





Imaginense programando en PHP, en un proyecto donde varios desarrolladores colaboran y constantemente se crean clases, funciones, etc. Es muy probable que en algún momento alguien cree alguna clase que ya exista, ¿Cómo hago para que no se pisen?



# Namespaces

- ◆ Solucionan el conflicto de nombres entre el código que se crea y las clases/funciones/constantas internas de PHP o la clases/funciones/constantas de terceros.





# Definición

La capacidad de apodar (o abreviar) Nombres\_Extra\_Largos diseñada para aliviar el primer problema, mejorando la legibilidad del código fuente.

## Ejemplo CORRECTO

```
<?php
    namespace MiProyecto;
    class Conexión { /*...*/ }
?>
```

## Ejemplo INCORRECTO

```
<html>
<?php
    namespace MiProyecto;
?>
```





## 2. Subespacios

```
<?php
```

```
    namespace MiProyecto\Sub\Nivel;
```

```
    const CONECTAR_OK = 1;
```

```
    class Conexión { /* ... */ }
```

```
    function conectar() { /* ... */ }
```

```
?>
```

Creamos:

- La **constante**: MiProyecto\Sub\Nivel\CONECTAR\_OK
- La **clase**: MiProyecto\Sub\Nivel\Conexión
- La **función**: MiProyecto\Sub\Nivel\conectar.



# Uso de namespaces


Dependiendo de nuestra posición hay 3 formas diferentes de acceder al nombre de una clase o función definida en un namespace.

Como ejemplo, imaginemos que en el espacio de nombres **Proyecto\Blog\Admin** está definida la clase **Usuario**.



# ejemplo.php

```
<?php
    namespace Proyecto\Blog\Admin;
    class Usuario {
        private $nombre;
        function getNombre(){
            return $this->nombre;
        }
    }
?>
```




Si queremos utilizar la clase **Usuario** y estamos en un namespace que **no está** dentro de la jerarquía que contiene dicha clase, necesitamos utilizar la **ruta completa** para poder acceder a ella.

A esta forma se le llama:

Nombre  
**completamente**  
calificado  
(Fully Qualified Name)






# Nombre completamente calificado

```
<?php
    namespace Prueba;
    include 'ejemplo.php';
    $usuario = new \Proyecto\Blog\Admin\Usuario();
    var_dump($usuario);

?>
```

A decorative graphic on the left side of the slide consists of a cluster of hexagons in various shades of blue and cyan. Some hexagons contain white icons: a lightbulb, a thumbs-up, a smartphone, a magnifying glass, and a gear. A larger, solid cyan hexagon is positioned in the center of this cluster.

Si estamos dentro de **algún** nivel en la jerarquía que contiene a la clase solicitada, no necesitamos utilizar la ruta completa si no que utilizaremos una **ruta relativa** al namespace en el que nos encontremos.

A esta forma de acceso se le llama:

Nombre  
Calificado  
(Qualified Name)




# Nombre calificado

```
<?php
```

```
    namespace Proyecto\Blog;  
    include 'ejemplo.php';  
    $usuario = new Admin\Usuario();  
    var_dump($usuario);
```


```
?>
```



Si estamos en el namespace actual, utilizamos la clase como siempre la habíamos utilizado hasta ahora, sin hacer referencia a ningún espacio de nombres.

A esta forma de acceso se le llama:

Nombre  
**no**  
Calificado  
(Unqualified Name)




# Nombre no calificado

```
<?php
```

```
    namespace Proyecto\Blog\Admin;  
    include 'ejemplo.php';  
    $usuario = new Usuario();  
    var_dump($usuario);
```

```
?>
```

A decorative graphic on the left side of the slide. It features a large, solid blue hexagon in the center. Surrounding it are several smaller hexagons of varying shades of blue and cyan. Some of these hexagons contain white icons: a lightbulb, a thumbs-up, a smartphone, a magnifying glass, and a gear. There is also a network-like icon with a central node and radiating lines.

Si en nuestro ejemplo.php tenemos que instanciar numerosas veces a la clase Usuario, o utilizar varias clases de ese namespace tendremos que escribir muchas veces toda la jerarquía:

```
new \Proyecto\Blog\Admin\Usuario() ???
```



# Importar namespaces

```
<?php
    namespace Prueba;
    require 'ejemplo.php';
    use Proyecto\Blog\Admin;
    $usuario = new Admin\Usuario();
    var_dump($usuario);

?>
```



La utilización del comando **use**

**no cambia**

el espacio en el que trabajamos.

