Funciones

Índice

[Funciones 3](#_Toc72054731)

[Argumentos de funciones 4](#_Toc72054732)

[Paso de argumentos por referencia 4](#_Toc72054733)

[Valores de argumentos predeterminados 5](#_Toc72054734)

[Devolver valores 6](#_Toc72054735)

[Funciones variables 7](#_Toc72054736)

[Funciones internas 8](#_Toc72054737)

[Funciones anónimas 9](#_Toc72054738)

[Funciones de flechas 10](#_Toc72054739)

[Objetos 11](#_Toc72054740)

[Definición de clases 12](#_Toc72054741)

[Instancia de una clase 12](#_Toc72054742)

[Constructor 13](#_Toc72054743)

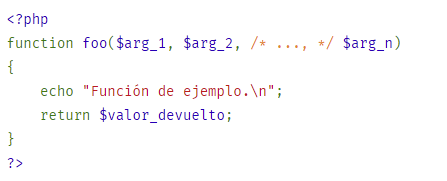
[Herencia 14](#_Toc72054744)

[Interfaces 15](#_Toc72054745)

[Clases anonimas 16](#_Toc72054746)

# Funciones

Una función puede ser definida empleando una sintaxis como la siguiente:



Cualquier código PHP válido puede aparecer dentro de una función, incluso otras funciones y definiciones de [clases](https://www.php.net/manual/es/language.oop5.basic.php#language.oop5.basic.class).

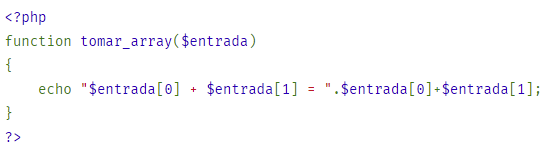
Los nombres de las funciones siguen las mismas reglas que las demás etiquetas de PHP. Un nombre de función válido comienza con una letra o guion bajo, seguido de cualquier número de letras, números o guiones bajos. Como expresión regular se expresaría así: ^[a-zA-Z\_\x80-\xff][a-zA-Z0-9\_\x80-\xff]\*$.

No es necesario definir una función antes de que sea referenciada, excepto cuando esta esté condicionalmente definida como se muestra en los dos ejemplos de abajo.

Cuando una función está definida de una forma condicional como en los dos ejemplos siguientes, sus definiciones deben ser procesadas antes de ser llamadas.

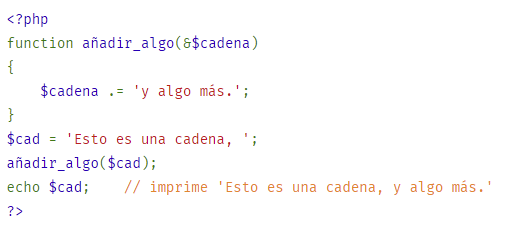
# Argumentos de funciones

Cualquier información puede ser pasada a las funciones mediante la lista de argumentos, la cual es una lista de expresiones delimitadas por comas. Los argumentos son evaluados de izquierda a derecha.



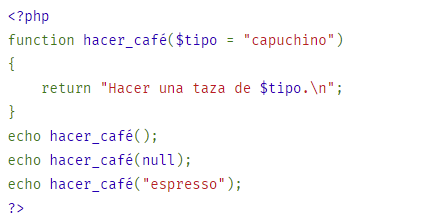
## Paso de argumentos por referencia

Por defecto, los argumentos de las funciones son pasados por valor (así, si el valor del argumento dentro de la función cambia, este no cambia fuera de la función). Para permitir a una función modificar sus argumentos, éstos deben pasarse por referencia.



