Estándar de Programación Lenguaje PHP

Grupo Nº 1 – Gestión de Calidad de Software

Sartini Nicolás – Muñoz Rut – Cadin Vanesa

Este documento refleja los estándares PSR-1 (Estándar de codificación básico) y PSR-12 (Guía de estilo de codificación ampliada) referidos a los estilos de codificación del lenguaje PHP presentados por PHP Framework Interop Group (PHP - FIG).

El mismo no solo busca definir la nomenclatura de las variables, objetos, métodos y funciones, sino que también hace referencia al orden y legibilidad del código escrito.

Tabla de contenido

Introducción **5**

PSR-1: Estándar de codificación básico **5**

Introducción *5*

1. Información general *5*

2. Archivos *6*

2.1. Etiquetas PHP 6

2.2. Codificación de caracteres 6

2.3. Efectos secundarios 6

3. Espacio de nombres y nombres de clases *7*

} **8**

4. Constantes de clase, propiedades y métodos *8*

4.1. Constantes 8

4.2. Propiedades 9

4.3. Métodos 9

PSR-12: Guía de estilo de codificación ampliada **9**

Introducción *9*

2. General *9*

2.1 Estándar de codificación básico 9

2.2 Archivos 10

2.3 Líneas 10

2.4 Sangría 10

2.5 Palabras clave y tipos 10

3. Declarar declaraciones, espacio de nombres e importar declaraciones *11*

4. Clases, propiedades y métodos *13*

new Foo(); **14**

4.1 Extiende e Implementa 14

4.2 Usando rasgos 15

4.3 Propiedades y constantes 17

4.4 Métodos y funciones 18

4.5 Argumentos de método y función 19

4.6 abstract, final y static 22

4.7 Llamadas a métodos y funciones 22

5. Estructuras de control *23*

5.1 if, elseif, else 24

} 24

5.2 switch, case 25

} 25

5.3 while, do while 26
} 26

} while ($expr); 26

5.4 for 27
5.5 foreach 28
5.6 try, catch, finally 28
} 28

6. Operadores 28
6.1. Operadores unarios 29
6.2. Operadores binarios 29
6.3. Operadores ternarios 29

$variable = $foo ? 'foo' : 'bar'; 29

7. Cierres 30

8. Clases anónimas 32

Convenciones del código PHP 33

Referencias: 33

Estándar de Programación Lenguaje PHP

Introducción

El presente documento detalla las pautas generales a ser consideradas en la codificación, como se mencionan en los Estándares PSR-1 (Estándar de codificación básico) y PSR-12 (Guía de estilo de codificación ampliada) de codificación del lenguaje PHP presentados por PHP Framework Interop Group (PHP - FIG).

PSR-1: Estándar de codificación básico

Introducción

Esta sección del estándar comprende lo que deben considerarse los elementos de codificación estándar que se requieren para garantizar un alto nivel de interoperabilidad técnica entre el código PHP compartido.

1. Información general

- Los archivos DEBEN usar solo etiquetas <?php y <?=.
- Los archivos DEBEN usar solo UTF-8 sin BOM para el código PHP.
- Los archivos deben declarar ya sea símbolos (clases, funciones, constantes, etc.) o producir efectos secundarios (por ejemplo, generar una salida, la configuración de .ini de cambio, etc.) pero no debe hacer ambos.
- Los espacios de nombres y las clases DEBEN seguir un PSR de "carga automática": [PSR-0 , PSR-4].
- Los nombres de las clases DEBEN declararse en formato StudlyCaps.
- Las constantes de clase DEBEN declararse en mayúsculas con separadores de subrayado.
- Los nombres de los métodos DEBEN declararse en formato camelCase.

2. Archivos

2.1. Etiquetas PHP

El código PHP DEBE usar las <?php ?>etiquetas largas o las <?= ?>etiquetas de eco corto ; NO DEBE utilizar las otras variaciones de etiqueta.

2.2. Codificación de caracteres

El código PHP DEBE usar solo UTF-8 sin BOM.

2.3. Efectos secundarios

Un archivo DEBE declarar nuevos símbolos (clases, funciones, constantes, etc.) y no causar otros efectos secundarios, o DEBE ejecutar lógica con efectos secundarios, pero NO DEBE hacer ambas cosas.

