Índice

[**¿Debería seguir usando versiones antiguas de PHP?** 2](#_Toc108979713)

[**Si es un proyecto legacy** 3](#_Toc108979714)

[**Actualización en sistemas distribuidos** 3](#_Toc108979715)

[**Si es un proyecto nuevo que saldrá a producción** 3](#_Toc108979716)

[**Si es un proyecto nuevo que usarás para experimentar** 3](#_Toc108979717)

[**Documentación: la biblia de PHP** 3](#_Toc108979718)

[**¿Qué es PHP Doc?** 4](#_Toc108979719)

[**¿Para qué sirven los entornos virtuales?** 5](#_Toc108979720)

[**¿Qué es un entorno virtual?** 5](#_Toc108979721)

[**¿Qué son los Virtual Hosts?** 6](#_Toc108979722)

[**¿Cómo configurar un virtual host?** 6](#_Toc108979723)

[**Constantes en PHP** 7](#_Toc108979724)

[**¿Qué es una constante?** 7](#_Toc108979725)

[**Diferencias entre const y define** 7](#_Toc108979726)

[**Constantes mágicas** 8](#_Toc108979727)

[**¿Qué son las variables variable?** 8](#_Toc108979728)

[**Scope de PHP** 9](#_Toc108979729)

[**Ámbito local** 9](#_Toc108979730)

[**La palabra reservada global** 9](#_Toc108979731)

[**Formas de acceder a las variables globales** 10](#_Toc108979732)

[**Variables superglobales** 10](#_Toc108979733)

[**Redirecciones** 11](#_Toc108979734)

[**header()** 11](#_Toc108979735)

[**Consideraciones** 12](#_Toc108979736)

[**Match** 12](#_Toc108979737)

[**Funciones variables** 13](#_Toc108979738)

[**Parámetros por referencia** 14](#_Toc108979739)

[**¿Qué es el paso de parámetros por referencia?** 15](#_Toc108979740)

[**¿Qué es el paso de parámetros por referencia?** 15](#_Toc108979741)

[**Argumentos a profundidad** 15](#_Toc108979742)

[**Arrays como parametros opcional** 15](#_Toc108979743)

[**Trailing commas** 15](#_Toc108979744)

[**Recibir clases como parámetros** 16](#_Toc108979745)

[**Named arguments** 17](#_Toc108979746)

[**Funciones anónimas** 17](#_Toc108979747)

[**Arrow functions** 19](#_Toc108979748)

[**Diferencias con las funciones anónimas** 19](#_Toc108979749)

[**Declaraciones de tipo escalar** 19](#_Toc108979750)

[**Declaraciones de tipo de devolución** 20](#_Toc108979751)

# **¿Debería seguir usando versiones antiguas de PHP?**

Muchas veces puede que estés trabajando sobre un proyecto “legacy” o estés iniciando un

proyecto nuevo.

**Legacy** quiere decir proyecto con código antiguo.

## **Si es un proyecto legacy**

Cuando estás trabajando en proyectos que ya llevan tiempo de uso, muchas veces no es recomendable hacer el update. ¿Qué tan costos sería hacer ese update? ¿Vale la pena?

## **Actualización en sistemas distribuidos**

Muchas veces actualizar un sistema distribuido es más fácil, ya que simplemente toca actualizar pequeñas partes mientras lo demás sigue funcionando.

## **Si es un proyecto nuevo que saldrá a producción**

Para el caso de proyectos nuevos, ¡siempre es mejor utilizar la última versión estable! Muchas veces las podemos encontrar como **Stable version** o incluso como **LTS**.

Esto nos asegura que nuestro sistema funcionará sin fallas por parte del lenguaje (y si las hay, serán corregidas).

## **Si es un proyecto nuevo que usarás para experimentar**

Acá puedes usar cualquier versión, incluso versiones beta. Esto te ayudará a explorar el

futuro de cualquier lenguaje y descubrir cosas nuevas.

# **Documentación: la biblia de PHP**

Leer la [documentación](https://www.php.net/) de php diferencia entre un programador **senior** de un **junior**.

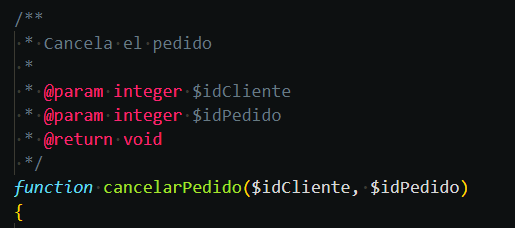
Es muy importante ya que es la que te dicta cómo funciona el lenguaje y que novedades puede tener.