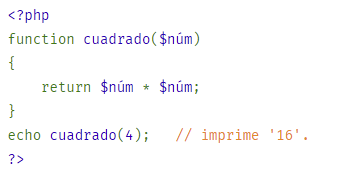
## Valores de argumentos predeterminados

Una función puede definir valores predeterminados al estilo de C++ para argumentos escalares como sigue:



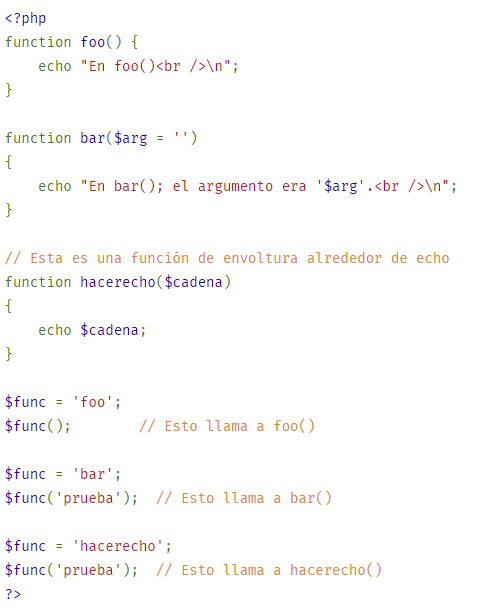
# Devolver valores

Los valores son devueltos usando la sentencia opcional return. Se puede devolver cualquier tipo, incluidos arrays y objetos. Esto causa que la función finalice su ejecución inmediatamente y pase el control de nuevo a la línea desde la que fue llamada. Véase [return](https://www.php.net/manual/es/function.return.php) para más información.



# Funciones variables

PHP admite el concepto de funciones variables. Esto significa que si un nombre de variable tiene paréntesis anexos a él, PHP buscará una función con el mismo nombre que lo evaluado por la variable, e intentará ejecutarla. Entre otras cosas, esto se puede usar para implementar llamadas de retorno, tablas de funciones, y así sucesivamente.



# Funciones internas

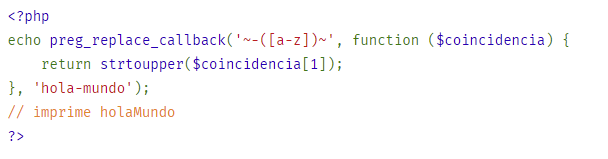
PHP se estandariza con muchas funciones y construcciones. También existen funciones que necesitan extensiones específicas de PHP compiladas, si no, aparecerán errores fatales "undefined function" ("función no definida"). Por ejemplo, para usar las funciones de [image](https://www.php.net/manual/es/ref.image.php) tales como [imagecreatetruecolor()](https://www.php.net/manual/es/function.imagecreatetruecolor.php), PHP debe ser compilado con soporte para GD. O para usar [mysql\_connect()](https://www.php.net/manual/es/function.mysql-connect.php), PHP debe ser compilado con soporte para [MySQLi](https://www.php.net/manual/es/book.mysqli.php). Hay muchas funciones de núcleo que está incluidas en cada versión de PHP, tales como las funciones de [string](https://www.php.net/manual/es/ref.strings.php) y de [variable](https://www.php.net/manual/es/ref.var.php). Una llamada a [phpinfo()](https://www.php.net/manual/es/function.phpinfo.php) o [get\_loaded\_extensions()](https://www.php.net/manual/es/function.get-loaded-extensions.php) mostrará las extensiones que están cargadas en PHP. Observe también que muchas extensiones están habilitadas por defecto y que el manual de PHP está dividido por extensiones. Véase [configuración](https://www.php.net/manual/es/configuration.php), [instalación](https://www.php.net/manual/es/install.php), y capítulos individuales de extensiones para más información sobre cómo configurar PHP.

Interpretar y comprender un prototipo de una función está explicado dentro de la sección del manual titulada [cómo interpretar la definición de una función](https://www.php.net/manual/es/about.prototypes.php). Es importante comprender lo que devuelve una función o si una función funciona directamente con un valor pasado. Por ejemplo, [str\_replace()](https://www.php.net/manual/es/function.str-replace.php) devolverá la cadena modificada mientras que [usort()](https://www.php.net/manual/es/function.usort.php) funciona con la variable actual pasada. Cada página del manual también tiene información específica para cada función, como información sobre parámetros de funciones, cambios de comportamiento, valores devueltos en caso de éxito o fallo, e información de disponibilidad. Conocer estas importantes diferencias (a menudo imperceptibles) es crucial para escribir código de PHP correcto.

# Funciones anónimas

Las funciones anónimas, también conocidas como cierres (closures), permiten la creación de funciones que no tienen un nombre especificado. Son más útiles como valor de los parámetros de llamadas de retorno, pero tienen muchos otros usos.

