Índice

Funciones

Cuando utilizamos en múltiples ocasiones un mismo fragmento de código debemos usar funciones (**functions**). Una función es una herramienta para encapsular y ejecutar un mismo código. Entre sus ventajas veremos que nos ayuda a que los ficheros tengan un menor tamaño y sean más fáciles de mantener.

Las funciones deben definirse antes de utilizarlas (aunque no sea en el mismo fragmento de código php). Las funciones pueden no tener argumento o tener argumentos por valor o por referencia.

Los nombres de las funciones siguen las mismas reglas de los identificadores de PHP, es decir, deben comenzar por una letra o un guión bajo (_) y el resto de caracteres pueden ser letras, números o guiones bajos (se pueden utilizar caracteres no ingleses como acentos, eñes, etc), pero los nombres de funciones no distinguen entre mayúsculas o minúsculas.

Se recomienda que los nombres de las funciones sigan el estilo camelCase (es decir, sin espacios ni guiones, con la primera palabra en minúsculas y el resto con la primera letra en mayúsculas).

La sintaxis clásica de la función es:

```
<?php
// Declarar
  function nombreFuncion($argumento1,$argumento2, ...) {
    instrucciones de la función;
    return ...;
  }

// Llamar
nombreFuncion($parametros);
?>
```

Veamos cuales son los elementos del código anterior:

- La palabra **function** es una palabra reservada:
- **nombreFuncion**: Es el nombre que daremos a la función, podemos escribir aquí cualquier palabra, y este será el nombre que tenga la función.
- **()**: Escribimos luego los paréntesis, y dentro de ellos los argumentos, en caso de que sean necesarios.
    - **$argumentos**: Son los datos necesarios para poder ejecutar la función, por ejemplo una función que realice la suma de dos números, necesitará que le pasemos como argumentos esos dos números. Una función puede necesitar o no argumentos, por lo que no es obligatorio ponerlos si no es necesario. Si necesita más de un argumento, estos se pondrán separados por comas. El argumento puede ser cualquier variable, la cual la utilizaremos dentro de la función para operar con ella y obtener el resultado.

- **{Instrucciones}**: Entre llaves pondremos las instrucciones que haga falta para ejecutar la función.
- **return** : la palabra clave return va seguido del resultado que devuelve la función.

La sintaxis moderna de la función es:

```
<?php
// Declarar
function nombreFuncion(tipo_de_parametro $parametros):
tipo_return
{
    instrucciones de la función
    return ...;
}

// Llamar
nombreFuncion($parametros);
?>
```

En el siguiente ejemplo tenemos una función clásica, con sintaxis moderna. En dicho ejemplo tenemos declarada una función y posteriormente es ejecutada seguido de un echo para ver el resultado. Lo único que hace es devolver un texto cuando se llama.

Ejemplo de función y de su uso:

```
<?php
 /**
 * Función con educación
 * @return {string}
 */
 function saludarAhora(): string
 {
   return "<p>Hola, soy una función.</p>";
 }

 echo saludarAhora();

?>
```

```
<p>Hola, soy una función.</p>
```

Las partes que componen una función son las siguientes:

- Las 4 primeras líneas es el formato de comentarios. Una función, por mucha prisa que tengas, debe estar comentada con el formato del ejemplo.

- La palabra **function** es una palabra reservada. A continuación de la palabra reservada debe de ir el nombre de la función. Después unos paréntesis con los argumentos. Si no tiene, se dejan vacíos pero siempre tienen que estar presentes. Luego dos puntos. Por último el tipo de valor resultante.
- Llaves para envolver el código {}.
- Y dentro del código la palabra reservada **return** seguido del valor a devolver.

La palabra **return** establece el valor de respuesta de una función a cualquier variable, y después PHP sale de la función. Si se usa sólo **return**, se sale de la función sin devolver ningún valor.

Nota: Si se omite return, el valor devuelto será null.

```
<?php
 /**
 * Función con educación
 */
 function saludarAhora(): void
 {
    ...
    return
    echo "<p>Hola, soy una función.</p>";
 }

 echo saludarAhora();

?>
```

Nunca se verá "Hola, soy una función." porque ya se ha salido de la función.

Como hemos indicado anteriormente PHP no distingue entre mayúsculas y minúsculas en los nombres de las funciones, con lo cual no se pueden definir dos funciones con el mismo nombre.

```
<?php
 /**
  * Función sintaxis clásica
 */
 function prueba()
 {
    return;
 }

 /**
  * Función sintaxis moderna
 */
```

```
 function Prueba():void
 {
   return;
 }
?>
```

```
Fatal error: Cannot redeclare Prueba() (previously declared in ejemplo.php:4) in ejemplo.php on line 14
```

En la versión clásica de la definición de una función en PHP, en las que no se indica el tipo, se asocia automáticamente un tipo de datos a la variable, dependiendo de su valor. Dado que los tipos de datos no se establecen en un sentido estricto, puede hacer cosas como añadir una cadena a un entero sin causar un error.

