

Bloque A

Instrucciones básicas de programación

Unidad 3. Funciones



Contenidos

1. Introducción
2. Definición y llamada de funciones
3. Secuencialidad y retorno en funciones
4. Uso de parámetros y argumentos
5. Nombrado y ámbito de las funciones
6. Funciones y tipos de datos compuestos
7. Declaración y uso de tipos en funciones

Contenidos (II)

- 8. Retornos múltiples y valores por defecto
- 9. Argumentos con nombre
- 10. Diseño y documentación de funciones
- 11. Ejemplo final
- 12. Resumen de la unidad
- 13. Referencias



Introducción

Introducción

Una sola página web suele realizar **muchas tareas**.

Las funciones organizan el código agrupando las sentencias necesarias para realizar una tarea individual.

Una página PHP puede contener cientos de líneas de código y realizar varias tareas distintas. Por lo tanto, **es importante que el código esté cuidadosamente organizado** para que nosotros mismos (y otros) puedan entender fácilmente lo que está haciendo.

Introducción

En la unidad anterior, vimos que los **bloques de código** ayudan a organizar nuestro código colocando un conjunto de sentencias relacionadas entre llaves.

Estas llaves le indican al intérprete de PHP donde comienzan y terminan las sentencias relacionadas. Esto significa que un bloque de código puede ser **omitido** (utilizando sentencias condicionales) o **repetido** (utilizando bucles).

Introducción

Una función agrupa todas las sentencias necesarias para realizar una tarea individual dentro de un bloque de código.

También proporciona al bloque de código un **nombre de función** que describe la tarea que realiza (lo que ayuda a encontrar el código que realiza una tarea individual).

La **llave de apertura** ({) indica al intérprete de PHP dónde comienzan las sentencias para realizar la tarea, y la **llave de cierre** (}) correspondiente le indica dónde terminan.

Introducción

Cuando el intérprete PHP encuentra una función, no ejecuta el código inmediatamente. Espera hasta que otra sentencia en la página PHP **llame a la función** utilizando su nombre para decir que la tarea necesita ser realizada; sólo entonces ejecutará las sentencias en el bloque de código.

También podemos indicarle al intérprete PHP que utilice la función varias veces, para ahorrar repetir las mismas líneas de código cuando necesitemos realizar la tarea más de una vez.



Definición y llamada de funciones

Usando funciones

Cada tarea que realiza una página PHP puede requerir muchas sentencias PHP.

Las sentencias necesarias para realizar una tarea individual se pueden almacenar dentro de una función, y luego ser llamadas cuando sea necesario.

Definiendo y Llamando a una función

Una función se crea dándole un nombre que describe la tarea que realiza. A esto le siguen las sentencias necesarias para realizar la tarea en un bloque de código. Los programadores denominamos a esto la **definición de la función**.

Cuando la página necesita realizar la tarea, el nombre de la función se utiliza para decirle al intérprete de PHP que ejecute las sentencias en ese bloque de código. Los programadores nos referimos a esto como **Llamar a la función**.

Usando funciones

Obtener datos de las funciones

Cuando una función ha realizado su tarea, normalmente **devuelve** un valor para indicar el resultado de la tarea que ha realizado. Por ejemplo:

- Si se utiliza una función para registrar a un usuario en el sitio, la función podría devolver *verdadero* cuando un usuario se registra con éxito y *falso* cuando no lo hace.
- Si se utiliza una función para calcular el coste total de un pedido, devolverá ese total.

Usando funciones

Definición de funciones que necesitan datos

A menudo, las funciones necesitan información para realizar su tarea.

Si la tarea es registrar a un usuario en el sitio, la función puede necesitar dos datos: la dirección de correo electrónico del usuario y su contraseña.

Los **parámetros** son como nombres de variables que representan cada dato que la función necesita para realizar su tarea. Los valores reales utilizados cuando se llama a la función se conocen como **argumentos**.

Usando funciones

Las funciones que conocerás en esta unidad realizan tareas muy sencillas con el fin de explicar cómo crear una función y por qué se utilizan. En unidades posteriores, las funciones se utilizarán para realizar tareas más complicadas.

Cómo funcionan las variables con las funciones

El código dentro de una función no puede acceder a las variables que se declararon fuera de la función; por lo tanto, cualquier dato que la función necesite debe pasarse a la función utilizando un parámetro.

Del mismo modo, las variables declaradas en una función no pueden ser accedidas por el código fuera de la función. Esta es la razón por la que una función se diseña a menudo para devolver un valor al código que la llamó (después de que la función se haya ejecutado).

Usando funciones

Especificar tipos de datos

Las **declaraciones de tipo** le indican al intérprete de PHP el tipo de datos que se espera que sea

- Pasado a una función (como argumento)
- Devueltos por una función

Usar declaraciones de tipos ayuda a asegurar que la función recibe datos que puede utilizar para realizar su tarea. También ayudan a localizar problemas en el código cuando no funciona como se esperaba.

Usando funciones

Valoresopcionales y por defecto

Cuando se crea una función, se puede especificar que uno o varios de los parámetros (elementos de información que necesita para realizar su tarea) son **opcionales** y no es necesario especificarlos.

Cuando indicamos que un parámetro es opcional, debemos proporcionar un **valor predeterminado para el parámetro**. Se trata de un valor que el script debe utilizar cuando se llama a la función sin un valor para ese parámetro.

Definiendo y llamando a una función

Una **definición** de función almacena las sentencias que realizan una tarea entre llaves para formar un bloque de código, y les da un nombre para describir la tarea. A continuación, se **llama** a la función cuando es necesario realizar la tarea.

Para **definir (o crear) una función**, se utiliza:

- La palabra clave `function` (esto indica que está definiendo una nueva función)
- Un **nombre** que describa la tarea que realiza esta función seguido de un par de paréntesis
- **Llaves para contener el código** que realiza la tarea

No hay punto y coma después de la llave de cierre.

Definiendo y llamando a una función

La siguiente función contiene dos sentencias:

- La primera almacena el año actual en una variable llamada `$year` (veremos con detalle como funciona más adelante).
- La segunda escribe un aviso de copyright en la página utilizando el valor almacenado en la variable `$year`.

Cuando defines una función, **el código no se ejecuta, sólo se almacena en la definición de la función** para su utilización posterior.

Definiendo y llamando a una función

```
KEYWORD          NAME
[---]           [---]
function write_copyright_notice()
OPENING CURLY BRACE — {
    $year = date('Y');
    echo '&copy; ' . $year;
CLOSING CURLY BRACE — }
```

Definiendo y llamando a una función

Cuando necesitemos realizar una tarea definida en una función, especificaremos el nombre de la función seguido de paréntesis. Esto indica al intérprete de PHP que se desea ejecutar las sentencias de la función.



Podemos llamar a la misma función tantas veces como deseemos dentro del mismo archivo PHP. Una vez que una función ha realizado su tarea, el intérprete PHP pasa a la línea de código siguiente a la que llamó a la función.

Ejemplo: Funciones básicas

En el siguiente ejemplo se muestra una página que utiliza dos funciones: La primera **muestra un logotipo** y la segunda función **crea un aviso de copyright**.

1. La función `write_logo()` se define.
2. Entre llaves, una única sentencia **muestra un logotipo**.
3. Se define una función llamada `write _copyright_notice()`. Las llaves contienen dos sentencias.
4. El año actual se almacena en una variable llamada `$year`.
5. El símbolo de copyright © se escribe en la página, seguido del año obtenido previamente.

Ejemplo: Funciones básicas

6. Se llama a la primera función para añadir un logotipo en la parte superior de la página.
7. Se llama a la misma función para añadir un logotipo al pie de página.
8. Se llama a la segunda función para añadir el aviso de copyright en el pie de página de la página.

Ejemplo: Funciones básicas

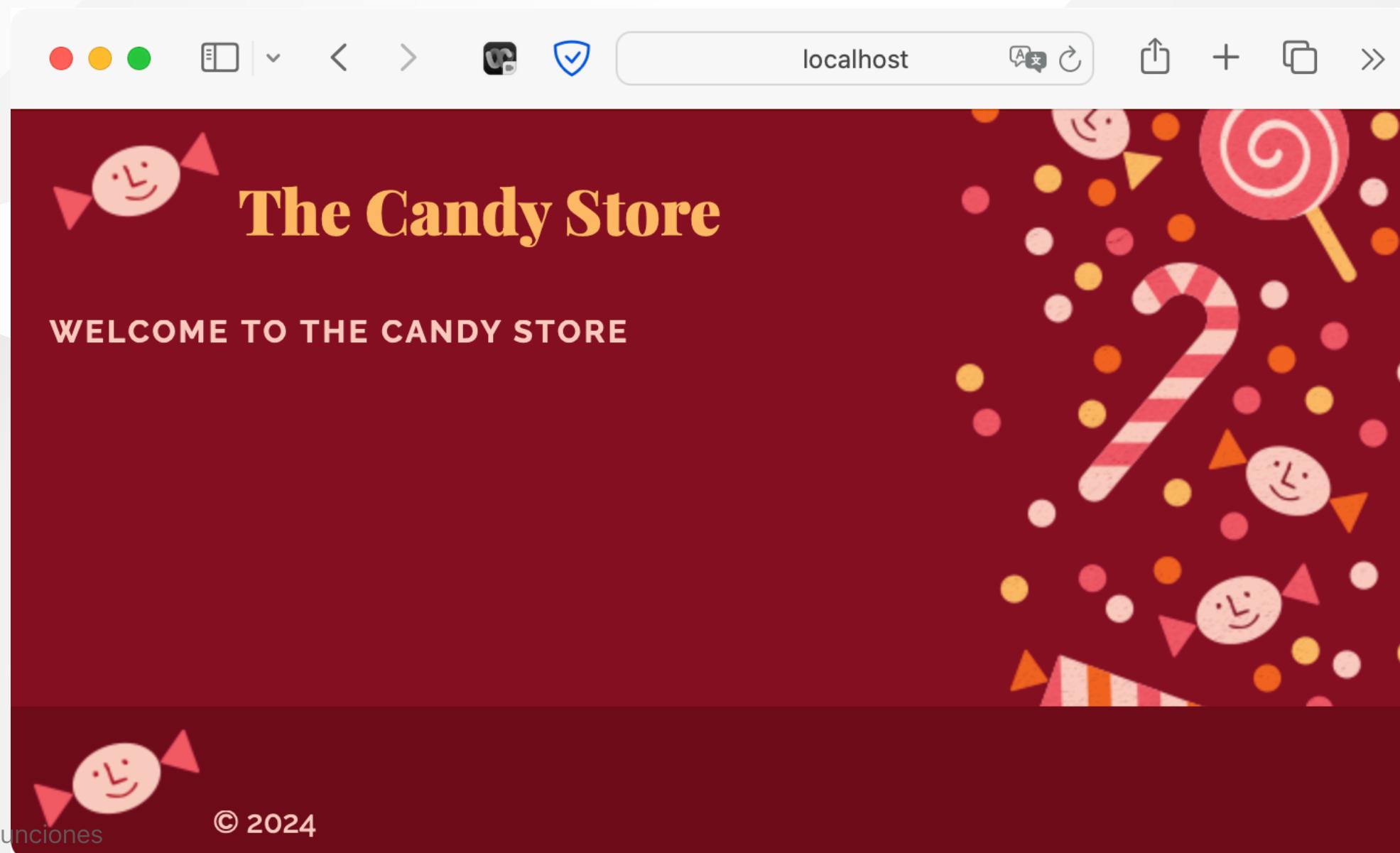
```
<?php
function write_logo()
{
    echo '';
}

function write_copyright_notice()
{
    $year = date('Y');
    echo '&copy; ' . $year;
}
?>
```

Ejemplo: Funciones básicas

```
<!DOCTYPE html>
<html>
  <head>
    <title>Basic Functions</title>
    <link rel="stylesheet" href="css/styles.css">
  </head>
  <body>
    <header>
      <h1><?php write_logo() ?> The Candy Store</h1>
    </header>
    <article>
      <h2>Welcome to the Candy Store</h2>
    </article>
    <footer>
      <?php write_logo() ?>
      <?php write_copyright_notice() ?>
    </footer>
  </body>
</html>
```

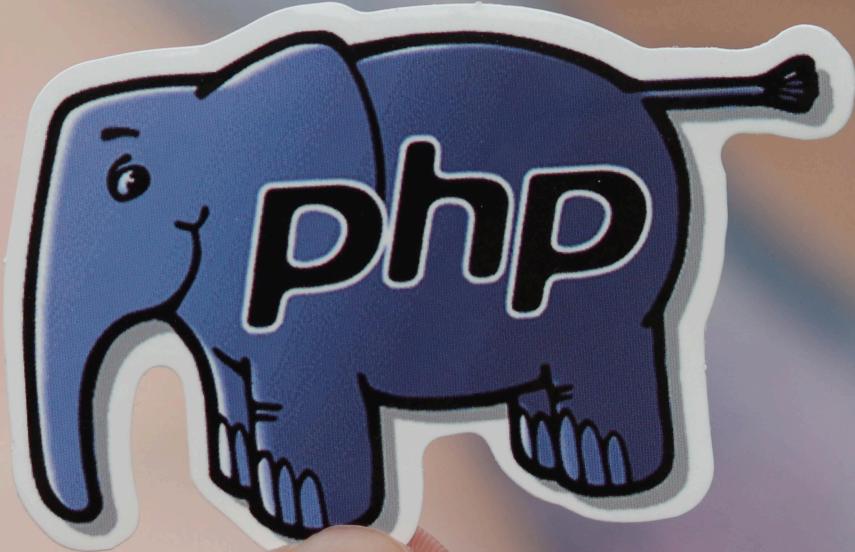
Ejemplo: Funciones básicas



Ejemplo: Funciones básicas

Sobre el ejemplo anterior, realiza las siguientes modificaciones:

- Después del paso 5, dentro de la función, escribe el nombre de la empresa (*The Candy Store*) después del aviso de copyright.



