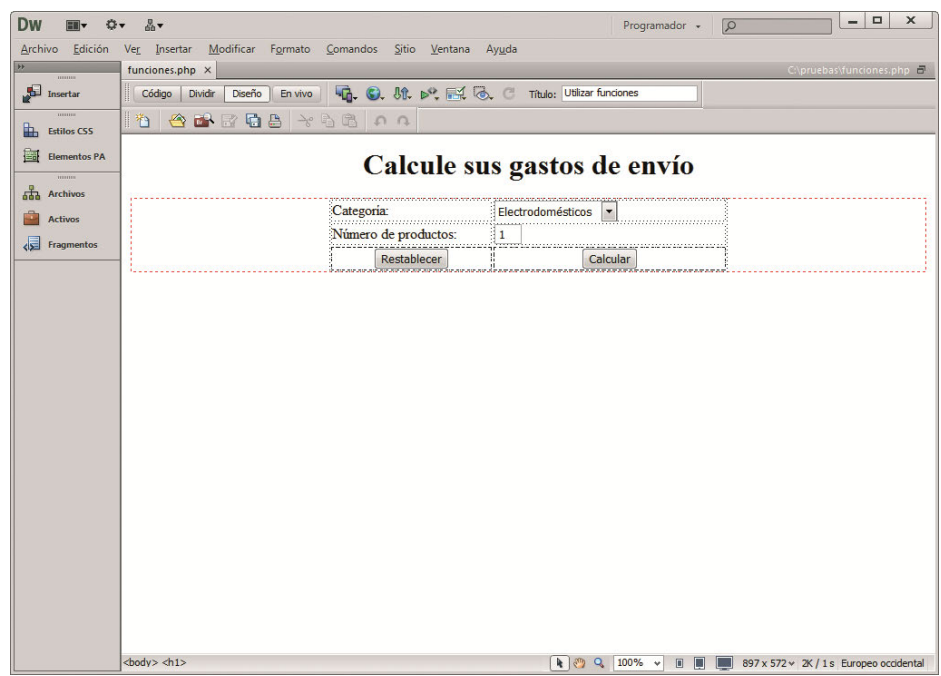


1. INTRODUCCIÓN

Cuando un proyecto tiene cierta magnitud, se hace muy difícil poder desarrollarlo sin dividirlo en trozos más pequeños. Las **funciones** (junto a las clases, que más tarde estudiaremos) son la herramienta que proporciona PHP para dividir la aplicación en unidades más pequeñas y fáciles de manejar.

Desde el punto de vista de la programación, una **función** no es más que un bloque de código bien delimitado al que le damos un nombre y que realiza alguna tarea bien definida. Cuando en nuestro programa necesitamos llevar a cabo esa tarea, llamamos a la función.

Veamos un ejemplo donde todo esto queda más claro. Fíjate en la página de la figura siguiente.

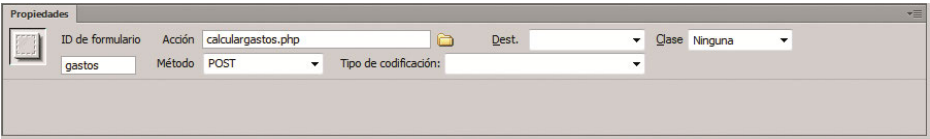


Se trata de un típico formulario que permite calcular los gastos de envío de los productos que ha elegido el usuario.

Así, se indica a qué categoría pertenece el producto comprado y cuántas unidades serán enviadas. A partir de estos datos, se calcula el importe total en cuanto a gastos de envío.

Más tarde estudiaremos detenidamente los formularios y cómo acceder a los datos que introduce el usuario en ellos. Por ahora, lo único que me interesa es que sepas que entre las propiedades de este formulario indicamos en **Acción** qué página o archivo es el encargado de procesar esa información.

En este caso, vemos que la información suministrada por el usuario será procesada por otra página llamada **calculargastos.php**. Pues en esta última página es donde definiremos la función para calcular los gastos del envío.



2. CREAR FUNCIONES

Se utiliza la palabra **function** para definir o crear una función. En este caso estaremos indicando qué hace dicha función.

Tras la palabra **function** escribiremos el nombre de la función y unos paréntesis. En el interior de esos paréntesis podemos escribir parámetros.

Los **parámetros** sirven para comunicar la función y el código que la utiliza. Por ejemplo, en nuestro caso, la función debe saber a qué categoría pertenece el producto comprado y cuántas unidades se tienen que enviar. Pues bien, eso se lo comunicaremos mediante parámetros.

```
function gastos($categoria, $unidades)
{
    switch ($categoria)
    {
        case 1:
            $recargo = 1.85;
            break;
        case 2:
            $recargo = 3.05;
            break;
        case 3:
            $recargo = 6.01;
            break;
        default:
            $recargo = 0;
    }
    return ($recargo + ($unidades * 2.25));
}
```



A la hora de utilizar la función, se hace indistinta la combinación de mayúsculas y minúsculas que utilices. Esto no es así con las variables.