La frase "efectos secundarios" significa la ejecución de la lógica que no está directamente relacionada con la declaración de clases, funciones, constantes, etc., *simplemente por incluir el archivo* .

Los "efectos secundarios" incluyen, entre otros: generar resultados, uso explícito de requireo include, conectarse a servicios externos, modificar la configuración de ini, emitir errores o excepciones, modificar variables globales o estáticas, leer o escribir en un archivo, etc. .

El siguiente es un ejemplo de un archivo con declaraciones y efectos secundarios; es decir, un ejemplo de lo que se debe evitar:

```php
<?php
// side effect: change ini settings
ini_set('error_reporting', E_ALL);

// side effect: loads a file
include "file.php";

// side effect: generates output
echo "<html>\n";

// declaration
function foo()
{
```

```
    // function body
}
```

El siguiente ejemplo es de un archivo que contiene declaraciones sin efectos secundarios; es decir, un ejemplo de qué emular:

```php
<?php
// declaration
function foo()
{
    // function body
}

// conditional declaration is *not* a side effect
if (! function_exists('bar')) {
    function bar()
    {
        // function body
    }

}
```

3. Espacio de nombres y nombres de clases

Los espacios de nombres y las clases DEBEN seguir un PSR de "carga automática": [PSR-0 , PSR-4].

Esto significa que cada clase está en un archivo por sí misma y está en un espacio de nombres de al menos un nivel: un nombre de proveedor de nivel superior.

Los nombres de las clases DEBEN declararse en formato StudlyCaps.

El código escrito para PHP 5.3 y posteriores DEBE usar espacios de nombres formales.

Por ejemplo:

```php
<?php
// PHP 5.3 and later:
namespace Vendor\Model;

class Foo
{
}
```

El código escrito para 5.2.xy antes DEBERÍA usar la convención de pseudo-espacio de nombres de Vendor_prefijos en nombres de clases.

```php
<?php
// PHP 5.2.x and earlier:
class Vendor_Model_Foo
{
}
```

4. Constantes de clase, propiedades y métodos

El término "clase" se refiere a todas las clases, interfaces y rasgos.

4.1. Constantes

Las constantes de clase DEBEN declararse en mayúsculas con separadores de subrayado. Por ejemplo:

```php
<?php
namespace Vendor\Model;

class Foo
{
    const VERSION = '1.0';
    const DATE_APPROVED = '2012-06-01';
}
```

4.2. Propiedades

En esta guía se evita intencionadamente ninguna recomendación sobre el uso de $StudlyCaps, $camelCase o $under_score nombres de propiedades.

Cualquiera que sea la convención de nomenclatura que se utilice DEBE aplicarse de manera coherente dentro de un alcance razonable. Ese alcance puede ser a nivel de proveedor, a nivel de paquete, a nivel de clase o a nivel de método.

4.3. Métodos

Los nombres de los métodos DEBEN declararse en formato camelCase().

PSR-12: Guía de estilo de codificación ampliada

Introducción

Esta especificación amplía y reemplaza la guía de estilo de codificación PSR-2 , y requiere el cumplimiento del estándar de codificación básico PSR-1.

Al igual que PSR-2 , la intención de esta especificación es reducir la fricción cognitiva al escanear código de diferentes autores. Lo hace enumerando un conjunto compartido de reglas y expectativas sobre cómo formatear el código PHP. Este PSR busca proporcionar una forma establecida en que las herramientas de estilo de codificación pueden implementar, los proyectos pueden declarar la adherencia y los desarrolladores pueden relacionarse fácilmente entre diferentes proyectos. Cuando varios autores colaboran en varios proyectos, es útil tener un conjunto de pautas para usar entre todos esos proyectos. Por lo tanto, el beneficio de esta guía no está en las reglas en sí, sino en compartir esas reglas.

2. General

2.1 Estándar de codificación básico

El código DEBE seguir todas las reglas descritas en PSR-1 .

El término 'StudlyCaps' en PSR-1 DEBE interpretarse como PascalCase, donde la primera letra de cada palabra está en mayúscula, incluida la primera letra.

2.2 Archivos

Todos los archivos PHP DEBEN usar el final de línea Unix LF (salto de línea) solamente.

Todos los archivos PHP DEBEN terminar con una línea que no esté en blanco, terminada con un solo LF.

