



PHP – Funciones



Ciclo: DAW
Módulo: DWES
Curso: 2020-2021
Autor: César Guijarro Rosaleny



Introducción.....	3
Declaración de funciones.....	4
Argumentos.....	7
Paso de argumentos por referencia.....	7
Paso de valores predeterminados.....	8
Declaraciones de tipo.....	9
Lista de argumentos de longitud variable.....	11
Devolver valores.....	13
Funciones variables.....	14
Funciones anónimas.....	15
Funciones de flecha.....	18
Bibliografía.....	19

Introducción

PHP, como cualquier lenguaje estructurado, permite modular el código mediante funciones.

En este tema veremos como definir funciones en PHP, como pasar argumentos a las funciones y como devolver valores.

Además, desde la versión 5.3.0 PHP permite utilizar **funciones anónimas**. Veremos qué son y como implementarlas

Por último, desde la versión 7.4.0 también permite las **funciones flecha**. Estas funciones nos permiten utilizar una sintaxis más concisa para las funciones anónimas.

Declaración de funciones

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

```
<?php
function foo($arg_1, $arg_2, /* ..., */ $arg_n) {
    echo "Función de ejemplo.\n";
    return $valor_devuelto;
}
```

En primer lugar usamos la palabra reservada **function** seguido del nombre que queramos dar a nuestra función. A continuación, definimos la lista de argumentos separados por coma y encerrados entre paréntesis.

El código de la función irá encerrado entre llaves (**{ }**). Cualquier código PHP válido puede aparecer dentro de la función, incluso otras funciones y definiciones de clases. Para devolver valores, usaremos **return**.

Al contrario que otros lenguajes, en PHP no es necesario definir una función antes de que sea referenciada (excepto cuando ésta esté condicionalmente definida). Por lo tanto, el siguiente código es perfectamente válido.

```
<?php
bar(4);

function bar($numero) {
    echo $numero;
};
```

Podemos definir funciones dentro de otras funciones, pero para poder llamarlas necesitamos llamar antes a la función padre.

```
<?php
function foo() {
    function bar() {
        echo "Hola mundo";
    }
}

bar();
```

Si llamamos directamente a la función `bar()` como en el ejemplo anterior, PHP nos mostrará un error diciendo que la función no está definida.

(!) Fatal error: Uncaught Error: Call to undefined function bar() in /var/www/html/DWS/clases/T2/clase17.php on line 21				
(!) Error: Call to undefined function bar() in /var/www/html/DWS/clases/T2/clase17.php on line 21				
Call Stack				
#	Time	Memory	Function	Location
1	0.1573	398440	{main}()	.../clase17.php:0

Sin embargo, si llamamos antes a la función `foo()` (aunque no haga nada), PHP ya nos permitirá usar la función `bar()`.

```
<?php
function foo() {
    function bar() {
        echo "Hola mundo";
    }
}

foo();
bar();
```

← → ↺ ⓘ loca

Hola mundo

NOTA: Los nombres de las funciones **son insensibles a mayúsculas-minúsculas**, aunque es una buena idea llamar a las funciones tal y como aparecen en sus declaraciones.

No se pueden redefinir funciones. Una vez definida una función, no podemos crear otra con el mismo nombre.

```
<?php
function foo() {

};

function foo() {

};
```

 Fatal error: Cannot redeclare foo() (previously declared in /var/www/html/DWS/clases/T2/clase17.php:32) in /var/www/html/DWS/clases/T2/clase17.php on line 36

PHP permite recursividad.

```
<?php
function recursividad($a) {
    if ($a < 20) {
        echo "$a\n";
        recursividad($a + 1);
    }
}
```

Argumentos

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. Para hacer que un argumento a una función se pase por referencia, hay que anteponer al nombre del argumento un ampersand (&).

