Convenciones de desarrollo

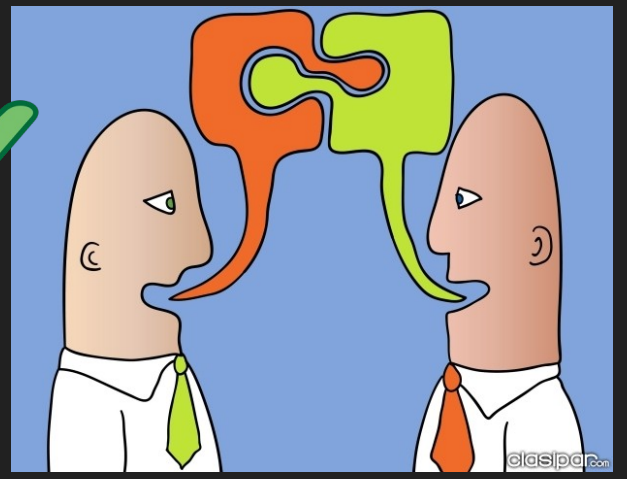
Mauricio Beltrán – INDAP 2014

¿Qué es una convención?

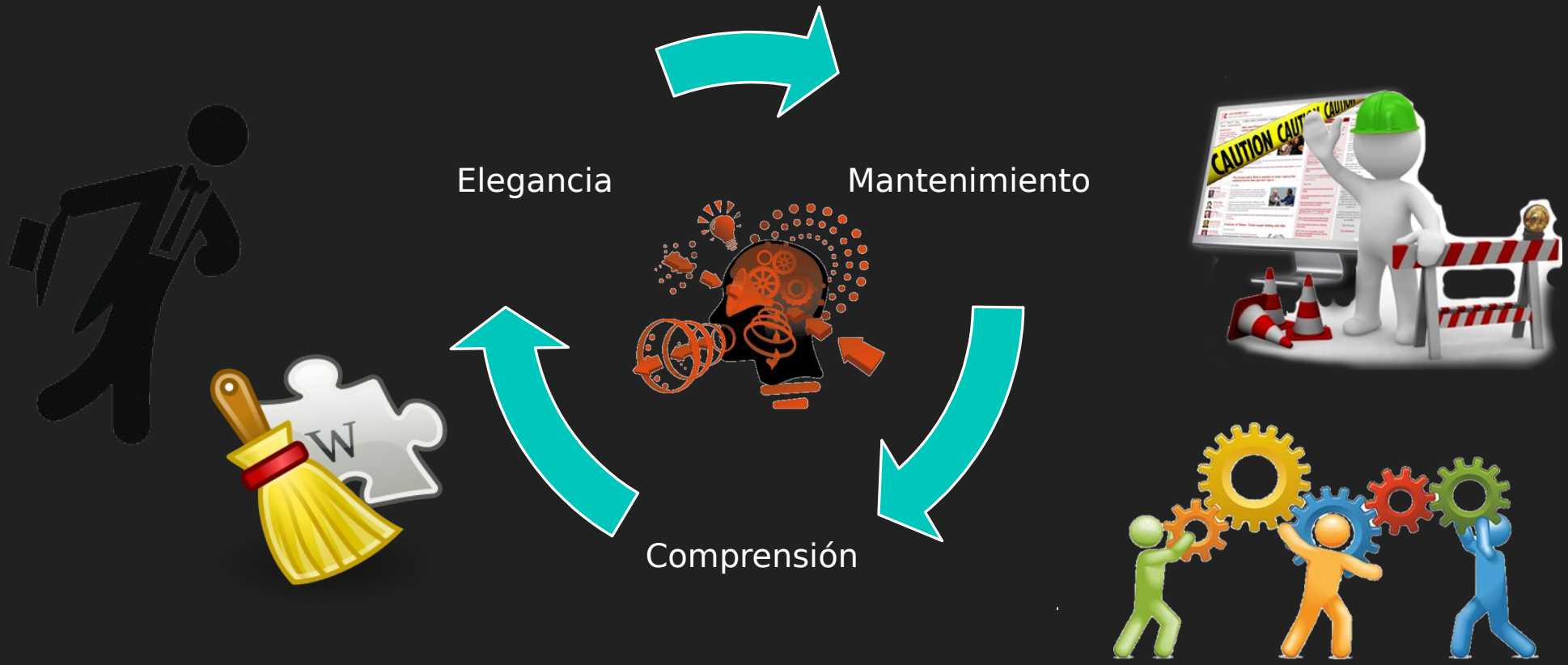
“Son normas o prácticas aceptadas socialmente por un acuerdo general o por la costumbre”

¿Para que sirven?