También encontraras las **built-in-functions**, son funciones predefinidas en el lenguaje tales como: **empty**() **is\_array**(), **array\_diff\_assoc**(), etc.

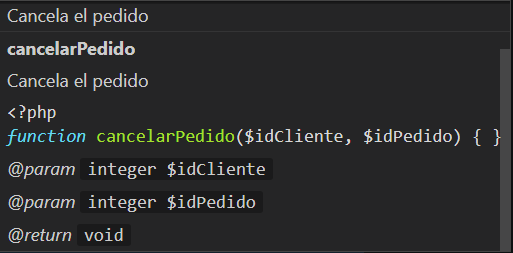
# **¿Qué es PHP Doc?**

Es un estándar informal que tenemos para documentar código en PHP, de esta forma los editores de código pueden mostrar sugerencias cuando implementemos el código que estamos comentando.

Es una adaptación de **Javadoc** y, además de ayudar a los editores de código, también puede ser útil para algunos generadores de documentación como **phpDocumentor**.



Una vez puesto los comentarios relacionados a los requerimientos de la funcion, al hacer un **hover** sobre la funcion nos mostrará la leyenda de esa misma documentación donde sea que usemos esa función.



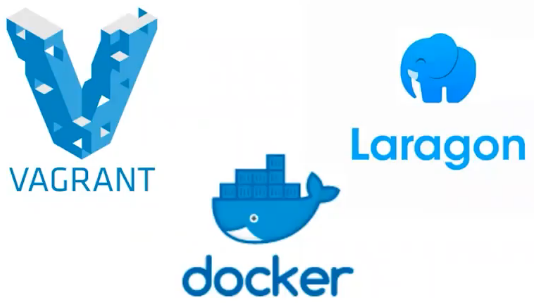
Es una buena práctica documentar tus funciones y clases durante la realización del código.

# **¿Para qué sirven los entornos virtuales?**

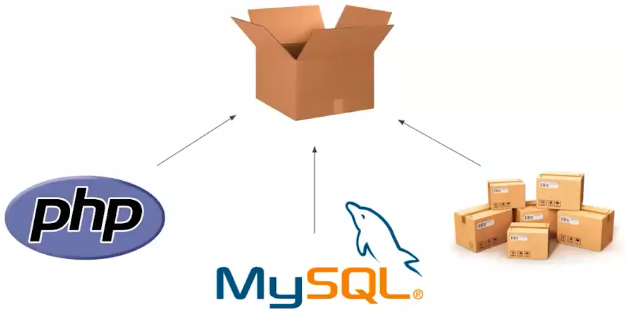
Muchas veces necesitamos trabajar con distintos proyectos que usan diferentes versiones de PHP. Aunque en algunos sistemas operativos es fácil cambiar entre versiones, muchas veces es recomendado utilizar entornos virtuales.

## **¿Qué es un entorno virtual?**

Un entorno virtual nos permite encapsular todo lo necesario para que un proyecto pueda funcionar con las versiones que este mismo requiere. Esto a su vez permite que podamos tener varios proyectos con distintas versiones.

****Entornos Virtuales Populares:

* **Docker**
* **Laragon**
* **Vagrant**



# **¿Qué son los Virtual Hosts?**

Es una forma de que varias páginas web puedan funcionar al mismo tiempo en un mismo servidor. Los podemos definir por direcciones IP o por nombres de dominio.

## **¿Cómo configurar un virtual host?**

Para configurar un virtual host tenemos que hacer dos cosas:

* Decirle a Apache que exponga X página web en tal dominio.
* Decirle al navegador que cuando escribamos X dominio redirija a la web que Apache está exponiendo.

# **Constantes en PHP**

## **¿Qué es una constante?**

Es algo que **nunca cambia**, es decir, una vez que se le asigne un valor ya no es posible modificarlo ni sustituirlo.

## **Diferencias entre const y define**

**const**

* Funciona dentro y fuera de la definición de una clase.
* Define las constantes en tiempo de compilación.
* No podemos usarlos dentro de bloques de código.
* Solo acepta tipos de dato escalares.
* Siempre es case sensitive.

**define**

* Solo funciona fuera de la definición de una clase.
* Define las constantes en tiempo de ejecución.
* Si podemos usarlos dentro de bloques de código.
* Acepta cualquier expresión.
* Puedes definir case insensitive pasando **true** en el tercer argumento. (Esto solo para versiones anteriores a PHP 8, ya que a partir de la version 8 el valor **true** no es válido.)