Secuencialidad y retorno en funciones

El código no siempre se ejecuta en secuencia

Una **definición de función** almacena las sentencias necesarias para realizar una tarea.
Estas sentencias sólo **se ejecutan cuando se llama a la función**.

Esto significa que **el código no siempre se ejecuta en el mismo orden en que se escribe**.

El código no siempre se ejecuta en secuencia

Cuando la gente mira por primera vez el código PHP, es común pensar que las sentencias se ejecutarán en el orden en que están escritas. En la práctica, **el intérprete de PHP puede ejecutar las sentencias en un orden muy diferente.**

A menudo se ven **funciones definidas cerca de la parte superior de una página PHP**. (Si la página también declara variables en la parte superior de la página, esas variables normalmente vienen antes de las definiciones de las funciones).

Las funciones **son llamadas más adelante en el código** donde la tarea necesita ser realizada.

El código no siempre se ejecuta en secuencia

Como se puede ver en el **ejemplo siguiente**, aunque las funciones pueden estar escritas cerca de la parte superior de una página, una definición de función sólo almacena las declaraciones en un bloque de código (y da a la función un nombre para identificar lo que hace).

El intérprete de PHP no ejecuta ese código hasta que la función es llamada. Esto puede significar que las sentencias se ejecuten en un orden muy diferente al que aparecen en el código.

El código no siempre se ejecuta en secuencia

```
<?php  
① function write_logo()  
{  
②     echo '';  
}  
  
③ function write_copyright_notice()  
{  
④     $year = date('Y');          // Get and store year  
⑤     echo '&copy; ' . $year;    // Write copyright notice  
}  
?  
?
```

El código no siempre se ejecuta en secuencia

```
<!DOCTYPE html>
<html>
  <header>
    ⑥      <h1><?php write_logo(); ?> The Candy Store</h1>
  </header>
  <article>
    <p>Welcome to The Candy Store</p>
  </article>
  <footer>
    ⑦      <?php write_logo(); ?>
    ⑧      <?php write_copyright_notice(); ?>
  <footer>
</html>
```

El código no siempre se ejecuta en secuencia

Paso	Acción del intérprete PHP
6	La primera línea a ejecutar es el Paso 6. Llama a la función <code>write_logo()</code> .
1,2	El intérprete de PHP va al Paso 1 donde se definió la función, luego ejecuta el Paso 2.
6	Cuando la función se ha ejecutado, el intérprete vuelve a la línea de código que llamó a la función.
7	Ahora pasa a la siguiente línea de código PHP. El paso 7 llama de nuevo a la función <code>write_logo()</code> .

El código no siempre se ejecuta en secuencia

Paso	Acción del intérprete PHP
1,2	El intérprete de PHP va al Paso 1 donde se definió la función, y luego ejecuta el Paso 2.
7	Cuando la función se ha ejecutado, el intérprete vuelve a la línea de código que llamó a la función.
8	Ahora pasa a la siguiente línea de código PHP. El paso 8 llama a la función <code>write_copyright_notice()</code> .

El código no siempre se ejecuta en secuencia

Paso	Acción del intérprete PHP
3, 4, 5	Se mueve al Paso 3 donde la función fue declarada, luego ejecuta los Pasos 4-5
8	Una vez que la función ha terminado, vuelve a la línea que la llamó.

El código no siempre se ejecuta en secuencia

Es posible que se llame a una función **antes** de definirla, pero es mejor definir las funciones antes de llamarlas.

Si varias páginas necesitan utilizar las mismas funciones, las definiciones de las funciones pueden **almacenarse en un archivo *include***.

Obtener datos de las funciones

Las **funciones rara vez escriben datos directamente en la página** (como en el ejemplo anterior). Más a menudo, una función **crea un nuevo valor y lo devuelve a la sentencia que la llamó**.

Para devolver un valor, se utiliza la palabra clave `return`, seguida del valor que se desea devolver.

Obtener datos de las funciones

La función a continuación es similar a las de los ejemplos anteriores, pero **crea un aviso de copyright y lo almacena en una variable** llamada `$message`.

Luego **devuelve el valor almacenado en la variable** `$message` (en lugar de escribir el HTML directamente en la página).

```
function create_copyright_notice()
{
    $year      = date('Y');
    $message = '&copy; ' . $year;
    return $message;
}
```

KEYWORD VALUE TO RETURN

Obtener datos de las funciones

Cuando una función devuelve un valor y desea mostrar esos datos en la página, utiliza el comando `echo` (o la *abreviatura de echo*) y, a continuación, llama a la función.

```
<?= create_copyright_notice() ?>
```

Se considera una **mejor práctica devolver un valor desde una función y luego escribirlo en la página**, en lugar de utilizar un comando `echo` dentro de la función, por varias razones relacionadas con la separación de lógica y presentación, la testabilidad, la reutilización del código y el mantenimiento.

Obtener datos de las funciones

También puedes **almacenar el valor devuelto por una función en una variable**.

Para ello, utiliza un nombre de variable, seguido del operador de asignación, y luego llama a la función.

```
$copyright_notice = create_copyright_notice();
```

Ejemplo: funciones que devuelven un valor

En este ejemplo, las funciones están adaptadas para devolver valores.

1. En primer lugar, se define `create_logo()`.
2. Se utiliza la palabra clave `return`, seguida del HTML necesario para crear la imagen.
3. Se define `create_copyright_notice()`. Entre llaves, hay tres sentencias que:
4. Obtiene el año actual y lo almacena en una variable llamada `$año`.
5. Crea una variable llamada `$message`, y almacena en ella el símbolo de copyright `©` seguido del año actual.

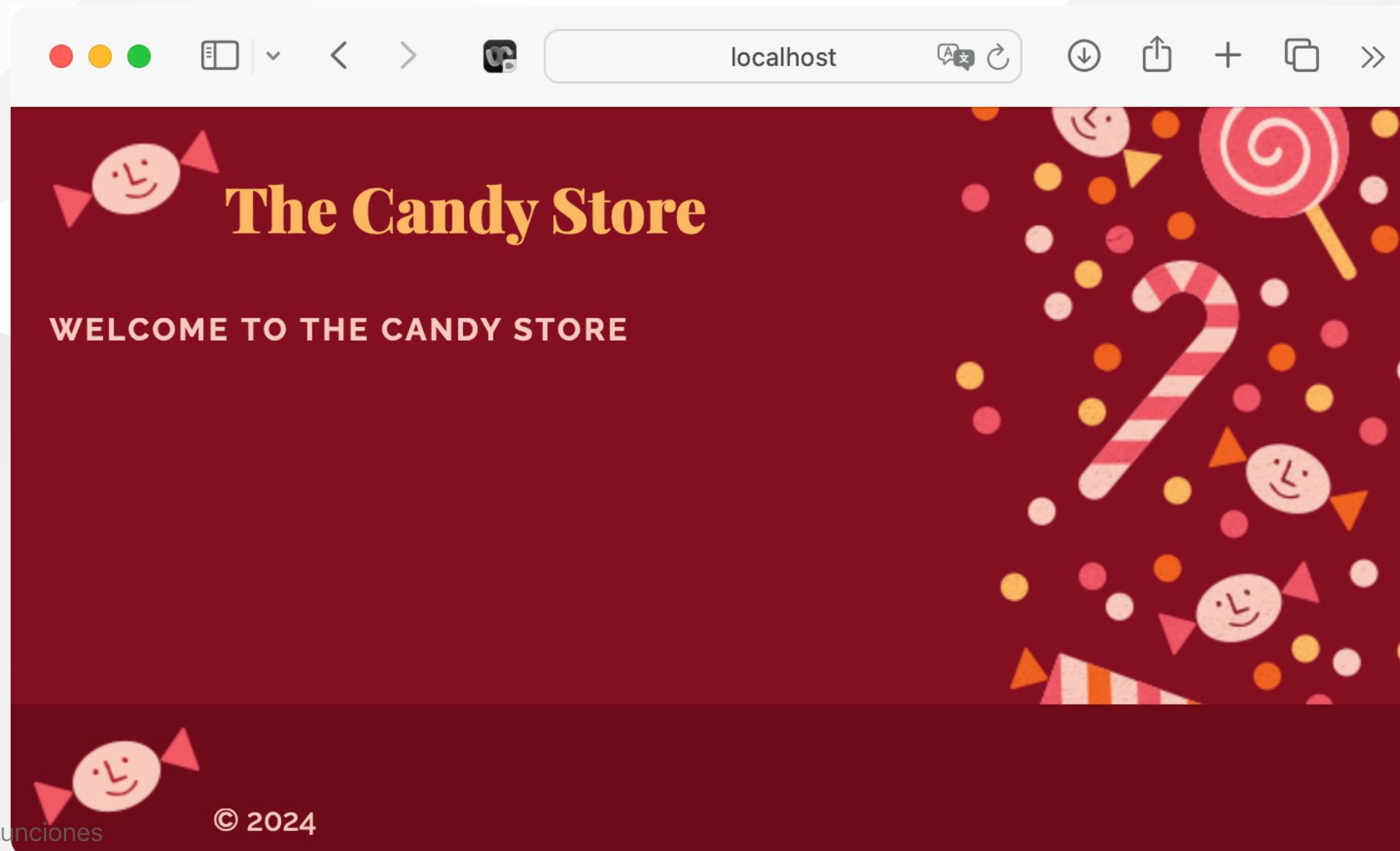
Ejemplo: funciones que devuelven un valor

6. Devuelve el valor almacenado en la variable `$mensaje`.
7. Se llama a la primera función, y el valor que se devuelve se escribe en la página utilizando la abreviatura de echo.
8. Se llama de nuevo a la primera función para repetir el logo.
9. Se llama a la segunda función y **el valor devuelto se escribe en la página** utilizando la abreviatura de eco.

Ejemplo: funciones que devuelven un valor

```
<?php  
① function create_logo()  
{  
②     return '';  
}  
  
③ function create_copyright_notice()  
{  
④     $year    = date('Y');  
⑤     $message = '&copy; ' . $year;  
⑥     return $message;  
}  
?> ...  
<header>  
⑦     <h1><?= create_logo() ?>The Candy Store</h1>  
</header>  
<article>  
    <h2>Welcome to The Candy Store</h2>  
</article>  
<footer>  
⑧     <?= create_logo() ?>  
⑨     <?= create_copyright_notice() ?>  
</footer>
```

Ejemplo: funciones que devuelven un valor



Ejemplo: funciones que devuelven un valor

Sobre el ejemplo anterior, realiza las siguientes modificaciones:

- En el paso 5, añade el nombre de la empresa a la variable `$message` .

Definir funciones que necesitan información

Los **parámetros** son como nombres de variables que representan valores que una función necesita para realizar su tarea.

Los valores que representa el parámetro **pueden cambiar cada vez que se llama a la función.**

Definir funciones que necesitan información

Al definir una función, si esta necesita datos para realizar su tarea:

- Enumera los datos que necesita
- Proporciona a cada uno un **nombre de variable** (empezando por `$`) que **describa el tipo de datos que representa**
- Coloca estos nombres **entre los paréntesis que siguen al nombre de la función.**
- Separa cada nombre con una coma
- Estos son los parámetros de la función

Los parámetros **actúan como variables**, pero sólo pueden ser utilizados por las sentencias entre llaves de la definición de la función. El código **frente a la definición de la función no puede acceder a ellos**.

Definir funciones que necesitan información

La función `calculate_cost()` que se muestra a continuación **calcula el total cuando los usuarios compran uno o varios artículos iguales**. Para realizar esta tarea, necesita dos parámetros:

- `$price` representa el **precio de un artículo**
- `$quantity` representa la **cantidad** de ese artículo

Dentro de esta función, `$price` y `$quantity` actúan como variables. **Representan los valores que se pasan a la función cuando se llama.**

El código dentro de la definición de la función **calcula el coste total multiplicando el precio por la cantidad**. Este valor **se devuelve al código que llamó a la función** utilizando la palabra clave `return`.

Definir funciones que necesitan información

```
function calculate_cost($price, $quantity)
{
    return $price * $quantity;
}
```

PARAMETERS

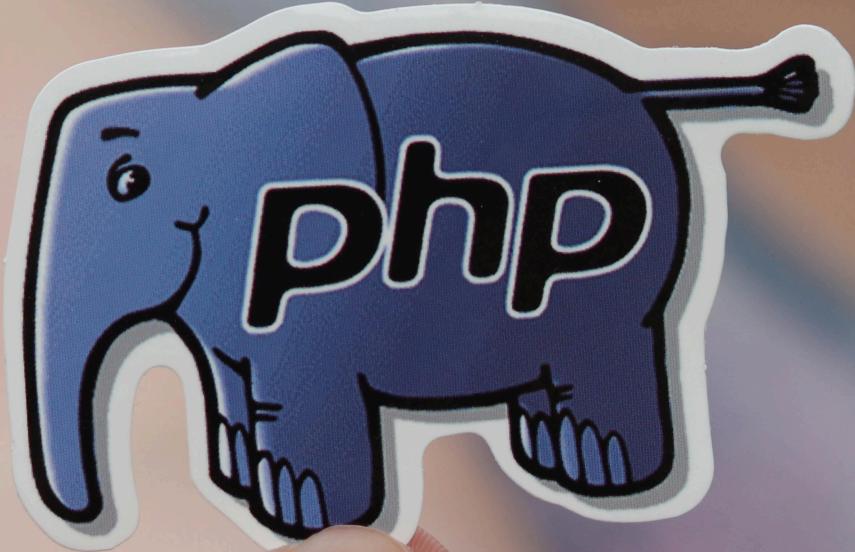
PARAMETER NAMES ARE USED LIKE
VARIABLES WITHIN THE FUNCTION

Definir funciones que necesitan información

Cuando el intérprete de PHP llega a la llave de cierre, **olvida todos los valores almacenados en la función.**

Esto es importante porque **una página puede llamar a una función varias veces, utilizando valores diferentes cada vez.**

NOTA: En PHP 8, la definición de la función puede añadir una coma después del nombre del último parámetro (no sólo entre parámetros) haciendo el código más uniforme. Esto causaría un error en versiones anteriores de PHP.



