

Programação Funcional em PHP

saia da zona de conforto

Marcel Gonçalves dos Santos
@marcelgsantos



Marcel Gonçalves dos Santos
@marcelgsantos
desenvolvedor web full-stack
pensandonaweb.com.br





sp.femug.com
@**femugsp**

phpsp

phpsp.org.br
@phpsp



Learning OOP in PHP

Tutoriais, vídeos, slides, livros sobre OOP, OOD, design patterns, refatoração e arquitetura.

bit.ly/Learning-OOP-in-PHP

Interaja nas mídias sociais!



- fale sobre o evento, palestrantes e conteúdo
- tire fotos do evento e publique
- interaja com outros participantes do evento
- tire dúvidas ou dê feedbacks para os palestrantes

Concorra a um livro da Casa do Código!



- 1. seguir @marcelgsantos no Twitter**
- 2. tuitar utilizando as hashtags #phpmgconf, #funcional e #php**
- 3. não vale tuíte em branco e retuíte**
- 4. ler e preencher este simples formulário**
bit.ly/sorteio-phpmg

**O que é programação
funcional?**

é um paradigma de programação que utiliza
funções puras e foca na **transformação do
estado**

**O que é paradigma de
programação?**

são **modelos** ou **estilos de programação**
suportados por linguagens que agrupam
certas **características comuns**

*Os paradigmas de programação definem como os
códigos são estruturados.*

principais paradigmas de programação
os dois principais paradigmas são o
imperativo e o declarativo

paradigma imperativo

**descreve a resolução de um problema através
de comandos que o computador pode
compreender e executar**

paradigma imperativo

os paradigmas **procedural** e **orientado a objetos** são exemplos de paradigmas imperativos

paradigma declarativo

**permite especificar o que deve ser
computado e não como deve ser computado**

paradigma declarativo

os paradigmas **funcional** e **lógico** são
exemplos de paradigmas declarativos

paradigmas de linguagens de programação

imperativo

procedural - C e Pascal

orientado a objetos - C++, Java, JavaScript, PHP, Python e Ruby

declarativo

lógico - Prolog

funcional - Clojure, Elixir, Elm, Erlang, F#, Haskell, Lisp, OCaml e Scala

Programação Funcional



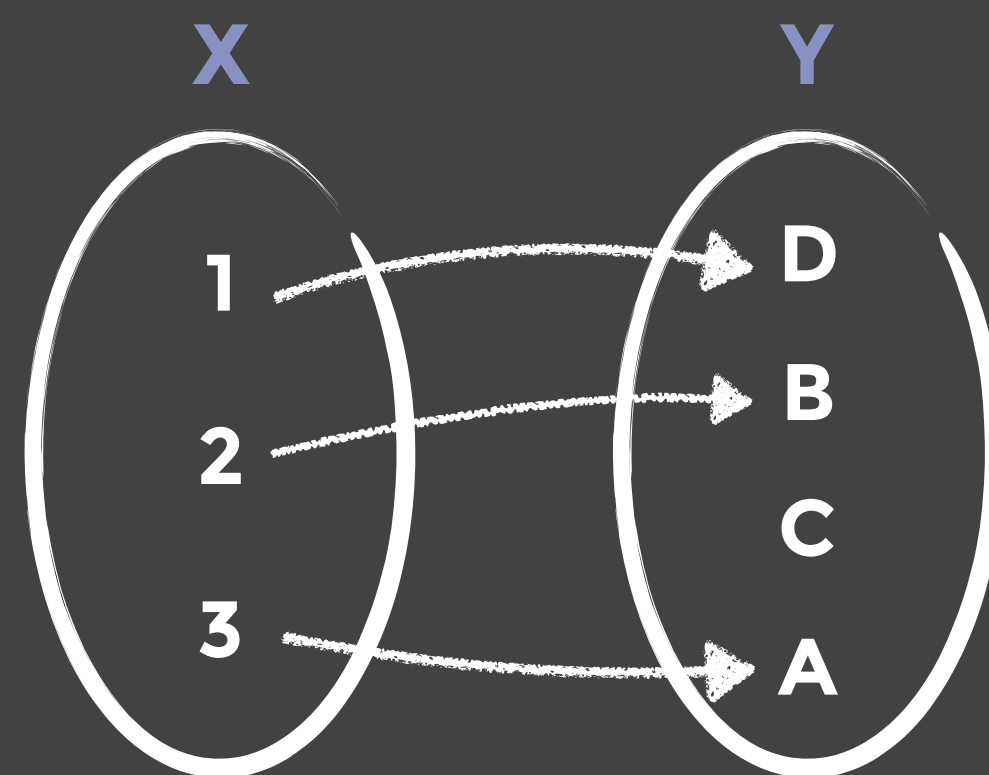
paradigma de programação que utiliza
funções puras e foca na **transformação do**
estado

baseado no **cálculo lambda**
proposto por **Alonzo Church**
na década de 30



na programação funcional as **funções** são
tratadas como conceito principal

Uma função matemática trata-se de um simples mapeamento entre o domínio e o contra-domínio.