La etiqueta de cierre ?> DEBE omitirse de los archivos que solo contienen PHP.

2.3 Líneas

NO DEBE haber un límite estricto en la longitud de la línea.

El límite flexible de la longitud de la línea DEBE ser de 120 caracteres.

Las líneas NO DEBEN tener más de 80 caracteres; las líneas más largas que esa DEBERÍAN dividirse en varias líneas posteriores de no más de 80 caracteres cada una.

NO DEBE haber espacios en blanco al final de las líneas.

PUEDEN agregarse líneas en blanco para mejorar la legibilidad y para indicar bloques de código relacionados, excepto donde esté explícitamente prohibido.

NO DEBE haber más de una declaración por línea.

2.4 Sangría

El código DEBE usar una sangría de 4 espacios para cada nivel de sangría, y NO DEBE usar tabulaciones para sangrar.

2.5 Palabras clave y tipos

Todos los tipos y palabras clave reservados de PHP [1] [2] DEBEN estar en minúsculas.

Los nuevos tipos y palabras clave que se agreguen a futuras versiones de PHP DEBEN estar en minúsculas.

DEBE usarse una forma corta de palabras clave de tipo, es decir, bool en lugar de boolean, int en lugar de integer, etc.

3. Declarar declaraciones, espacio de nombres e importar declaraciones

El encabezado de un archivo PHP puede constar de varios bloques diferentes. Si está presente, cada uno de los bloques a continuación DEBE estar separado por una sola línea en blanco y NO DEBE contener una línea en blanco. Cada bloque DEBE estar en el orden que se indica a continuación, aunque los bloques que no son relevantes pueden omitirse.

- Etiqueta de apertura <?php .
- Docblock a nivel de archivo.
- Una o más declaraciones de declaración.
- La declaración de espacio de nombres del archivo.
- Una o más declaraciones use de importación basadas en clases .
- Una o más declaraciones use de importación basadas en funciones .
- Una o más declaraciones use de importación basadas en constantes .
- El resto del código en el archivo.

Cuando un archivo contiene una combinación de HTML y PHP, se puede seguir utilizando cualquiera de las secciones anteriores. Si es así, DEBEN estar presentes en la parte superior del archivo, incluso si el resto del código consiste en una etiqueta PHP de cierre y luego una mezcla de HTML y PHP.

Cuando la etiqueta de apertura <?php está en la primera línea del archivo, DEBE estar en su propia línea sin otras declaraciones a menos que sea un archivo que contenga marcas fuera de las etiquetas de apertura y cierre de PHP.

Las declaraciones de importación nunca DEBEN comenzar con una barra invertida inicial, ya que siempre deben estar completamente calificadas.

El siguiente ejemplo ilustra una lista completa de todos los bloques:

```php
<?php

/**
 * This file contains an example of coding styles.
 */

declare(strict_types=1);
```

```php
namespace Vendor\Package;

use Vendor\Package\{ClassA as A, ClassB, ClassC as C};
use Vendor\Package\SomeNamespace\ClassD as D;
use Vendor\Package\AnotherNamespace\ClassE as E;

use function Vendor\Package\{functionA, functionB, functionC};
use function Another\Vendor\functionD;

use const Vendor\Package\{CONSTANT_A, CONSTANT_B, CONSTANT_C};
use const Another\Vendor\CONSTANT_D;

/**
 * FooBar is an example class.
 */
class FooBar
{
    // ... additional PHP code ...

}
```

NO SE DEBEN utilizar espacios de nombres compuestos con una profundidad de más de dos. Por lo tanto, la siguiente es la profundidad máxima de composición permitida:

```php
<?php

use Vendor\Package\SomeNamespace\{
    SubnamespaceOne\ClassA,
    SubnamespaceOne\ClassB,
    SubnamespaceTwo\ClassY,
    ClassZ,

};
```

Y no se permitiría lo siguiente:

```php
<?php

use Vendor\Package\SomeNamespace\{
    SubnamespaceOne\AnotherNamespace\ClassA,
    SubnamespaceOne\ClassB,
    ClassZ,

};
```

Cuando desee declarar tipos estrictos en archivos que contienen marcas fuera de las etiquetas de apertura y cierre de PHP, la declaración DEBE estar en la primera línea del archivo e incluir una etiqueta PHP de apertura, la declaración de tipos estrictos y la etiqueta de cierre.

