Conceptos Básicos php IV. Funciones

Índice

1. Funciones

PHP es un lenguaje estructurado, y como tal dispone de funciones. Las funciones no son más que fragmentos de código, que podrían verse como pequeños programas, y que pueden ejecutarse posteriormente en resto del código. Gracias a esto permiten:

- Reutilizar código que se usa frecuentemente.
- Estructurar lógicamente el código de la aplicación para que se más comprensible.
- Separar el código en diferentes ficheros para poder compartirlo en diferentes páginas *PHP*.

Desde el siguiente enlace podéis acceder a la información que os proporciona Manual PHP

1.1 Funciones definidas por el usuario.

1.1.1 Declaración de una función

Es recomendable que antes de poder utilizar una función ésta debe ser escrita aunque hay en casos que no es necesario. Por lo tanto, el lugar en el que se suelen situar las funciones es al comienzo del archivo, de tal forma que estén disponibles a partir de ese momento. Situarlas al comienzo también ayuda a una estructuración más lógica del código. Las funciones se declaran, al igual que las variables, con un nombre. 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 guión bajo, seguido de cualquier número de letras, números
o guiones bajos.

Éste irá precedido de la palabra *function*, unos paréntesis (para albergar los datos de entrada, llamados argumentos o parámetros) y unas llaves que incluyen el cuerpo de la función. Dentro del cuerpo de la función podemos declarar variables, llamadas a otras funciones y demás sentencias.

```
function nombre_funcion (arg1, arg2, arg3, ...) {
      // bloque de código
return valor; //Opcional
}
```

5.1.2 Llamada a una función

Para utilizar una función se escribe su nombre, seguido de paréntesis y dentro de estos se escriben los datos que se quieren pasar a la función (o variables que contienen
dichos datos).

```
nombre_funcion (arg1, arg2, arg3, ...);
```

Si la función devuelve un dato, éste se puede asignar a una variable o usar directamente como parte de una expresión.

Ejemplo *A continuación se muestra una función muy sencilla, que no tiene argumentos.*

```
<?php
function hola_mundo() {

echo "¡Hola Mundo!";
}
hola_mundo();
?>
```

1.1.3 Devolución de una valor.

Las funciones son mucho más útiles si pueden devolver un dato. Para ello usan la sentencia ***return***. Una vez que se llega a esta instrucción no se ejecuta el código que se pueda encontrar a continuación.

Ejemplo. *La siguiente función devuelve aleatoriamente un día de la semana.*

```
<?php
    function dia_semana() {
    $semana = array("lunes", "martes", "miércoles","jueves", "viernes",
    "sabado", "domingo"); $dia =
    $semana[rand(0, 6)]; return
    $dia;
    }
```

```
$dia_cine = dia_semana();
echo "El próximo $dia_cine voy al cine.";
?>
```

Ejemplo. *La siguiente función devuelve un array.*

```
<?php
function números_pequeños()

{
return array(0,1,2);
}
//La función list hace una asignación múltiple

list($cero,$uno,$dos) = números_pequeños(); echo
$cero,$uno,$dos;

?>
```

1.1.4 Declaración de tipos. Type Hinting

- Las funciones obligan a que los parámetros sean de cierto tipo
- Si el valor dado es de un tipo incorrecto, se generará un error
- 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
- A no ser que lo declaremos de forma estricta PHP intentará adaptar los tipos para evitar el error.

Tipos válidos

Tipo	Descripción	Versión de PHP mínima
nombre de clase/interfaz	El parámetro debe ser una *instanceof* del nombre de la clase o interfaz dada.	PHP 5.0.0
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
array	El parámetro debe ser un array.	PHP 5.1.0
callable	El parámetro debe ser un callable válido.	PHP 5.4.0
bool	El parámetro debe ser un valor de tipo boolean.	PHP 7.0.0
float	El parámetro debe ser un número de tipo float.	PHP 7.0.0
int	El parámetro debe ser un valor de tipo integer.	PHP 7.0.0
string	El parámetro debe ser un string.	PHP 7.0.0

Ejemplos

```
function suma(int $a, int $b)
{
    return $a + $b;
}
$resultado = suma(5, 3);
```

Declaración de tipo de devolución

- Añadido en PHP 7
- Especifican el tipo del valor que serán devuelto desde una función
- Están disponibles los mismos tipos de la tabla anterior