## **Constantes mágicas**

Son constantes cuyos **valores sí cambian**. Realmente no son constantes, pero se les llama así porque son palabras que hacen referencia a algo que depende del entorno en el que se está ejecutando el script.

Ejemplos de constantes mágicas:

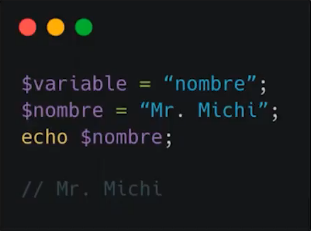
* **\_\_LINE\_\_**: Nos indica en que línea está actualmente el script.
* **\_\_FILE\_\_**: Nos indica cual es el archivo que se está ejecutando actualmente.
* **\_\_DIR\_\_**: Nos brinda la dirección del archivo del cual es llamada.
* **ClassName::class** Nos brinda el nombre completo de una clase.

# **¿Qué son las variables variable?**

Simplemente, PHP interpreta la variable que está en donde debería estar el nombre para saber cual es la variable que realmente debe modificar.



Internamente lo que hace PHP es lo siguiente.



Por está razón, tenemos que nuestra variable **$$variable** se llama **$nombre** y su valor es Mr. Michi

# **Scope de PHP**

El **scope** (o ámbito de las variables) **es el contexto en donde una variable es definida**.

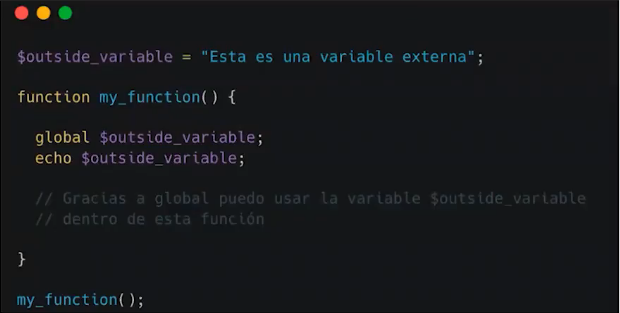
Usualmente, la mayor parte del tiempo una variable tiene un ámbito simple, es decir, está disponible en todo el archivo e incluso en otros archivos donde se haga un **require**.

## **Ámbito local**

Cuando empezamos a usar funciones el ámbito de cualquier variable definida dentro de la funcion pasa a ser un ámbito local. En otras palabras, esa variable **solo existe dentro de la funcion**.

## **La palabra reservada global**

Cuando queremos que una variable externa exista dentro de una funcion sin necesidad de pasarla como parámetro podemos usar la palabra reservada **global**.



## **Formas de acceder a las variables globales**

* Con la palabra reservada **global**.
* Con la variable superglobal **$GLOBALS**.

# **Variables superglobales**

Son variables predefinidas por PHP que están disponibles en cualquier parte del código, incluso sin la necesidad de usar la palabra reservada global.

Algunos ejemplos

* **$GLOBALS**: Nos brinda información de todas las variables globales que existen.
* **$\_SERVER**: Nos brinda información del servidor.
* **$\_GET**: Nos brinda información de todas las variables o parámetros que son pasadas por URL.
* **$\_POST**: Nos brinda información de las variables que son enviadas a través del método POST de HTTP.
* **$\_FILES**: Nos brinda información de variables de subida de ficheros HTTP con el método POST.
* **$\_COOKIE**: Nos brinda información de todas las cookies.
* **$\_SESSION**: Nos brinda información de todas las variables de sesiones.
* **$\_REQUEST**: Por defecto contiene el contenido o valores de las variables superglobales $\_GET, $\_POST y $\_COOKIE.
* **$\_ENV**: Nos brinda información del entorno en el que se está ejecutando PHP.

# **Redirecciones**

## **header()**

La función **header** nos permite definir encabezados de una petición. Esto se puede hacer desde cliente a servidor, servidor a cliente y servidor a servidor.

En el modelo cliente/servidor es común querer enviar información extra (autenticación, control de cookies, control de caché, etc.). Esto lo podemos hacer con la funcion header.

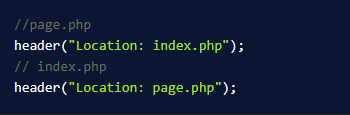
Entre las cosas que se pueden mandar son:

* Autenticación.
* Control de cookies.
* Control de cache.
* Entre otros.