Por ejemplo:

```
<?php declare(strict_types=1) ?>
<html>
<body>
    <?php
        // ... additional PHP code ...
    ?>
</body>

</html>
```

Las declaraciones de declaración NO DEBEN contener espacios y DEBEN ser exactas declare(strict_types=1) (con un terminador de punto y coma opcional).

Las declaraciones de declaración en bloque están permitidas y DEBEN tener el formato siguiente. Tenga en cuenta la posición de los tirantes y el espaciado:

```
declare(ticks=1) {
    // some code

}
```

4. Clases, propiedades y métodos

El término "clase" se refiere a todas las clases, interfaces y rasgos.

Cualquier llave de cierre NO DEBE ir seguida de ningún comentario o declaración en la misma línea.

Al crear una instancia de una nueva clase, los paréntesis DEBEN estar siempre presentes incluso cuando no se pasen argumentos al constructor.

```
new Foo();
```

4.1 Extiende e Implementa

Las palabras clave extends e implements DEBEN declararse en la misma línea que el nombre de la clase.

La llave de apertura para la clase DEBE ir en su propia línea; la llave de cierre para la clase DEBE ir en la siguiente línea después del cuerpo.

Las llaves de apertura DEBEN estar en su propia línea y NO DEBEN ir precedidas o seguidas de una línea en blanco.

Las llaves de cierre DEBEN estar en su propia línea y NO DEBEN ir precedidas de una línea en blanco.

```
<?php

namespace Vendor\Package;

use FooClass;
use BarClass as Bar;
use OtherVendor\OtherPackage\BazClass;

class ClassName extends ParentClass implements \ArrayAccess, \Countable
{
    // constants, properties, methods
}
```

Las listas de implements y, en el caso de interfaces, extends PUEDEN dividirse en varias líneas, donde cada línea subsiguiente se sangra una vez. Al hacerlo, el primer elemento de la lista DEBE estar en la siguiente línea y DEBE haber solo una interfaz por línea.

```php
<?php

namespace Vendor\Package;

use FooClass;
use BarClass as Bar;
use OtherVendor\OtherPackage\BazClass;

class ClassName extends ParentClass implements
    \ArrayAccess,
    \Countable,
    \Serializable
{
    // constants, properties, methods

}
```

4.2 Usando rasgos

La palabra clave use utilizada dentro de las clases para implementar rasgos DEBE declararse en la siguiente línea después de la llave de apertura.

```php
<?php

namespace Vendor\Package;

use Vendor\Package\FirstTrait;

class ClassName
{
    use FirstTrait;

}
```

Cada rasgo individual que se importa a una clase DEBE incluirse uno por línea y cada inclusión DEBE tener su propia declaración use de importación.

```php
<?php

namespace Vendor\Package;
```

```php
use Vendor\Package\FirstTrait;
use Vendor\Package\SecondTrait;
use Vendor\Package\ThirdTrait;

class ClassName
{
    use FirstTrait;
    use SecondTrait;
    use ThirdTrait;

}
```

Cuando la clase no tiene nada después de la declaración use de importación, la llave de cierre de la clase DEBE estar en la siguiente línea después de la declaración use de importación.

```php
<?php

namespace Vendor\Package;

use Vendor\Package\FirstTrait;

class ClassName
{
    use FirstTrait;

}
```

De lo contrario, DEBE tener una línea en blanco después de la declaración use de importación.

```php
<?php

namespace Vendor\Package;

use Vendor\Package\FirstTrait;

class ClassName
{
    use FirstTrait;

    private $property;

}
```

Al usar los operadores insteadof y as, deben usarse de la siguiente manera, tomando nota de la sangría, el espaciado y las nuevas líneas.

```php
<?php

class Talker
{
    use A;
    use B {
        A::smallTalk insteadof B;
    }
    use C {
        B::bigTalk insteadof C;
        C::mediumTalk as FooBar;
    }
}
```

4.3 Propiedades y constantes

La visibilidad DEBE declararse en todas las propiedades.

La visibilidad DEBE declararse en todas las constantes si la versión mínima de PHP de su proyecto admite visibilidades constantes (PHP 7.1 o posterior).

La varpalabra clave NO DEBE usarse para declarar una propiedad.

