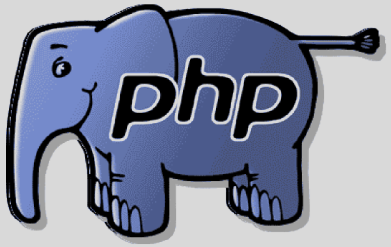


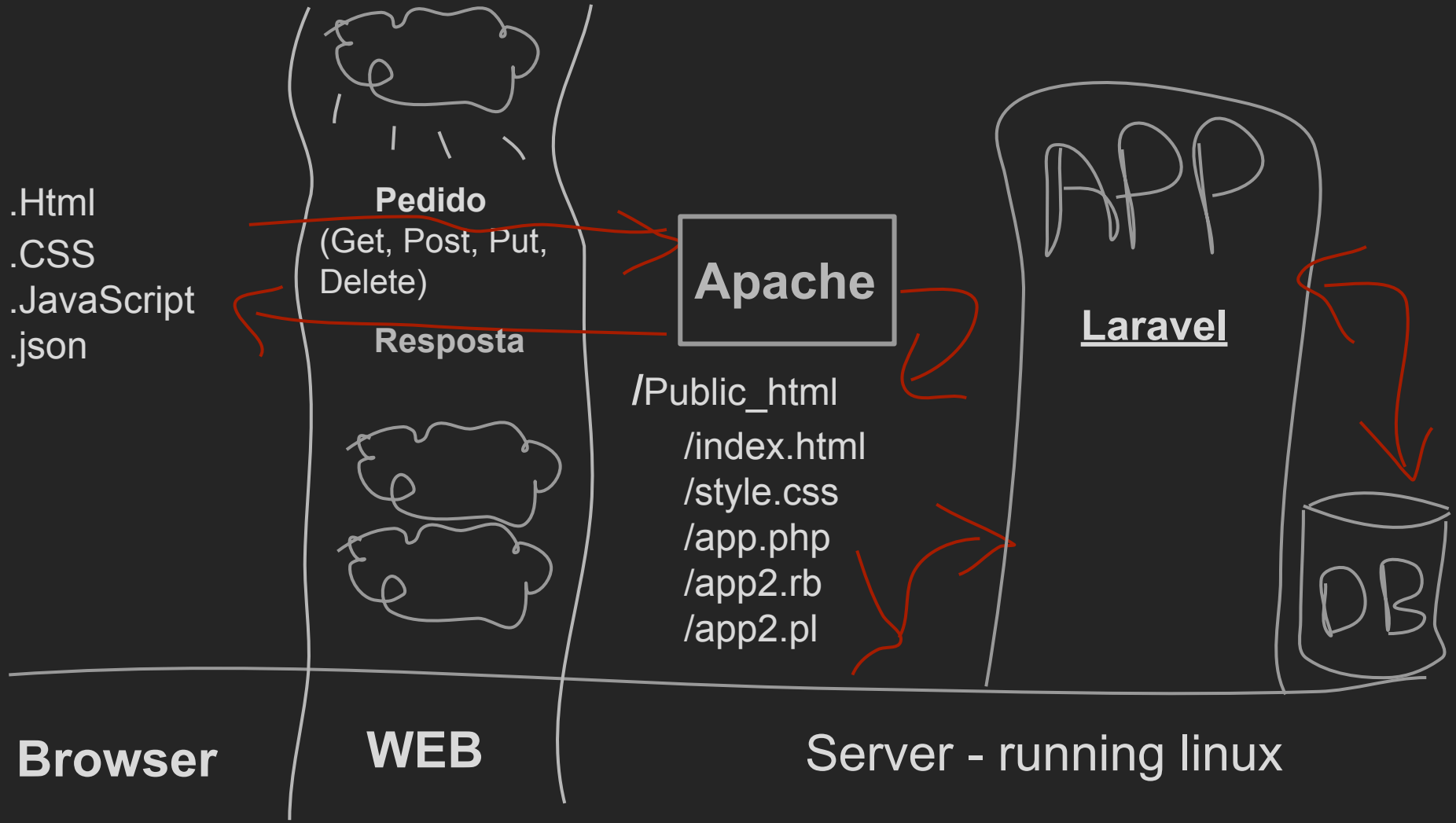
WEB DEVELOPMENT

Introduction to Php by

Rui Oliveira

Content

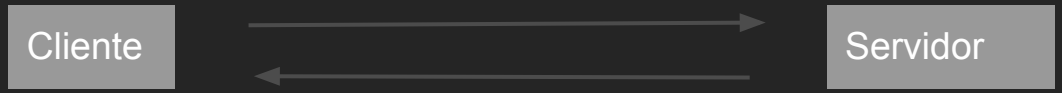
1. Basics
2. Objects
3. Multi-file
4. Particularities
5. Selected Topics