Nos permitirá utilizar implementaciones de otros namespaces haciendo uso del nombre calificado (ruta relativa).





# Alias

```
<?php
    namespace Prueba;
    require 'ejemplo.php';
    use Proyecto\Blog\Admin\Usuario as clase;
    $usuario = new clase();
    var_dump($usuario);
?>
```



# \_\_NAMESPACE\_\_

La constante `__NAMESPACE__` contiene el nombre del espacio de nombres actual. Si el espacio de nombres actual es el global, la constante devolverá una cadena vacía.

```
<?php
    namespace Prueba;
    echo __NAMESPACE__ . "<br/>"; //Prueba
    namespace Pepe;
    echo __NAMESPACE__; // Pepe
?>
```



# Ejemplo Global

```
<?php
```

```
    namespace Prueba;
```

```
    function strlen($cadena) {
```

```
        echo $cadena . "<br/>";
```

```
    }
```

```
    echo strlen('hola'); // strlen del namespace actual
```

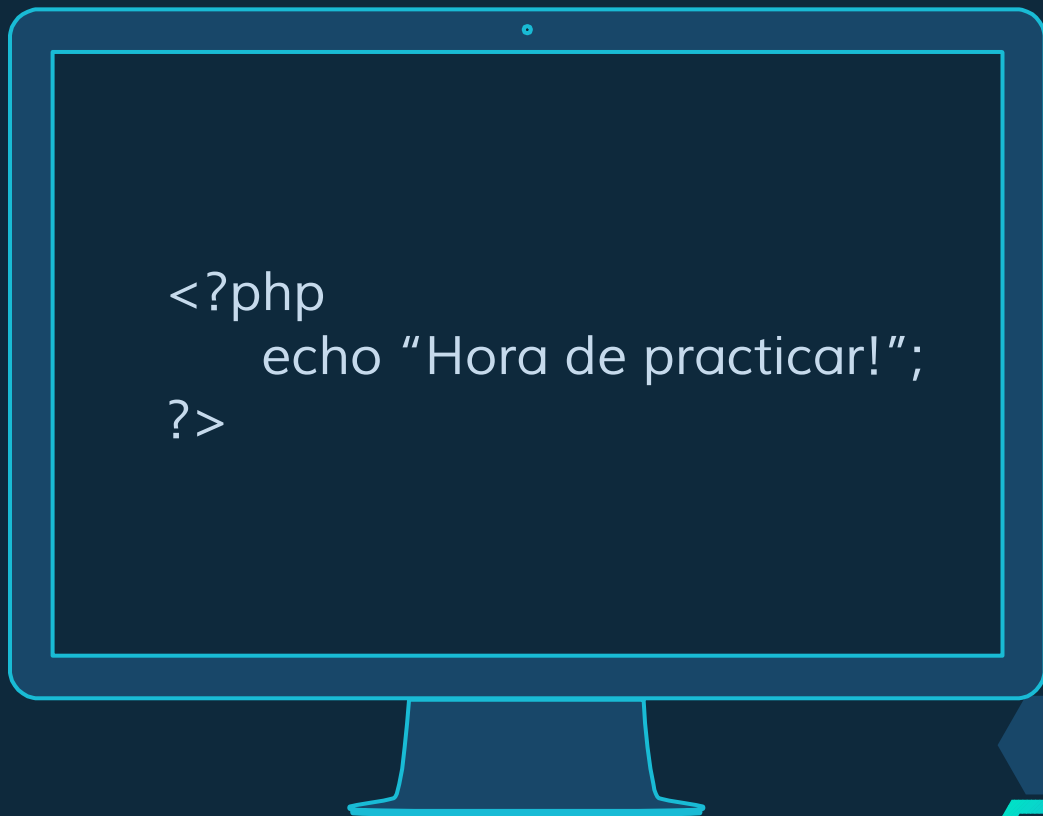
```
    echo \strlen('hola'); // función global strlen
```

```
?>
```



# ¡A practicar!

Ejercicios del 1 y 2






# Traits

PHP es un lenguaje con herencia simple. Esto quiere decir que una clase sólo puede heredar de otra clase.