NO DEBE haber más de una propiedad declarada por declaración.

Los nombres de propiedad NO DEBEN tener el prefijo de un guión bajo para indicar visibilidad protegida o privada. Es decir, un prefijo de subrayado no tiene ningún significado explícitamente.

DEBE haber un espacio entre la declaración de tipo y el nombre de la propiedad.

Una declaración de propiedad se parece a lo siguiente:

```php
<?php

namespace Vendor\Package;

class ClassName
```

```
{
    public $foo = null;
    public static int $bar = 0;

}
```

4.4 Métodos y funciones

La visibilidad DEBE declararse en todos los métodos.

Los nombres de los métodos NO DEBEN tener un prefijo con un solo subrayado para indicar visibilidad protegida o privada. Es decir, un prefijo de subrayado no tiene ningún significado explícitamente.

Los nombres de métodos y funciones NO DEBEN declararse con un espacio después del nombre del método. La llave de apertura DEBE ir en su propia línea, y la llave de cierre DEBE ir en la siguiente línea que sigue al cuerpo. NO DEBE haber un espacio después del paréntesis de apertura, y NO DEBE haber un espacio antes del paréntesis de cierre.

Una declaración de método tiene el siguiente aspecto. Tenga en cuenta la ubicación de los paréntesis, las comas, los espacios y las llaves:

```
<?php

namespace Vendor\Package;

class ClassName
{
    public function fooBarBaz($arg1, &$arg2, $arg3 = [])
    {
        // method body
    }

}
```

Una declaración de función se parece a la siguiente. Tenga en cuenta la ubicación de los paréntesis, las comas, los espacios y las llaves:

```
<?php

function fooBarBaz($arg1, &$arg2, $arg3 = [])
```

```
{
    // function body
}
```

4.5 Argumentos de método y función

En la lista de argumentos, NO DEBE haber un espacio antes de cada coma, y DEBE haber un espacio después de cada coma.

Los argumentos de método y función con valores predeterminados DEBEN ir al final de la lista de argumentos.

```
<?php

namespace Vendor\Package;

class ClassName
{
    public function foo(int $arg1, &$arg2, $arg3 = [])
    {
        // method body
    }
}
```

Las listas de argumentos PUEDEN dividirse en varias líneas, donde cada línea subsiguiente se sangra una vez. Al hacerlo, el primer elemento de la lista DEBE estar en la siguiente línea y DEBE haber solo un argumento por línea.

Cuando la lista de argumentos se divide en varias líneas, el paréntesis de cierre y la llave de apertura DEBEN colocarse juntos en su propia línea con un espacio entre ellos.

```
<?php

namespace Vendor\Package;

class ClassName
{
    public function aVeryLongMethodName(
        ClassTypeHint $arg1,
        &$arg2,
```

```
        array $arg3 = []
    ) {
        // method body
    }

}
```

Cuando tenga una declaración de tipo de retorno presente, DEBE haber un espacio después de los dos puntos seguido de la declaración de tipo. Los dos puntos y la declaración DEBEN estar en la misma línea que el paréntesis de cierre de la lista de argumentos sin espacios entre los dos caracteres.

```php
<?php

declare(strict_types=1);

namespace Vendor\Package;

class ReturnTypeVariations
{
    public function functionName(int $arg1, $arg2): string
    {
        return 'foo';
    }

    public function anotherFunction(
        string $foo,
        string $bar,
        int $baz
    ): string {
        return 'foo';
    }

}
```

En las declaraciones de tipos que aceptan valores NULL, NO DEBE haber un espacio entre el signo de interrogación y el tipo.

```php
<?php

declare(strict_types=1);

namespace Vendor\Package;

class ReturnTypeVariations
{
    public function functionName(?string $arg1, ?int &$arg2):
?string
    {
        return 'foo';
    }

}
```

Cuando se usa el operador de referencia & antes de un argumento, NO DEBE haber un espacio después de él, como en el ejemplo anterior.

NO DEBE haber un espacio entre el operador variable de tres puntos y el nombre del argumento:

```php
public function process(string $algorithm, ...$parts)
{
    // processing

}
```

Al combinar el operador de referencia y el operador de tres puntos variadic, NO DEBE haber ningún espacio entre los dos:

```php
public function process(string $algorithm, &...$parts)
{
    // processing

}
```

4.6 abstract, final y static

Cuando están presentes, las declaraciones abstract y final DEBEN preceder a la declaración de visibilidad.