Vemos que la función utilizará dos parámetros. Fíjate que el nombre de la función no incluye el signo del dólar, pero sí los parámetros porque actuarán como variables en el interior de la función.

En el cuerpo de la función se establece el recargo que se aplica teniendo en cuenta la categoría del producto que ha comprado el cliente. Recuerda que este detalle lo elige el usuario en el formulario inicial.

Además, la función debe devolver el cálculo efectuado. Esto se consigue con la palabra **return**.

```
return ($recargo + ($unidades * 2.25));
```

Con la palabra **return**, la función devolverá la suma del recargo aplicado más 2.25 unidades monetarias por cada unidad comprada del producto.

La palabra **return** hace que la función finalice inmediatamente, devolviendo el control a la siguiente línea del código donde se ha utilizado. En este caso es la última línea de la función, pero si no fuese así, el resto no llegaría a ejecutarse.

Vemos, por lo tanto, que la función **gastos** realiza una tarea bien definida, devolviendo el valor buscado y nada más.

Sin embargo, definir una función no hace que su código se ejecute. Para eso, es necesario **llamarla**.

3. LLAMAR A UNA FUNCIÓN

Como se ha comentado, el hecho de que escribamos la definición de la función, incluyendo las instrucciones que la componen, no produce ningún resultado en la página web.

Es necesario utilizar esa función en nuestro código para que se ejecute lo que hemos preparado para ello. Esto se conoce como "**llamar a la función**".

Para llamar a una función, simplemente tenemos que escribir su nombre. Eso sí, si la función devuelve un valor, deberíamos recoger dicho valor para utilizarlo en algún sitio.

Por ejemplo, vamos a imprimir en la página web el resultado que devuelve nuestra función:

```
echo gastos($arg_categoria, $arg_unidades);
```

Si en la definición indicábamos que entre los paréntesis se incluían los posibles parámetros que necesita la función, durante la llamada pasa lo mismo.




No olvides incluir los paréntesis al definir y llamar a una función aunque no se utilicen parámetros. Es obligatorio hacerlo.

Pero en este caso estarás dando valor a los parámetros que necesita la función, por lo que se conocen como **argumentos** de la función. Durante la definición se llaman **parámetros** y durante la llamada, **argumentos**.

Por ejemplo, necesitamos pasar la categoría y el número de unidades compradas. Pues utilizaremos dos variables como argumentos y después veremos de dónde sacamos sus valores: **\$arg_categoria** y **\$arg_unidades**.

Es importante el orden en que se escriben los argumentos, ya que serán emparejados según ese orden con sus correspondientes parámetros en la función. Es decir, que el valor de la variable **\$arg_categoria** será el que recoja el parámetro **\$categoria** de la función y **\$arg_unidades** dará valor al parámetro **\$unidades** de la función.

Con esto estaremos llamando a nuestra función para calcular e imprimir el importe de los gastos de envío.



A partir de PHP 4 no es necesario que la definición de una función aparezca antes que su llamada, aunque es mejor hacerlo así por motivos de claridad.

Solo nos falta saber qué valor tendrán estos argumentos. Como se ha indicado, vendrán dados por lo que introduzca el usuario en el formulario que vimos al principio de la lección:

```
$arg_categoria = $_POST["Categorias"];  
$arg_unidades = $_POST["Bultos"];
```

¿De dónde salen estos valores? Por ahora no nos interesa conocer cómo se pasan los valores introducidos mediante formularios, pero entiende que con la expresión **\$_POST["Categorias"]** estarás accediendo a la categoría que elija el usuario en el formulario inicial, y lo mismo respecto al número de unidades que allí indique estaría almacenado en **\$_POST["Bultos"]**.

Por lo tanto, una vez tenemos los valores que ha indicado el usuario en el formulario, se los pasamos a la función para que calcule el importe y finalmente lo imprimimos en esta página web.

En la figura siguiente se puede ver el resultado para un caso específico. La página ha recogido los detalles introducidos en el formulario, que han sido pasados a la función **gastos** para obtener el resultado.

Las ventajas de utilizar funciones son muchas. Imagínate que necesita calcular los gastos de envío en otro lugar de su sitio web.

Será muy fácil hacerlo simplemente llamando a la función. Si después decides cambiar el recargo por modalidad o por las unidades enviadas, solo tendrás que hacerlo en la definición de la función y todos los cálculos se actualizarán automáticamente.



4. PASO DE PARÁMETROS

Sabemos que los parámetros permiten comunicar el código que llama a la función con la propia función. Así, hemos visto que al llamar a la función **gastos**, le "*pasábamos*" la categoría y el número de productos indicados por el usuario.

Los parámetros también pueden servir para lo contrario, es decir, para hacer que la función proporcione resultados al código que la ha llamado. Esto puede ser necesario cuando no es suficiente con que devuelva un único valor mediante la palabra **return**.