Uso de parámetros y argumentos

Llamar a funciones que necesitan información

Cuando se llama a una función que tiene parámetros, **el valor de cada parámetro se especifica entre paréntesis después del nombre de la función.**

Los valores utilizados al llamar a una función se denominan **argumentos**.

Llamar a funciones que necesitan información

Argumentos como valores

A continuación, cuando se llama a la función `calculate_cost()`, se le proporcionan los valores que debe utilizar.

Los valores se suministran en el mismo orden en que se especificaron los parámetros en la definición de la función.

El número 3 se utiliza para el precio del artículo y el número 5 se utiliza para la cantidad comprada, por lo que `calculate_cost()` devolverá el número 15 y este valor se almacena en una variable llamada `$total`.

```
$total = calculate_cost(3, 5);
```

Llamar a funciones que necesitan información

Argumentos como variables

Esta vez, cuando se llama a `calculate_cost()`, utiliza nombres de variables en lugar de valores:

- `$cost` representa el precio del artículo
- `$units` representa la cantidad comprada

Si se utilizan nombres de variables como argumentos, **no es necesario que los nombres de las variables coincidan con los nombres de los parámetros.**

Llamar a funciones que necesitan información

Argumentos como variables

Cuando se llama a la función de abajo, el intérprete de PHP enviará los valores almacenados en las variables `$cost` y `$units` a la función.

Dentro de la función, esos valores están representados por los nombres de los parámetros `$price` y `$quantity` (los nombres se especificaron entre paréntesis en la primera línea de la definición de la función).

```
$cost = 4;  
$units = 6;  
$total = calculate_cost($cost, $units);
```

Llamar a funciones que necesitan información

Parámetros vs Argumentos

A menudo se utilizan indistintamente los términos parámetro y argumento, pero existe una sutil diferencia.

En el ejemplo anterior, cuando se define la función, puedes ver los nombres `$price` y `$quantity`. Dentro de las llaves de la función, esas palabras actúan como variables. Estos nombres son los **parámetros**.

En esta página, cuando se llama a la función, el código especifica números que se utilizarán para realizar el cálculo o variables que contienen números. Estos valores que se pasan al código (la información que necesita para calcular el coste de este tipo concreto de caramelo) se denominan **argumentos**.

Ejemplo: Función con parámetros

1. En este ejemplo, `calculate_total()` se define en la parte superior de la página. **Calcula el total cuando alguien compra uno o más artículos del mismo artículo y, a continuación, añade un impuesto sobre las ventas del 20%. Necesita dos datos, por lo que tiene dos parámetros:**
 - `$price` representará el precio de un artículo individual
 - `$cantidad` representa el número de unidades del artículo que se compra
2. En la definición de la función, una variable llamada `$cost` almacena el coste del número de unidades requerido. Este se calcula multiplicando el valor almacenado en `$price` por el valor en `$quantity`.
3. A continuación, el impuesto sobre las ventas correspondiente a esos artículos se almacena en una variable denominada `$tax`. Se calcula multiplicando el valor en `$cost` (creado en el paso 2) por 0,2.

Ejemplo: Función con parámetros

4. Para obtener el total, se suman los valores de `$cost` y `$tax`.
5. El total se devuelve desde la función al código que la llamó.
6. La función se llama tres veces. Cada vez, utiliza diferentes precios y cantidades, y el valor devuelto por la función se escribe en la página.

Ejemplo: Función con parámetros

```
<?php
function calculate_total($price, $quantity)
{
    $cost  = $price * $quantity;
    $tax   = $cost  * (20 / 100);
    $total = $cost + $tax;
    return $total;
}
?>
<!DOCTYPE html>
<html>
    <head>
        <title>Functions with Parameters</title>
        <link rel="stylesheet" href="css/styles.css">
    </head>
    <body>
        <h1>The Candy Store</h1>
        <p>Mints: $<?= calculate_total(2, 5) ?></p>
        <p>Toffee: $<?= calculate_total(3, 5) ?></p>
        <p>Fudge: $<?= calculate_total(5, 4) ?></p>
    </body>
</html>
```

Ejemplo: Función con parámetros

The screenshot shows a web browser window with the address bar set to "localhost". The main content area features a dark red background with a festive illustration of a lollipop, a candy cane, and several smiling faces made of circles and triangles. On the left side, the text "The Candy Store" is displayed in a large, gold-colored serif font. Below it, there is a list of items and their prices:

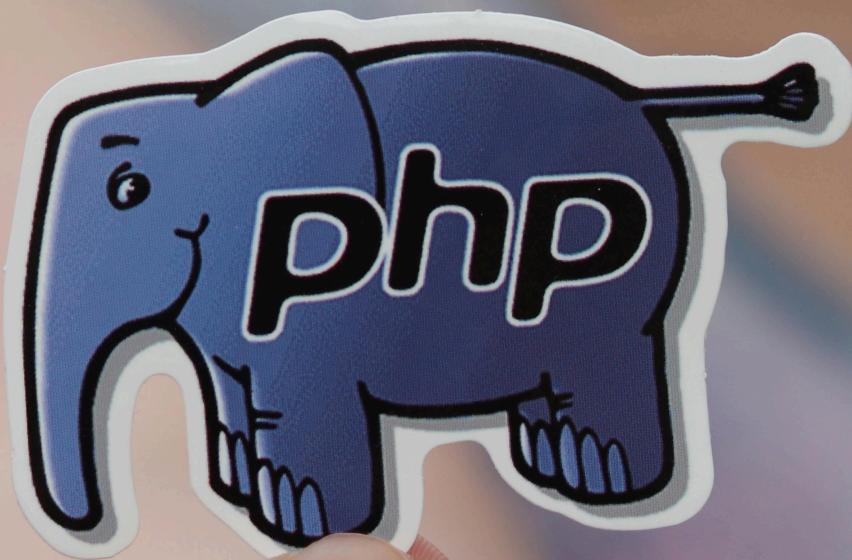
- Mints: \$12
- Toffee: \$18
- Fudge: \$24

The browser interface includes standard navigation buttons (back, forward, search) and a tab labeled "VC".

Ejemplo: Función con parámetros

Sobre el ejemplo anterior, realiza las siguientes modificaciones:

- En el Paso 6, vuelve a llamar a la función para mostrar el precio de 4 paquetes de chicles a 1,50 \$ cada uno.



Nombrado y ámbito de las funciones

Nombrando funciones

El nombre de una función debe **describir claramente la tarea que realiza**.

Suele estar formado por una **palabra que describe lo que hace la función y el tipo de información con la que trabaja o que devuelve**.

Nombrando funciones

Las **reglas** para los nombres de las funciones son **las mismas que para las variables**. Deben comenzar con una letra, seguida de cualquier combinación de letras, números o guiones bajos.

No se pueden tener dos funciones con el mismo nombre en la misma página PHP.

En los ejemplos que estamos estudiando, todos los nombres de funciones van en **minúsculas**. Si el nombre de la función requiere más de una palabra, **un guión bajo las separa**.

Comprobarás que las funciones utilizan diferentes reglas de nomenclatura; lo importante es utilizar una estrategia de nomenclatura coherente en todo el proyecto.

Nombrando funciones

Para ayudar a crear un nombre que describa la tarea realizada por la función, deberías:

- **Indicar qué hace la función** (por ejemplo, calcular, obtener o actualizar).
- seguido del **tipo de información que devuelve o procesa** (por ejemplo, fecha, total o mensaje)

A continuación se muestran **dos ejemplos de nombres de función descriptivos** que ya has visto en esta unidad:

- `calculate_total()` calcula el coste total de los artículos en venta.
- `create_copyright_notice()` crea un aviso de copyright.

Nombrando funciones

WHAT IT DOES

`calculate_total()`

DATA IT RETURNS

WHAT IT DOES

`create_copyright_notice()`

DATA IT RETURNS

Ámbito (scope)

Cuando se llama a una función, el código se ejecuta en su propio **ámbito**; no puede acceder ni actualizar valores almacenados en variables fuera de la función.

Ámbito (scope)

El código de una función se ejecuta independientemente del resto de la página.

- Cualquier información que una función necesite para hacer su trabajo debemos pasársela utilizando parámetros, que actuarán como variables dentro de la función.
- Cuando se llama a la función, las sentencias de la función pueden crear variables y darles valores.
- La función puede entonces devolver un valor al código que llamó a la función.
- Cuando la función se ha ejecutado, los parámetros y variables creados en la función se destruyen.

Ámbito (scope)

Dado que el código de la función se ejecuta de forma separada del resto de la página:

- La función **no puede acceder o actualizar variables fuera de la función** (razón por la cual la información se pasa como parámetros).
- El código posterior no puede acceder a las variables creadas dentro de la función porque se destruyen en cuanto la función ha hecho su trabajo.

Los programadores decimos que cada vez que se llama a una función, el intérprete de PHP ejecuta el código de esa función en el **ámbito local** (*local scope*). El código fuera de la función en la parte principal de la página está en **ámbito global** (*global scope*).

Ámbito (scope)

El lugar donde se declara una variable, en ámbito local o global, determina si otro código puede o no acceder a ella.

En el diagrama siguiente, hay dos variables que se llaman ambas `$tax`; cada una se ejecuta en un ámbito diferente:

- A: La primera variable llamada `$tax` se crea en ámbito global. (Está fuera de la función).
- B: La segunda variable llamada `$tax` se crea dentro de la definición de la función, en ámbito local.

Lo ideal sería que dos variables no compartieran nombre en el mismo script, pero esto muestra cómo las variables se tratan de forma totalmente independiente la una de la otra.

Ámbito (scope)

```
<?php  
① $tax = 20;  
    function calculate_total($price, $quantity)  
    {  
        $cost  = $price * $quantity;  
②        $tax   = $cost  * (20 / 100);  
        $total = $cost  + $tax;  
        return $total;  
    }  
?>
```

Ejemplo: Demostrando el ámbito (scope)

1. Una variable llamada `$tax` se declara en ámbito **global** para que pueda ser utilizada por cualquier código fuera de la función.
2. Se define la función `calcular_total()`. Necesita el precio y la cantidad de un artículo. Las variables creadas dentro de esta función están en ámbito **local**.
3. El coste se calcula multiplicando el precio del artículo por la cantidad necesaria. El resultado se almacena en la variable `$coste`.
4. El impuesto a pagar se calcula multiplicando el coste por el tipo impositivo (que es 20 dividido por 100). El resultado se almacena en una variable llamada `$tax`. Nota: Esto no sobrescribe el valor almacenado en la variable `$tax` creada en el Paso 1.

Ejemplo: Demostrando el ámbito (scope)

5. Para obtener el total, el valor en `$cost` se suma al valor que se almacenó en `$tax` en el Paso 4.
6. Se devuelve el total. Cuando la función se haya ejecutado, el intérprete PHP borrará todos los parámetros y las variables creadas en la función.
7. La función se llama tres veces con nuevos valores cada vez.
8. Se muestra el valor almacenado en `$tax` (en el ámbito global en el Paso 1).

Ejemplo: Demostrando el ámbito (scope)

```
<?php
$tax = '20';

function calculate_total($price, $quantity)
{
    $cost  = $price * $quantity;
    $tax   = $cost  * (20 / 100);
    $total = $cost  + $tax;
    return $total;
}
?>
<!DOCTYPE html>
<html>
    <head>
        <title>Global and Local Scope</title>
        <link rel="stylesheet" href="css/styles.css">
    </head>
    <body>
        <h1>The Candy Store</h1>
        <p>Mints: $<?= calculate_total(2, 5) ?></p>
        <p>Toffee: $<?= calculate_total(3, 5) ?></p>
        <p>Fudge: $<?= calculate_total(5, 4) ?></p>
        <p>Prices include tax at: <?= $tax ?>%</p>
    </body>
</html>
```

Ejemplo: Demostrando el ámbito (scope)

The screenshot shows a web browser window with the URL "localhost" in the address bar. The page itself has a dark red background and features a title "The Candy Store" in large, gold-colored serif font. Below the title, there is a list of items and their prices: "Mints: \$12", "Toffee: \$18", and "Fudge: \$24". A note at the bottom states "Prices include tax at: 20%". To the right of the text, there is a colorful illustration of various candies, including a large striped candy cane, several lollipops, and some wrapped candies.

Mints: \$12

Toffee: \$18

Fudge: \$24

Prices include tax at: 20%

DWES - U3. Funciones

74

Ejemplo: Demostrando el ámbito (scope)

Sobre el ejemplo anterior, realiza las siguientes modificaciones:

- En el Paso 1, cambia el tipo impositivo a 25. Cambiará el tipo impositivo mostrado en la parte inferior de la página, pero no los totales escritos en el Paso 7.

Los totales del paso 7 siguen siendo los mismos porque en el paso 4, dentro de la función `calculate_total()` se utiliza un tipo impositivo del 20%. Esto muestra cómo las dos variables llamadas `$tax` funcionan independientemente.

- Realiza la modificación apropiada para que el cambio del tipo impositivo anterior se refleje también en los totales descritos en el Paso 7.

Variables globales y estáticas

En casos limitados, el código de una función puede acceder o actualizar una variable global y recordar un valor almacenado en una variable de la función después de que ésta haya terminado de ejecutarse.

VARIABLES GLOBALES Y ESTÁTICAS

Accediendo o actualizando variables globales desde una función

El código dentro de una función puede acceder o actualizar un valor almacenado en una variable que fue declarada en ámbito global **si se le indica al intérprete de PHP que puede acceder a ella.**

Al principio del bloque de código de la función (antes de que se utilice la variable), añade la palabra clave `global` seguida del nombre de la variable. Esto permite que el código de la función acceda o actualice su valor.

```
global $cost;
```

Se considera una buena práctica pasar valores a una función utilizando parámetros, pero la posibilidad de hacerlo se menciona aquí porque es posible que veas código que accede a una variable global utilizando esta técnica.

Variables globales y estáticas

Conservar un valor en una función tras su ejecución

Cuando una función ha terminado de ejecutarse, normalmente borra cualquier variable local que haya sido creada dentro de la función.