Por ejemplo, si ejecutamos el siguiente código en un navegador:

```
<?php
function anyadir_algo($cadena) {
    $cadena .= 'y algo más.';
}
$cad = 'Esto es una cadena, ';
anyadir_algo($cad);
echo $cad;
```

La salida será:



Esto es una cadena,

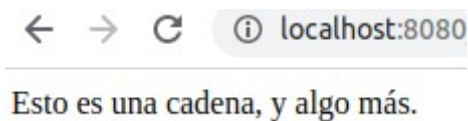
Ya que al pasar el argumento *\$cad* a la función por valor, en realidad le estamos pasando una copia de la variable, no la variable en sí.

Si queremos pasar una referencia a la variable, deberemos definir la función de la siguiente forma:

```
<?php
function anyadir_algo(&$cadena) {
    $cadena .= 'y algo más.';
}
$cad = 'Esto es una cadena, ';
anyadir_algo($cad);
echo $cad;
```



Lo que hará que la variable *\$cad* cambie su valor al ejecutarse la función.



Paso de valores predeterminados

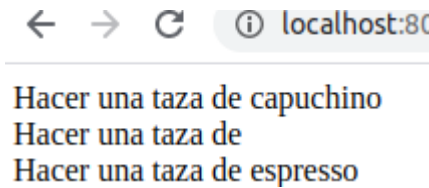
Una función puede definir valores predeterminados para argumentos escalares como sigue:

```
<?php
function hacer_café($tipo = "capuchino") {
    return "Hacer una taza de $tipo.\n";
}

echo hacer_café();
echo hacer_café(null);
echo hacer_café("espresso");
```



El resultado del ejemplo anterior sería:



← → ↻ ⓘ localhost:80

Hacer una taza de capuchino
Hacer una taza de
Hacer una taza de espresso

Declaraciones de tipo

Las declaraciones de tipo permiten a las funciones requerir que los parámetros sean de cierto tipo durante una llamada. Si el valor dado es de un tipo incorrecto, se generará un error (en PHP 7 se lanzará una excepción *TypeError*).

Para especificar una declaración de tipo, debe anteponerse el nombre del tipo al nombre del parámetro. Se puede hacer que una declaración acepte valores NULL si el valor predeterminado del parámetro se establece a NULL.

```
<?php
function foo(int $numero) {
    return $numero + 1;
}

echo foo(2);
```



← → ↻ ⓘ loc

3

```
<?php
function foo(int $numero) {
    return $numero + 1;
}

echo foo("Hola");
```

! Fatal error: Uncaught TypeError: Argument 1 passed to foo() must be of the type int, string given, called in /var/www/html/DWS/clases/T2/clase18.php on line 27 and defined in /var/www/html/DWS/clases/T2/clase18.php on line 23

! TypeError: Argument 1 passed to foo() must be of the type int, string given, called in /var/www/html/DWS/clases/T2/clase18.php on line 27 in /var/www/html/DWS/clases/T2/clase18.php on line 23

Call Stack

#	Time	Memory	Function	Location
1	0.1349		398088 {main}()	.../clase18.php:0
2	0.1350		398088 foo()	.../clase18.php:27

A partir de PHP 7 podemos usar **declaración estricta de tipos**. Por ejemplo, si definimos una función con un argumento de tipo *string* y le pasamos un *integer*, PHP convertirá ese entero en una cadena.

```
<?php
function foo(string $palabra) {
    echo "La palabra es $palabra";
}

foo(1);
```

← → ↺ ⓘ

La palabra es 1



Pero podemos habilitar el modo estricto en el fichero añadiendo la línea **function**.

```
<?php
declare(strict_types=1);

function foo(string $palabra) {
    echo "La palabra es $palabra";
}

foo(1);
```

Con lo que, ahora sí, PHP nos mostraría el error correspondiente.