Cuando esté presente, la declaración static DEBE ir después de la declaración de visibilidad.

```php
<?php

namespace Vendor\Package;

abstract class ClassName
{
    protected static $foo;

    abstract protected function zim();

    final public static function bar()
    {
        // method body
    }

}
```

4.7 Llamadas a métodos y funciones

Al realizar una llamada a un método o función, NO DEBE haber un espacio entre el método o el nombre de la función y el paréntesis de apertura, NO DEBE haber un espacio después del paréntesis de apertura, y NO DEBE haber un espacio antes del paréntesis de cierre. En la lista de argumentos, NO DEBE haber un espacio antes de cada coma, y DEBE haber un espacio después de cada coma.

```php
<?php

bar();
$foo->bar($arg1);

Foo::bar($arg2, $arg3);
```

Las listas de argumentos PUEDEN dividirse en varias líneas, donde cada línea subsiguiente se sangra una vez. Al hacerlo, el primer elemento de la lista DEBE estar en la siguiente línea y

DEBE haber solo un argumento por línea. Un solo argumento dividido en varias líneas (como podría ser el caso de una función o matriz anónima) no constituye una división de la lista de argumentos en sí.

```php
<?php

$foo->bar(
    $longArgument,
    $longerArgument,
    $muchLongerArgument

);
```

```php
<?php

somefunction($foo, $bar, [
  // ...
], $baz);

$app->get('/hello/{name}', function ($name) use ($app) {
    return 'Hello ' . $app->escape($name);

});
```

5. Estructuras de control

Las reglas generales de estilo para las estructuras de control son las siguientes:

- DEBE haber un espacio después de la palabra clave de estructura de control
- NO DEBE haber un espacio después del paréntesis de apertura
- NO DEBE haber un espacio antes del paréntesis de cierre
- DEBE haber un espacio entre el paréntesis de cierre y la llave de apertura
- El cuerpo de la estructura DEBE ser sangrado una vez
- El cuerpo DEBE estar en la siguiente línea después de la abrazadera de apertura.
- La llave de cierre DEBE estar en la siguiente línea después del cuerpo

El cuerpo de cada estructura DEBE estar encerrado por tirantes. Esto estandariza el aspecto de las estructuras y reduce la probabilidad de introducir errores a medida que se agregan nuevas líneas al cuerpo.

5.1 if, elseif, else

Una estructura if se parece a la siguiente. Tenga en cuenta la ubicación de paréntesis, espacios y llaves; else y elseif están en la misma línea que la llave de cierre del cuerpo anterior.

```
<?php

if ($expr1) {
    // if body
} elseif ($expr2) {
    // elseif body
} else {
    // else body;

}
```

La palabra clave elseif DEBE usarse en lugar de else if para que todas las palabras clave de control se vean como palabras individuales.

Las expresiones entre paréntesis PUEDEN dividirse en varias líneas, donde cada línea subsiguiente se sangra al menos una vez. Al hacerlo, la primera condición DEBE estar en la siguiente línea. El paréntesis de cierre y la llave de apertura DEBEN colocarse juntos en su propia línea con un espacio entre ellos. Los operadores booleanos entre condiciones DEBEN estar siempre al principio o al final de la línea, no una combinación de ambos.

```
<?php

if (
    $expr1
    && $expr2
) {
    // if body
} elseif (
    $expr3
    && $expr4
) {
    // elseif body
}
```

5.2 switch, case

Una estructura switch se parece a la siguiente. Tenga en cuenta la ubicación de los paréntesis, los espacios y las llaves. La declaración case DEBE tener una sangría de una vez y la palabra clave break (u otras palabras clave de terminación) DEBE tener sangría al mismo nivel que el casecuerpo. DEBE haber un comentario //, cuando la caída es intencional en un cuerpo case no vacío .

```php
<?php

switch ($expr) {
    case 0:
        echo 'First case, with a break';
        break;
    case 1:
        echo 'Second case, which falls through';
        // no break
    case 2:
    case 3:
    case 4:
        echo 'Third case, return instead of break';
        return;
    default:
        echo 'Default case';
        break;
}
```