En la versión moderna, las declaraciones de tipo permiten a las funciones requerir que los parámetros sean de cierto tipo durante una llamada. La declaración moderna obliga a utilizar las cosas de la manera prevista, así que se recomienda su uso siempre que sea posible

Return con tipos alternativos

A partir de la versión 7.1 de PHP disponemos de la posibilidad de indicar si un return devuelve un tipo concreto o un null. Para ello solo habrá que añadir un interrogante en su inicio.

```
<?php
/**
 * Método que indica el tipo de metal que debe tener una medalla a partir del resultado
 * @param {int} $posicion - Posición
 * @return {string|null} - Tipo de metal
 */
function tipoDeMedalla(int $posicion): ?string
{
    $medalla;
    switch ($posicion) {
        case 1:
            $medalla = "Oro";
            break;
        case 2:
            $medalla = "Plata";
            break;
        case 3:
            $medalla = "Bronce";
            break;
        default:
            $medalla = null;
    }
    return $medalla;
```

```
}

echo "<p>" . tipoDeMedalla(2) . "</p>";

echo "<p>" .tipoDeMedalla(34) . "</p>";
?>
```

```
<p>Plata</p>
<p>null</p>
```

Y la versión PHP 8 enriquece más las posibilidades, ya que es posible indicar 2 tipos diferentes.

```
<?php
/*
 * Método que duplica un número en positivo
 * @param {int} $numero - Número a duplicar
 * @return {float|string} - Resultado o mensaje de ayuda
 */
function duplicarPositivo(float $numero): float|string
{
    $valor;
    if ($numero > 0) {
        $valor = $numero * 2;
    } else {
        $valor = 'No puedes usar número en negativo o cero';
    }
    return $valor;
}

echo "<p>" . duplicarPositivo(12.1) . "</p>";

echo "<p>" . duplicarPositivo(-45) . "</p>";

?>
```

```
<p>24.2</p>
<p>No puedes usar número en negativo o cero</p>
```

A partir de PHP 8 incluso tenemos la opción de utilizar **mixed,** que es equivalente a union type (object|resource|array|string|int|float|bool|null).

Funciones con argumentos

En las funciones pueden incluirse argumentos. Los argumentos de las funciones son un conjunto de expresiones delimitadas por comas evaluadas de izquierda a derecha que se emplean para pasar información a las funciones.

Se pueden pasar argumentos por valor, por referencia y por valor predeterminado.

Pasar argumentos por valor

Por defecto los argumentos de las funciones se pasan por valor, si el valor del argumento dentro de la función cambia, éste no cambia fuera de la función.

```
<?php
/**
 * Corta un texto a 10 letras y añade puntos suspensivos al final
 * @param {string} $text - Texto a tratar
 * @return {string}
 */
function resumen(string $text): string
{
    return substr($text, 0, 10) . '...';
}

echo "<p>". resumen("La vida es un cuento contado por un idiota,
lleno de ruido y de furia, que no tiene ningún sentido.") .
"</p>";
?>
```

```
<p>La vida es...</p>
```

A la hora de pasar los argumentos, PHP arreglará las incompatibilidades de tipos de forma automática

```
<?php
/**
 * Incrementa uno un número
 * @param {int} $num - Número a incrementar
 * @return {int}
 */
function incrementar(int $num): int
{
    return $num + 1;
}

echo "<p>". incrementar(4.5) . "</p>";

?>
```

```
<p>5</p>
```

Pero si quieres ser estricto deberás desactivar esta ayuda. En otras palabras, que si encuentra algún problema de tipos muestre un error y no aplique una solución mágica.