.Html
.CSS
.JavaScript
.json
Pedido
(Get, Post, Put, Delete)
Resposta
Apache
/Public_html
/index.html
/style.css
/app.php
/app2.rb
/app2.pl
APP
Laravel
DB
Browser
WEB
Server - running linux

Php Basics

Data Types
If .. elseif .. else
foreach

Simple php file




```
<!DOCTYPE html>

<html>
	<head>
	</head>

	<body>
		<div>Hello Rui Oliveira</div>
	</body>
</html>
```

```
<?php
	$var = 'Rui Oliveira';
?>


<!DOCTYPE html>

<html>
	<head>
	</head>

	<body>
		<div>Hello <?php echo $var; ?
></div>
	</body>
</html>
```

Data Types

```
<?php    //tipos Scalar

    $i = 100; // integer
    $f = 14.04; // float
    $s = “Ubuntu $f”; // String “Ubuntu 14.04”
    $s = ‘Ubuntu $f’; // String “Ubuntu $f”
    $b = true; // boolean
    $i = $i + $f // $i passou a ser float
```

?> Variáveis não tem um tipo associado. É dinamicamente tipada.

?> Os valores de um array podem ser de tipos diferentes

?> “ ” variáveis são substituídas

?> ‘ ’ variáveis não são substituídas

```
<?php    //Array

    $array = [‘ola’,12]; // array(‘10’,’12’);
    //Adicionar no fim
    $array[] = ‘Rui Oliveira’;
    $array[0] == ‘ola’; //aceder por índice
```

```
<?php    //Array Chave->valor (Associativo)

    $array = [‘key1’ => 12,’key2’ => ‘someText’];
    //agora podemos aceder por chave
    $array[‘key1’]  == 12; // true
```

If … elseif … elseif … else .

```php
<?php

$x = '1';
If ( $x == 1) {
        echo "TRUE";
} elseif ($x === 1)  {
        echo "FALSE";
} elseif ($x === '1') {
        echo "TRUE";
} else {
}
// 1 == '1' => true
// 1 === '1' => false
// 'Ola' == true => true
// 'Ola' === true => false
```

Operador:

?> ==, compara valor

?> ===, compara valor e tipo

For / Foreach

```php
<?php
for ($i = 1; $i <= 10; $i++) {
    // o que fará isto?
}

$arr = array(1, 2, 3, 4);
foreach ($arr as $value) {
    echo $value;
}

$arr = array('key1' => 1, 'a' =>  2, 'c' => 3);
foreach ($arr as $key => $value) {
    echo "array de $key tem $value";
}
```

?> Para percorrer todo um array usar **foreach**

?> Valores de um array não podem ser alteradas num foreach, em casos normais.

?> Operador & permite receber variáveis por referencia.

```php
<?php
//Como poder alterar array no foreach
$arr = array(1, 2, 3, 4);
foreach ($arr as &$value) {
        $value = $value * $value;
}
//arr == [1,4,9,16]
```

Funções sobre String's e Array's

http://php.net/manual/en/ref.array.php
http://php.net/manual/en/ref.strings.php

Exemplos:

?> array_filter(), array_keys(), key_exists()

?> strtolower(),str_replace()

Exercício

Php Objects

Class
Extends
Interface
Abstract Class

Class

```php
<?php    //Class's
class Pessoa {
    public static $my_static = 7; //variável de class
    protected $altura;  //variável de instancia
   /** Construtor da class  */
   public function __construct( $value ) {
        $this->altura = $value;
    }
   /** Setter de exemplo, com valor predefinido */
    public function setAltura( $value = 181 ) {
        $this->altura = $value;
    }
   /** Método que usa variável de class e de instancia*/
    public function sum(){
         return $this->altura + self::$my_static;
    }
}
```

Class idênticas ao Java.

Dentro da class:

?> self::*$atributo* ou *método()* - *Estático*

?> $this->*atributo* ou *método()* - *Instância*

?> **Uso de self e $this é obrigatório**

Fora da class:

?> class::*$atributo* ou *método()* - *Estático*

?> $var->*atributo* ou *método()* - *Instância*

```php
<?php //class use case
$myVar = new Pessoa(10);
$myVar->setAltura();

echo $myVar->sum(). ' static: '. Pessoa::$my_static;
```

Extends

?> Não existe herança múltipla

?> Funcionamento idêntico ao Java

?> São herdados todos os métodos.
(excepto os private)