Uno de los headers que podemos mandar son **redirects**, es decir, el servidor puede responder al navegador “Oye, redirige al usuario a esta página.”



También podemos redirigir a archivos específicos, pero hay que tener cuidado de no generar un bucle, es decir, no hacer lo que está en la siguiente imagen.



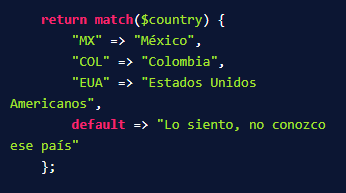
Si redirigimos desde **index.php** a **page.php** y desde **page.php** a **index.php** y esto puede hacer que se rompa la página.

## **Consideraciones**

* Los encabezados siempre deben ser enviados **antes** de enviar el cuerpo de la petición, es decir, **antes de mandar el contenido**. Eso quiere decir que, si imprimes HTML o cualquier otra cosa que genere cuerpo, **al intentar redirigir con header obtendrás un error**.
* Recuerda que los encabezados deben ser los primeros en enviarse antes de enviar una respuesta o body.
* Recuerda utilizar la funcion **exit**() o die al terminar tus headers para que los crawler se detengan de analizar.

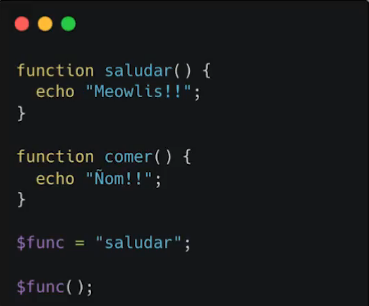
# **Match**

La expresión match ramifica la evaluación basada en una comprobación de identidad de un valor. De forma similar a una sentencia **switch**.



# **Funciones variables**

En pocas palabras, son lo mismo que las **variables variables**, pero aplicado a funciones. La ventaja es que podemos empezar a mandar a llamar funciones a partir de cadenas de texto dinámicas.



# **Parámetros por referencia**

Cuando declaramos una variable en PHP, lo que sucede es que se ocupa un lugar en la memoria de tú computadora para guardar el valor de dicha variable. ¿Cómo sabe PHP qué lugar en la memoria se ocupó? Bueno, PHP lo sabe porque guarda una **referencia** a ese lugar.

Supongamos que tenemos los siguientes nombres direcciones de memorias.

* **Carlos.**
* **Mariana.**
* **José.**
* **Raúl.**
* **Etc..**

Declaramos una variable en PHP.

**$edad = 14;**

La computadora lo guardará de forma aleatoria en algunas de las direcciones antes mencionadas, digamos que lo guarda en Mariana, es decir:

* **Carlos.**
* **Mariana.**
  + $edad = 14;
* **José.**
* **Raúl.**
* **Etc..**

Ahora dentro de la dirección llamada **Mariana** se encuentra almacenada la variable **$edad = 14;**

**Mariana** es la **referencia** porque es el nombre que me dice en qué lugar de la memoria se guardó mi variable, es decir, **hace referencia hacia el espacio donde está guardado**.

## **¿Qué es el paso de parámetros por referencia?**

Cuando pasamos parámetros a una función, lo que hacemos es hacer una **copia** de la variable, es decir, estamos guardando **el mismo valor** en **diferentes** espacios de memoria.

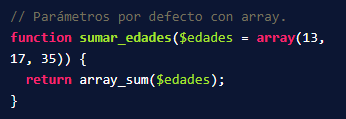
## **¿Qué es el paso de parámetros por referencia?**

Hacer un paso por referencia significa que, **en lugar de mandar una copia** de la variable, estamos mandando la **referencia** **de dicha variable**. Es decir, estamos apuntando al **mismo espacio en memoria** gracias a que tenemos su referencia.

# **Argumentos a profundidad**

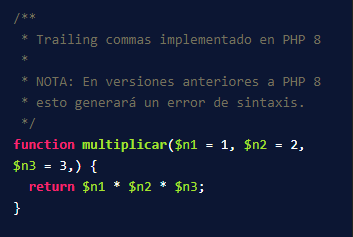
## **Arrays como parametros opcional**

Existe la funcionalidad de poder colocar arrays como argumentos.



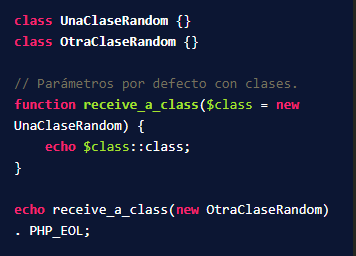
## **Trailing commas**

Los trailing commas son comas que van el apartado de los parametros de tu función, lo que destaca es que estos parámetros pueden terminar con coma sin marcar error. Sólo para versiones superiores a PHP 8.



