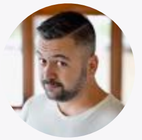
Guia de codificação (PSRs)

Uma análise das principais recomendações do PHP Framework Interop Group.

https://fullstackphpdeveloper.com

upinside
vem aprender com a gente!

```
{
  "name": "UpInside/fsphp",
  "description": "Full Stack PHP Developer",
  "minimum-stability": "stable",
  "authors": [
    {
      "name": "Robson V. Leite",
        "email": "cursos@upinside.com.br"
        "homepage": "https://upinside.com.br",
        "role": "Developer"
    }
  ],
  "config": {
    "vendor-lib": "vendor"
  },
  "autoload": {
    "psr-4": {
      "Source\\": "source/"
    }
  }
}
```



Robson V. Leite

CEO UpInside Treinamentos
cursos@upinside.com.br
www.upinside.com.br

<?= "page X"; ?>

PSR-1: Padrão básico de codificação:

O que DEVE ser considerado como codificação padrão para garantir um alto nível de interoperabilidade técnica entre códigos compartilhados.

/Tags:

Use as tags **<?php ;?>** para abertura e fechamento e **<?= ;?>** para saída. Nunca use outras variações (ex: <%, %>, <%=)

file.php

```
<?php
  echo "Olá mundo!";
?>

<h1><?= "Olá mundo!"; ?></h1>
```

/Codificação de caracteres:

Arquivos devem usar apenas UTF-8 sem BOM para código PHP. (IDE, Charset, Etc.)

/Efeitos colaterais (side effects):

Você pode (incluir um arquivo, conectar a um serviço, gerar uma saída) OU (declarar uma classe, criar uma função, definir uma constante). Mas nunca faça ambos no mesmo arquivo.

file.php

```
<?php

  require "config.php"; /Arquivo com constantes
  require "functions.php"; //Arquivo de funções

  echo HELLO;
  useMyFunction();

  DEFINE("HELLO", "Olá mundo!");

  mySecondeFunction() {
     //Function Body
  }
```

PSR-1: Padrão básico de codificação:

/Classes PHP:

Cada classe deve estar em seu próprio arquivo e ter pelo menos um nível de namespace (padrão PSR-4). O nome da classe deve ser declarada em **StudlyCaps**.

/Constantes da classe devem ser declaradas em maiúscula e quando preciso podem ser separadas por under_score.

/Propriedades da classe podem ser escritas em $StudlyCaps, $camelCase ou $under_score, não existe uma recomendação rígida para elas, mas é sempre importante escolher uma e usar sempre a mesma. Se for $camelCase, use sempre $camelCase e nunca as outras.

/Métodos da classe devem ser declarados sempre em camelCase()

MyClass.php

```php
<?php

namespace Source;

class MyClass
{
  const COURSE = "FSPHP";
  const CLASS_NAME = "PSR-1";

  public $ModuleName; //APENAS $StudlyCaps
  public $className; //OU APENAS $camelCase
  public $class_time; //OU APENAS $under_score

  public function getClassName()
  {
     return $this->className;
  }

  public function getClassTime()
  {
     //Erro pois já usamos $camelCase acima.
     return $this->class_time;
  }
}
```

PSR-2: Guia de estilo de codificação:

Reduzir o atrito cognitivo ao escanear códigos de diferentes autores. A PSR-2 enumerando um conjunto compartilhado de regras e expectativas sobre como formatar o código PHP.

/Padrão básico de codificação: O código deve seguir todas as recomendações listadas na PSR-1.

/Arquivos: Sempre devem terminar com uma linha em branco (Unix linefeed).

/Somente PHP: Um arquivo somente com PHP deve omitir a tag **?>** de fechamento.

IDE **/Linhas:** Procure manter suas linhas com no máximo 80 caracteres, quando necessário você poderá usar até 120.

IDE **/Espaços finais:** Certifique-se que o último caractere da linha não é um espaço em branco.

/Legibilidade: Pode adicionar linhas em branco para separar blocos de código.

IDE **/Recuo:** Use 4 espaços para indentar seu código, nunca o TAB.

IDE **/Declaração:** Não deve ter mais de uma por linha de código.

/true, false, null: São palavras-chave e constantes do PHP, devem ser escritas sempre em letra minúscula.

IDE A IDE é sua ferramenta de programação, PHPStorm, netBeans entre outras. Elas já implementam as PSR`s ou tem recursos para automatizar essa implementação.

PSR-2: Guia de estilo de codificação:

/Namespaces: Após declarar um, sempre deixe uma linha em branco para então continuar seu código.

/use: Quando presentes devem ser declarados após os namespaces.

DEVE haver um use por declaração.