Se le puede decir al intérprete de PHP que recuerde un valor almacenado en una variable que fue creada en una función si esa variable es creada como una **variable estática**.

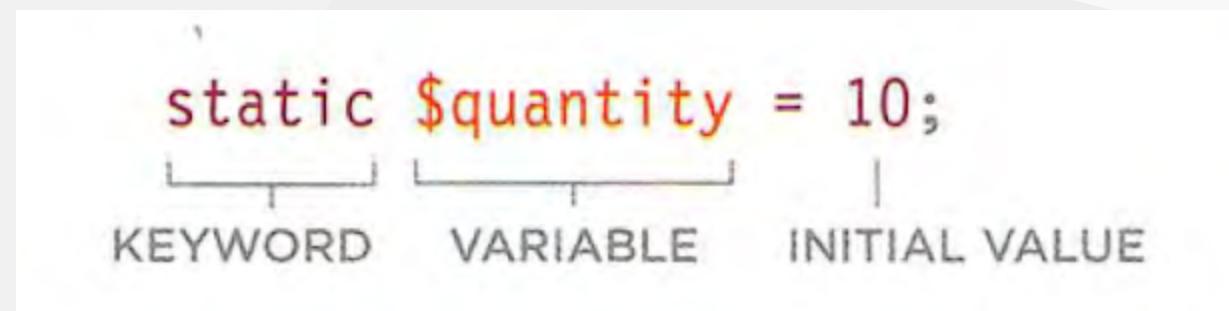
Variables globales y estáticas

Conservar un valor en una función tras su ejecución

Para crear una variable estática, utiliza:

- La palabra clave `static`
- seguida de un nombre de variable
- A continuación, un **valor inicial que debe mantener la primera vez que se llama a la función**

Cuando la función termine de ejecutarse, esta variable y el valor que almacena no se borrarán (pero sólo estarán disponibles para el código dentro de la función).



Ejemplo: Acceder a variables externas a una función

A continuación veremos un ejemplo de uso de variables globales y estáticas:

1. Se crea una variable llamada `$tax_rate` en el ámbito global.
2. `calculate_running_total()` crea un total acumulado.
3. La palabra clave global permite a la función acceder/actualizar el valor de la variable global `$tax_rate`.
4. La palabra clave `static` indica que la variable `$running_total` (y su valor) no deben borrarse cuando la función haya terminado de ejecutarse. (Obtiene un valor inicial de 0 cuando se crea).

Ejemplo: Acceder a variables externas a una función

5. `$total` contiene el precio de un producto multiplicado por la cantidad que desea el cliente.
6. `$tax` contiene la cantidad de impuestos a pagar en esos artículos utilizando el valor de la variable global `$tax_rate` creada en el Paso 1.
7. `$running_total` contiene el:
 - Valor en `$running_total`
 - Más el valor en `$total`
 - Más el valor en `$tax`
8. El valor en `$running_total` se devuelve pero **no se borra porque es una variable estática**.
9. La función se llama tres veces. Cada vez, el total de este elemento se añade al total anterior.

Ejemplo: Acceder a variables externas a una función

```
<?php
$tax_rate = 0.2;

function calculate_running_total($price, $quantity)
{
    global $tax_rate;
    static $running_total = 0;
    $total = $price * $quantity;
    $tax   = $total * $tax_rate;
    $running_total = $running_total + $total + $tax;
    return $running_total;
}
?>
```

Ejemplo: Acceder a variables externas a una función

```
<!DOCTYPE html>
<html>
  <head>
    <title>Global and Static Variables</title>
    <link rel="stylesheet" href="css/styles.css">
  </head>
  <body>
    <h1>The Candy Store</h1>
    <table>
      <tr><th>Item</th><th>Price</th><th>Qty</th>
      <th>Running total</th></tr>
      <tr><td>Mints:</td><td>$2</td><td>5</td>
        <td>$<?= calculate_running_total(2, 5) ?></td></tr>
      <tr><td>Toffee:</td><td>$3</td><td>5</td>
        <td>$<?= calculate_running_total(3, 5) ?></td></tr>
      <tr><td>Fudge:</td><td>$5</td><td>4</td>
        <td>$<?= calculate_running_total(5, 4) ?></td></tr>
    </table>
  </body>
</html>
```

Ejemplo: Acceder a variables externas a una función

The screenshot shows a web browser window with the URL "localhost" in the address bar. The page title is "The Candy Store". On the left, there is a table with the following data:

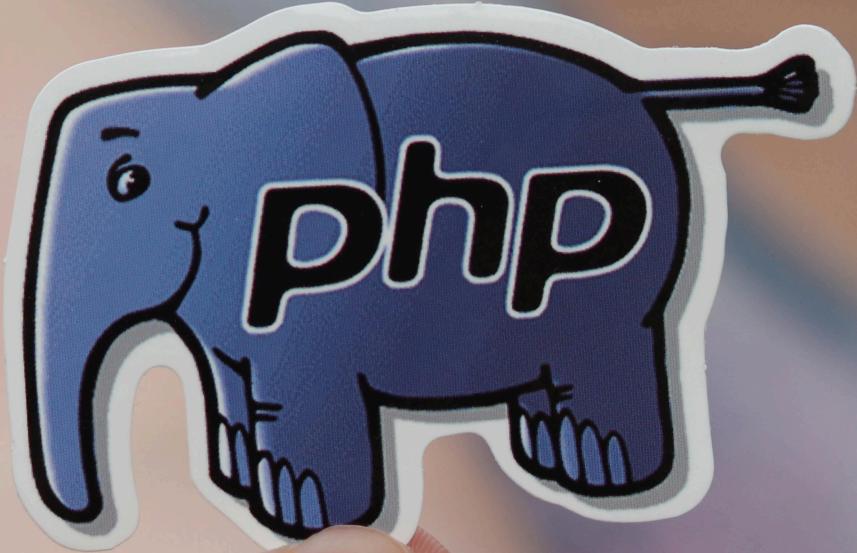
ITEM	PRICE	QTY	RUNNING TOTAL
Mints:	\$2	5	\$12
Toffee:	\$3	5	\$30
Fudge:	\$5	4	\$54

On the right side of the page, there is a decorative background featuring a large candy cane, a lollipop, and various colorful candies and shapes.

Ejemplo: Acceder a variables externas a una función

Sobre el ejemplo anterior, realiza las siguientes modificaciones:

- Añade más productos a la tabla y calcular el total acumulado para cada producto utilizando la función `calculate_running_total()`.
- Modificar la tasa de impuestos (`$tax_rate`) y observar cómo afecta el total acumulado.
- Añadir una nueva variable global que represente un descuento global (por ejemplo, `$global_discount`) y modificar la función `calculate_running_total()` para aplicar este descuento antes de calcular el impuesto.



Funciones y tipos de datos compuestos

Funciones y tipos de datos compuestos

Los tipos de datos **compuestos** (como los arrays) pueden almacenar múltiples valores.

Las funciones pueden aceptar un tipo de datos compuesto como argumento y devolver un tipo de datos compuesto de la función.

Funciones y tipos de datos compuestos

Uso de un tipo de dato compuesto como argumento

Al definir una función, los parámetros pueden escribirse de forma que acepten un tipo de datos escalar o compuesto:

- Los tipos de datos **escalares** contienen un dato: una cadena, un número o un booleano.
- Los tipos de datos **compuestos** pueden contener varios datos. En la unidad 1 conociste los *arrays* y en la unidad 4 próximamente aprenderemos sobre otro tipo de datos compuestos llamados *objetos*.

En el ejemplo que veremos a continuación, puedes ver un caso en el que un array que contiene tres tipos de cambio diferentes se pasa a la función como un único parámetro.

Funciones y tipos de datos compuestos

Uso de un tipo de dato compuesto como valor de retorno

Una función sólo debe realizar una única tarea (no múltiples tareas) pero **una tarea individual puede generar más de un valor que necesita ser devuelto.**

Si desea devolver más de un valor de una función, debe crear un array o un objeto en la función y luego devolverlo. Esto se debe a que una función sólo puede devolver un valor escalar o compuesto.

Funciones y tipos de datos compuestos

Uso de un tipo de dato compuesto como valor de retorno

Tan pronto como el intérprete PHP ha ejecutado una sentencia que comienza con la palabra clave `return`, deja de ejecutar el código en la función y vuelve a la línea de código que llamó a la función (incluso si hay más sentencias en la definición de la función que no se han ejecutado).

En el ejemplo siguiente, la función `calculate_prices()` calcula tres precios para un artículo en tres monedas y devuelve los precios como un array.

Ejemplo: Funciones y tipos de datos compuestos

1. La variable `$us_price` contiene el precio en dólares estadounidenses de un artículo.
2. La variable `$rates` almacena un array asociativo de tres tipos de cambio.
3. La función `calculate_prices()` calcula los precios de un artículo en tres divisas y los devuelve como un array. Tiene dos parámetros: el precio en dólares y el array con los tipos de cambio.
4. Se crea un array y se almacena en una variable llamada `$prices`. El primer elemento contiene el precio del Reino Unido, que se calcula multiplicando el precio de EE.UU. por el tipo de cambio del Reino Unido. A continuación, se añaden al array los precios de la UE y Japón.

Ejemplo: Funciones y tipos de datos compuestos

5. La función devuelve el array que contiene los tres nuevos precios.
6. Se llama a la función, y el array que devuelve se almacena en una variable llamada `$global_prices`.
7. El precio de EE.UU. se escribe utilizando la variable del paso 1.
8. Se muestran otros precios del array creado en el paso 6.

Ejemplo: Funciones y tipos de datos compuestos

```
<?php
$us_price = 4;
$rates = [
    'uk' => 0.81,
    'eu' => 0.93,
    'jp' => 113.21,
];

function calculate_prices($usd, $exchange_rates)
{
    $prices = [
        'pound' => $usd * $exchange_rates['uk'],
        'euro'  => $usd * $exchange_rates['eu'],
        'yen'   => $usd * $exchange_rates['jp'],
    ];
    return $prices;
}

$global_prices = calculate_prices($us_price, $rates);
?>
```

Ejemplo: Funciones y tipos de datos compuestos

```
<!DOCTYPE html>
<html>
  <head>
    <title>Functions with Multiple Values</title>
    <link rel="stylesheet" href="css/styles.css">
  </head>
  <body>
    <h1>The Candy Store</h1>
    <h2>Chocolates</h2>
    <p>US $<?= $us_price ?></p>
    <p>(UK &pound; <?= $global_prices['pound'] ?> |
      EU &euro; <?= $global_prices['euro'] ?> |
      JP &yen; <?= $global_prices['yen'] ?>)</p>
  </body>
</html>
```

Ejemplo: Funciones y tipos de datos compuestos

The screenshot shows a web browser window with the address bar set to "localhost". The main content area displays a page titled "The Candy Store". On the left, there is a section titled "CHOCOLATES" with the price "US \$4" and exchange rates "(UK £ 3.24 | EU € 3.72 | JP ¥ 452.84)". To the right of this text is a vibrant, colorful illustration of various candies, including a large striped candy cane, a lollipop, and several smaller circular and triangular shapes.

The Candy Store

CHOCOLATES

US \$4

(UK £ 3.24 | EU € 3.72 | JP ¥ 452.84)

localhost

DWES - U3. Funciones

95

Ejemplo: Funciones y tipos de datos compuestos

Sobre el ejemplo anterior, realiza las siguientes modificaciones:

- Añadir más monedas al array de tipos de cambio y actualizar la función `calculate_prices()` para calcular los precios en esas nuevas monedas:
 - AUD (Dólar Australiano): 1.30
 - CAD (Dólar Canadiense): 1.25
- Modificar el código para incluir una función que formatee los precios con dos decimales y el símbolo de la moneda correspondiente.
- Crear una tabla HTML que muestre los precios de varios productos en diferentes monedas utilizando las funciones creadas.



Declaración y uso de tipos en funciones

Declaración de tipos en argumentos y retornos

Al definir una función, podemos **especificar qué tipo de datos debe tener cada argumento y qué tipo de datos debe devolver la función.**

Esto puede resultar útil ya que mejora la robustez, legibilidad, mantenibilidad y rendimiento del código. Además, facilita el trabajo en equipo y el uso de herramientas avanzadas de desarrollo (autocompletado y refactorización asistida en IDEs por ejemplo).

Esto resulta en un **código más seguro y fácil de entender**, contribuyendo a un desarrollo más eficiente y menos propenso a errores.

Declaración de tipos en argumentos y retornos

Algunas tareas requieren datos de un **tipo específico**. Por ejemplo, las funciones que realizan operaciones aritméticas requieren números como argumentos, y las funciones que procesan texto necesitan cadenas.

Una definición de función puede especificar los tipos de datos que espera cada parámetro y el tipo de datos que debe devolver la función. Esto nos ayuda a los programadores porque la primera línea de la definición de función les **muestra claramente qué tipo de datos debe tener cada argumento y qué tipo de datos debe devolver la función**.

Declaración de tipos en argumentos y retornos

A continuación, en la primera línea de la definición de la función:

- Dentro de los paréntesis, se especifica un **tipo de argumento antes de cada nombre** de parámetro para indicar el tipo de datos que debe utilizar el argumento.
- Después de los paréntesis, hay dos puntos seguidos de un tipo de retorno para indicar **el tipo de datos que devolverá la función**.

En este caso, ambos argumentos deben ser un número entero, y el valor devuelto también debe ser un número entero.

Declaración de tipos en argumentos y retornos

```
function calculate_total(int $price, int $quantity): int
{
    return $price * $quantity;
}
```

Diagram illustrating the declaration of types in arguments and returns:

- ARGUMENT TYPE**: Points to the type declarations for the parameters (\$price and \$quantity).
- PARAMETERS**: Groups the two parameters (\$price and \$quantity).
- COLON**: Points to the colon character separating the parameters from the return type.
- RETURN TYPE**: Points to the type declaration for the return value (int).