 Fatal error: Uncaught TypeError: Argument 1 passed to foo() must be of the type string, int given, called in /var/www/html/DWS/clases/T2/clase18.php on line 35 and defined in /var/www/html/DWS/clases/T2/clase18.php on line 31				
 TypeError: Argument 1 passed to foo() must be of the type string, int given, called in /var/www/html/DWS/clases/T2/clase18.php on line 35 in /var/www/html/DWS/clases/T2/clase18.php on line 31				
Call Stack				
#	Time	Memory	Function	Location
1	0.1378	398160	{main}()	.../clase18.php:0
2	0.1378	398160	foo()	.../clase18.php:35

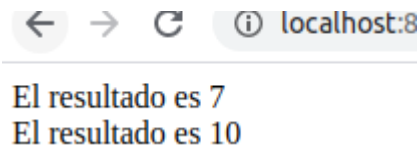
Lista de argumentos de longitud variable

PHP tiene soporte para lista de argumentos de longitud variable en funciones definidas por el usuario. Esto se implementa utilizando el token **...** en PHP 5.6 y posteriores. Los argumentos serán pasados a la función como un array.

Los valores son devueltos usando la sentencia opcional **return**. Se puede devolver cualquier tipo, incluidos arrays y objetos. Ésto causa que la función finalice su ejecución inmediatamente y pase el control de nuevo a la línea desde la que fue llamada. Si se omite *return* el valor devuelto será NULL.

```
<?php
function suma(...$numeros) {
    $resultado = 0;
    foreach ($numeros as $n) {
        $resultado += $n;
    }
    return "El resultado es $resultado<br>";
}

echo suma(2, 5);
echo suma(1, 2, 3, 4);
```



El resultado es 7
El resultado es 10

Se puede especificar argumentos normales antes del token `...`. En este caso, solamente los argumentos al final que no coincidan con un argumento posicional serán añadidos al array generado por `...`.

También es posible añadir una declaración de tipo antes del símbolo `...`. Si está presente, todos los argumentos capturados por `...` deben ser objetos de la clase implicada.

```
<?php
function suma(string $cadena, int ...$numeros) {
    $resultado = 0;
    foreach ($numeros as $n) {
        $resultado += $n;
    }
    return "$cadena $resultado<br>";
}

echo suma("El resultado es ", 5, 8, 7);
echo suma("La suma es ", 5, 8, 7);
```

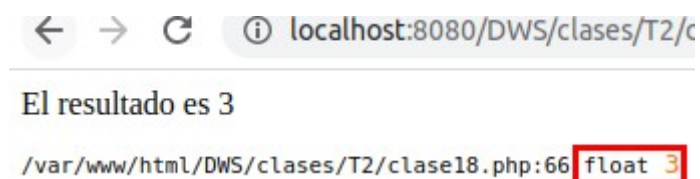
Devolver valores

PHP 7 añade soporte para las declaraciones de tipo de devolución. De forma similar a las declaraciones de tipo de argumento, las declaraciones de tipo de devolución especifican el tipo del valor que será devuelto desde una función.

```
<?php
function sum($a, $b): float {
    return $a + $b;
}

$resultado = sum(1, 2);
echo "El resultado es $resultado";
var_dump($resultado);
```

Si nos fijamos en el resultado, aunque parezca que la función devuelve un entero con *var_dump* vemos que lo que realmente está devolviendo es un *float*.



← → ↻ ⓘ localhost:8080/DWS/clases/T2/c

El resultado es 3

/var/www/html/DWS/clases/T2/clase18.php:66 float 3

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.

```
<?php
function foo() {
    echo "En foo(<br>";
}

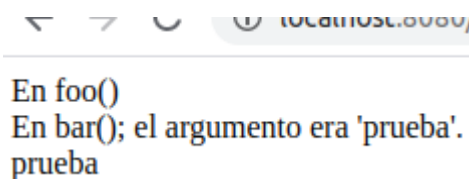
function bar($arg = ") {
    echo "En bar(); el argumento era '$arg'.<br>";
}

function hacerecho($cadena) {
    echo $cadena;
}

$func = 'foo';
$func(); // Esto llama a foo()

$func = 'bar';
$func('prueba'); // Esto llama a bar()

$func = 'hacerecho';
$func('prueba'); // Esto llama a hacerecho()
```



```
En foo()
En bar(); el argumento era 'prueba'.
prueba
```