Ejemplo

```
function suma(int $a, int $b) : int
 {
        return $a + $b;
}
$resultado = suma('5', 3);
```

Tipificación estricta

PHP fuerza a los valores de un tipo erróneo a ser del tipo escalar esperado si es posible

Ejemplo:

  - Una función que espera un string y recibe un int obtendrá una variable de tipo string

- En el modo estricto esto no se permite
- Si los tipos no coinciden nos dará error
- Excepción: se puede pasar un int a una función que espere un float
- El modo estricto se habilita para cada fichero php • Habilitar el modo estricto **declare**(strict_types=1);

Ejemplo

```
declare(strict_types=1);  function suma(int $a, int $b) : int
{
        return $a + $b;
}
$resultado = suma('5',3);
```

1.1.4 Argumentos.

Aún así, estas funciones que hemos visto son muy sencillas y no permiten hacer gran cosa. Lo más habitual es que a las funciones se les pasen datos, para que luego operen con éstos, y que al terminar la función devuelva el resultado. A los datos que recibe una función se les llama argumentos o parámetros.

En la declaración de la función, tras la palabra clave ***function***, va el nombre de la función seguido por una lista de argumentos entre paréntesis y separados por comas. Dentro del cuerpo de la función estos datos se pueden utilizar como una variable cualquiera.
PHP admite el paso de argumentos por valor (lo predeterminado), el paso por referencia, y valores de argumentos predeterminados. Las Listas de argumentos de longitud variable también están soportadas.

Ejemplo *Función sencilla que recibe un parámetro y devuelve un resultado. Luego es llamada dentro de un bucle, para mostrar el cuadrado de los números del 1 al 10.*

```
<?php
function cuadrado($numero) {
return $numero * $numero;
}
for ($i = 1; $i <= 10; $i++) {
echo "$i al cuadrado es igual a " . cuadrado($i) . ".<br>";
}
?>
```

1.1.5 Paso de argumentos por valor.

Cuando se pasan valores a las funciones, podemos pasarlos de 2 formas distintas: por valor y por referencia. El comportamiento predefinido en *PHP* se conoce como paso por valor.

En el paso de **argumentos por valor**, la variable que recibe el valor hace una copia del mismo, convirtiéndose en una nueva variable, y por tanto actuando a partir de ese momento como una variable totalmente independiente.

El aspecto práctico más importante consiste en que si cambiamos en algún momento el valor del argumento dentro de la función (copia), no se ve modificado fuera de ella (original).

1.1.6 Paso de argumentos por referencia.

En ocasiones es preferible no hacer la copia del dato que se pasa, sino trabajar directamente sobre el dato original.

Esto, por ejemplo, puede interesar cuando se trabaje con arrays, ya que el copiado de muchos datos puede perjudicar al tiempo de ejecución del programa, o más comúnmente cuando se quiere alterar el valor de la variable para obtener algún efecto.

A esto se le llama paso de parámetros por referencia. Para indicar que un parámetro se pasa por referencia se antepone el símbolo ampersand, "&", en la declaración de la función.

Una misma función puede emplear argumentos pasados por valor y por referencia, en una misma declaración.

Ejemplo *Ejecutando el siguiente código podemos observar los diferentes resultados obtenidos entre utilizar la misma function empleando un paso de parámetros por valor o por referencia. Como es de esperar, mediante el paso por referencia la variable exterior se verá afectada por los cambios ocurridos en el interior de la función.*

```
<?php
function duplicar_por_valor($argumento) {
$argumento = $argumento * 2;
echo "Dentro de la función vale $argumento.<br>";
}

function duplicar_por_referencia(&$argumento) {
$argumento = $argumento * 2;
echo "Dentro de la función vale $argumento.<br>";
}
$numero1 = 5;
```

```
echo "Antes de llamar a la función vale $numero1.<br>";
duplicar_por_valor($numero1);
echo "Después de llamar a la función vale $numero1.<br>"; echo
"<br>"; $numero2 = 7;
echo "Antes de llamar a la función vale $numero2.<br>";
duplicar_por_referencia($numero2);
echo "Después de llamar a la función vale $numero2.<br>";
?>
```