Declaración de tipos en argumentos y retornos

La tabla que se muestra a continuación ilustra los tipos de datos utilizados en las declaraciones de argumentos y tipos de retorno. **PHP 8 añadió:**

- tipos de **unión** para especificar que un argumento o tipo de retorno puede ser uno de un conjunto de tipos. Cada tipo está separado por el símbolo `|`. Por ejemplo, `int | float` indica un entero o un flotante.
- `mixed` que indica que un argumento o tipo de retorno puede ser **cualquiera de los tipos de datos** (se llama **pseudotipo** porque las variables no pueden tener este tipo).

`mixed` equivale a la unión tipo

`object|resource|array|string|float|int|bool|null`.

Declaración de tipos en argumentos y retornos

DATA TYPE	DESCRIPTION
<code>string</code>	String
<code>int</code>	Integer (whole number)
<code>float</code>	Floating point number (decimal)
<code>bool</code>	Boolean (true or false / 0 or 1)
<code>array</code>	Array
<code>className</code>	Class of object (see Chapter 4)
<code>mixed</code>	A mix of the above data types (PHP8)

NOTA: Si un argumento o tipo de retorno puede ser *nulo* (en lugar de un valor) puedes utilizar un signo de interrogación antes del tipo de dato. Por ejemplo, `?int` indica que el valor será un entero o *null*. En PHP 8, esto también podría indicarse con un tipo de unión `int | null`.

Ejemplo: Usando declaración de tipos

1. En este ejemplo, la primera línea de la definición de la función especifica:

- Declaraciones de **tipo de argumento** para los parámetros `$price` y `$quantity`, para mostrar que sus valores deben ser enteros.
- Una declaración del tipo de retorno para indicar que la función debe devolver un número entero.

Las declaraciones de tipo no afectan al funcionamiento de este ejemplo; **sólo indican qué tipos de datos deben ser los argumentos y los tipos de retorno**. A continuación veremos cómo hacer que estos valores utilicen los tipos de datos correctos.

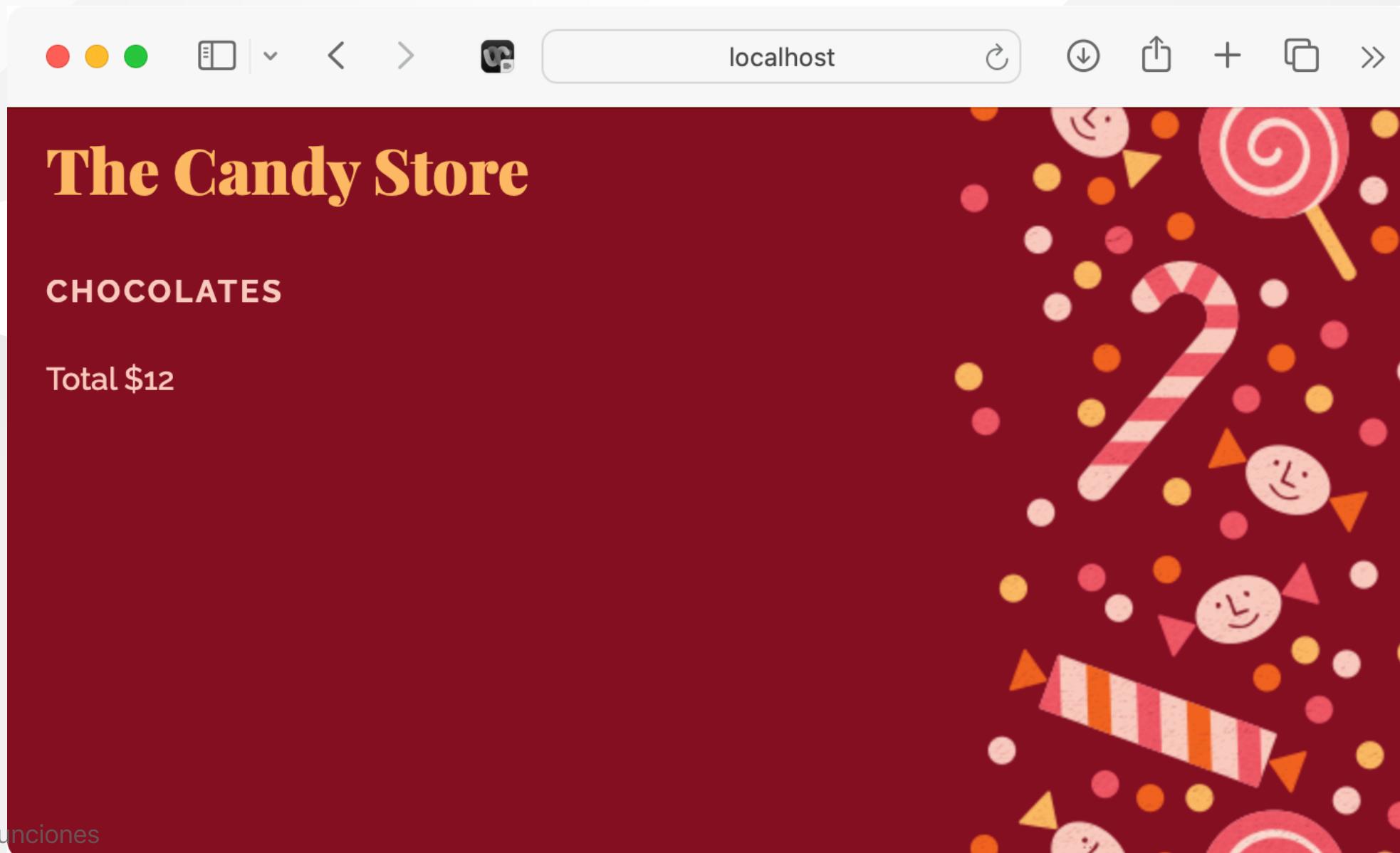
Ejemplo: Usando declaración de tipos

```
<?php
$price    = 4;
$quantity = 3;

function calculate_total(int $price, int $quantity): int
{
    return $price * $quantity;
}

$total = calculate_total($price, $quantity);
?>
<!DOCTYPE html>
<html>
    <head>
        <title>Type Declarations</title>
        <link rel="stylesheet" href="css/styles.css">
    </head>
    <body>
        <h1>The Candy Store</h1>
        <h2>Chocolates</h2>
        <p>Total $<?= $total ?></p>
    </body>
</html>
```

Ejemplo: Usando declaración de tipos



Ejemplo: Usando declaración de tipos

Sobre el ejemplo anterior, realiza las siguientes modificaciones:

- Cambia el valor almacenado en la variable `$price` para que sea una cadena en lugar de un entero, por ejemplo

```
$price = '1';
```

Cuando actualices la página deberías ver el mismo resultado porque necesitas activar los tipos estrictos (lo vemos a continuación).

- Si estás ejecutando PHP 8, utiliza tipos de unión para indicar que los valores pueden ser enteros o números de coma flotante.

Habilitación de tipos estrictos

Cuando una definición de función tiene declaraciones de argumentos y/o tipos de retorno, puedes decirle al intérprete de PHP que **genere un error si una función es llamada utilizando un tipo de datos incorrecto, o si devuelve un tipo de datos incorrecto.**

Habilitación de tipos estrictos

Cuando los argumentos de una función son del tipo de datos escalar incorrecto, el intérprete de PHP puede intentar convertirlos al tipo de datos que espera recibir.

Por ejemplo, para ayudarle a intentar procesar datos, el intérprete de PHP puede convertir

- La cadena '1' al entero 1.
- Un valor booleano de true en un entero de 1.
- Un valor booleano de false en un entero de 0.
- Un entero de 1 en un valor booleano de true.
- Un entero de 0 en un valor booleano de false.

Estos son ejemplos de manipulación de tipos, que se introdujo en la unidad 1.

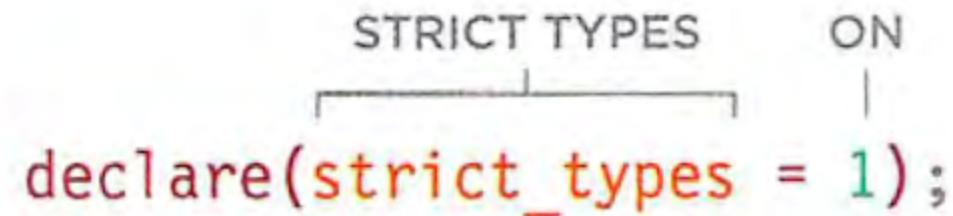
Habilitación de tipos estrictos

Podemos indicar al intérprete de PHP que habilite tipos estrictos, de forma que emita un error si una función:

- **Es llamada utilizando un argumento que es del tipo de datos incorrecto** (cuando se utilizó una declaración de tipo de argumento).
- **Devuelve un valor que es un tipo de datos incorrecto** (cuando se proporcionó una declaración de tipo de retorno).

Habilitación de tipos estrictos

Los errores que genera pueden ayudar a rastrear el origen de los problemas en el código PHP, pero el intérprete PHP **necesita que se le diga que compruebe los tipos en la página que llama a la función** utilizando la siguiente construcción de declaración:



The diagram shows the PHP declaration for strict types. It consists of two parts: 'STRICT TYPES' above a horizontal line and 'ON' to its right. A vertical line descends from the end of the 'ON' text to the word 'strict_types' in the declaration below. The declaration itself is written in red text: 'declare(strict_types = 1);'. The 'strict_types' part is underlined.

```
STRICT TYPES      ON
|
declare(strict_types = 1);
```

Esta debe ser **la primera declaración en la página**, y sólo activa tipos estrictos para funciones llamadas en esa página.

Habilitación de tipos estrictos

A veces las definiciones de funciones se colocan en un fichero *include* para que puedan ser llamadas por varias páginas de un sitio (como veremos en la unidad 6).

Los tipos estrictos **no necesitan estar habilitados en un fichero que sólo contenga definiciones de funciones**, pero deben estar habilitados en las páginas **que llamen a las funciones** si quiere que el intérprete PHP genere un error cuando se utilice un tipo de datos incorrecto.

Ejemplo: Usando tipos estrictos

Este ejemplo es casi el mismo que el anterior, pero **se habilitan los tipos estrictos en la primera sentencia para mostrar un error si los argumentos o el valor de retorno son del tipo de datos incorrecto.**

1. La construcción `declare` **habilita los tipos estrictos para la página.**
2. Se declaran dos variables para contener un precio y una cantidad.
3. Se define la función `calculate_total()`. Las:
 - **Declaración de tipo de argumento** indica que ambos parámetros esperan enteros.
 - **La declaración del tipo de retorno** especifica que la función devuelve un número entero.

Ejemplo: Usando tipos estrictos

4. La función multiplica el precio por la cantidad y devuelve este valor.
5. Se llama a la función, utilizando las variables creadas en el Paso 2, y el resultado que se devuelve se almacena en una variable llamada `$total`.
6. Se escribe el total en la página.

Ejemplo: Usando tipos estrictos

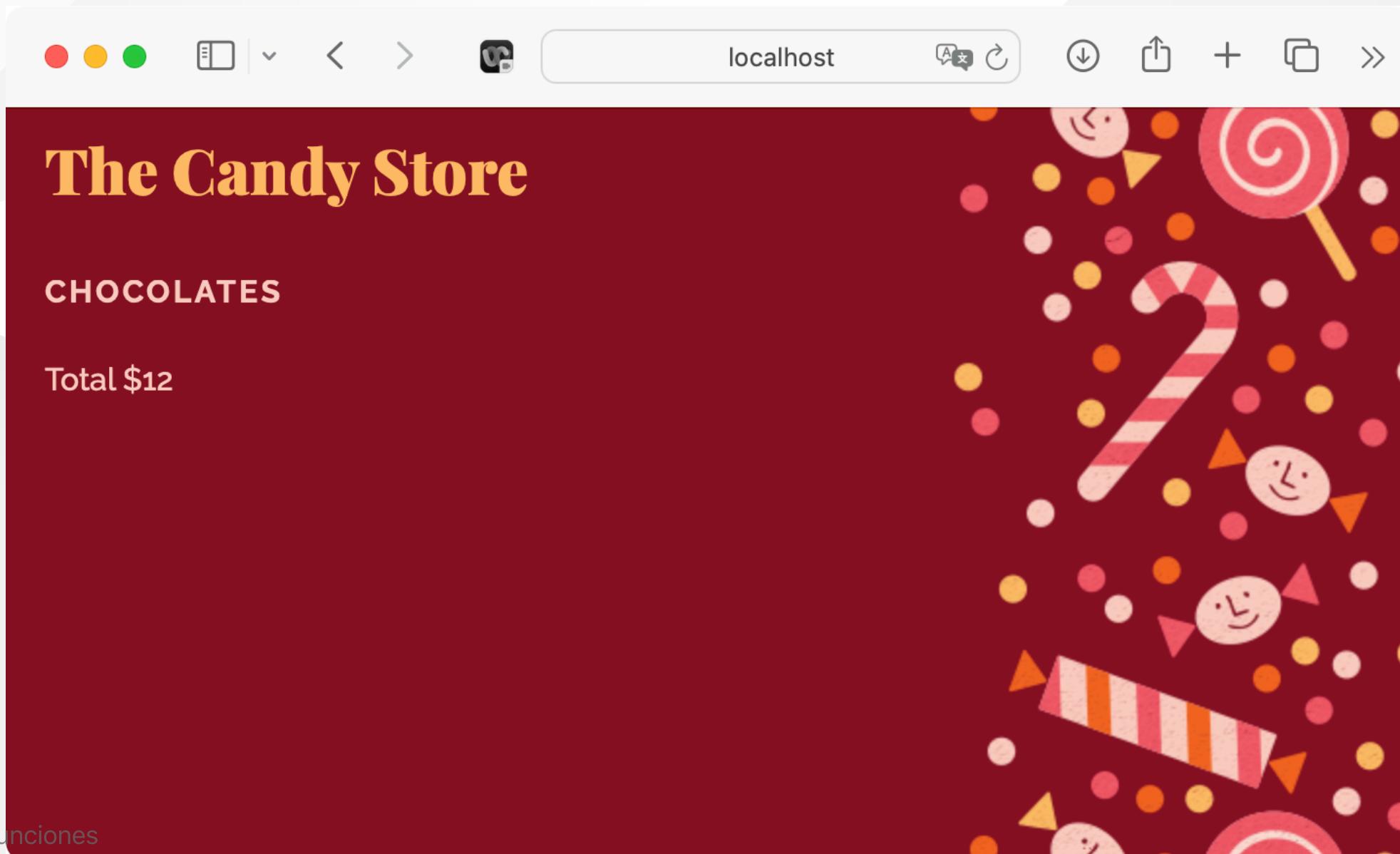
```
<?php
declare(strict_types = 1);

$price      = 4;
$quantity   = 3;

function calculate_total(int $price, int $quantity) : int {
    return $price * $quantity;
}

$total = calculate_total($price, $quantity);
?>
<!DOCTYPE html>
<html>
    <head>
        <title>Return Type Declarations</title>
        <link rel="stylesheet" href="css/styles.css">
    </head>
    <body>
        <h1>The Candy Store</h1>
        <h2>Chocolates</h2>
        <p>Total $<?= $total ?></p>
    </body>
</html>
```