?> Métodos podem ser modificados.

```php
<?php    //Class's
class Aluno extends Pessoa {
    protected $nome;
    public function __construct( $value , $nome) {
        parent::__construct($value);
        $this->nome = $nome;
    }
    public function calcSuperAltura(){
         return $this->altura * 100;
    }
    /** override de função */
    public function sum(){
        $aux = parent::sum();
        return $aux * 2;
    }
}
```

```php
<?php // Use case
$myVar = new Aluno(10, 'RuiOliveiras');
$myVar->setValue();
echo $myVar->sum();
```

Interface

```php
<?php    //Class's
interface Programador {
    public function learnLanguage ($language);
    public function knowLanguage($language);
}
class AlunoLei extends Pessoa
     implements Programador {
     protected $knowledge;
    /** __contrutor(...) */
    /** método da interface*/
    public function learnLanguage ($language){
        $this->knowledge[] = $language;
    }
    /** método da interface */
    public function knowLanguage($language){
          return in_array($language, $this-
>knowledge)';
    }
}
```

?> Class pode ter várias interfaces

?> Class deve implementar todos os métodos de interface.

```php
<?php
function webDevelop (Programador &$myCoder ){
    if (! $myCoder->knowLanguage('php')) {
        $myCoder->learnLanguage('php');
     }
}
```

Abstract Class

```php
<?php    //Class's
abstract class Pessoa{
  protected $nome;   /** Construtor da class  */
  public function __construct( $nome ) {
      $this->nome = $nome;
   }

  public function getNome() {
      return $this->nome;
   }
   /** função para saber se Pessoa esta ocupada
       nada data $date */
   abstract public function isBusyAt (\DateTime
$date);
}
```

?> Abstract idêntico a Java

?> **Type hinting** - permite receber parâmetros de determinado tipo, apesar de ser dinamicamente tipada.

```php
<?php use case
// Pode-se obrigar a função receber Class ou Interface ou Array
// Não se pode obrigar a receber Scalar (int, string, ...).
function verifica (Pessoa $myCoder ){
    //Será que esta pessoa implementa Programador ?
    if ($myCoder instanceof Programador) {
        //significa que $myCoder é Programdor
        //criar objeto nativo do PHP para data e horas.
        $now  = new \DateTime();
        if (! $myCoder->isBusyAt($now)) {
            // A class que instanciou $myCoder teve implementar
            // isBussyAt, apenas assumimos que existe.
        }
    }
}
```

Exercício

Multi-File

Namespace
Require vs Include
Require_once vs Include

Namespace

```
# ficheiro modelo/Pessoa.php
<?php
namespace modelo/Pessoa;
class pessoa { … }

# ficheiro modelo/Aluno.php
<?php
namespace modelo/Pessoa;
use Modelo/Pessoa;
class Aluno extends Pessoa { … }
```

Boas práticas:

?> Cada Class deve ser um Ficheiro.

?> Nome da class === Nome Ficheiro.

?> O namespace deve ser igual ao path do ficheiro.

Factos:

?> O namespace não precisa de ser o path do ficheiro (apenas é boa prática que o seja)

?> Para usar o *use* o namespace já deve ter sido interpretado pelo PHP para saber do que se trata.

?> Namespace e equivalente ao package do java

PHP > 5.3.0

Require vs Include

```
<?php    //require
# ficheiro app.php
require “modelo/Pessoa.php”;
require “modelo/Aluno.php”;
use modelo/Pessoa;
pessoa = new Pessoa(...); //...
```

Require - se ficheiro não existir dá erro fatal e pára a execução

Include - se ficheiro não existir dá warning mas a execução continua

?> Em situações críticas é preferível usar o require e tratar a excepção imediatamente.

```
<?php //include
# ficheiro app.php
include “modelo/Pessoa.php”;
include “modelo/Aluno.php”;
use modelo/Pessoa;
pessoa = new Pessoa(...); //...
```

Require_once vs Include_once

```
<?php //include
# ficheiro app.php
include_once “modelo/Pessoa.php”;
require_once “modelo/Aluno.php”;

use modelo/Pessoa;

pessoa = new Pessoa(...); //...
```

?> Garante que um ficheiro apenas é importado uma vez.

?> Na maior parte das situações reais usamos require_once.