DEVE haver um espaço após o bloco de declaração do use.

MyClass.php

```php
<?php

namespace Source\Controller;

use Source\Model\MyClass, Source\Conection\MyClass;
use Source\Model\MyClass;
use Source\Conection\MyClass;

class MyClass
{
   //Class Body
}
```

/Classes, propriedades e métodos:

Entenda classes como todas as classes, interfaces e traits.

/extends e implements devem SEMPRE ser declarados na mesma linha do nome da classe, se ambos, primeiro o extends.

IDE A chave de abertura da classe deve seguir na próxima linha após o nome e a chave de fechamento deve seguir na próxima linha após o corpo.

MyClass.php

```php
<?php

namespace Source\Controller;

use Source\Parent\ParentClass
use Source\Parent\iClass

class MyClass extends ParentClass implements iClass
{
   //Class Body
}
```

PSR-2: Guia de estilo de codificação:

/Classes, propriedades e métodos:

/em uma lista de implements você pode declarar todas as interfaces em uma linha ou declarar uma por linha.

IDE Em uma por linha, cada linha subsequente deve ser recuada uma vez e o primeiro item deve estar na próxima linha e deve haver apenas uma interface por linha.

/Propriedades devem sempre declarar a visibilidade (public, protected, private) e nunca deve usar em sua declaração a palavra-chave **var**.

Não deve haver mais de uma propriedade declarada por linha e nunca use **underscore_** ou **_underscore** para declarar visibilidade protegida ou privada.

MyClass.php

```php
<?php

namespace Source\Controller;

use Source\Parent\iClass

class MyClass implements
    iClass,
    \ArrayAccess,
    \Countable
{
   //Class Body
}
```

MyClass.php

```php
<?php

namespace Source\Controller;

class Course
{
   var $courseName;
   private $courseName_;
   public $courseName;
   protected $courseAuthor = "Robson V. Leite";
   private $coursePrice;
}
```

PSR-2: Guia de estilo de codificação:

/Classes, propriedades e métodos:

/Métodos assim como propriedades devem ter sua visibilidade declarada e não usar underscore_ ou _underscore.

Nomes de métodos não podem conter espaços nele ou após ele. A chave de abertura deve estar na sua própria linha, a de fechamento na próxima linha após o corpo e devem ser declarados com $camelCase().

/Argumentos do método: Na lista de argumentos deve haver um espaço depois da vírgula mas não antes, argumentos com valores padrão devem ir ao final da lista.

IDE A lista de argumentos também podem ser declaradas na próxima linha. Quando isso ocorrer cada argumento assim como os parênteses devem estar em uma linha subsequente e você deve adicionar um recuo para todos.

MyClass.php

```php
<?php

namespace Source\Controller;

class Course
{
   private $courseName;
   private $courseAuthor;
   private $coursePrice;

   public function setName($name)
   {
      $this->courseName = $name;
   }

   public function setMore($course, $price = "997")
   {
      //Method Body
   }

   public function setAll(
      $course,
      $author,
      $price = "997"
   )
   {
      //Method Body
   }
}
```

PSR-2: Guia de estilo de codificação:

/abstract, final e static:

/abstract e final quando presentes, devem ser declaradas ANTES da declaração de visibilidade.

Quando presente, a **static** deve ser declarada DEPOIS da declaração de visibilidade.

MyClass.php

```php
<?php

namespace Source\Controller;

abstratic class Course
{
    private static $courseName;

    abstract public function getCourse()
    {
        //Method Body
    }

    final public function getPrice()
    {
        //Method Body
    }
}
```

/chamada de métodos funções:

Chamadas de métodos e funções não use espaços entre o nome declarado e os parênteses de abertura e fechamento.

Na lista de argumentos deve haver um espaço depois da vírgula mas não antes.

A lista de argumentos também pode ser dividida em várias linhas.

file.php

```php
<?php

 myFunction();
 myFunction($arg, $argB);

 $course->getCourse();
 $course->setPrice("997");

 $course->setAll(
    "FS PHP",
    "Robson V. Leite"
 );
```

PSR-2: Guia de estilo de codificação:

/Regras gerais da PSR para estruturas de controle:

DEVE haver um espaço após a palavra-chave da estrutura de controle.

NÃO DEVE haver um espaço após o parêntese de abertura nem antes do parêntese de fechamento.

DEVE haver um espaço entre o parêntese de abertura e a chave de abertura.

O corpo da estrutura de controle **DEVE** ser recuado uma vez.

A chave de fechamento **DEVE** estar na próxima linha após o corpo da estrutura.