Entendimiento



¿Por que las vamos a implementar?



PSR-1

Los archivos DEBEN utilizar solamente las etiquetas

<?php ?>

<?= ?>

```php
<?php
// Esta es una declaración correcta
?>

<?
// Esta no
```

```php
<?= // Esta es para impresion de contenido ?>
```

Los archivos **DEBEN** emplear solamente la codificación UTF-8 sin BOM para el código PHP.

Los archivos DEBERÍAN declarar **cualquier** estructura (clases, funciones, constantes, etc...) **o** realizar partes de la lógica de negocio (por ejemplo, generar una salida, cambio de configuración, etc...) pero NO DEBERÍAN hacer las dos cosas

```php
<?php

class Foo
{
    const FOO_BAR = 'bar';

    public function __construct()
    {
        // content
    }
}

?>
```

```php
<?php

// efecto secundario: cambiar configuracion inicial
ini_set('error_reporting', E_ALL);

// efecto secundario: cargar ficheros
include "archivo.php";

// efecto secundario: generar salida
echo "<html>\n";

// declaración
function foo()
{
    // cuerpo de la función
}

?>
```

Los nombres de las clases DEBEN declararse en notación `StudlyCaps`

```php
<?php

// esto esta bien
class FooBar
{
    //
}

?>

<?php

// esto No esta bien
class Foo_Bar
{
    //
}

?>

<?php

// esto No esta bien
class fooBar
{
    //
}

?>

<?php
```

Las constantes de las clases DEBEN declararse en mayúsculas con guiones bajos como separadores `CONSTANTE_DE_CLASE`

```php
<?php

// esto esta bien
const FOO_BAR = '';

class FooBar
{
    //
}

?>

<?php

// esto esta bien
const FOOBAR = '';

class FooBar
{
    //
}

?>

<?php

// esto NO esta bien
const Foo_Bar = '';

class FooBar
{
    //
}

?>
```

Los nombres de los métodos DEBEN declararse en notación `camelCase`.

```php
<?php

class FooBar
{
    // esto NO esta bien
    public function fooBar()
    {

    }
}

?>

<?php

class FooBar
{
    // esto NO esta bien
    public function FooBar()
    {

    }
}

?>

<?php

class FooBar
{
    // esto NO esta bien
    public function foo_bar()
    {

    }
}

?>
```

PSR-2

Codificación estándar básica

- Todos los ficheros PHP DEBEN usar el final de línea Unix LF.
- Todos los ficheros PHP DEBEN terminar con una línea en blanco.
- La etiqueta de cierre DEBE ser omitida en los ficheros que sólo contengan código PHP.

```php
<?php
namespace FOO;

use FOO;

class FooBar
{
    const FOO_BAR='';
    public function fooBar()
    {
        // content
    }
}
```

Los espacios de nombres y las clases DEBEN seguir el estándar PSR-1

El código DEBE usar 4 espacios como indentación, no tabuladores

```php
<?php

class FooBar
{
    const FOO_BAR='';
    public function fooBar()
    {
        // content
    }
}

?>
```

NO DEBE haber un límite estricto en la longitud de la línea; el límite DEBE estar en 120 caracteres; las líneas DEBERÍAN tener 80 caracteres o menos.

DEBE haber una línea en blanco después de la declaración del `namespace`, y DEBE haber una línea en blanco después del bloque de declaraciones `use`

```php
<?php
namespace FOO;

use FOO;

class FooBar
{
    const FOO_BAR='';
    public function fooBar()
    {
        // content
    }
}

?>
```

Las llaves de apertura de las clases DEBEN ir en la línea siguiente, y las llaves de cierre DEBEN ir en la línea siguiente al cuerpo de la clase

Las llaves de apertura de los métodos DEBEN ir en la línea siguiente, y las llaves de cierre DEBEN de ir en la línea siguiente al cuerpo del método.

```php
<?php
namespace FOO;

use FOO;

class FooBar
{
    const FOO_BAR='';
    public function fooBar()
    {
        // content
    }
}

?>
```