?> Não queremos voltar a interpretar ficheiros já interpretados

Exercício

Particularities

Métodos Magicos
Variaveis SuperGlobais
Boas praticas gerais

Métodos Mágicos

```php
<?php
public mixed __get ( string $name )
public void __set ( string $name , mixed $value )
public string __toString ( void )
public array __debugInfo ( void )
public mixed  __call ( string $name , array $arguments )
```

?> São métodos que dão comportamento alternativo á class

?> Existem outros, estes são os mais relevantes

Métodos Mágicos ~ __get e __set

```php
<?php // __get e __set

class Pessoa3{
    public $aux;
    protected $array;

    function _set($name, $value) {
        $array[$name] = $value;
    }
    function __get($name) {
        return $array[$name];
     }
}
```

```php
<?php // __get e __set

$p =new Pessoa3();

$p->ola = “OLA __set”;  // usa __set
echo $p->ola;                 //usa __get
$p->aux = “ola aux”; //Não usa __set

//$p->aux existe e é public, logo não usa __set
//__get e __set são apenas usados quando
//variaveis não existem ou não são public
```

Métodos Mágicos ~ __toString e __debugInfo

<?php //__toString

```php
class Pessoa2{
    function __toString()
    {return “Eu Sou o” + $this->nome;}
}
$p = new Pessoa2();
echo (string) $p;
//__toString e usado em situações de cast.
```

<?php //__debugInfo

```php
class Pessoa1{
    public __debugInfo()
    {return array(“name” => $this->getName());}
}
$p = new Pessoa1();
var_dump($p);
// __debugInfo é apenas para controlar o que o var_dump imprime. //
Por defeito imprime tudo
```

Métodos Mágicos ~ __call

```php
<?php //__call
class Aluno extends Pessoa {
    protected static $access = [‘nome’, ‘email’];
    // public function__contructor(...) {...}

    function __call($name, $args) {
        $methodPrefix = substr($name, 0, 3);
        $variable = strtolower(substr($name, 3));

        if ($methodPrefix === ‘get’ &&
            in_array($variable, $this->access)) {
                return $this->{$variable};
        } elseif ($methodPrefix === ‘set’ &&
            in_array($variable, $this->access)) {
                $this->{$variable} = $args[0];
        }
    }
}
```

?> __call serve para criar métodos dinâmicos

?> Pode ser muito difícil fazer debug a código que use muito o __call.

?> Autocomplete dos IDE’s não vai ajudar nestes métodos

```php
<?php // test __call
$aluno = new Aluno();
$aluno->getNome();
// 1º não existe getNome definido
// 2º executa __call(‘getNome’, []);
// 3º retorna variavel $nome do aluno

$aluno = new Aluno();
$aluno->setNome(‘Rui Oliveiras’);
// 1º não existe setNome definido
// 2º executa __call(‘setNome’, [‘Rui Oliveiras’]);
// 3º coloca $nome igual a ‘Rui Oliveiras’;
```

Variáveis Super Globais

?> Variáveis Super globais podem ser acedidas em qualquer contexto.
?> Principais variáveis Super globais:

- **$_GET** - parâmetros da url
- **$_POST** - dados de submissão de formulário
- **$_FILES** - ficheiros submetidos num formulário
- **$_COOKIE** - cookies do browser
- **$_REQUEST** - junção de _GET _POST e _COOKIE

?> Normalmente as Frameworks PHP abstraem-se do uso direto destas variáveis.

PSR ~ Good practices in Php

?> Existe PSR 0,1,2,3,4

?> PSR 0 - Autoloading Standards

?> PSR 1 - Code Standard

?> PSR 2 - Code Style

?> http://www.php-fig.org/psr/psr-0/

Exemplos de regras de PSR

PSR0 > Namespace deve convertido diretamente no path da class.

PSR1 > Nunca fechar **<?php** com **?>** para garantir que ficheiro apenas contém PHP

PSR2 > Apenas usar espaços na identação.

Exercício

Selected Topics

Composer ~ Gestão de dependências
Routing ~ pedidos web

Composer ~ gestão de dependências

?> Composer um ficheiro php: *composer.phar*

?> Download: https://getcomposer.org/composer.phar

?> É executado: *php composer.phar*

?> As dependências ficam em: *composer.json*

?> Exemplo *Laravel*: https://packagist.org/search/?q=laravel

?> *Autoload.php:* faz require (Ver multi-file) automaticamente dos ficheiros das dependências.

Routing ~ Pedidos web

?> Forma de Servidores comunicar com Clientes.

?> GET: site.org/signin ~ pagina de login

?> GET: site.org/article/:articleId ~ pagina de um artigo

Tipos de pedidos:

?> GET: pedido para obter
?> POST: pedido para adicionar
?> DELETE: pedido para apagar
?> PUT: pedido para alterar

Restful ~ Web service

?> Normalmente usa-se JSON ou XML

?> Cada bloco deve representar um recurso

GET: site.org/api/users # retorna todos os Users
GET: site.org/api/users/1 # retorna info do user 1
GET: site.org/api/users/1/friends # retorna amigos do 1
POST: site.org/api/users # Adiciona novo User
PUT: site.org/api/users/1 # Edita User 1

ruiOliveiras

Contact: rui96pedro@gmail.com