Vamos a añadir un nuevo parámetro a la función donde almacenar el cálculo realizado. En lugar de devolver ese valor con **return**, utilizaremos dicho parámetro.

```
function gastos($categoria, $unidades, $importe)
{
    switch ($categoria)
    {
        case 1:
            $recargo = 1.85;
            break;
        case 2:
            $recargo = 3.05;
            break;
        case 3:
            $recargo = 6.01;
            break;
        default:
            $recargo = 0;
    }
    $importe = ($recargo + ($unidades * 2.25));
}
```

Ahora tendremos que modificar también la llamada a la función:

```
gastos($arg_categoria, $arg_unidades, $arg_importe);
echo $arg_importe;
```

Ahora ya no tiene sentido que imprimamos en pantalla el resultado devuelto por la función, ya que no devuelve ningún valor con **return** sino que hacemos lo propio con la variable que sirve de argumento **\$arg_importe**, que es donde está almacenado el cálculo.

Sin embargo, si mostramos el resultado en la página, veremos que no se imprime el cálculo de los gastos de envío ¿Por qué? El código es correcto, pero le falta un pequeño detalle que pasamos a explicar.

Por defecto, el **paso de parámetros** en PHP es **por valor**, lo que quiere decir que la función trabaja con una copia de los argumentos pasados en la llamada y no con las variables realmente utilizadas.

Esto quiere decir que, aunque en el interior de la función hemos almacenado el valor calculado en el parámetro **\$importe**, realmente tras ejecutarla, el argumento utilizado no ha sufrido ninguna modificación porque se ha estado trabajando con una copia.

Eso hace que al finalizar la función, **\$arg_importe** mantenga el mismo valor que tenía antes de ejecutarla.



Es frecuente utilizar el paso por referencia cuando el parámetro es un array, sobre todo si tiene muchos elementos. De esta forma, no se crea una copia en memoria del array, sino que se trabaja directamente con él.

Hay otra forma de pasar los parámetros: **por referencia**. En este caso, la función trabaja realmente con la variable que sirve de argumento en la llamada a la función y no con una copia, lo que permite modificar su valor.

Para ello, es necesario indicarlo en la definición de la función añadiendo un signo ampersand **&** delante del parámetro:

```
function gastos($categoria, $unidades, &$importe)
```

Ahora sí que obtendremos el resultado esperado. Es verdad que en esta ocasión era más apropiado utilizar la palabra **return** para que el cálculo lo devolviera la función y no utilizar un parámetro más, pero ahora ya sabes que puedes utilizar parámetros que se puedan modificar pasándolos por referencia.

Finalmente, si necesitas devolver más de un valor tras ejecutar una función, puedes hacerlo devolviendo un **array**, ya sea mediante la palabra **return** o con un parámetro más de la función.

5. PARÁMETROS POR DEFECTO

Cuando realizas una llamada a una función, tienes que proporcionar el mismo número de argumentos y en el mismo orden en que aparecen en su definición.

Sin embargo, PHP permite crear parámetros **opcionales** o para los que se indica un **valor por defecto**.

La idea es que si nosotros no especificamos el valor de dicho parámetro en la llamada a la función, entonces tomará el valor que tenga por defecto, por lo que la llamada seguirá siendo válida.

Para crear un parámetro opcional debemos escribir su nombre y seguidamente un igual más el valor que tomará por defecto. Además, los parámetros de este tipo tienen que aparecer después de todos los parámetros de la función que no sean opcionales.

```
function gastos($categoria, $unidades, &$importe,
               $urgente = FALSE)
```

El valor por defecto tiene que ser una expresión constante y no una variable o llamada a una función.

¿Qué estamos indicando con este parámetro? Muy sencillo, hemos añadido un servicio urgente o rápido de envío de los productos y queremos que se tenga en cuenta a la hora de calcular el importe total.

Por defecto, asignamos el valor **FALSE** a dicho parámetro, por lo que si no se indica en la llamada, se utilizará ese valor.

```
function gastos($categoria, $unidades, &$importe,
               $urgente = FALSE)
{
    switch ($categoria)
    {
        case 1:
            $recargo = 1.85;
            break;
        case 2:
            $recargo = 3.05;
            break;
        case 3:
            $recargo = 6.01;
            break;
        default:
            $recargo = 0;
    }
    if ($urgente)
        $recargo = $recargo + 50;
    $importe = ($recargo + ($unidades * 2.25));
}
```

Con el nuevo parámetro representamos que se añadirán 50 unidades monetarias al total de los gastos de envío si el usuario ha elegido el envío urgente.

Establecer un valor por defecto para un parámetro evita la obligación de tener que proporcionarlo en la llamada a la función, pero nada más. Podrás dar el valor adecuado (incluso coincidiendo con el establecido por defecto) siempre que quieras y ese será el valor que realmente se utilice en el interior de la función.