La visibilidad DEBE estar declarada en todas las propiedades y métodos; `abstract` y `final` DEBEN estar declaradas antes de la visibilidad; `static` DEBE estar declarada después de la visibilidad.

```php
abstract class BaseModuloNodo extends BaseModulo
{

    /**
     * constante de tipo
     */
    const TIPO = 'nodo';

    /**
     * Constructor de la clase
     */
    function __construct()
    {
        parent::__construct();
    }

    /**
     * Metodo que retorna el tipo
     * @final
     * @return string constante tipo
     */
    final public function getTipo()
    {
        return $this::TIPO;
    }
```

Las llaves de apertura de las estructuras de control DEBEN estar en la misma línea, y las de cierre DEBEN ir en la línea siguiente al cuerpo

```
 * @param  int $timeout tiempo maximo de espera de una conexon, en segund
 * @return int
 */
private function chequearEstadoDeConexion($host, $puerto=389, $timeout=1)
{
    $sock = fsockopen($host, $puerto, $errno, $errstr, $timeout);
    if (!$sock){
        return 0;
    } else {
        fclose($sock);
        return 1;
    }
}
```

Los paréntesis de apertura en las estructuras de control NO DEBEN tener un espacio después de ellos, y los paréntesis de cierre NO DEBEN tener un espacio antes de ellos

Indentación

El código DEBE usar una indentación de 4 espacios, y NO DEBE usar tabuladores para la indentación.

Las Palabras clave de PHP DEBEN estar en minúsculas, asi como true, false y null.

Clases, propiedades y métodos

- Las palabras clave `extends` e `implements` DEBEN declararse en la misma línea del nombre de la clase.
- La llave de apertura de la clase DEBE ir en la línea siguiente; la llave de cierre DEBE ir en la línea siguiente al cuerpo de la clase.
- La lista de `implements` PUEDE ser dividida en múltiples líneas, donde las líneas subsiguientes serán indentadas una vez. Al hacerlo, el primer elemento de la lista DEBE estar en la línea siguiente, y DEBE haber una sola interfaz por línea.

```php
<?php
namespace Proveedor\Paquete;

use FooClase;
use BarClase as Bar;
use OtroProveedor\OtroPaquete\BazClase;

class NombreDeClase extends ClasePadre implements
    \ArrayAccess,
    \Countable,
    \Serializable
{
    // constantes, propiedades, métodos
}
```

Propiedades

- La visibilidad DEBE ser declarada en todas las propiedades.
- La palabra clave `var` NO DEBE ser usada para declarar una propiedad.
- NO DEBE declararse más de una propiedad por sentencia.
- Los nombres de las propiedades NO DEBERÍAN usar un guion bajo como prefijo para indicar si son privadas o protegidas.

```php
<?php
namespace Proveedor\Paquete;

class NombreDeClase
{
    public $foo = null;

    private function bar()
    {
        // cuerpo funcion
    }
}
```

Argumentos de los métodos

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 con valores por defecto del método DEBEN ir al final de la lista de argumentos.

```php
<?php
namespace Proveedor\Paquete;

class NombreDeClase
{
    public $foo = null;

    private function bar($arg1, $arg2, $arg0="default")
    {
        // cuerpo funcion
    }
}
```

La lista de argumentos PUEDE dividirse en múltiples líneas, donde cada línea será indentada una vez. Cuando se dividan de esta forma, el primer argumento DEBE estar en la línea siguiente, y DEBE haber sólo un argumento por línea.

Cuando la lista de argumentos se divide en varias líneas, el paréntesis de cierre y la llave de apertura DEBEN estar juntos en su propia línea separados por un espacio.

```php
<?php
namespace Proveedor\Paquete;

class NombreDeClase
{
    public function metodoConNombreLargo(
        ClassTypeHint $arg1,
        &$arg2,
        array $arg3 = []
    ) {
        // cuerpo del método
    }
}
```

Llamadas a métodos y funciones

- Cuando se realize una llamada a un método o a una función, NO DEBE haber un espacio entre el nombre del método o 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 espacio antes de cada coma y DEBE haber un espacio después de cada coma.

- La lista de argumentos PUEDE dividirse en múltiples líneas, donde cada una se indenta una vez. Cuando esto suceda, el primer argumento DEBE estar en la línea siguiente, y DEBE haber sólo un argumento por línea.

```php
<?php
bar();
$foo->bar($arg1);
Foo::bar($arg2, $arg3);
```