Las expresiones entre paréntesis PUEDEN dividirse en varias líneas, donde cada línea subsiguiente se sangra al menos una vez. Al hacerlo, la primera condición DEBE estar en la siguiente línea. El paréntesis de cierre y la riostra de apertura DEBEN colocarse juntos en su propia línea con un espacio entre ellos. Los operadores booleanos entre condiciones DEBEN estar siempre al principio o al final de la línea, no una combinación de ambos.

```php
<?php

switch (
    $expr1
    && $expr2
) {
    // structure body

}
```

5.3 while, do while

Una declaración while se parece a la siguiente. Tenga en cuenta la ubicación de los paréntesis, los espacios y las llaves.

```php
<?php

while ($expr) {
    // structure body
}
```

Las expresiones entre paréntesis PUEDEN dividirse en varias líneas, donde cada línea subsiguiente se sangra al menos una vez. Al hacerlo, la primera condición DEBE estar en la siguiente línea. El paréntesis de cierre y la riostra de apertura DEBEN colocarse juntos en su propia línea con un espacio entre ellos. Los operadores booleanos entre condiciones DEBEN estar siempre al principio o al final de la línea, no una combinación de ambos.

```php
<?php

while (
    $expr1
    && $expr2
) {
    // structure body
}
```

De manera similar, una declaración do while se parece a la siguiente. Tenga en cuenta la ubicación de los paréntesis, los espacios y las llaves.

```php
<?php

do {
    // structure body;
} while ($expr);
```

Las expresiones entre paréntesis PUEDEN dividirse en varias líneas, donde cada línea subsiguiente se sangra al menos una vez. Al hacerlo, la primera condición DEBE estar en la siguiente línea. Los operadores booleanos entre condiciones DEBEN estar siempre al principio o al final de la línea, no una combinación de ambos.

```php
<?php

do {
    // structure body;
} while (
    $expr1
    && $expr2

);
```

5.4 for

Una declaración for se parece a la siguiente. Tenga en cuenta la ubicación de los paréntesis, los espacios y las llaves.

```php
<?php

for ($i = 0; $i < 10; $i++) {
    // for body

}
```

Las expresiones entre paréntesis PUEDEN dividirse en varias líneas, donde cada línea subsiguiente se sangra al menos una vez. Al hacerlo, la primera expresión DEBE estar en la siguiente línea. El paréntesis de cierre y la riostra de apertura DEBEN colocarse juntos en su propia línea con un espacio entre ellos.

```php
<?php

for (
    $i = 0;
    $i < 10;
    $i++
) {
    // for body

}
```

5.5 foreach

Una declaración foreach se parece a la siguiente. Tenga en cuenta la ubicación de los paréntesis, los espacios y las llaves.

```php
<?php

foreach ($iterable as $key => $value) {
    // foreach body

}
```

5.6 try, catch, finally

Un bloque try-catch-finally se parece al siguiente. Tenga en cuenta la ubicación de los paréntesis, los espacios y las llaves.

```php
<?php

try {
    // try body
} catch (FirstThrowableType $e) {
    // catch body
} catch (OtherThrowableType | AnotherThrowableType $e) {
    // catch body
} finally {
    // finally body

}
```

6. Operadores

Las reglas de estilo para los operadores se agrupan por aridad (el número de operandos que toman).

Cuando se permite el espacio alrededor de un operador, se PUEDEN utilizar varios espacios para facilitar la lectura.

Todos los operadores que no se describen aquí quedan sin definir.

6.1. Operadores unarios

Los operadores de incremento / decremento NO DEBEN tener ningún espacio entre el operador y el operando.

```
$i++;

++$j;
```

Los operadores de conversión de tipos NO DEBEN tener ningún espacio entre paréntesis:

```
$intValue = (int) $input;
```

6.2. Operadores binarios

Todos los operadores binarios de aritmética , comparación , asignación , bit a bit , lógicos , de cadena y de tipo DEBEN ir precedidos y seguidos de al menos un espacio:

```
if ($a === $b) {
    $foo = $bar ?? $a ?? $b;
} elseif ($a > $b) {
    $foo = $a + $b * $c;

}
```

6.3. Operadores ternarios

El operador condicional, también conocido simplemente como operador ternario, DEBE estar precedido y seguido por al menos un espacio alrededor de los caracteres ? y : :

```
$variable = $foo ? 'foo' : 'bar';
```

Cuando se omite el operando del medio del operador condicional, el operador DEBE seguir las mismas reglas de estilo que otros operadores de comparación binaria :

```
$variable = $foo ?: 'bar';
```

7. Cierres

Los cierres DEBEN declararse con un espacio después de la palabra clave function y un espacio antes y después de la palabra clave use.