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

/**
```

```php
 * Incrementa uno un número
 * @param {int} $num - Número a incrementar
 * @return {int}
 */
function incrementar(int $num): int
{
    return $num + 1;
}

echo "<p>". incrementar(4.5 ). "</p>";

?>
```

```
Fatal error: Uncaught TypeError: incrementar(): Argument #1 ($num) must be of type int, float given, called in ejemplo.php on line 14
```

Pasar argumentos por referencia

Para permitir a una función que cambie un argumento, se tiene que pasar por referencia, y se consigue anteponiendo el símbolo **ampersand &** en el argumento:

```php
<?php

/**
 * Incrementa uno un número
 * @param {array} $animales - Lista de animales
 */
function cambiarAnimales(array &$animales): void
{
    $animales = ["mono", "gorila", "léon"];
}

$array = ["perro", "gato", "avestruz"];
cambiarAnimales($array);

foreach($array as $animal){
    echo "<p>$animal</p>";
}

?>
```

```
<p>mono</p>
<p>gorila</p>
<p>léon</p>
```

Valores de argumentos predeterminados

En PHP se pueden definir funciones con argumentos predeterminados, de manera que si en la llamada a la función no se envía un argumento, la función asume un valor

predeterminado. Lógicamente, los argumentos predeterminados deben ser los últimos en la lista de argumentos, para evitar ambigüedades.

Los argumentos predeterminados se establecen en la definición de la función, igualando el nombre del argumento a su valor predeterminado.

```
<?php
/**
 * Saluda a una persona
 * @param {string} $nombre - Nombre
 * @return {string}
 */
 function saludar(string $nombre = 'Anónimo'): string
 {
    return "<p>Hola, persona llamada " . $nombre . ". Por lo que
veo tu nombre mide " . strlen($nombre) . " caracteres.</p>";
 }

echo saludar();

echo saludar("Guillermo");
?>
```

```
<p>Hola, persona llamada Anónimo. Por lo que veo tu nombre mide
8 caracteres.</p>
<p>Hola, persona llamada Guillermo. Por lo que veo tu nombre
mide 9 caracteres.</p>
```

También se pueden poner arrays y de tipo null como valores predeterminados:

```
<?php
/**
 * Devuelve el listado de películas a ver
 * @param {array} $titulos - Títulos de películas
 * @param {array} $genero - Género de las películas
 * @return {string}
 */
function verPeliculas($titulos = array("Avatar"), $genero =
null){
  $generoPelicula = is_null($genero) ? "" : " cuyo género es:
$genero";

return "<p>Hoy vamos a ver: ".join(", ", $titulos) .
$generoPelicula . "</p>";
}

echo verPeliculas();
echo verPeliculas(array("La jungla de cristal", "Transporter"),
"acción");
```

```
?>
```

```
<p>Hoy vamos a ver: Avatar</p>
<p>Hoy vamos a ver: La jungla de cristal, Transporter cuyo género es: acción</p>
```

Tipos admitidos

Las declaraciones de tipos permiten a las funciones exigir que los parámetros proporcionados sean de un tipo determinado.

Para especificar una declaración de tipo se antepone el nombre del tipo al nombre del parámetro. Se puede hacer que la declaración acepte valores null si el valor predeterminado del parámetro se establece a null.

Tipo	Descripción	Versión mínima PHP
string	El parámetro debe ser un string.	PHP 7.0.0
int	El parámetro debe ser un valor de tipo integer.	PHP 7.0.0
float	El parámetro debe ser un número de tipo float.	PHP 7.0.0
bool	El parámetro debe ser un valor de tipo boolean.	PHP 7.0.0
array	El parámetro debe ser un array.	PHP 5.1.0
self	El parámetro debe ser una instanceof de la misma clase donde está definido el método. Esto solamente se puede utilizar en clases y métodos de instancia.	PHP 5.0.0
nombre de clase/interfaz	El parámetro debe ser una instanceof del nombre de la clase o interfaz dada.	PHP 5.0.0

Funciones variables

Las funciones variables son llamadas cuando se añade un paréntesis a una variable y se ejecuta su valor.

Si se llama a una variable que tiene paréntesis anexos a ella, PHP buscará una función con el valor que tiene dicha variable, e intentará ejecutarla.

Las funciones variables no funcionan con constructores del lenguaje como echo, print, unset(), isset(), empty(), include, require, etc.

```
<?php
```

```php
/**
 * Función saludo con educación
 * @return {string}
 */
 function saludar(): string
 {
    return "<p>Hola que tal</p>";
 }
/**
 * Función despedida con educación
 * @param {string} $despedida - despedida
 * @return {string}
 */
 function despedirse(string $despedida = ''): string
 {
   return "<p>Nos vemos otro día, $despedida</p>";

 }

$saludo = "saludar";
$despedida = "despedirse";

echo $saludo(); // Llama a saludar()
echo $despedida("Juan"); // Llama a despedirse()
?>
```

```html
<p>Hola que tal</p>
<p>Nos vemos otro día, Juan</p>
```

Ámbito de las variables

El ámbito de las variables (o scope) divide un script PHP en entornos, lo que afecta a la definición y uso de las variables:

Las variables que se declaran fuera de funciones o clases se encuentran en el ámbito global (global scope), disponibles en cualquier parte del script.

Las funciones son bloques independientes, las variables que se definen dentro de una función tienen ámbito local (local scope) y no afectan a otras en el script global, de la misma forma que las variables globales no están disponibles dentro de las funciones:

```php
<?php
/**
 * Función saludo con educación
 */
 function saludar(): void
```

```
 {
    $saludo = "Hola";
    echo $saludo;
    return;
 }

 $saludo = "Buenos días";
 saludar(); // Devuelve: Hola
 echo $saludo; // Devuelve: Buenos días
?>
```

La ejecución del script empieza en **$saludo = "Buenos días"**, y después llama a la función **saludar()**. Esta función establece **$saludo** a **"Hola"** y lo imprime, y después devuelve el control al script principal donde se imprime **$saludo**. Puede comprobarse que sus valores son diferentes y no se afecta el uno al otro.

Si vas a usar variables que están presentes en el código, puedes enriquecer el contenido de la función usando use.

```
<?php
$tienda = "frutería";

function () use ($tienda) {
    return "Estoy en la $tienda";
}
?>
```

Es posible emplear también una variable global dentro de una función con la palabra **global** o con el array **$GLOBALS**:

Si se asigna una referencia a una **variable declarada como global** dentro de una función, la referencia se verá sólo dentro de la función. Si se quiere que el resultado sea visible en el ámbito global también se utiliza la matriz **$GLOBALS**:

```
<?php

// --- Ejemplo con la palabra global
$x = 2;
$y = 20;

/**
 * Función devuelve suma de variables globales
 * @return {int}
 */
function suma():int
{
```

```
    global $x, $y;
    return $x + $y;
}

echo "<p>" . suma() . "</p>"; // Devuelve: 22
// --- Ejemplo con $GLOBALS

$x = 4;
$y = 21;

/**
 * Función devuelve suma de variables globales
 * @return {int}
 */
function otraSuma():int
{
    return $GLOBALS["x"] + $GLOBALS["y"];
}

echo "<p>" . otraSuma() . "</p>"; // Devuelve: 25

?>
```

```
<p>22</p>
<p>25</p>
```

Cuando se hace global a una variable, es lo mismo que referenciar el nombre de la misma al elemento **GLOBALS** en cuestión:

```
<?php
$x = 10;
/**
 * Función devuelve uno más
 * @return {int}
 */
function funcionUno():int
{
    global $x;
    $x++;
    return $x;
}
/**
 * Función devuelve uno más
 * @return {int}
 */
function funcionDos():int
```

```
{
    $x =& $GLOBALS["x"];
    $x++;
    return $x;
}
funcionUno();
funcionDos();
echo "<p>" . $x . "</p>"; // Devuelve: 12
?>
```

```
<p>12</p>
```

La diferencia entre utilizar la palabra global y el array $GLOBALS se produce cuando se hacen referencias dentro de las funciones:

```
<?php
$x = "Soy X";
$y = "Soy Y";
/**
 * Función devuelve uno cadena
 */
function funcionUno():
{
    global $x, $y;
    $y = &$x;
}

/**
 * Función devuelve una cadena
 */
function funcionDos(){
    global $x;
    $GLOBALS['y'] = &$x;
}

funcionUno();
echo "<p>" . $y . "</p>"; // Devuelve: Soy Y
funcionDos();
echo "<p>" . $y . "</p>"; // Devuelve: Soy X
?>
```

```
<p>Soy Y</p>
<p>Soy X</p>
```

En **funcionUno()** se han importado **$x** e **$y** con la palabra **global** y se ha referenciado **$y** a **$x**, pero esto ha ocurrido sólo dentro de la función. En **funcionDos()** se ha empleado **$GLOBALS** con **$y** y su valor ha permanecido referenciado después de la función.

Anónimas

Las funciones anónimas son funciones sin nombre a las que se accede mediante variables o mediante otras funciones

```
<?php
  function () {
    return 'Soy anónima';
  }
?>
```

En el siguiente ejemplo incrementamos en 1 cada número del array

```
<?php
 $numeros = [10, 20, 30, 40];
 $numerosIncrementados = array_map(function ($numero) {
    return $numero + 1;
 }, $numeros);

 echo "<pre>";
 var_dump($numerosIncrementados);
 echo "</pre>";
?>
```

```
Array(4) {
  [0]=>
  int(11)
  [1]=>
  int(21)
  [2]=>
  int(31)
  [3]=>
  int(41)
}
```

Bibliotecas

Las bibliotecas son archivos php que se pueden incluir en cualquier otro archivo php. Las bibliotecas se suelen utilizar para centralizar fragmentos de código que se utilizan en varias páginas. De esa manera, si se quiere hacer alguna modificación, no es necesario hacer el cambio en todos las páginas sino únicamente en la biblioteca.

Por ejemplo, si definimos en la biblioteca una función que imprima la cabecera de las páginas, desde cualquier página se puede incluir la biblioteca mediante la construcción **include** y llamar a la función como si se hubiera definido en la propia página:

- biblioteca.php