Muchas veces queremos heredar distintos métodos de muchas clases, lo cual haría muy engorroso el código que estamos escribiendo.





Imagine:


```
<?php
    class AbstractValidate extends AbstractCache {};
    class AbstractSocial extends AbstractValidate {};
    class Post extends AbstractSocial {};
?>
```



# ¿Qué son los traits?

Un trait es un grupo de métodos que queremos incluir dentro de una clase.

```
<?php
    trait Sharable {
        public function share($item)
        {
            return 'share this item';
        }
    }
?>
```




```
<?php
    trait Sharable {
        public function share($item)
        {
            return 'share this item';
        }
    }

    class Post {
        use Sharable;
    }

    class Comment {
        use Sharable;
    }

?>
```






# Ahora podríamos

```
<?php
```

```
    $post = new Post;  
    echo $post->share(""); // 'share this item'
```

```
    $comment = new Comment;  
    echo $comment->share(""); // 'share this item'
```

```
?>
```



```
<?php
    interface Sociable {
        public function like();
        public function share();
    }

    trait Sharable {
        public function share($item)
        {
            // share this item
        }
    }

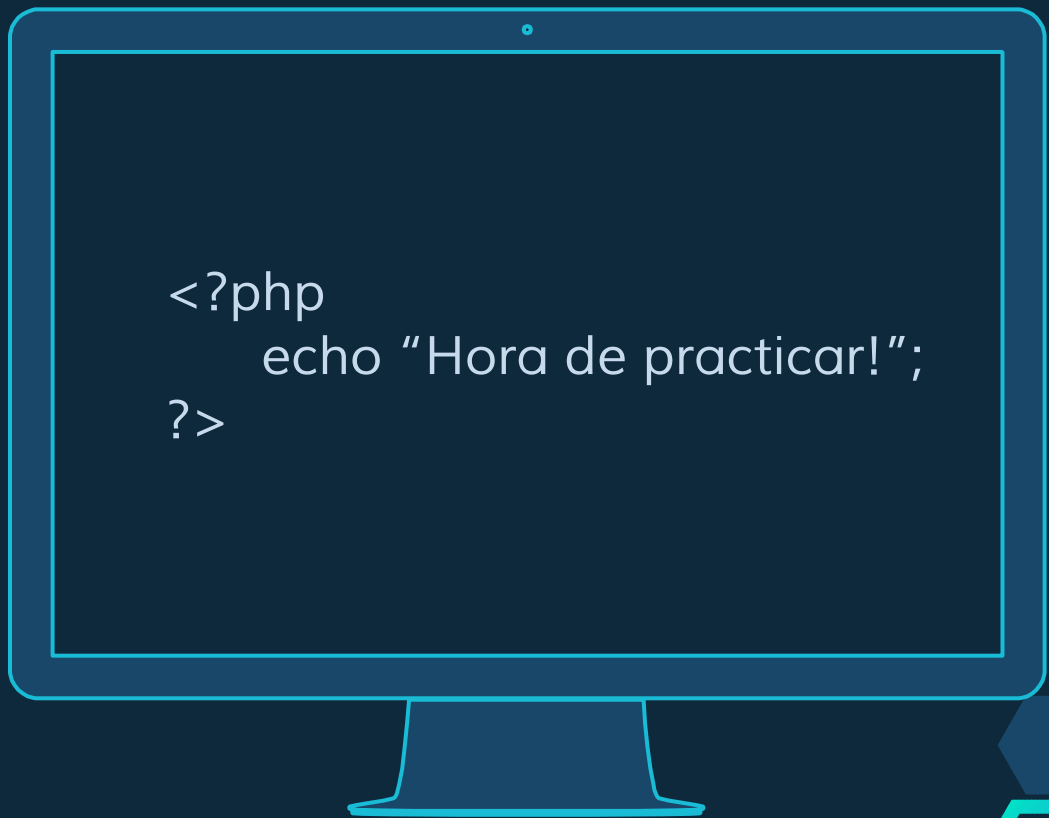
    class Post implements Sociable {
        use Sharable;
        public function like()
        {
            // like this item
        }
    }
?>
```



Un trait es básicamente "copiar y pegar"  
código durante la ejecución.



¡A practicar!



```
<?php  
    echo "Hora de practicar!";  
?>
```





# Gracias!

## ¿Preguntas?