La llave de apertura DEBE ir en la misma línea, y la llave de cierre DEBE ir en la siguiente línea que sigue al cuerpo.

NO DEBE haber un espacio después del paréntesis de apertura de la lista de argumentos o de la lista de variables, y NO DEBE haber un espacio antes del paréntesis de cierre de la lista de argumentos o la lista de variables.

En la lista de argumentos y la lista de variables, NO DEBE haber un espacio antes de cada coma, y DEBE haber un espacio después de cada coma.

Los argumentos de cierre con valores predeterminados DEBEN ir al final de la lista de argumentos.

Si un tipo de retorno está presente, DEBE seguir las mismas reglas que con las funciones y métodos normales; si la palabra clave use está presente, los dos puntos DEBEN seguir el useparéntesis de cierre de la lista sin espacios entre los dos caracteres.

Una declaración de cierre se parece a lo siguiente. Tenga en cuenta la ubicación de los paréntesis, las comas, los espacios y las llaves:

```
<?php

$closureWithArgs = function ($arg1, $arg2) {
    // body
};

$closureWithArgsAndVars = function ($arg1, $arg2) use ($var1, $var2) {
    // body
};

$closureWithArgsVarsAndReturn = function ($arg1, $arg2) use ($var1, $var2): bool {
    // body

};
```

Las listas de argumentos y las listas de variables PUEDEN dividirse en varias líneas, donde cada línea subsiguiente se sangra una vez. Al hacerlo, el primer elemento de la lista DEBE estar en la siguiente línea y DEBE haber solo un argumento o variable por línea.

Cuando la lista final (ya sea de argumentos o variables) se divide en varias líneas, el paréntesis de cierre y la llave de apertura DEBEN colocarse juntos en su propia línea con un espacio entre ellos.

Los siguientes son ejemplos de cierres con y sin listas de argumentos y listas de variables divididas en varias líneas.

```php
<?php

$longArgs_noVars = function (
    $longArgument,
    $longerArgument,
    $muchLongerArgument
) {
    // body
};

$noArgs_longVars = function () use (
    $longVar1,
    $longerVar2,
    $muchLongerVar3
) {
    // body
};

$longArgs_longVars = function (
    $longArgument,
    $longerArgument,
    $muchLongerArgument
) use (
    $longVar1,
    $longerVar2,
    $muchLongerVar3
) {
    // body
};

$longArgs_shortVars = function (
    $longArgument,
    $longerArgument,
    $muchLongerArgument
) use ($var1) {
    // body
};
```

```
$shortArgs_longVars = function ($arg) use (
    $longVar1,
    $longerVar2,
    $muchLongerVar3
) {
   // body

};
```

Tenga en cuenta que las reglas de formato también se aplican cuando el cierre se usa directamente en una función o llamada a un método como argumento.

```
<?php

$foo->bar(
    $arg1,
    function ($arg2) use ($var1) {
        // body
    },
    $arg3

);
```

8. Clases anónimas

Las clases anónimas DEBEN seguir las mismas pautas y principios que los cierres en la sección anterior.

```
<?php


$instance = new class {};
```

La llave de apertura PUEDE estar en la misma línea que la palabra clave class siempre que la lista de interfaces implements no se ajuste. Si la lista de interfaces se ajusta, la llave DEBE colocarse en la línea que sigue inmediatamente a la última interfaz.

```php
<?php

// Brace on the same line
$instance = new class extends \Foo implements \HandleableInterface {
    // Class content
};

// Brace on the next line
$instance = new class extends \Foo implements
    \ArrayAccess,
    \Countable,
    \Serializable
{
    // Class content

};
```

Convenciones del código PHP

Referencias:

PSR-1: Basic Coding Standard - https://www.php-fig.org/psr/psr-1/

PSR-12: Extended Coding Style - https://www.php-fig.org/psr/psr-12/