Funciones y bibliotecas

Una función es un grupo de instrucciones, independiente del programa principal, que se puede reutilizar a lo largo de un programa. Las funciones nos ahorran tener que repetir el grupo de instrucciones, sustituyéndolos por simples llamadas a las funciones.

En esta lección se tratan primero las funciones sin argumentos y después las funciones con argumentos. Todo lo que se comenta sobre las funciones sin argumentos (forma de definirlas, nombre, etc.) se aplica también a las funciones con argumentos.

Funciones sin argumentos

Las funciones se identifican por su nombre. Las funciones se definen mediante la palabra reservada **function** seguida del nombre de la función y unos paréntesis, seguidos de un conjunto de instrucciones escritos entre llaves. La llave que abre el bloque de instrucciones se suele escribir al principio de la línea siguiente, no a continuación del nombre de la función.

```
function nombreDelaFuncion()
{
    bloque_de_sentencias
}
```

Las funciones se aprovechan en el resto del programa escribiendo el nombre de la función seguido de paréntesis. PHP ejecuta el bloque de instrucciones de la función cada vez que se llama a la función.

```
<?php
// Definición de la función prueba()
function prueba()
{
    print "<p>I will not waste chalk.</p>\n";
    print "\n";
      print  "<p>I  will  not  skateboard  in  the  halls.
</p>\n";
    print "\n";
    print "<p>I will not burp in class.</p>\n";
    print "\n";
```

```
<p>Programa de prueba.</p>

<p>I will not waste chalk.</p>

<p>I will not skateboard in the halls.</p>

<p>I will not burp in class.</p>

<p>I will not waste chalk.</p>

<p>I will not skateboard in the halls.</p>
```

```
}

print "<p>Programa de prueba.</p>\n";
print "\n";
// Llamadas a la función prueba()
prueba();
prueba();
?>
```

```
<p>I will not burp in class.</p>
```

- Si se llama a una función definida en el mismo fichero, la función puede estar definida antes o después de la llamada.

```
<?php
// Definición de la función prueba()
function prueba()
{
    print "<p>Hola!</p>\n";
}

// Llamada a la función prueba()
prueba();
?>
```

```
<p>Hola!</p>
```

```
<?php
// Llamada a la función prueba()
prueba();

// Definición de la función prueba()
function prueba()
{
  print "<p>Hola!</p>\n";
}
?>
```

```
<p>Hola!</p>
```

Los nombres de las funciones siguen las mismas reglas de los identificadores de PHP, es decir, deben comenzar por una letra o un guion 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.

La guía de estilo PEAR para PHP 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).

- Si no se llama a una función en el programa principal, las funciones no tienen ningún efecto en la salida del programa (excepto si contienen errores de sintaxis que impidan la ejecución del programa).

```php
<?php
// Definición de la función prueba()
function prueba()
{
    print "<p>I will not waste chalk.</p>\n";
    print "\n";
     print "<p>I will not skateboard in the halls.
</p>\n";
    print "\n";
    print "<p>I will not burp in class.</p>\n";
    print "\n";
}

print "<p>Programa de prueba.</p>\n";
?>
```

```
<p>Programa de prueba.</p>
```

- PHP no distingue ente mayúsculas y minúsculas en el nombre de las funciones.

```php
<?php
// Definición de la función prueba()
function prueba()
{
    print "<p>I will not waste chalk.</p>\n";
    print "\n";
}
```

```
<p>I will not waste chalk.</p>

<p>I will not waste chalk.</p>

<p>I will not waste chalk.</p>
```

```php
// Llamadas a la función prueba()
prueba();
PRUEBA();
PrUeBa();
?>
```

- En un programa no se pueden definir dos veces la misma función.

```php
<?php
function prueba()
{
    print "<p>Hola!</p>\n";
}

function Prueba()
{
    print "<p>Adios!</p>\n";
}
?>
```

Fatal error: Cannot redeclare Prueba() (previously declared in ejemplo.php:4) in **ejemplo.php** on line **7**

Independencia de las funciones respecto al programa principal

Las funciones son independientes del resto del programa. En particular, las variables que aparecen en una función son independientes de las variables del programa principal. En principio, ni la función tiene acceso a las variables del programa principal, ni el programa principal tiene acceso a las variables de la función.