```
<?php
 /**
 * Función que imprime
 * @param {string} $titulos - Título
 */
  function cabecera(string $titulo)
  {
    echo "<!DOCTYPE html>";
    echo "<html lang=\"es\">";
    echo "<head>";
    echo " <meta charset=\"utf-8\">";
    echo " <title>$titulo</title>";
    echo " <meta name=\"viewport\" content=\"width=device-width,
initial-scale=1.0\">";
    echo " <link rel=\"stylesheet\" href=\"estilo.css\"
title=\"Color\">";
    echo " </head>";
    echo "<body>";
    echo " <h1>$titulo</h1>";
  }
?>
```

- pagina_1.php

```
<?php
  include "biblioteca.php";
  cabecera("Página de ejemplo");
  echo "<p>Esta página es válida</p>";
?>
</body>
</html>
```

```
<!DOCTYPE html>
<html lang="es">
<head>
 <meta charset="utf-8">
 <title>Página de ejemplo</title>
 <meta name="viewport" content="width=device-width,
initial-scale=1.0">
```

```
 <link rel="stylesheet" href="estilo.css" title="Color">
</head>
<body>
 <h1>Página de ejemplo</h1>
 <p>Esta página es válida</p>
</body>
</html>
```

- pagina_2.php

```
<?php
  include "biblioteca.php";
  cabecera("Otra página de ejemplo");
  echo "<p>Esta página también es válida</p>";
?>
</body>
</html>
```

```
<!DOCTYPE html>
<html lang="es">
<head>
 <meta charset="utf-8">
 <title>Otra página de ejemplo</title>
 <meta name="viewport" content="width=device-width,
initial-scale=1.0">
 <link rel="stylesheet" href="estilo.css" title="Color">
</head>
<body>
 <h1>Otra página de ejemplo</h1>
 <p>Esta página también es válida</p>
</body>
</html>
```

Se pueden crear todas las bibliotecas que se necesiten e incluir cualquier número de bibliotecas en cualquier punto de un programa. Las bibliotecas pueden a su vez incluir otras bibliotecas.

Normalmente, las bibliotecas suelen contener funciones, definiciones de constantes o inicialización de variables, pero en realidad pueden incluir cualquier tipo de código php, que se ejecutará en la posición en la que se incluya la biblioteca. También se puede incluir parte de la lógica del programa (generar salida, modificar variables, etc.), pero se recomienda que no se hagan las dos cosas en un mismo fichero.
En el ejemplo siguiente, las bibliotecas modifican variables, lo que afecta a su valor.

- biblioteca-1.php

```
<?php
```

```
  $i = 1;
?>
```

- biblioteca_2.php

```
<?php
  $i = $i + 10;
?>
```

- Programa:

```
<?php
  include "biblioteca_1.php";
  echo "<p>Ahora \$i vale $i</p>";
  include "biblioteca_2.php";
  echo "<p>Ahora \$i vale $i</p>";
  include "biblioteca_2.php";
  echo "<p>Ahora \$i vale $i</p>";
?>
</body>
</html>
```

```
<p>Ahora $i vale 1</p>
<p>Ahora $i vale 11</p>
<p>Ahora $i vale 21</p>
```

Existe una construcción similar a **include**, la construcción **require**. La diferencia con respecto a **include** es que **require** produce un error si no se encuentra el archivo (y no se procesa el resto de la página), mientras que **include** sólo produce un aviso (y se procesa el resto de la página).

En un mismo archivo php se pueden incluir varias construcciones **include** o **require**, pero si las bibliotecas incluidas contienen definiciones de funciones, al incluir de nuevo la definición de la función se genera un error.

Para que no ocurra este problema se pueden utilizar las funciones **include_once** o **require_once**, que también incluyen los ficheros pero que, en caso de que los ficheros ya se hayan incluido, entonces no los incluyen.

Las bibliotecas están habitualmente en el mismo servidor que los programas, pero podrían estar en otros servidores y acceder a ellas por HTTP, no como ficheros locales. La directiva de configuración **allow_url_include** permite acceder a bibliotecas por HTTP (suele estar desactivada).

Las cuatro construcciones (**include**, **require**, **include_once** y **require_once**) pueden utilizarse escribiendo entre paréntesis el nombre de los ficheros o sin escribir paréntesis.