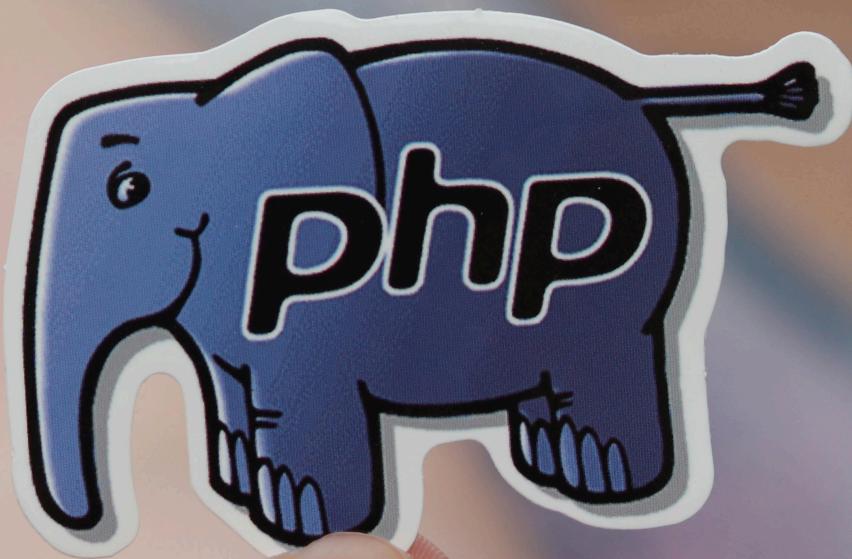
Ejemplo: Usando tipos estrictos



Ejemplo: Usando tipos estrictos

Sobre el ejemplo anterior, realiza las siguientes modificaciones:

- En el paso 2, establezca el valor de la variable \$price en una cadena: \$price = '4';
Cuando actualices la página, se mostrará un mensaje de error.
- Si estás ejecutando PHP 8, en el Paso 2 utiliza 4.5 para el precio, y en el Paso 3 utiliza tipos de unión para especificar que los argumentos y los valores de retorno pueden ser un *int* o un *float*.



Retornos múltiples y valores por defecto

Múltiples sentencias *return*

Las funciones pueden **devolver valores diferentes dependiendo del resultado de una sentencia condicional** dentro de la función.

La siguiente función devuelve diferentes mensajes dependiendo del valor pasado como argumento.

Tan pronto como una función ha procesado una sentencia return, **el intérprete PHP vuelve a la línea de código que llamó a la función**. Ninguna de las sentencias subsiguientes en esa función se ejecuta.

Múltiples sentencias *return*

La siguiente función tiene tres sentencias *return*:

1. Una condición comprueba si el valor del parámetro `$stock` es 10 o más. Si es así, se procesa la primera sentencia *return* y no se ejecuta ninguna sentencia posterior de la función.
2. Si el valor es mayor que 0 y menor que 10, se procesa la segunda sentencia *return*, y no se ejecuta ningún código posterior en la función.
3. Si la función aún se está ejecutando, `$stock` debe tener un valor de 0 para que se procese la última sentencia *return*.

Múltiples sentencias *return*

```
function get_stock_message($stock)
{
    if ($stock >= 10) {
        return 'Good availability';
    }
    if ($stock > 0 && $stock < 10) {
        return 'Low stock';
    }
    return 'Out of stock';
}
```



The code illustrates a function named `get_stock_message` that takes a parameter `$stock`. It contains three `return` statements. The first two are grouped by a brace and are labeled with circled numbers 1 and 2. The third return statement is also labeled with circled number 3 and is not grouped by a brace, indicating it is the final return value of the function.

Ejemplo: Múltiples sentencias *return*

1. La variable `$stock` contiene el nivel de existencias.
2. La función `get_stock_message()` comprueba el nivel de existencias y devuelve uno de los tres mensajes.
3. Una sentencia condicional comprueba si el stock es mayor o igual que 10. Si es así, se procesa la primera sentencia de retorno. En caso afirmativo, se procesa la primera sentencia `return`. Devuelve el mensaje "Good availability" y no se ejecuta más código de la función.

Ejemplo: Múltiples sentencias *return*

4. Si el stock no fuera 10 o más, la función seguiría ejecutándose y la siguiente condición comprobaría si el stock es mayor que 0 y menor que 10. Si es así, se procesa la segunda sentencia return. Si es así, se procesa la segunda sentencia return. Devuelve el mensaje "Low stock", y no se ejecuta más código de la función.
5. Si la función sigue ejecutándose, no puede haber existencias, por lo que la sentencia return final devuelve un mensaje que dice "Out of stock".
6. Se llama a la función y el valor devuelto se escribe en la página.

Ejemplo: Múltiples sentencias *return*

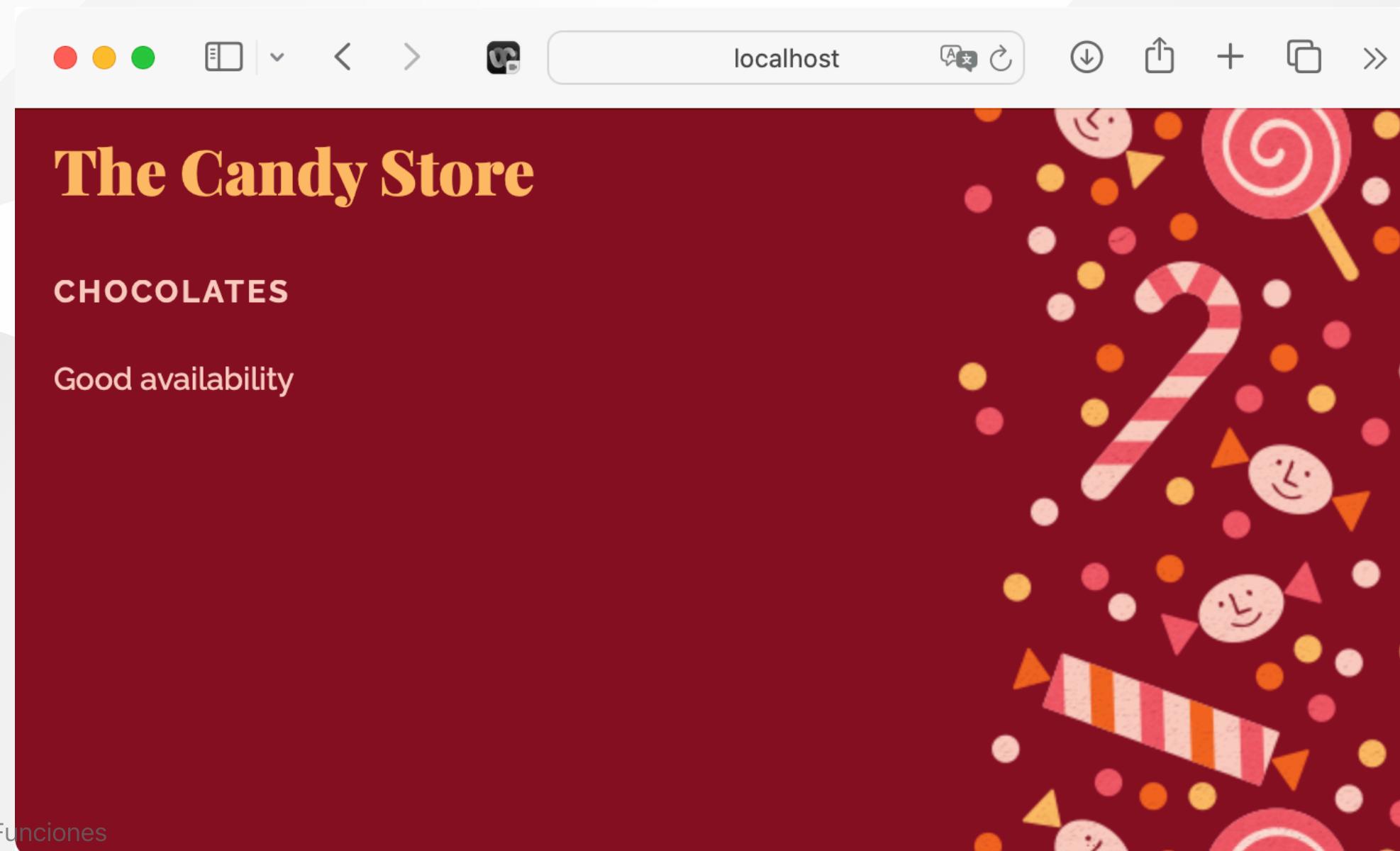
```
<?php
$stock = 25;

function get_stock_message($stock)
{
    if ($stock >= 10) {
        return 'Good availability';
    }
    if ($stock > 0 && $stock < 10) {
        return 'Low stock';
    }
    return 'Out of stock';
}
?>
```

Ejemplo: Múltiples sentencias *return*

```
<!DOCTYPE html>
<html>
  <head>
    <title>Multiple Return Statements</title>
    <link rel="stylesheet" href="css/styles.css">
  </head>
  <body>
    <h1>The Candy Store</h1>
    <h2>Chocolates</h2>
    <p><?= get_stock_message($stock) ?></p>
  </body>
</html>
```

Ejemplo: Múltiples sentencias *return*



Ejemplo: Múltiples sentencias *return*

Sobre el ejemplo anterior, realiza las siguientes modificaciones:

- Modificar la función `get_stock_message()` para que retorne "Exactly 10 items in stock" cuando el stock sea exactamente 10.
- Crear una tabla HTML que muestre los mensajes de stock para varios productos utilizando la función `get_stock_message()`.

Parametros opcionales y valores por defecto

Algunas tareas pueden tener información opcional. Estos datos **no son necesarios para que la función realice su trabajo**, pero se puede proporcionar un valor cuando se llama a la función.

Para que un parámetro sea opcional, se le da un valor por defecto, que **se utiliza cuando se llama a la función sin proporcionar un valor para ese parámetro**.

El valor por defecto se indica después del nombre del parámetro en la definición de la función. La sintaxis es la misma que si asignara un valor a una variable.

Parametros opcionales y valores por defecto

La función que se muestra a continuación es llamada utilizando dos argumentos, por lo que el parámetro final utilizaría el valor por defecto de 0.

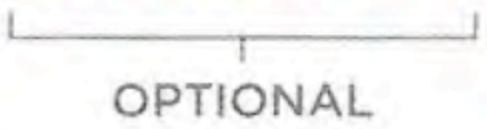
Los parámetrosopcionales se colocan después de los parámetros requeridos porque, hasta PHP 8, cuando se llamaba a una función, los argumentos tenían que estar en el orden en que los parámetros estaban listados en la definición de la función.

Aprenderemos sobre los parámetros con nombre de PHP 8 a continuación, pero es probable que los desarrolladores continúen colocando los parámetrosopcionales después de los requeridos por pura convención.

Parametrosopcionalesyvalorespordefecto

```
function calculate_cost($cost, $quantity, $discount = 0)
{
    $cost = $cost * $quantity;
    return $cost - $discount;
}

$cost = calculate_cost(5, 3);
```



OPTIONAL

Ejemplo: Parámetros opcionales y valores por defecto

1. La función `calculate_cost()` calcula el coste de uno o más artículos basándose en tres datos, el:

- Coste
- La cantidad
- Descuento

Al llamar a la función, **el último argumento es opcional porque el parámetro recibe un valor por defecto de 0**. La función se llama entonces tres veces.

2. La primera vez que se llama, se le da un coste de 5, una cantidad de 10 y un descuento de 5, por lo que la función restará 5 del total de 50 y devolverá 45.

Ejemplo: Parámetros opcionales y valores por defecto

3. La segunda vez que se llama a la función, se le da un coste de 3 y una cantidad de 4, pero no se proporciona el descuento, por lo que se aplica el valor por defecto de 0. Como resultado, la función devuelve un coste de 5, una cantidad de 10 y un descuento de 5. Como resultado, la función devuelve un coste de 12.
4. La tercera vez que se llama, se le da un coste de 4, una cantidad de 15 y un descuento de 20. Al igual que en el paso 2, restará un descuento (esta vez es 20) del coste (que es 60) para devolver un valor de 40.

Ejemplo: Parámetros opcionales y valores por defecto

```
<?php
function calculate_cost($cost, $quantity, $discount = 0)
{
    $cost = $cost * $quantity;
    return $cost - $discount;
}
?>
<!DOCTYPE html>
<html>
    <head>
        <title>Default Values for Parameters</title>
        <link rel="stylesheet" href="css/styles.css">
    </head>
    <body>
        <h1>The Candy Store</h1>
        <h2>Chocolates</h2>
        <p>Dark chocolate $<?= calculate_cost(5, 10, 5) ?></p>
        <p>Milk chocolate $<?= calculate_cost(3, 4) ?></p>
        <p>White chocolate $<?= calculate_cost(4, 15, 20) ?></p>
    </body>
</html>
```

Ejemplo: Parámetros opcionales y valores por defecto

The screenshot shows a web browser window with the address bar set to "localhost". The main content is a page titled "The Candy Store" with a red background. On the left, there's a section titled "CHOCOLATES" listing three items: "Dark chocolate \$45", "Milk chocolate \$12", and "White chocolate \$40". To the right of this text is a decorative graphic featuring a large pink and white striped candy cane, several smaller lollipops, and various colorful, smiling circular and triangular shapes against a dark red background.