## **Recibir clases como parámetros**

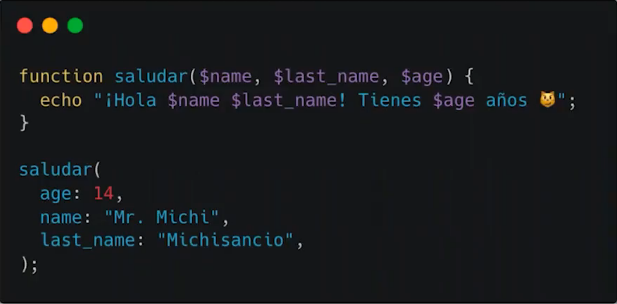
**No instancias** sino la **inicialización** de una (solo php > 8)



# **Named arguments**

Los named arguments son una forma de pasarle parámetros a una función basándose en el **nombre del parámetro** en lugar de la **posición**. Es decir, puedo simplemente mencionar a qué parámetro le quiero pasar algún valor.

Sólo disponibles desde las versiones de PHP 8 o superiores.

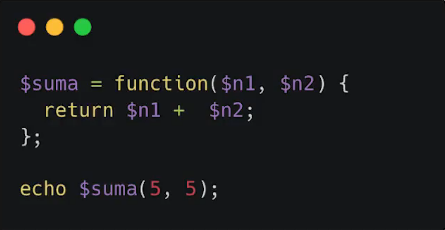


# **Funciones anónimas**

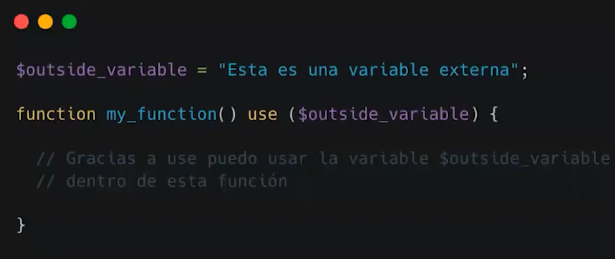
Las funciones anónimas o también conocidas como **Closures** son funciones que podemos definir sin necesidad de asignarles un nombre. Son muy similares a los **callbacks** de Javascript

Las podemos guardar dentro de variables, pero su uso más común es pasarlas como parámetro de otra función.

Las funciones anónimas son una instancia de la clase **Closure**.



Cuando queremos que una variable **externa** exista **dentro** de un **Closure** sin necesidad de pasarla como parámetro podemos usar la palabra reservada “**use**”

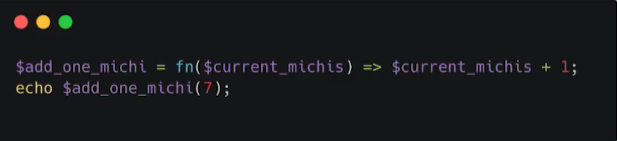


Con “**use**” cambiamos el ámbito de la variable de **global** a **local** y la podemos usar dentro del **closure**, pero debemos tener en cuenta que solo podemos leerla, pero **no** cambiar su valor.

# **Arrow functions**

**Son funciones que se pueden definir en una sola línea**. Su sintaxis es muy similar a las arrow functions de JavaScript, mientras que su funcionamiento es muy similar a las lambdas de python. Este feature fue introducido en PHP 7.4

Este tipo de funciones también son implementadas usando la clase **Closure**.



## **Diferencias con las funciones anónimas**

Las arrow functions **NO** declaran un scope local.

# **Declaraciones de tipo escalar**

Cuando declaramos una funcion es posible indicar que tipo de datos queremos recibir en nuestros parametros. Esto nos da la certeza de que estamos trabajando con el tipo de dato que realmente necesitamos.

En otras palabras, podemos tipar nuestras funciones. Este comportamiento, por defecto, es de tipo **coercitivo,** es decir, es opcional, pero podemos hacer que sea obligatorio poniendo la palabra reservada **declare** al inicio de nuestro script.

# **Declaraciones de tipo de devolución**

PHP 7 añade soporte para **declaraciones de tipo de devolución**. Similarmente a las **declaraciones de tipo de argumento**, las declaraciones de tipo de devolución **especifican el tipo del valor que será devuelto por una función**. Están disponibles los mismos tipos tanto para las declaraciones de tipo de devolución como para las declaraciones de tipo de argumento.