Estado



o **estado de um programa** é representado
pelos **valores dos dados** armazenados na
memória...

...em qualquer ponto de execução do
programa

o **estado** de uma aplicação é alterado a
cada **interação** feita pelo usuário ou pelo
próprio sistema...

...e pode ser representado por uma
estrutura de dados

**a maioria dos bugs são relacionados ao
controle de estado**

Funções Puras



funções puras

- 1. ter** parâmetros de entrada
- 2. não** depender do estado externo
- 3. retorno** baseado nos valores de entrada
- 4. não** devem causar efeitos colaterais

por que utilizar funções puras?

**são reutilizáveis, componíveis, fáceis de
testar, fáceis de cachear e paralelizáveis**

transparência referencial

propriedade que garante que a saída de
uma função pura sempre será a mesma
dado um mesmo conjunto de argumentos


```
// referential transparency
```

```
$add = fn($x, $y) => $x + $y;
```

```
var_dump($add(2, 3) === 5); // true
```

```
var_dump(5 === 5); // true
```

pode não ser fácil criar **funções puras**

porém, a **restritividade** ajuda a melhorar o
foco

Mais sobre Funções



funções de primeira classe

são funções que podem ser tratadas como valores, ou seja, podem ser atribuídas a variáveis, passadas como argumentos e retornadas de uma função


```
// first-class function
```

```
$add = fn($x, $y) => $x + $y;  
$numbers = [1, 2, 3, 4, 5];
```

```
$sum = array_reduce($numbers, $add);  
$sum10 = array_reduce($numbers, $add, 10);
```

```
print $sum; // 15  
print $sum10; // 25
```

funções de alta ordem
são funções que operam sobre outras
funções


```
// high-order function
```

```
$add = fn($x, $y) => $x + $y;
```

```
$numbers = [1, 2, 3, 4, 5];
```

```
$sum = array_reduce($numbers, $add);
```

```
$sum10 = array_reduce($numbers, $add, 10);
```

```
print $sum; // 15
```

```
print $sum10; // 25
```

```
// high-order function
```

```
$add = fn($x, $y) => $x + $y;
```

```
$numbers = [1, 2, 3, 4, 5];
```

```
$sum = array_reduce($numbers, $add);
```

```
$sum10 = array_reduce($numbers, $add, 10);
```

```
print $sum; // 15
```

```
print $sum10; // 25
```

Os conceitos de funções de primeira classe e funções de alta ordem estão intimamente relacionados, uma não existiria sem a outra.

funções anônimas

**funções que não possuem nome e que,
geralmente, são passadas como argumento
ou atribuídas**

escopo

**uma função lambda ou função anônima tem
o seu próprio escopo como qualquer função
no PHP**

escopo

**o escopo é tratado de forma diferente entre
as linguagens JavaScript e PHP**

escopo

no JavaScript uma função anônima pode acessar uma variável do escopo externo enquanto no PHP isto não é permitido

closures

**funções que possuem acesso à valores do
escopo externo**

```
// closure function
```

```
function greet($greeting) {  
    return function ($name) use ($greeting) {  
        return "$greeting $name!";  
    };  
}
```

```
$greet = fn($greeting) => fn($name) => "$greeting $name!";
```

```
print greet('Hello')('Mary'); // Hello Mary!  
print $greet('Hello')('Mary'); // Hello Mary!
```

closures

as closures em PHP utilizam a abordagem **early-binding**, ou seja, as variáveis da closure terão os valores que tinham quando a closure foi definida