Funciones anónimas

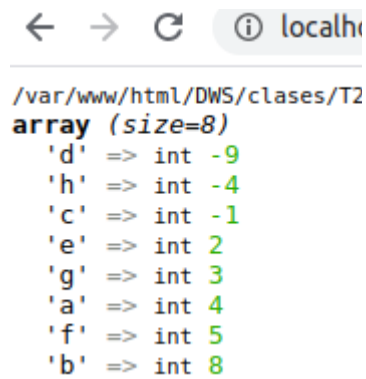
A partir de PHP 5.3.0, las **funciones anónimas**, también conocidas como **cierres (closures)**, permiten la creación de funciones que **no tienen un nombre especificado**. Son más útiles como valor de los parámetros de callbacks, pero tienen muchos otros usos.

Por ejemplo, vamos a utilizar la función `uasort()` de PHP, la cual ordena un array con una función de comparación definida por el usuario y mantiene la asociación de índices.

```
<?php
function cmp($a, $b) {
    if ($a == $b) {
        return 0;
    }
    return ($a < $b) ? -1 : 1;
}

$array = [
    'a' => 4,
    'b' => 8,
    'c' => -1,
    'd' => -9,
    'e' => 2,
    'f' => 5,
    'g' => 3,
    'h' => -4
];

uasort($array, 'cmp');
var_dump($array);
```



```

array (size=8)
  'd' => int -9
  'h' => int -4
  'c' => int -1
  'e' => int 2
  'g' => int 3
  'a' => int 4
  'f' => int 5
  'b' => int 8

```

Podríamos utilizar una *función anónima* para mejorar la organización y aumentar la legibilidad del código.

```

<?php
$array = [
    'a' => 4,
    'b' => 8,
    'c' => -1,
    'd' => -9,
    'e' => 2,
    'f' => 5,
    'g' => 3,
    'h' => -4
];

uasort($array, function ($val1, $val2) {
    return $val1 == $val2 ? 0 : ($val1 < $val2 ? -1 : 1);
});
var_dump($array);

```


Las *funciones anónimas* también se pueden usar como valores de variables. La asignación de un *función anónima* a una variable emplea la misma sintaxis que cualquier otra asignación, incluido el punto y coma final:

```
<?php
    $saludo = function($nombre) {
        echo"Hola $nombre<br>";
    };

    $saludo('Mundo');
    $saludo('PHP');
```



Hola Mundo
Hola PHP

Para acceder a variables globales dentro de las funciones anónimas, podemos utilizar **use**.

```
<?php
    $sempezar = "Hola ";
    $saludo = function($nombre) use($sempezar) {
        echo"$sempezar$nombre<br>";
    };

    $saludo('Mundo');
    $saludo('PHP');
```



Hola Mundo
Hola PHP

Funciones de flecha

Las **funciones de flecha** fueron introducidas en PHP 7.4 como una sintaxis más concisa para las *funciones anónimas*.

Las *funciones de flecha* tienen la forma básica **fn (lista de argumentos) => expresión**.

```
<?php
    $suma = fn($x, $y) => $x + $y;

    $resultado = $suma(2, 3);
    echo "El resultado es $resultado";
```

El resultado es 5

Las *funciones de flecha* **sólo permiten una expresión**, lo que significa que **no pueden contener varias líneas de código**. Es por eso que **se omiten las llaves de inicio y final de función y la palabra reservada **return****.

El ejemplo anterior usando funciones anónimas es equivalente a:

```
<?php
    $suma = function($x, $y) {
        return $x + $y;
    };

    $resultado = $suma(2, 3);
    echo "El resultado es $resultado";
```

Bibliografía

<https://www.php.net/manual/es/language.functions.php>