Las funciones anónimas están implementadas utilizando la clase [Closure](https://www.php.net/manual/es/class.closure.php).



# Funciones de flechas

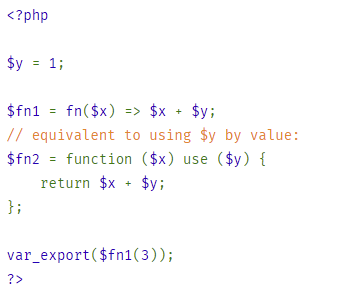
Las funciones de flecha fueron introducidas en PHP 7.4 como una sintaxis más concisa para [las funciones anónimas](https://www.php.net/manual/es/functions.anonymous.php).

Tanto las funciones anónimas como las funciones de flecha se implementan utilizando la clase [Closure](https://www.php.net/manual/es/class.closure.php).

Las funciones de las flechas tienen la forma básica fn (argument\_list) => expr.

Las funciones de las flechas soportan las mismas características que [las funciones anónimas](https://www.php.net/manual/es/functions.anonymous.php), excepto que el uso de variables del ámbito padre siempre es automático.

Cuando una variable utilizada en la expresión se define en el ámbito padre será implícitamente capturada por valor. En el siguiente ejemplo, las funciones *$fn1* y *$fn2* se comportan de la misma manera.



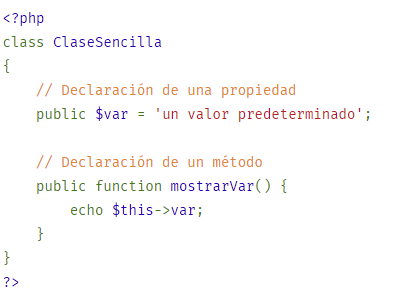
# Objetos

PHP trata los objetos de la misma manera que las referencias o manejadores, lo que significa que cada variable contiene una referencia a un objeto en lugar de una copia de todo el objeto.

## Definición de clases

La definición básica de una clase comienza con la palabra reservada class, seguida de un nombre de clase, y continuando con un par de llaves que encierran las definiciones de las propiedades y métodos pertenecientes a dicha clase.

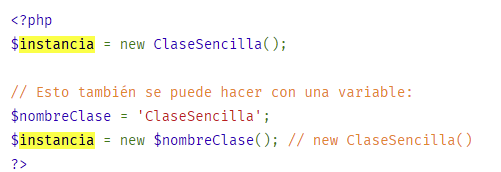
El nombre de clase puede ser cualquier etiqueta válida, siempre que no sea una [palabra reservada](https://www.php.net/manual/es/reserved.php) de PHP. Un nombre válido de clase comienza con una letra o un guión bajo, seguido de una cantidad arbitraria de letras, números o guiones bajos. Como expresión regular, se expresaría de la siguiente forma: ^[a-zA-Z\_\x80-\xff][a-zA-Z0-9\_\x80-\xff]\*$.



## Instancia de una clase

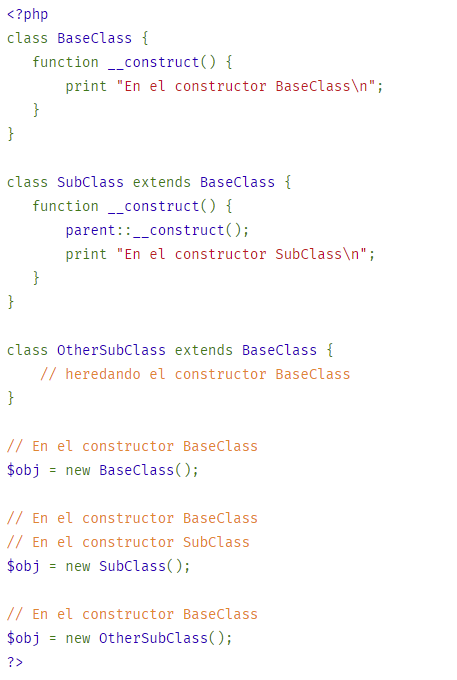
Para crear una instancia de una clase, se debe emplear la palabra reservada new. Un objeto se creará siempre a menos que el objeto tenga un [constructor](https://www.php.net/manual/es/language.oop5.decon.php) que arroje una [excepción](https://www.php.net/manual/es/language.exceptions.php) en caso de error. Las clases deberían ser definidas antes de la instanciación (y en algunos casos esto es un requerimiento).