closures

isso pode ser sobrescrito utilizando

passagem por referência ou redefinindo

uma closure

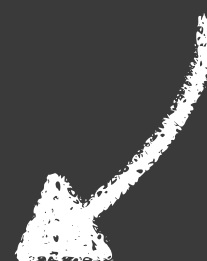
funções como objetos

no PHP as funções são consideradas objetos

a classe deve implementar
o método `__invoke`

```
// function as object
```

```
class Hello {  
    public function __invoke($name) {  
        return "Hello $name!";  
    }  
}
```



```
$hello = new Hello;  
print $hello('John'); // Hello John!
```

recursão

é quando uma função é definida em termos de si própria, ou seja, quando a função chama ela mesma

```
// recursive function
```

```
function factorial($number) {  
    if ($number == 1) {  
        return 1;  
    } else {  
        return ($number * factorial($number - 1));  
    }  
}
```

```
print factorial(5); // 120
```



```
// recursive function (using closure syntax)

$factorial = function ($number) use (&$factorial) {
    if ($number == 1) {
        return 1;
    } else {
        return ($number * $factorial($number - 1));
    }
};

print $factorial(5); // 120
```

**diferença entre função e procedimento
(procedure)**

**uma função recebe um valor e retorna um
resultado; um procedimento é um conjunto
de comandos executados numa ordem**

*"Don't think of functions as a collection of instructions. Think of them as non-destructive operations on input `double = n => n * 2;`"*

Eric Elliott, 2016.

https://twitter.com/_ericelliott/status/685172918784004097

memoize

técnica que permite que funções custosas
sejam cacheadas para execuções posteriores
mais rápidas

```
// expensive cost function memoized

$factorial = $memoize(function ($number) use (&$factorial) {
    sleep(1); // it takes 1s to run!

    if ($number == 1) {
        return 1;
    } else {
        return ($number * $factorial($number - 1));
    }
});

print $factorial(5); // 120 (after 5s)
print $factorial(7); // 5040 (after 2s)
```

Imutabilidad



a **imutabilidade** diz que um dado não pode ser alterado após a sua criação

a **imutabilidade** permite maior confiança e evita que erros ocorram

o PHP não possui suporte completo a **dados imutáveis** de forma nativa

porém, algumas técnicas como **value objects** podem ser utilizadas para alcançar a imutabilidade

Currying e Aplicação Parcial



o **currying** é a técnica que permite transformar uma função que recebe múltiplos argumentos...

...em uma função que recebe apenas **um argumento** e que retorna uma função que aceita os argumentos restantes

```
// uncurried function
```

```
function greet($greeting, $name) {  
    return "$greeting $name";  
}
```

```
print greet('Good morning', 'Alice'); // Good morning Alice
```

```
$greetMorning = greet('Good morning');
```

```
// PHP Fatal error:  Uncaught ArgumentCountError: Too
```

```
// few arguments to function greet(), 1 passed
```

```
// "curried" function

function greet($greeting) {
    return function ($name) use ($greeting) {
        return "$greeting $name";
    };
}

$greetMorning = greet('Good morning');

print $greetMorning('Alice'); // Good morning Alice
print greet('Good morning')('Alice'); // Good morning Alice
print greet('Good morning', 'Alice'); // error!
```

```
// "curried" function using arrow function (PHP 7.4)
```

```
$greet = fn($greeting) => fn($name) => "$greeting $name!";
```

```
$greetMorning = $greet('Good morning');
```

```
print $greetMorning('Alice'); // Good morning Alice
```

```
print $greet('Good morning')('Alice'); // Good morning Alice
```

```
print $greet('Good morning', 'Alice'); // error!
```



```
// curried function using Functional library
```

```
$greet = curry(function ($greeting, $name) {  
    return "$greeting $name";  
});
```

```
print $greet('Good morning')('Alice'); // Good morning Alice  
print $greet('Good morning', 'Alice'); // Good morning Alice
```

a **aplicação parcial** é quando se executa uma função e passa apenas **parte de seus argumentos**

a **aplicação parcial** permite fazer a
especialização de uma função mais genérica

```
// specialization from a curried function
// using partial application

$greet = curry(function ($greeting, $name) {
    return "$greeting $name";
});

$greetMorning = $greet('Good morning');

print $greetMorning('Alice'); // Good morning Alice
```


currying e aplicação parcial são recursos
muito utilizados em programação funcional

na programação funcional deve-se levar em
consideração a **ordem dos parâmetros**

os parâmetros mais **genéricos** devem vir
mais para o início e os parâmetros mais
específicos devem vir mais para o final

o PHP não possui **suporte nativo** para **currying** como nas linguagens **puramente funcionais** Elm ou Haskell