Ejemplo. *Aunque el lenguaje no nos permite hacer que una función devuelva dos valores, tenemos la posibilidad de pasar por referencia las variables a devolver y, así, modificar su contenido en el cuerpo de la función.*
Aquí se emplea esta estratagema para intercambiar el contenido de dos variables.

```
<?php
function intercambiar(&$argumento1, &$argumento2) {
$auxiliar = $argumento1;
$argumento1 = $argumento2;
$argumento2 = $auxiliar;
}
$numero1 = 5;
$numero2 = 8; echo "Antes: ($numero1,
$numero2)<br>"; intercambiar($numero1,
$numero2); echo "Despues: ($numero1,
$numero2)";
?>
```

1.1.7. Uso de parámetros predeterminados en funciones

Una función puede definir valores predeterminados pero hay que tener en cuenta que deben aparecen en la parte derecha de la lista de argumentos porque la asignación se realiza de izquierda a derecha.

Ejemplo *Función que muestra como definir una función con un argumento con valor predeterminado.*

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

Ejemplo *Función con dos parámetros con valor predeterminado. Uso condicional de uno de ellos.*

```
function hacer_café($tipos = array("capuchino"), $fabricanteCafé = NULL)
{
  $aparato = is_null($fabricanteCafé) ? "las manos" : $fabricanteCafé;    return "Hacer una
taza de ".join(", ", $tipos)." con $aparato.\n";
}
echo hacer_café()."</br>"; echo hacer_café(array("capuchino",
"lavazza"), "una tetera");
```

Argumentos con nombre (Nuevo en PHP 8)

Desde PHP 8.0 podemos pasar los argumentos con el nombre (además de por posición, como hemos hecho hasta ahora). Los argumentos con nombre se pasan poniendo el nombre como prefijo del parámetros separado por dos puntos: $resultado = funcion(arg1 : valor1, arg2 : valor2);

Esta característica complementa los parametros opcionales permitiendonos saltar su valor:

```
<?php
function funcionArgumentosNombre($a, $b = 2, $c = 4) {
  echo "$a $b $c";
}
funcionArgumentosNombre(c: 3, a: 1); // "1 2 3"
```

Tanto los parámetros opcionales como los obligatorios pueden tener nombre, pero los argumentos con nombre se tienen que poner después de los que no lo tienen.

```
<?php
funcionArgumentosNombre(1, c: 3); // "1 2 3"
```

1.1.8. Uso listas de argumentos de longitud variable

En PHP 5.6 y posteriores, las listas de argumentos pueden incluir el token … para denotar que la función acepta un número variable de argumentos. Los argumentos serán pasados a la variable dada como un array, por ejemplo:

```
 <?php
function sum(...$números){
     $acc=0;
       foreach($números as $n) {
    $acc+=$n;
    }
     return $acc;
}

echo sum(1,2,3,4);
?>
```

1.1.9 Alcance de las variables.

Dentro de las funciones también podemos declarar nuevas variables, pero, ¿que pasa si hay una variable dentro de una función que se llama igual que una variable externa?
La respuesta es que la variable interna es diferente de la que está fuera, y por lo tanto su valor será totalmente independiente.

A esto se le llama *alcance de una variable*, y tiene las siguientes implicaciones:

- Las variables que se declaran dentro de una función sólo existen el cuerpo de dicha función.

- Las variables creadas fuera de las funciones son globales a la página, son accesibles en cualquier momento, independientemente del punto del código en que nos encontremos. Para acceder a ellas tenemos que anteponer la palabra **global** o utilizar la matriz **$GLOBALS**.

- Las variables locales, que son las declaradas dentro de una función, tienen preferencia sobre las variables globales.

Ejemplo. Este código muestra el alcance de una variable dentro de una función.

```
<?php function
mi_ciudad() { $ciudad =
"Madrid";
echo "Dentro de la función vale $ciudad.<br>";
}
$ciudad = "Barcelona";
echo "Antes de llamar a la función vale $ciudad.<br>"; mi_ciudad();
echo "Después de llamar a la función vale $ciudad.<br>" ?>
```

Ejemplo. Puede darse el caso que queramos acceder a una variable global dentro del cuerpo de la función. Para conseguirlo le antepondremos la palabra clave global a la primera referencia de la variable, con lo que el interprete PHP sabe que estamos llamando a la variable externa.

```
<?php function mi_ciudad() {
global $ciudad; $ciudad =
"Madrid";
echo "Dentro de la función vale $ciudad.<br>";
}
$ciudad = "Barcelona";
echo "Antes de llamar a la función vale $ciudad.<br>"; mi_ciudad();
echo "Después de llamar a la función vale $ciudad.<br>" ?>
```