- En el siguiente ejemplo, tanto el programa principal como la función utilizan una variable con el mismo nombre ($a), pero para el programa se trata de dos variables distintas.

```php
<?php
// Definición de la función prueba()
function prueba()
{
    $a = 42;
```

```
<p>La variable a es 100.</p>

<p>La variable a es 42.</p>

<p>La variable a es 100.</p>
```

```
    print "<p>La variable a es $a.</p>\n";
    print "\n";
}

// Damos un valor a la variable
$a = 100;
print "<p>La variable a es $a.</p>\n";
print "\n";
// Llamamos a la función, que da un valor distinto a una variable con el mismo nombre
prueba();
// Pero en el programa principal, la variable no ha cambiado su valor
print "<p>La variable a es $a.</p>\n";
?>
```

- En el siguiente ejemplo, la función genera un aviso ya que se hace referencia a una variable no definida. Esto se debe a que la función no da valor a la variable $a y la función no tiene acceso a la variable $a del programa principal. Otro detalle a señalar es que el aviso se genera cuando el programa principal llama a la función, es decir, cuando se ejecuta la función.

```
<?php
// Definición de la función prueba()
function prueba()
{
    // Intentamos escribir el valor de la variable $a
     // pero como no está definida, se produce un aviso
    print "<p>La variable a es $a.</p>\n";
    print "\n";
}

// Damos un valor a la variable
$a = 100;
```

```
<p>Programa de prueba.</p>

<p>La variable a es 100.</p>

<br />
<b>Warning</b>:  Undefined variable $a in <b>ejemplo.php</b> on line <b>5</b><br />
<p>La variable a es .</p>

<p>La variable a es 100.</p>
```

```php
print "<p>La variable a es $a.</p>\n";
print "\n";
// Llamamos a la función
prueba();
// Volvemos a escribir la variable
print "<p>La variable a es $a.</p>\n";
?>
```

- En el siguiente ejemplo, es el programa principal el que genera el aviso ya que se hace referencia a una variable no definida. Aunque hemos llamado a la función que da valor a una variable $a, el aviso se produce porque el programa no tiene acceso a la variable $a de la función.

```php
<?php
// Definición de la función prueba()
function prueba()
{
     // La variable $a sólo está definida en la función, no fuera de ella
    $a = 42;
    print "<p>La variable a es $a.</p>\n";
    print "\n";
}

// Llamamos a la función
prueba();
// Intentamos escribir el valor de la variable $a
// pero como no está definida, se produce un aviso
print "<p>La variable a es $a.</p>\n";
?>
```

```
<p>La variable a es 42.</p>

<br />
<b>Warning</b>:  Undefined variable $a in
<b>ejemplo.php</b> on line <b>13</b><br />
<p>La variable a es .</p>
```

Variables globales

Podemos hacer que una función tenga acceso a las variables del programa principal, indicando en el cuerpo de la función los nombres de las variables precedidas de la palabra reservada **global**.

- En el siguiente ejemplo, la función puede escribir el valor de la variable definida en el programa.

```php
<?php
function prueba()
{
    // Declaramos la variable $a como global
    global $a;

    print "<p>La variable a es $a.</p>\n";
    print "\n";
}

$a = 100;
prueba();
?>
```

```
<p>La variable a es 100.</p>
```

- El acceso a las variables globales es completo, es decir, las variables se pueden modificar, como muestra el siguiente ejemplo:

```php
<?php
function prueba()
{
    global $a;

    // Modificamos la variable $a en la función
    $a = 50;
}

$a = 100;
prueba();
print "<p>La variable a es $a.</p>\n";
?>
```

```
<p>La variable a es 50.</p>
```

Funciones con argumentos (valores)

Podemos enviar datos a una función incluyendo argumentos en su llamada. En la definición de la función deben indicarse los argumentos que se van a recibir, escribiéndolos entre los paréntesis como variables, separadas por comas. En el programa principal, los datos que se quieren enviar a la función simplemente se incluyen en la llamada a la función escribiéndolos entre los paréntesis, separados por comas. La función guarda los valores en los argumentos (como variables) en el orden indicado en la definición. Los argumentos se pueden utilizar o no en la función, pero si una función se define con argumentos, la llamada a la función debe incluir todos los argumentos indicados en la definición.

```
function nombreDelaFuncion($argumento_1, $argumento_2, etc ...) {
    bloque_de_sentencias
}
```

En este apartado se trata el caso más sencillo, es decir, cuando los argumentos son simplemente valores (número, cadenas, matrices, etc.). En los dos apartados siguientes se tratan los casos en los que los argumentos son variables del programa principal.