The Candy Store

CHOCOLATES

Dark chocolate \$45

Milk chocolate \$12

White chocolate \$40

localhost

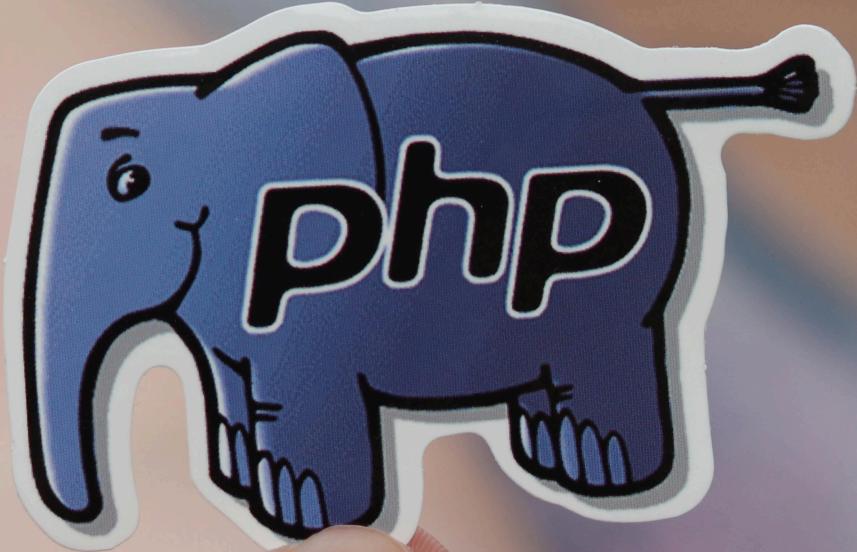
DWES - U3. Funciones

134

Ejemplo: Parámetros opcionales y valores por defecto

Sobre el ejemplo anterior, realiza las siguientes modificaciones:

- Añade más productos con diferentes costes, cantidades y descuentos, utilizando la función `calculate_cost()` para calcular los costos totales.
- Modificar la función `calculate_cost()` para incluir un parámetro opcional adicional para los impuestos (por defecto, 0).
- Crear una tabla HTML que muestre los costos calculados para varios productos con diferentes combinaciones de parámetros.



Argumentos con nombre

Argumentos con nombre

Cuando se llama a una función en **PHP 8**, se pueden **anteponer los nombres de los parámetros a los argumentos**.

Esto significa que **no es necesario dar los argumentos en el mismo orden** en que aparecen los nombres de los parámetros en la **definición de la función**.

Argumentos con nombre

Algunas funciones tienen muchos parámetros. En PHP 8, cuando se llama a una función, se pueden añadir nombres de parámetros antes de los argumentos. Estos se llaman **argumentos con nombre** (o parámetros con nombre). Estos:

- Indican claramente lo que hace cada argumento.
- Nos permiten **omitar argumentos opcionales sin dar un valor por defecto o utilizando comillas vacías** (se muestra a continuación).

Al llamar a una función sin argumentos con nombre, los **argumentos deben aparecer en el mismo orden que los parámetros en la definición de la función**.

Argumentos con nombre

La definición de la función no cambia, sólo la forma en que se proporcionan los argumentos cuando se llama. El ejemplo que veremos a continuación tiene cuatro parámetros:

- `$cost` (obligatorio) es el coste de un artículo
- `$quantity` (obligatorio) es el número de ese artículo
- `$discount` (opcional) es la cantidad a descontar
- `$tax` (opcional) es el porcentaje de impuestos a pagar

Argumentos con nombre

Para utilizar el valor por defecto para `$discount` y luego especificar un valor para `$tax`, se debe especificar el valor por defecto o utilizar comillas vacías como argumento porque el parámetro `$discount` viene antes de `$tax`.

```
calculate_cost(5, 10, 0, 5); or calculate_cost(5, 10, '', 5);
```

Argumentos con nombre

Cuando se utilizan argumentos con nombre, **el nombre se separa del argumento mediante dos puntos** y los argumentos **pueden estar en cualquier orden**.

Si se supone que un argumento utiliza su valor por defecto (especificado en la definición de la función) no es necesario proporcionarle un valor ni utilizar comillas en blanco:

```
calculate_cost(quantity: 10, cost: 5, tax: 5);
```

Argumentos con nombre

Los argumentos sin nombre de parámetro pueden aparecer antes que los argumentos con nombre si aparecen en el mismo orden que los parámetros en la definición de la función.

A continuación, se utilizarán los dos primeros valores para coste y cantidad; después se proporciona el impuesto; y observa que no se especifica el valor para el descuento:

```
calculate_cost(5, 10, tax: 5);
```

Ejemplo: Argumentos con nombre

1. La función `calculate_cost()` calcula el coste de uno o más artículos basándose en cuatro datos; el:

- **Coste** (obligatorio)
- **Cantidad** (obligatoria)
- **Descuento** (opcional - por defecto 0)
- **Impuesto** (opcional - por defecto 20%)

Dado que este ejemplo utiliza PHP 8, se puede añadir una coma después del último parámetro en la definición de la función (no sólo entre los parámetros). Esto mejora la uniformidad en el código (porque una coma puede aparecer después de cada parámetro).

Ejemplo: Argumentos con nombre

Después de que la función ha sido definida, es llamada tres veces.

2. La función se llama **con cuatro argumentos con nombre**. Como son nombrados, los argumentos pueden aparecer en cualquier orden.
3. Los argumentos con nombre se utilizan para el coste, la cantidad y el impuesto. Se utiliza **el descuento por defecto**.
4. Los **dos primeros valores no utilizan argumentos con nombre**, por lo que se utilizan para los dos primeros parámetros (`$cost` y `$quantity`). No se da ningún valor para `$discount` por lo que el argumento final debe ser nombrado para que pueda ser utilizado para el parámetro `$tax` .

Ejemplo: Argumentos con nombre

```
<?php
function calculate_cost($cost, $quantity, $discount = 0, $tax = 20,
{
    $cost = $cost * $quantity;
    $tax = $cost * ($tax / 100);
    return ($cost + $tax) - $discount;
}
?>
<!DOCTYPE html>
<html>
<head>
    <title>Default Values for Parameters</title>
    <link rel="stylesheet" href="css/styles.css">
</head>
<body>
    <h1>The Candy Store</h1>
    <h2>Chocolates</h2>
    <p>Dark chocolate $<?= calculate_cost(quantity: 10, cost: 5, tax: 5, discount: 2) ?></p>
    <p>Milk chocolate $<?= calculate_cost(quantity: 10, cost: 5, tax: 5) ?></p>
    <p>White chocolate $<?= calculate_cost(5, 10, tax: 5) ?></p>
</body>
</html>
```

Ejemplo: Argumentos con nombre

The screenshot shows a web browser window with the address bar set to "localhost". The main content is a page titled "The Candy Store" with a red background. On the left, there's a section titled "CHOCOLATES" listing three items: "Dark chocolate \$50.5", "Milk chocolate \$52.5", and "White chocolate \$52.5". To the right of this text is a decorative graphic featuring a large pink and white striped candy cane, several smaller colorful lollipops, and various small, smiling circular faces and geometric shapes.

The Candy Store

CHOCOLATES

Dark chocolate \$50.5

Milk chocolate \$52.5

White chocolate \$52.5

localhost

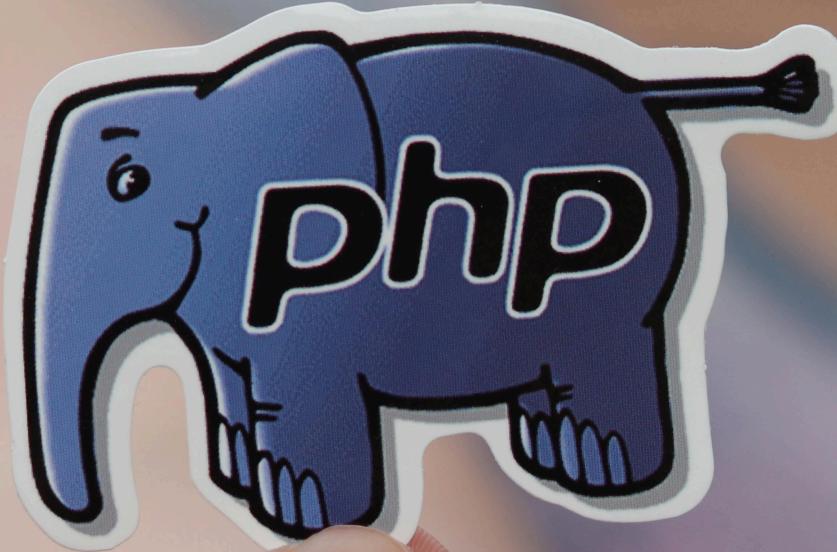
DWES - U3. Funciones

146

Ejemplo: Argumentos con nombre

Sobre el ejemplo anterior, realiza las siguientes modificaciones:

- Añadir más productos con diferentes costos, cantidades, descuentos y tasas de impuestos, utilizando la función `calculate_cost()` con argumentos con nombre para calcular los costos totales.
- Modificar la función `calculate_cost()` para incluir un parámetro opcional adicional para el envío (`$shipping`), por defecto 0.
- Crear una tabla HTML que muestre los costos calculados para varios productos con diferentes combinaciones de parámetros.



Diseño y documentación de funciones

Cómo enfocar la creación de una función

Seguir estos cuatro pasos le ayudará a escribir funciones.

1: Describe brevemente la tarea

Utiliza una combinación de lo que hace (por ejemplo, obtener, calcular, actualizar o guardar) seguida del tipo de datos con los que trabaja. Esto se convertirá en el **nombre de la función**.

El nombre de la función nunca cambia.

Cómo enfocar la creación de una función

2: ¿Qué datos se necesitan para realizar la tarea?

Cada dato se convierte en un **parámetro**.

Los valores pasados al parámetro (los argumentos) pueden cambiar cada vez que se llama a la función.

Cómo enfocar la creación de una función

3: ¿Qué instrucciones se deben seguir para realizar la tarea?

Las instrucciones se representarán utilizando **sentencias** dentro de las llaves que contienen el bloque de código de la función.

Las instrucciones que se sigan serán las mismas cada vez que se llame a la función.

Cómo enfocar la creación de una función

4: ¿Cuál es el resultado esperado?

Será el valor devuelto por la función. Se considera una buena práctica que una función devuelva **un valor**. Si estás realizando una tarea que no calcula un nuevo valor o recupera alguna información, entonces la función devolverá a menudo *verdadero* o *falso* para indicar si ha funcionado o no.

El valor que se devuelve puede cambiar cada vez que la función recibe nuevos valores con los que trabajar.

¿Por qué usar funciones?

Escribir el código que realiza una tarea en una función tiene sus ventajas.

Reusabilidad

Como has visto varias veces en esta unidad, si una página necesita realizar la misma tarea varias veces (como calcular el coste de los artículos), entonces **sólo necesitas escribir el código para realizar esta tarea una vez**.

Cuando la página necesita realizar la tarea, llama a la función y le da los valores necesarios para hacer su trabajo.

¿Por qué usar funciones?

Mantenibilidad

Si las instrucciones necesarias para realizar una tarea cambian, **sólo hay que cambiar el código de la definición de la función** (no es necesario hacer cambios cada vez que se realiza la tarea).

Una vez actualizada la definición de la función, cada vez que se llame a esa función, ésta utilizará el código actualizado.

¿Por qué usar funciones?

Organización

Al colocar el código que realiza cada tarea en una función, **resulta más fácil encontrar todas las sentencias necesarias para realizar esa tarea.**

Testeabilidad

Al dividir el código en las tareas individuales que realiza, **podemos probar cada tarea por separado**, lo que facilita el **aislamiento de problemas**.

Documentando funciones

A menudo, los programadores necesitan utilizar funciones que no han escrito ellos; por ejemplo, cuando trabajan en un sitio de gran tamaño en un equipo de programadores.

La documentación ayuda a los programadores a aprender a utilizar esas funciones.

Documentando funciones

Para utilizar una función en tu página PHP, no necesitas entender cómo las declaraciones que aparecen en las llaves logran la tarea, sólo necesitas saber:

- **Qué debe hacer la función**
- **El nombre de la función**
- **Los parámetros que requiere**
- **Lo que debe devolver**

Documentando funciones

En la siguiente figura, puedes ver una página del [sitio web PHP.net](#). Es el hogar oficial de PHP y alberga la documentación del lenguaje PHP (un recurso muy útil mientras aprendes el lenguaje).

En ella se muestra una función que determina cuántos caracteres hay en una cadena. Esta página es un ejemplo típico de cómo se documentan las funciones. Normalmente verás:

1. El **nombre** de la función y su **descripción**
2. La **sintaxis para llamar a la función y utilizar sus parámetros** (pueden mostrarse los tipos de argumento y de retorno)
3. La **descripción de sus parámetros**
4. Qué **valor o valores devolverá la función**
5. **Ejemplo** de utilización de la función

Documentando funciones

strlen

(PHP 4, PHP 5, PHP 7, PHP 8)

strlen — Get string length

Description

```
strlen(string $string): int
```

Returns the length of the given **string**.

Parameters

string

The string being measured for length.

Return Values

The length of the **string** in bytes.