Si se emplea un string que contenga el nombre de una clase con new, se creará una nueva instancia de esa clase. Si la clase estuviera en un espacio de nombres, se debe utilizar su nombre completo al realizar esto.



## Constructor

PHP permite a los desarrolladores declarar métodos constructores para las clases. Aquellas que tengan un método constructor lo invocarán en cada nuevo objeto creado, lo que lo hace idóneo para cualquier inicialización que el objeto pueda necesitar antes de ser usado. Los constructores padres no son llamados implícitamente si la clase hija define un constructor. Para ejecutar un constructor padre, se requiere invocar a parent::\_\_construct() desde el constructor hijo. Si el hijo no define un constructor, entonces se puede heredar de la clase madre como un método de clase normal (si no fue declarada como privada).



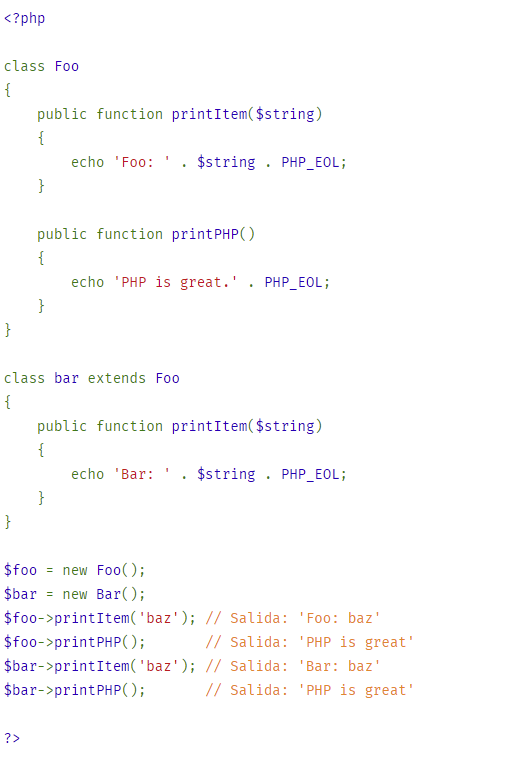
## Herencia

La herencia es un principio de programación bien establecido y PHP hace uso de él en su modelado de objetos. Este principio afectará la manera en que muchas clases y objetos se relacionan unas con otras.

Por ejemplo, cuando se extiende una clase, la subclase hereda todos los métodos públicos y protegidos de la clase padre. A menos que una clase sobrescriba esos métodos, mantendrán su funcionalidad original.

Esto es útil para la definición y abstracción de la funcionalidad y permite la implementación de funcionalidad adicional en objetos similares sin la necesidad de reimplementar toda la funcionalidad compartida.

A menos que la carga automática sea utilizada, entonces las clases deben ser definidas antes de ser usadas. Si una clase se extiende a otra, entonces la clase padre debe ser declarada antes de la estructura de clase hija. Esta regla se aplica a las clases que heredan de otras clases e interfaces.



## Interfaces

Las interfaces de objetos permiten crear código con el cual especificar qué métodos deben ser implementados por una clase, sin tener que definir cómo estos métodos son manipulados.

Las interfaces se definen de la misma manera que una clase, aunque reemplazando la palabra reservada class por la palabra reservada interface y sin que ninguno de sus métodos tenga su contenido definido.

Todos los métodos declarados en una interfaz deben ser públicos, ya que ésta es la naturaleza de una interfaz.

Para implementar una interfaz, se utiliza el operador implements. Todos los métodos en una interfaz deben ser implementados dentro de la clase; el no cumplir con esta regla resultará en un error fatal. Las clases pueden implementar más de una interfaz si se deseara, separándolas cada una por una coma. Es posible tener constantes dentro de las interfaces. Las constantes de interfaces funcionan como las [constantes de clases](https://www.php.net/manual/es/language.oop5.constants.php) excepto porque no pueden ser sobrescritas por una clase/interfaz que las herede. 

## Clases anonimas

En PHP 7 se ha añadido soporte para clases anónimas. Las clases anónimas son útiles cuando es necesario crear objetos sencillos y únicos.