- En el ejemplo siguiente, enviamos una cadena a la función para que la incluya en la salida de la función.

```
<?php
function saludo($nombre)
{
    print "<p>Hola, $nombre!</p>\n";
}

saludo("Don Pepito");
?>
```

```
<p>Hola, Don Pepito!</p>
```

- En el ejemplo siguiente, enviamos una cadena y un número a la función.

```
<?php
function saludo($nombre, $veces)
{
    for ($i = 0; $i < $veces; $i++) {
        print "<p>Hola, $nombre!</p>\n";
        print "\n";
    }
}
```

```
<p>Hola, Don Pepito!</p>

<p>Hola, Don Pepito!</p>

<p>Hola, Don Pepito!</p>
```

```php
saludo("Don Pepito", 3);
?>
```

- El ejemplo siguiente produce un error porque en él se llama sin argumentos a una función que tiene definido un argumento.

```php
<?php
// Definición de la función prueba($x) con 1 argumento
function prueba($x)
{
    // Esta función no hace nada
}

// LLamamos a la función sin enviar el argumento
prueba();
?>
```

```
Fatal error: Uncaught ArgumentCountError: Too few arguments to function prueba(), 0 passed in ejemplo.php on line 9 and exactly 1 expected in ejemplo.php:3
Stack trace:
#0 ejemplo.php(9): prueba()
#1 {main}
  thrown in ejemplo.php on line 3
```

- PHP no produce error si se envía un argumento a una función que no tiene definidos argumentos, pero el argumento enviado no se utiliza para nada.

```php
<?php
function saludo()
{
    print "<p>Hola!</p>\n";
}

saludo("Don Pepito");
?>
```

```
<p>Hola</p>
```

Funciones con argumentos: paso por valor

Cuando llamamos a una función escribiendo como argumento una variable del programa principal, lo único que se envía es el valor de la variable. En programación está situación se suele llamar "paso por valor".

- En el siguiente ejemplo, la función puede escribir el valor de la variable definida en el programa. La función conoce el valor, pero no sabe qué variable tenía ese valor.

```php
<?php
// Definición de la función prueba()
function prueba($x)
{
    print "<p>La variable es $x.</p>\n";
    print "\n";
}

// Damos un valor a las variables
$a = 100;
$b = 50;
print "<p>La variable a es $a y la variable b es $b.
</p>\n";
print "\n";
// Llamamos a la función
prueba($a);
prueba($b);
// Volvemos a escribir las variables
print "<p>La variable a es $a y la variable b es $b.
</p>\n";
?>
```

```
<p>La variable a es 100 y la variable b es 50.</p>

<p>La variable es 100.</p>

<p>La variable es 50.</p>

<p>La variable a es 100 y la variable b es 50.</p>
```

- En un paso por valor, la función recibe el valor pero no puede modificar la variable del programa principal, como muestra el siguiente ejemplo.

```php
<?php
function prueba($x)
{
    $x = 200;
    print "<p>La variable es $x.</p>\n";
    print "\n";
```

```
<p>La variable a es 100.</p>

<p>La variable es 200.</p>

<p>La variable a es 100.</p>
```

```php
}

$a = 100;
print "<p>La variable a es $a.</p>\n";
print "\n";
prueba($a);
print "<p>La variable a es $a.</p>\n";
?>
```

- El nombre de un argumento puede coincidir con el nombre de una variable del programa principal, pero PHP las trata como variables distintas.

```php
<?php
function saludo($nombre)
{
    $nombre = "desconocido";
    print "<p>Hola, $nombre!</p>\n";
    print "\n";
}

$nombre = "Don Pepito";
saludo($nombre);
print "<p>Hola, $nombre!</p>\n";
?>
```

```
<p>Hola, desconocido!</p>

<p>Hola, Don Pepito!</p>
```

Funciones con argumentos: paso por referencia

Por completar. No se utiliza en los ejercicios propuestos en estos apuntes.

manual de PHP: Paso por referencia

Funciones que devuelven valores

De la misma manera que el programa principal puede enviar valores a una función, una función puede devolver uno o varios valores al programa principal. La palabra reservada **return** permite indicar la variable que se devuelve.