Examples

Example #1 A strlen() example

Documentando funciones

Es importante distinguir entre dos tipos de funciones:

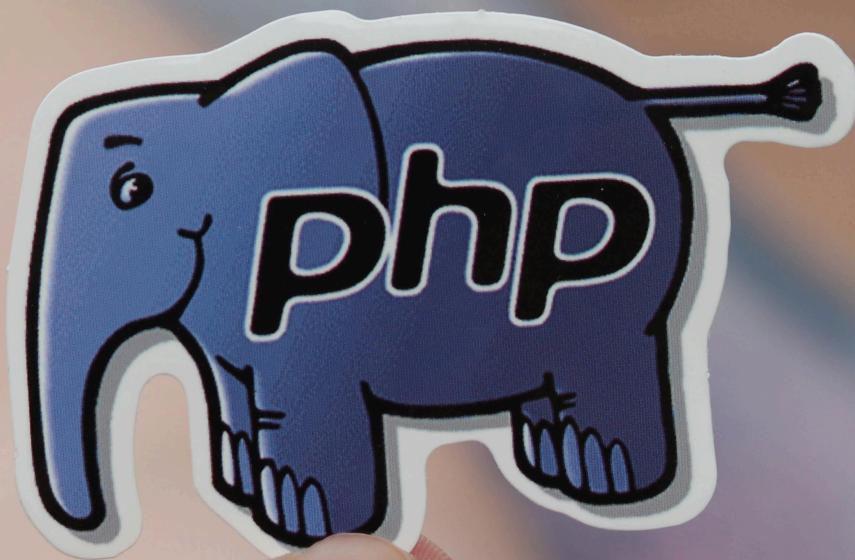
- Las **funciones definidas por el usuario** son funciones que son definidas en un archivo PHP por un programador que está utilizando el lenguaje PHP. (Todas las funciones en esta unidad han sido funciones definidas por el usuario).
- Las **funciones incorporadas (*built-in*)** son definidas por la gente que crea el lenguaje PHP, y sus definiciones son implementadas en el intérprete PHP. Esto significa que cualquiera puede llamar a esas funciones sin incluir la definición de la función en una página.

Documentando funciones

Las funciones incorporadas realizan **tareas que los programadores necesitan realizar habitualmente** al escribir código PHP.

Por lo tanto, **evitan que los programadores tengan que escribir su propio código** para realizar estas tareas, reinventando la rueda cada vez.

Aprenderemos más sobre las funciones incorporadas en la unidad 5.



Ejemplo final

Ejemplo final

Este ejemplo muestra una **página para monitorizar los niveles de stock en una tienda de golosinas.**

Se crea un array asociativo para contener los nombres de los productos que vende la tienda y los niveles de stock de cada uno. Estos valores se muestran en las dos primeras columnas de la tabla.

Ejemplo final

A continuación, se crean tres funciones para generar los valores que se muestran en las tres columnas siguientes:

- La primera función **mira los niveles de stock y crea un mensaje** indicando si se debe pedir más stock o no.
- La segunda **calcula el valor total de las existencias** de cada artículo vendido.
- La tercera **calcula el importe de los impuestos que habrá que pagar** cuando se hayan vendido todas las existencias restantes.

Ejemplo final

1. Los **tipos estrictos** están habilitados.
2. Se crea un **array multidimensional** y se almacena en una variable llamada `$candy` :
 - Las **claves** son los nombres de los tipos de caramelos vendidos.
 - Los **valores** son arrays que contienen el precio y el nivel de stock disponible de ese producto
3. Se declara una **variable global** para contener el tipo impositivo.
4. Se define una función llamada `get_reorder_message()` . Tiene un parámetro, el **nivel actual de existencias** de un producto (un *int*). Devuelve un **mensaje** (un *string*) diciendo si el artículo debe ser reordenado o no.

Ejemplo final

5. Se utiliza un operador ternario para devolver un mensaje. La condición comprueba si el nivel de existencias es inferior a 10.

- Si lo es, la función devuelve Yes.
- En caso contrario, la función devuelve No.

6. Se define una función llamada `get_total_value()`. Tiene dos parámetros:

- El **precio** del producto (`float`)
- La **cantidad disponible** de este producto (`int`)

Devuelve un `float` que indica el valor total de las existencias de este producto (en este caso, un `int` también es un número válido).

Ejemplo final

7. La función devuelve el precio del producto multiplicado por la cantidad de stock disponible.
8. Se define una función llamada `get_tax_due()`. Tiene tres parámetros
 - **Precio del producto** (float)
 - **Cantidad disponible** del producto (int)
 - **Tipo impositivo** como porcentaje con un valor por defecto de 0% (int)

Devuelve un *float*, **indicando la cantidad total de impuestos que se devengarán** cuando se vendan estos productos.

Ejemplo final

9. Se devuelve el **impuesto total a pagar**. Para calcularlo, se multiplica el valor total de las existencias (precio de un artículo multiplicado por la cantidad disponible) por el porcentaje de impuestos (el tipo impositivo dividido por 100).

Ejemplo final

10. Un bucle `foreach` recorre los productos del array almacenado en `$candy`. Entre paréntesis:

- `$candy` es la variable que almacena el array del Paso 2
- `$product_name` es el nombre de la variable que contendrá la **clave para el elemento actual** del array (el nombre del producto: caramelos, caramelos de menta o caramelos de azúcar).
- `$data` es la variable que representará el **valor del elemento actual**. Es el array que almacena el precio y el nivel de stock de ese producto.

Ejemplo final

11. Se crea una fila de tabla, y el nombre del producto que el bucle está procesando actualmente se escribe en un elemento `<td>`.
12. `$data` contiene un array con el precio y el nivel de stock de ese producto; el nivel de stock se escribe en la siguiente celda de la tabla.
13. Se llama a la función `get_reorder_message()`. El nivel de existencias se pasa como argumento. El valor devuelto se muestra en la tabla.
14. Se llama a la función `get_total_value()`. El primer parámetro es el precio del producto. El segundo parámetro es la cantidad disponible. El valor devuelto se escribirá en la tabla.

Ejemplo final

15. Se llama a la función `get_tax_due()`.

El primer parámetro es el precio del producto. El segundo parámetro es la cantidad disponible. El tercer parámetro es el tipo impositivo almacenado en el paso 3. El valor devuelto se escribirá en la tabla.

16. La llave de cierre termina el bloque de código, y el **bucle se repite para cada elemento del array**.

Ejemplo final

```
<?php  
① declare(strict_types = 1);  
② $candy = [  
    'Toffee' => ['price' => 3.00, 'stock' => 12],  
    'Mints'  => ['price' => 2.00, 'stock' => 26],  
    'Fudge'  => ['price' => 4.00, 'stock' => 8],  
];  
③ $tax = 20;  
  
④ function get_reorder_message(int $stock): string  
{  
⑤     return ($stock < 10) ? 'Yes' : 'No';  
}  
  
⑥ function get_total_value(float $price, int $quantity): float  
{  
⑦     return $price * $quantity;  
}  
  
⑧ function get_tax_due(float $price, int $quantity, int $tax = 0): float  
{  
⑨     return ($price * $quantity) * ($tax / 100);  
}  
?>
```

Ejemplo final

```
<!DOCTYPE html>
<html>
  <head> ... </head>
  <body>
    <h1>The Candy Store</h1>
    <h2>Stock Control</h2>
    <table>
      <tr>
        <th>Candy</th><th>Stock</th><th>Re-order</th><th>Total value</th><th>Tax due</th>
      </tr>
      ⑩      <?php foreach ($candy as $product_name => $data) { ?>
      ⑪      <tr>
      ⑫          <td><?= $product_name ?></td>
      ⑬          <td><?= $data['stock'] ?></td>
      ⑭          <td><?= get_reorder_message($data['stock']) ?></td>
      ⑮          <td>$<?= get_total_value($data['price'], $data['stock']) ?></td>
      ⑯          <td>$<?= get_tax_due($data['price'], $data['stock'], $tax) ?></td>
      </tr>
      ⑯      <?php } ?>
      </table>
    </body>
  </html>
```

Ejemplo final

The screenshot shows a web browser window with the URL "localhost" in the address bar. The page has a dark red background with a festive candy-themed illustration on the right side featuring a large candy cane, lollipops, and colorful circles. The title "The Candy Store" is displayed in a large, gold-colored serif font. Below it, the heading "STOCK CONTROL" is shown in a smaller, white, sans-serif font. A table with a white border and light gray header row lists the stock information for three products: Toffee, Mints, and Fudge. The table columns are labeled "PRODUCT", "STOCK", "RE-ORDER", "TOTAL VALUE", and "TAX DUE".

PRODUCT	STOCK	RE-ORDER	TOTAL VALUE	TAX DUE
Toffee	12	No	\$36	\$7.2
Mints	26	No	\$52	\$10.4
Fudge	8	Yes	\$32	\$6.4

Actividad propuesta

Crea un **sistema de gestión de inventarios para una librería**, donde se deberán implementar funciones que calculen el total de libros en stock, el valor total de los libros en inventario y el importe de impuestos a pagar.

Actividad propuesta

Instrucciones:

1. Configuración Inicial

- i. Crea un archivo PHP llamado `inventory.php`.
- ii. Configura la declaración de tipos estrictos.

2. Datos del inventario

- i. Define un array asociativo `$books` que contenga la siguiente información para cada libro:
 - a. Título del libro
 - b. Precio
 - c. Stock (cantidad en inventario)

Actividad propuesta

3. Tasa de Impuesto

- i. Define una variable `$tax_rate` con un valor de 12 (que representa el 12%).

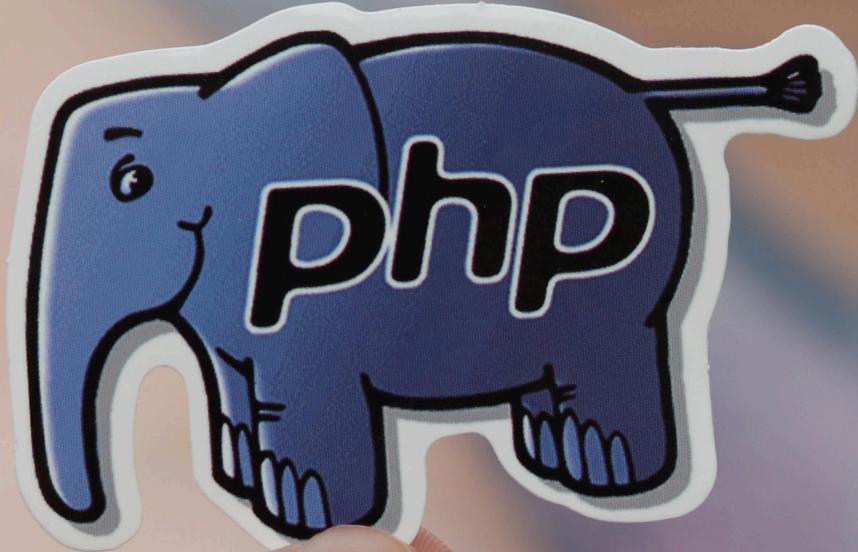
4. Funciones a Implementar

- i. `get_total_stock` : Recibe un arreglo de libros y devuelve el total de libros en inventario.
- ii. `get_inventory_value` : Recibe el precio y la cantidad en inventario de un libro y devuelve el valor total en inventario.
- iii. `calculate_tax` : Recibe el valor total de inventario y la tasa de impuesto, y devuelve el importe del impuesto a pagar.

Actividad propuesta

5. Interfaz Web

- Crea una estructura HTML básica que muestre los detalles de los libros en una tabla y los cálculos realizados mediante las funciones PHP.
- La página debe incluir los siguientes datos:
 - Título del libro
 - Precio
 - Stock
 - Valor total en inventario
 - Impuesto a pagar



Resumen de la unidad

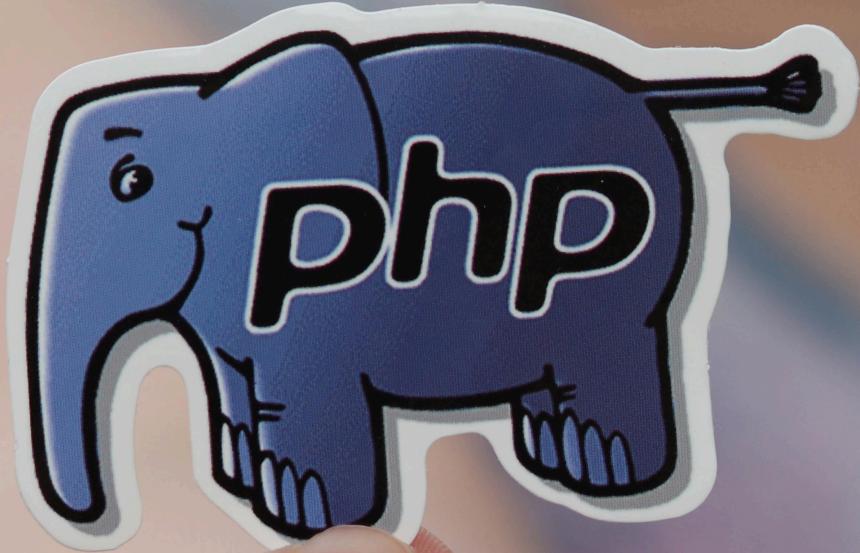
Resumen de la unidad

- Las **definiciones de función** dan un nombre a una función y utilizan un bloque de código para almacenar las sentencias para realizar una tarea.
- **Llamar** a una función le dice al intérprete de PHP que ejecute esas sentencias para realizar la tarea.
- La palabra clave `return` **envía datos de vuelta desde una función**.
- Los **parámetros** representan los datos que una función necesita para realizar su tarea. Los nombres de los parámetros actúan como variables en la función.

Resumen de la unidad

- Cuando una función se ha ejecutado, los parámetros y cualquier variable declarada en la función **se borran**.
- Cuando se llama a una función, los valores utilizados para los parámetros se conocen como **argumentos**.
- Las **declaraciones de tipo** especifican el tipo de datos de los argumentos.
- Los **tipos de retorno** especifican el tipo de datos que devuelve una función.
- Si un parámetro es **opcional**, se le asigna un **valor por defecto**.





Referencias

Referencias

- [PHP functions \(simplilearn.com\)](#)
- [Tipo de dato mixed \(php.net\)](#)
- [Parámetros/Argumentos con nombre \(blog php.watch\)](#)
- [PHP: Strict Types \(php.net\)](#)
- [PHP: Variables Globales y Estáticas \(php.net\)](#)

Bloque A

Instrucciones básicas de programación

Unidad 3. Funciones

