

- TRUE becomes "1"
- FALSE becomes ""
Strings

- PHP supports both single-quoted and double-quoted strings
- PHP also supports heredocs as a means to specify multi-line strings

```php
<<<identifier
here document
identifier

identifier might optionally be surrounded by double-quotes
identifier might also be surrounded by single-quotes, making the string a nowdoc in PHP terminology

print "<html lang="en-GB">
<head><title>Multi-line String</title></head>
</html>

print <<<EOF
<body>Some text
<img alt="Picture of Crowne Plaza" src="pic.png">
</body>
</html>
EOF;
```
**Strings**

PHP distinguishes between

- **single-quoted strings** and
- **double-quoted strings**

<table>
<thead>
<tr>
<th><strong>single-quoted strings</strong> ('taken literally')</th>
<th><strong>double-quoted strings</strong> ('interpreted'/'evaluated')</th>
</tr>
</thead>
<tbody>
<tr>
<td>'hello'</td>
<td>&quot;hello&quot;</td>
</tr>
<tr>
<td>'don't'</td>
<td>&quot;don't&quot;</td>
</tr>
<tr>
<td>'&quot;hello&quot;'</td>
<td>&quot;&quot;hello&quot;&quot;</td>
</tr>
<tr>
<td>'backslash'</td>
<td>&quot;backslash&quot;</td>
</tr>
<tr>
<td>'glass\table'</td>
<td>&quot;glass\table&quot;</td>
</tr>
<tr>
<td>'glass\table'</td>
<td>&quot;glass\table&quot;</td>
</tr>
</tbody>
</table>

'Strings' is represented as a table with two columns. The first column lists single-quoted strings, and the second column lists double-quoted strings, illustrating how they are interpreted.
Strings

• **Variable interpolation** is applied to double-quoted strings

```php
$title = "String Operators";
print "<title>$title</title>";
<title>String Operators</title>
```

• The **string concatenation** operator is denoted by ‘.’

• The string multiplication / repetition operator in PHP is

```php
string str_repeat(string_arg, number)
```

```php
$string = "<p>I shall not repeat myself.<p>\n";
print "<body>" . str_repeat($string,3) . ' </body>";
<body><p>I shall not repeat myself.<p>
<p>I shall not repeat myself.<p>
<p>I shall not repeat myself.<p>
</body>
```
Variables

- All PHP variable names start with $ followed by a PHP identifier.
- A PHP identifier consists of letters, digits, and underscores, but cannot start with a digit.
- PHP identifiers are case sensitive.
- In PHP, a variable does not have to be declared before it can be used.
- A variable also does not have to be initialised before it can be used, although initialisation is a good idea.
- Uninitialized variables have a default value of their type depending on the context in which they are used.

<table>
<thead>
<tr>
<th>Type</th>
<th>Default</th>
<th>Type</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>bool</td>
<td>FALSE</td>
<td>string</td>
<td>empty string</td>
</tr>
<tr>
<td>int/float</td>
<td>0</td>
<td>array</td>
<td>empty array</td>
</tr>
</tbody>
</table>

If there is no context, then the default value is NULL.
Assignments

- Just like Java, JavaScript and Python, PHP uses the equality sign `=` for assignments
  
  ```php
  $student_id = 200846369;
  ```

  As in JavaScript, this is an assignment expression

- The value of an assignment expression is the value assigned
  
  ```php
  $b = ($a = 0) + 1;
  // $a has value 0
  // $b has value 1
  ```
PHP also supports the standard **binary assignment** operators:

<table>
<thead>
<tr>
<th>Binary assignment</th>
<th>Equivalent assignment</th>
</tr>
</thead>
<tbody>
<tr>
<td>$a += b$</td>
<td>$a = a + b$</td>
</tr>
<tr>
<td>$a -= b$</td>
<td>$a = a - b$</td>
</tr>
<tr>
<td>$a *= b$</td>
<td>$a = a * b$</td>
</tr>
<tr>
<td>$a /= b$</td>
<td>$a = a / b$</td>
</tr>
<tr>
<td>$a %= b$</td>
<td>$a = a % b$</td>
</tr>
<tr>
<td>$a **= b$</td>
<td>$a = a ** b$</td>
</tr>
<tr>
<td>$a .= b$</td>
<td>$a = a . b$</td>
</tr>
</tbody>
</table>

// Convert Fahrenheit to Celsius:
// Subtract 32, then multiply by 5, then divide by 9
$temperature = 105; // temperature in Fahrenheit
$temperature -= 32; $temperature *= 5; $temperature /= 9; // converted to Celsius
Constants