Composição de Funções



a **composição** é o processo de **combinar**
uma ou mais funções para criar uma **nova**
função

```
// create a function using composition (traditional way)

$sentence = 'estava à toa na vida o meu amor me chamou pra
            ver a banda passar cantando coisas de amor';

$wordCount = function ($text) {
    return count(explode(' ', $text));
};

print $wordCount($sentence); // 19
```

é uma solução **elegante** e **legível** e ajuda a evitar a utilização do **aninhamento de funções**

a biblioteca Functional possui uma função
que permite criar uma **nova função** a partir
da **composição de funções**

```
// create a function using composition

$sentence = 'estava à toa na vida o meu amor me chamou pra
            ver a banda passar cantando coisas de amor';
$wordCount = compose(partial('explode', ' '), 'count');

print $wordCount($sentence); // 19
```


Diferenças entre OO e FP



um **objeto** possui características,
comportamentos e estado atual e realiza
operações sobre o seu próprio estado

uma **função** não faz alterações diretas no estado e, sim, retorna novas transformações do estado atual

**Caso de
Uso**



Caso de Uso 1

somar os preços dos produtos de um carrinho de compra

Passo 1

```
// shopping cart
$cart = [
    ['id' => 1, 'product' => 'iPhone', 'price' => 499],
    ['id' => 2, 'product' => 'Kindle', 'price' => 179],
    ['id' => 3, 'product' => 'MacBook Pro', 'price' => 1199],
];

$total = 0;

// get price from shopping cart and sum them
// imperative way (using for-loop and accumulator)
for ($i = 0; $i < count($cart); $i++) {
    $total += $cart[$i]['price'];
}

print $total; // 1877
```

Passo 2

```
// shopping cart
$cart = [
  ['id' => 1, 'product' => 'iPhone', 'price' => 499],
  ['id' => 2, 'product' => 'Kindle', 'price' => 179],
  ['id' => 3, 'product' => 'MacBook Pro', 'price' => 1199],
];
```

```
// get prices from shopping cart and sum them
// functional way (intermediate values)
$cartPrices = map(fn($item) => $item['price'], $cart);
$total = sum($cartPrices);
```

```
print $total; // 1877
```

faz a somatória da lista de
números e retorna o total

realiza o mapeamento da lista
de produtos (objetos) para
uma lista de preços (números)

Passo 3

```
// shopping cart
$cart = [
    ['id' => 1, 'product' => 'iPhone', 'price' => 499],
    ['id' => 2, 'product' => 'Kindle', 'price' => 179],
    ['id' => 3, 'product' => 'MacBook Pro', 'price' => 1199],
];

// get prices from shopping cart and sum them
// functional way (functional composition)
$totalCart = compose(
    map(fn($item) => $item['price']),
    'f\sum'
);

print $totalCart($cart); // 1877
```

cria uma nova função a partir da composição de funções e elimina valores intermediários

aplicação parcial da função map

Outras Bibliotecas



bibliotecas

- 1.** [lstrojny/functional-php](#)
- 2.** [kapolos/pramda](#)
- 3.** [sergiors/prelude](#) 🇧🇷
- 4.** [widmogrod/php-functional](#)
- 5.** [nikic/iter](#)
- 6.** [leocavalcante/siler](#) 🇧🇷
- 7.** [ihor/Nspl](#)

Conclusão



as linguagens de programação **tradicionais**
têm adotado **conceitos de programação**
funcional logo é importante conhecê-los

por mais que o PHP não tenha um **suporte**
amplo a programação funcional o **mindset**
funcional te ajudará a se tornar um(a)
programador(a) melhor

**a programação funcional não é sobre não
ter estado...**

...e sim sobre **eliminar** estado e efeito
colateral **sempre** que possível e **controlar**
efeitos colaterais quando necessário

foque na **transformação do estado** e evite
efeitos colaterais

conhecer bem os **paradigmas de programação** te permite escolher a **melhor ferramenta** para cada problema

*A programação funcional garante o que é
código bom em outros paradigmas.*

existem inúmeros conceitos relacionados a
programação funcional como functors,
monads, lazy evaluation, tail call
optimization...

vá em frente e divirta-se!