O corpo de qualquer estrutura **DEVE** estar entre chaves.

file.php

```php
<?php

if ($arg) {
    // if body
} elseif ($argB) {
    // elseif body
} else {
    // else body;
}

switch ($argTest) {
    case 0:
        //First Case
        break;
    default:
        //No tested In Case
        break;
}

try {
    // try body
} catch (ClassExceptionType $e) {
    // catch body
}
```

PSR-2: Guia de estilo de codificação:

/Closures:

DEVEM ser declaradas com um espaço depois de function e um espaço antes e depois de use.

A chave de abertura **DEVE** ir na mesma linha do nome, a chave de fechamento **DEVE** ir uma linha após o corpo.

NÃO DEVE haver espaço após o parêntese de abertura ou antes do parêntese de fechamento na lista de argumentos ou variáveis.

DEVE haver na lista de argumentos ou variáveis um espaço depois da vírgula mas nunca antes.

Argumentos com valor padrão **DEVEM** ir ao final da lista de argumentos.

file.php

```php
<?php

$myClosure = function($argA, $argB) {
    //Closure Body
};

$myClosureB = function($argA) use ($varA) {
    //Closure B Body
};

$myLongClosureArgs = function(
    $argA,
    $argB,
    $argC
) {
    //Closure Long Body
};

$myLongClosureArgsB = function(
    $argA,
    $argB
) use(
    $varA,
    $varB
) {
    //Closure Long B Body
};
```

PSR-4: Carregamento automático:

Este PSR descreve uma especificação para o carregamento automático e interoperável das classes, assim como mostra onde colocar os arquivos em seu projeto.

/Especificação:

Entenda classes como todas as classes, interfaces e traits.

1) Um nome de classe totalmente qualificado deve seguir o seguinte formato:

```
\<VendorNamespace>(\<SubNamespace>)*\<ClassName>
```

   1) O namespace completo **DEVE** ter um nome de nível superior (Vendor).
   2) O namespace **PODE** ter um ou mais sub-namespaces.
   3) O namespace **DEVE** terminar com o nome da classe.
   4) Underscore não tem qualquer efeito especial no namespace.
   5) Caracteres alfabéticos **PODEM** ter qualquer combinação de minúsculas e maiúsculas.
   6) Todos os namespaces **DEVEM** ser referenciados de forma única.
   7) O Vendor namespace e alguns dos primeiros níveis de sub-namespace **DEVEM** corresponder a um diretório base.
   8) Cada sub-namespace seguinte deve corresponder a um sub-diretório dentro do diretório base, cada separador de sub-namespace corresponde a um separador de diretório no sistema operacional.

2) Implementações de autoloader **NÃO DEVEM** lançar exceções, gerar erros de qualquer nível ou retornar um valor.

PSR-4: Carregamento automático:

Um exemplo de autoload:

autoload.php

```php
<?php

autoload_register(function ($class) {

    //Define o Vendor Namespace e a pasta raiz.
    $prefix = 'Source\\';
    $baseDir = __DIR__ . '/source/';

    //Verifica o namespace vendor na classe
    $len = strlen($prefix);
    if (strncmp($prefix, $class, $len) !== 0) {
        return;
    }

    //Obtém o nome da classe e substitui o namespace por diretório.
    $relativeClass = substr($class, $len);
    $file = $baseDir . str_replace('\\', '/', $relativeClass) . '.php';

    //Carrega o arquivo de classe se ele existir.
    if (file_exists($file)) {
        require $file;
    }

});
```

PSR-4: Carregamento automático:

Uma forma mais prática, inteligente e interoperável de criar seu autoload usando o **Composer** para gerenciar todas as dependências do projeto:

composer.json

```
{
  "name": "UpInside/fsphp",
  "description": "Full Stack PHP Developer",
  "minimum-stability": "stable",
  "authors": [
    {
      "name": "Robson V. Leite",
      "email": "cursos@upinside.com.br"
      "homepage": "https://upinside.com.br",
      "role": "Developer"
    }
  ],
  "config": {
    "vendor-lib": "vendor"
  },
  "autoload": {
    "psr-4": {
      "Source\\": "source/"
    }
  }
}
```

O exemplo ao lado mostra o arquivo de configuração **composer.json** do Composer.

Com ele o gerenciador de dependências PHP poderá automatizar todo o processo de carregamento de classes e componentes para você.

A configuração autoload:

```
"autoload": {
    "psr-4": {
      "Source\\": "source/"
    }
}
```

Neste ponto estamos informando que nossas classes estão no namespace fornecedor `Source`, e dentro da pasta raiz `source` do projeto. Depois de rodar o Composer, basta invocar o autoload:

```
require __DIR__ . "/vendor/autoload.php";
```