- **bool define**(string, expr [, case_insensitive])
  - defines a constant that is globally accessible within a script
  - *string* should be a string consisting of a PHP identifier (preferably all upper-case)
    The PHP identifier is the name of the constant
  - *expr* is an expression that should evaluate to a value of a scalar type
    (In PHP 7, *expr* can also be an array)
  - *case_insensitive* is an optional boolean argument, indicating whether the name of the constant is case-insensitive (default is FALSE)
  - returns TRUE on success or FALSE on failure

```php
define("PI",3.14159);
define("SPEED_OF_LIGHT",299792458,true);
// PHP 7
define("ANIMALS",["bird","cat","dog"]);```
## Constants

- To use a constant we simply use its **name**

```php
define("PI",3.14159);
define("SPEED_OF_LIGHT",299792458,true);
// PHP 7
define("ANIMALS",["bird","cat","dog"]);

circumference = PI * $diameter;
distance = speed_of_light * $time;
myPet = ANIMALS[1];
```

- Caveat: PHP does **not** resolve constants within double-quoted strings (or here documents)

```php
print "1 - Value of PI: \n";
1 - Value of PI: PI
print "2 - Value of PI: 
" . PI . " \n";
2 - Value of PI: 3.14159
```
Values, Variables and Types

PHP provides several functions that explore the type of an expression:

<table>
<thead>
<tr>
<th>PHP Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>string gettype(expr)</code></td>
<td>returns the type of <code>expr</code> as string</td>
</tr>
<tr>
<td><code>bool is_type(expr)</code></td>
<td>checks whether <code>expr</code> is of type <code>type</code></td>
</tr>
<tr>
<td><code>void var_dump(expr)</code></td>
<td>displays structured information about <code>expr</code> that includes its type and value</td>
</tr>
</tbody>
</table>

```php
<?php
print "Type of 23: 
print "Type of 23.0: 
print "Type of "23": 
if (is_int(23)) {
    echo "23 is an integer\n";
} else {
    echo "23 is not an integer\n";
}
?>
```

Type of 23: integer
Type of 23.0: double
Type of "23": string
23 is an integer
Type juggling and Type casting

• PHP automatically converts a value to the appropriate type as required by the operation applied to the value (type juggling)

\[
\begin{align*}
2 & . \ "\text{worlds}" & \sim & "2\text{worlds}" \\
"2" & * 3 & \sim & 6 \\
"1.23e2" & + 0 & \sim & 123 \\
"\text{hello}" & * 3 & \sim & 0 & \text{(in PHP 7 also a warning)} \\
"10\text{hello5}" & + 5 & \sim & 15 & \text{(in PHP 7 also a warning)}
\end{align*}
\]

• We can apply an identity function of the target type to force a type conversion

\[
\begin{align*}
"12" & * 1 & \sim & 12 \\
"12" & * 1.0 & \sim & 12.0 \\
"12.1" & * 1 & \sim & 12.1 \\
12 & . \ "" & \sim & "12" \\
12.1 & . \ "" & \sim & "12.1"
\end{align*}
\]

\[
\begin{align*}
!!1 & \sim \text{TRUE} \\
!!0 & \sim \text{FALSE} \\
!!1.0 & \sim \text{TRUE} \\
!!"" & \sim \text{FALSE} \\
!!"0" & \sim \text{FALSE} \\
!!"1" & \sim \text{TRUE} \\
!!0.0 & \sim \text{FALSE} \\
\text{FALSE} & * 1 & \sim 0
\end{align*}
\]

Conversion of arrays to strings or numbers does not work
Type juggling and Type casting

PHP also supports explicit type casting via `type`

- `(int) "12"` → `12`
- `(int) "10hello5"` → `10`
- `(int) "1.23e2"` → `1` in PHP 5
- `(int) "1.23e2"` → `123` in PHP 7
- `(int) (float) "1.23e2"` → `123` in both PHP 5 and 7
- `(int) "1.23e2h5"` → `1` in PHP 5
- `(int) "1.23e2h5"` → `123` in PHP 7
- `(int) 10.5` → `10`
- `(float) "1.23e2"` → `123.0`
- `(float) "1.23e2h5"` → `123.0`
- `(bool) "0"` → `FALSE` (was `true` in JavaScript)
- `(bool) "foo"` → `TRUE`
- `(array) "foo"` → `array(0 => "foo")`
Revision and Further Reading

• Read
  • Chapter 3: Introduction to PHP
  • Chapter 4: Expressions and Control Flow in PHP: Expressions

• Read
  • Language Reference: Types: Booleans
  • Language Reference: Types: Integers
  • Language Reference: Types: Floating Point Numbers
  • Language Reference: Types: Strings