Referências

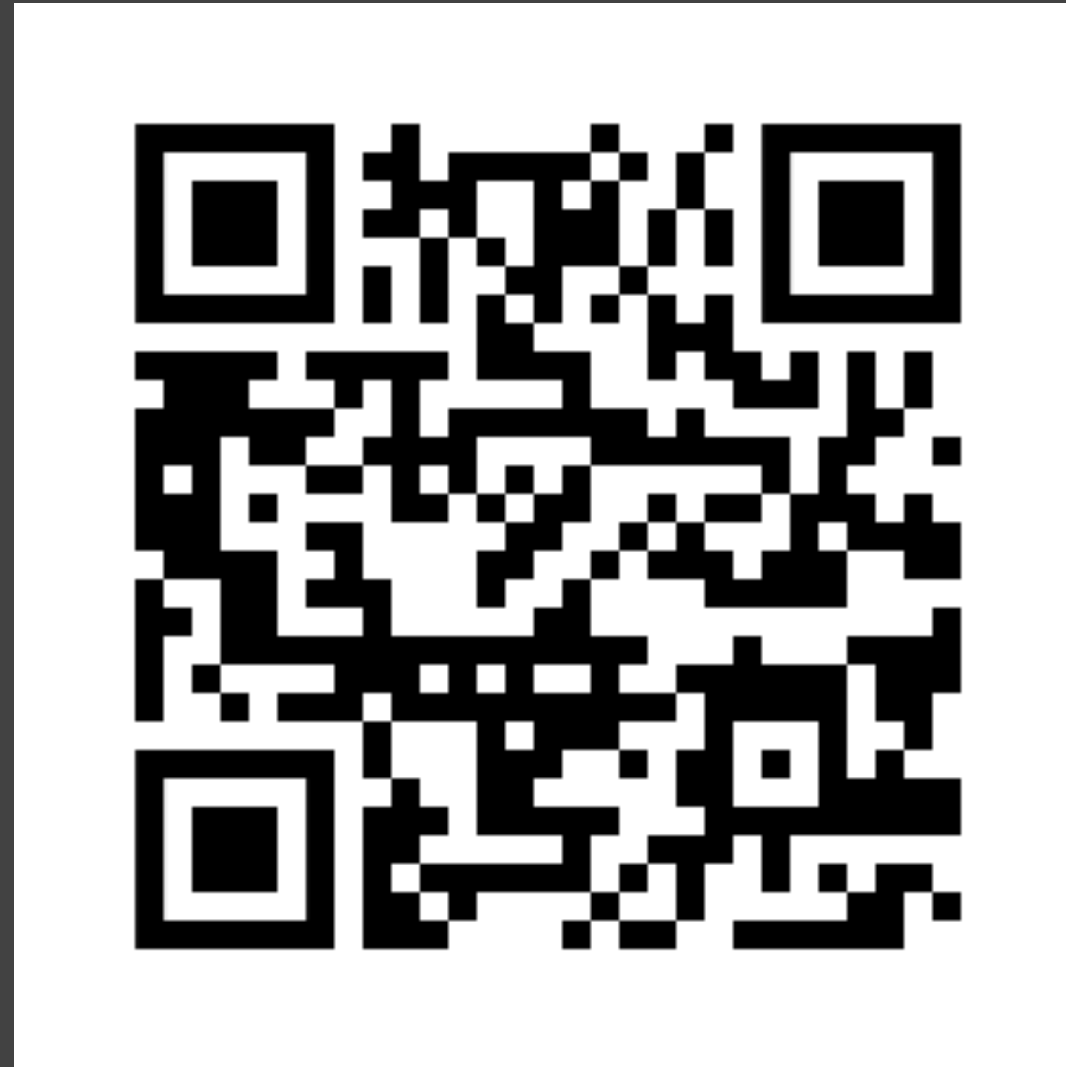


Getting Started with FP in PHP

Tutoriais, vídeos, slides, bibliotecas, pessoas influentes sobre programação funcional em PHP.

bit.ly/Getting-Started-with-FP-in-PHP

Avalie!



bit.ly/avalie-palestra-php-funcional

Anúncio!

Novas Funcionalidades do PHP 7.4



Curso presencial com
Marcel Gonçalves dos Santos

📍 Em SP no CT da Novatec

📅 01/10 (Terça-feira)

🕒 19h às 23h

INSCRIÇÕES ABERTAS

bit.ly/workshop-php74

Informações: será apresentado um histórico breve da linguagem, as principais funcionalidades do PHP 7.4 de forma prática como typed properties, arrow functions, spread operator, preloading, FFI entre outras e funcionalidades futuras do PHP 8.0.

Obrigado.
Perguntas?

@marcelgsantos



speakerdeck.com/marcelgsantos