Ejemplo El mismo ejemplo anterior pero utilizando la matriz $GLOBALS

```
<?php function
mi_ciudad() {
$GLOBALS ['ciudad'];
$GLOBALS ['ciudad']= "Madrid";
echo "Dentro de la función vale $GLOBALS ['ciudad'.<br>";
}
$ciudad = "Barcelona";
echo "Antes de llamar a la función vale $ciudad.<br>"; mi_ciudad();
echo "Después de llamar a la función vale $ciudad.<br>" ?>
```

No obstante, es recomendable usar en las funciones nombres de variables diferentes a los de las variables del programa principal.

Tampoco es conveniente usar variables globales dentro de las funciones. Es mejor pasar estas variables como parámetro, ya que mejora notablemente la fiabilidad y claridad del código.

1.2 Creación de bibliotecas.

Para conseguir un código lo más claro posible, es deseable que éste sea breve. Una forma de conseguirlo es extraer las funciones que se declaran a un archivo independiente con extensión

"*.php*". Una ventaja de esta estrategia es que se pueden hacer accesibles estas funciones a más de una página en *PHP*.

Ejemplo. A continuación se muestra una biblioteca de funciones, guardada en el archivo "utils.php", en la que se ha optado por comentarios, para mejorar la legibilidad y mantenibilidad del código.

```
<?php
// Biblioteca de funciones de usuario: utils.php //
Devuelve el argumento elevado al cuadrado function
cuadrado($numero) {
return $numero * $numero;
}
// Devuelve la raíz cuadrada del argumento function
raiz($numero) {
return sqrt($numero);
}
// Devuelve verdadero si el número es igual o mayor que cero function
es_positivo($numero) { return ($numero >= 0);
}
?>
```

Ejemplo En los ficheros de biblioteca se puede incluir cualquier tipo de código.
También es posible crear y encontrarse archivos de configuración o de constantes predefinidas.
En este caso se guarda en el archivo "config.php".

```
<?php
// Fichero de configuración
// Archivo config.php
DEFINE(PI, 3.1416);
DEFINE(NUMERO_E, 2.7183); DEFINE(EULER, 0.5772);
?>
```

1.2.1 Include() Y Require().

Para tener disponibles las funciones de un fichero externo hay que indicarle al código *PHP* que debe incorporarlas al script actual.

Mediante estas instrucciones se incluye el contenido del fichero importado en el punto exacto en el que realiza la importación.

La diferencia entre las funciones de importación disponibles reside en que *require()* lanza un error fatal en el caso de no encontrar el fichero a importar, mientras que *include()* no lo hace.

Ejemplo. La inclusión de bibliotecas se suele poner al principio del código para que estén disponibles en el resto del código.

```
<?php
        include("config.php"); include("utils.php");
$radio = 5;
$circunferencia = 2 * $radio * PI; $area = PI *
cuadrado($radio);
echo "Un círculo de rádio $radio tiene circunferencia "; echo
"$circunferencia y área $area.<br>";
$area = -8; if (es_positivo($area)) { $radio = raiz($area / PI);
echo "Un círculo de área $area tiene un rádio $radio";
} else {
```

```
echo "No se puede calcular, área negativa.";
}
?>
```

Hay que tener en cuenta el orden en el que se cargan los archivos externos, especialmente si unos hacen uso del código de otros.

1.2.2 Plantillas mediante include¶

Mediante el uso de la instrucción include también podemos separar fragmentos de código PHP/HTML que queramos reutilizar en nuestros sitios web y crear un sistema muy sencillo de plantillas. Por ejemplo, vamos a separar una página en tres partes, primero la parte superior en **`encabezado.php`**:

```
<!DOCTYPE html>
<html lang="es">
<head>
<meta charset="UTF-8">
<meta name="viewport" content="width=device-width, initial scale=1.0">
<title><?= $titulo ?></title>
</head>
<body>
```

La parte de abajo, por ejemplo, solo va a contener HTML y la colocamos en **`pie.html`**:

```
<footer>Página de ejemplo</footer>
</body>
</html>
```

Y luego nos centramos únicamente en el contenido que cambia en **`pagina.php`**:

```
<?php
$titulo = "Página con includes";
include("encabezado.php");
?>
<h1><?= $titulo ?></h1>
<?php
include("pie.html");
?>
```