- El ejemplo siguiente muestra una función que devuelve la suma de los valores recibidos:

```php
<?php
function suma($arg1, $arg2)
{
    return $arg1 + $arg2;
}

$a = 20;
$b = 30;
$suma = suma($a, $b);
print "<p>$a + $b = $suma</p>\n";
print "\n";
print "<p>$a + $b = " . suma($a, $b) . "</p>\n";
?>
```

```
<p>20 + 30 = 50</p>

<p>20 + 30 = 50</p>
```

- La ejecución de una función se interrumpe cuando se ejecuta un **return**:

```php
<?php
function suma($arg1, $arg2)
{
    return $arg1 + $arg2;
    // Esta instrucción no se ejecuta nunca.
    print "<p>Hola</p>\n";
}

$a = 20;
$b = 30;
print "<p>$a + $b = " . suma($a, $b) . "</p>\n";
?>
```

```
<p>20 + 30 = 50</p>
```

- Una función puede devolver varios valores, devolviendo una matriz. Desde PHP 7.1 (publicado en diciembre de 2016) para recoger los valores devueltos se pueden utilizar corchetes. Técnicamente, esta operación se denomina desestructurar la matriz y en versiones anteriores se debía hacer con la construcción del lenguaje **list()**. En el ejemplo siguiente, la función devuelve la suma y el producto de los valores recibidos:

```php
<?php
function sumaProducto($arg1, $arg2)
{
    return [$arg1 + $arg2, $arg1 * $arg2];
}

$a = 20;
$b = 30;
[$suma, $producto] = sumaProducto($a, $b);
print "<p>$a + $b = $suma</p>\n";
print "\n";
print "<p>$a * $b = $producto</p>\n";
?>
```

```
<p>20 + 30 = 50</p>

<p>20 * 30 = 600</p>
```

Funciones con 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.

El ejemplo siguiente muestra una función que calcula diferentes tipos de media (aritmética, geométrica, armónica). Los argumentos de la función son los números cuya media se debe calcular y el tipo de media a calcular. En el ejemplo, el tipo de media predeterminado es la media aritmética.

```php
<?php
// ESTA ES LA DEFINICIÓN DE LA FUNCIÓN calculaMedia
function    calculaMedia($arg1,    $arg2,    $arg3    =
```

```
<p>La media geométrica de 12 y 16 es 6.9282032302755.
</p>
```

```
"aritmética")
{
    if ($arg3 == "aritmética") {
        $media = ($arg1 + $arg2) / 2;
    } elseif ($arg3 == "geométrica") {
        $media = sqrt($arg1 * $arg2) / 2;
    } elseif ($arg3 == "armónica") {
            $media = 2 * ($arg1 * $arg2) / ($arg1 +
$arg2);
    }
    return $media;
}

// ESTO SON EJEMPLOS DE USO DE LA FUNCIÓN calculaMedia
$dato1 = 12;
$dato2 = 16;

// EL TERCER ARGUMENTO INDICA EL TIPO DE MEDIA A
CALCULAR
$media = calculaMedia($dato1, $dato2, "geométrica");
print "<p>La media geométrica de $dato1 y $dato2 es
$media.</p>\n";
print "\n";

// AL NO PONER EL TERCER ARGUMENTO, DEVUELVE LA MEDIA
ARITMÉTICA
$media = calculaMedia($dato1, $dato2);
print "<p>La media aritmética de $dato1 y $dato2 es
$media.</p>\n";
?>
```

```
<p>La media aritmética de 12 y 16 es 14.</p>
```

Funciones con un número de argumentos indeterminados

Por completar. No se utiliza en los ejercicios propuestos en estos apuntes.

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

- pagina-1.php

```php
<?php
include "biblioteca.php";
cabecera("Página de ejemplo");
```

```html
<!DOCTYPE html>
<html lang="es">
<head>
```

```
print "  <p>Esta página es válida</p>";
?>
</body>
</html>
```

```
  <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");
print "  <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.

Si una función está definida en una biblioteca, se debe incluir la biblioteca antes de llamar a la función o se generará un error.

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.

Sobre este punto en concreto, la guía de estilo PSR-1 comenta que en cualquier fichero de una biblioteca se pueden definir funciones, constantes, clases, etc. o se puede incluir parte de la lógica del programa (generar salida, modificar variables, etc.), pero prescribe 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
<?php
$i = 1;
?>
```

- biblioteca-2.php

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

- Programa:

```php
<?php
include "biblioteca-1.php";
print "<p>Ahora \$i vale $i</p>\n";
include "biblioteca-2.php";
print "<p>Ahora \$i vale $i</p>\n";
include "biblioteca-2.php";
print "<p>Ahora \$i vale $i</p>\n";
?>
```

```
<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.

La guía de estilo del proyecto PEAR prescribe el uso de la construcción **require_once** en el caso de bibliotecas cuya inclusión no dependa de ninguna condición y que no se utilicen paréntesis alrededor de los nombres de las bibliotecas.

Última modificación de esta página: 23 de enero de 2022