Chapter 13

How to create and use functions

Objectives

Applied

1. Create any of the functions that your applications require. These functions may need to pass arguments by value or reference, provide default values for arguments, or provide for a variable number of arguments.
2. Call any of the functions that your applications require.
3. Create and use function libraries and namespaces.

Objectives (continued)

Knowledge

1. Describe the creation and use of functions.
2. Distinguish between passing an argument by value and passing an argument by reference.
3. Describe local scope and global scope as it applies to the variables within functions, and describe the scope of functions themselves.
4. Describe the use of function libraries and namespaces.

The syntax for a function

```
function function_name([$param_1,
                        $param_2, ... ,
                        $param_n]) {
    // Code for function
    [return [value];]
}
```

A function with no parameters that returns a value

```
function coin_toss() {
    $coin = (rand_int(0, 1) == 0) ? 'heads' : 'tails';
    return $coin;
}
```


A function with one parameter

```
function display_error($error) {
    echo '<p class="error">' . $error . '</p>';
}
```

A function with three parameters

```
function avg_of_3($x, $y, $z) {
    $avg = ($x + $y + $z) / 3;
    return $avg;
}
```

Calling functions that return values

```
$average = avg_of_3(5, 2, 8);  // $average is 5
echo coin_toss();              // Displays heads or tails
```


A function call that doesn't return a value

```
display_error('Value out of range.');
```

A function call that discards the return value

```
$list = array('Apples', 'Oranges', 'Grapes');
$last = array_pop($list);
array_pop($list);       // Removes Oranges
                        // Discards return value
```

Key terms

- function
- parameter
- parameter list
- return statement
- argument
- argument list
- function call
- calling a function

An argument passed by value

```
function add_3_by_val($value) {
    $value += 3;
    echo '<p>Number: ' . $value . '</p>';
}

$number = 5;
add_3_by_val($number);                    // Displays 8
echo '<p>Number: ' . $number . '</p>';    // Displays 5
```

An argument passed by reference

```
function add_3_by_ref(&$value) {
    $value += 3;
    echo '<p>Number: ' . $value . '</p>';
}

$number = 5;
add_3_by_ref($number);                    // Displays 8
echo '<p>Number: ' . $number . '</p>';    // Displays 8
```


How to modify a string that's passed by reference

```
function wrap_in_tag(&$text, $tag) {
    $before = '<'  . $tag . '>';
    $after  = '</' . $tag . '>';
    $text = $before . $text . $after;
}

$message = 'Value out of range.';
wrap_in_tag($message, 'p');
echo $message;            // <p>Value out of range.</p>
```


How to return multiple values

```
function array_analyze($array, &$sum, &$prod, &$avg) {
    $sum = array_sum($array);
    $prod = array_product($array);
    $avg = $sum / count($array);
}

$list = array(1, 4, 9, 16);
array_analyze($list, $s, $p, $a);
echo '<p>Sum: ' . $s . '<br />
         Product: ' . $p . '<br />
         Average ' . $a .
     '</p>';
```


A variable with global scope

```
$a = 10;            // $a has global scope
function show_a() {
    echo $a;        // Inside function, $a is NULL
}
show_a();           // Displays nothing
```

How to access a global variable from a function

```
$b = 10;              // $b has global scope
function show_b() {
    global $b;        // $b refers global variable $b
    echo $b;
}
show_b();             // Displays 10
```


Another way to access a global variable

```
$c = 10;                  // $c has global scope
function show_c() {
    $c = $GLOBALS['c'];   // $c refers to global $c
    echo $c;
}
show_c();                 // Displays 10
```

A variable with local scope

```
function show_d() {
    $d = 10;          // $d has local scope within function
    echo $d;
}
echo $d;              // Outside function, $d is NULL
```

Key terms

- scope
- local scope
- global scope

A function with one default parameter

```
function get_rand_bool_text($type = 'coin') {
    $rand = random_int(0, 1);
    switch ($type) {
        case 'coin':
            $result = ($rand == 1) ? 'heads' : 'tails';
            break;
        case 'switch':
            $result = ($rand == 1) ? 'on' : 'off';
            break;
    }
    return $result;
}
```


A function with an optional parameter

```
function is_leap_year($date = NULL) {
    if (!isset($date)) {
        $date = new DateTime();
    }
    if ($date->format('L') == '1') return true;
    else return false;
}
```


A function with one required and two default parameters

```
function display_error($error,
                       $tag = 'p',
                       $class = 'error') {
    $opentag  = '<'  . $tag . ' class="' . $class . '">';
    $closetag = '</' . $tag . '>';
    echo $opentag . $error . $closetag;
}
```


Calling a function with a default parameter value

Omitting optional parameters

```
echo get_rand_bool_text();
echo display_error('Out of range');
$is_leap_year = is_leap_year();
```

Providing optional parameters

```
echo get_rand_bool_text('switch');
echo display_error('Out of range', 'li');
$is_leap_year =
    is_leap_year(new DateTime('March 15, 2018'));
```


The syntax for type declarations

For the return type

```
function function_name(parameter_list) : return_type {}
```

For the parameter list

```
(type_1 $param_1, type_2 $param_2, ... , type_n $param_n)
```

A function with one parameter type declaration

```
function display_error(string $error) {
    echo '<p class="error">' . $error . '</p>';
}
```

A function with type declarations for its parameters and return value

```
function avg_of_3(int $x, int $y, int $z) : float {
    $avg = ($x + $y + $z) / 3;
    return $avg;
}
```


Function calls without strict typing

```
display_error('Value out of range.');
                    // Displays 'Value out of range.'
display_error(1);   // Displays '1'

$average = avg_of_3(5, 2, 8);          // $average is 5
$average = avg_of_3(5.1, 2.7, 8.2);    // $average is 5
```


How to enable strict types mode

```
declare(strict_types=1); // must be first line of script
```

Function calls with strict typing

```
display_error('Value out of range.');
                    // Displays 'Value out of range.'
display_error(1);   // Fatal error

$average = avg_of_3(5, 2, 8);          // $average is 5
$average = avg_of_3(5.1, 2.7, 8.2);    // Fatal error
```

A typical error message when the data type isn't correct

```
TypeError: Argument 1 passed to avg_of_3() must
be of the type integer, float given
```


Key terms

- scalar value
- type declarations
- strict types mode

Functions for working with variable-length parameter lists

```
func_get_args()
func_num_args()
func_get_arg($i)
```

A function that adds a list of numbers

```
function add() {
    $numbers = func_get_args();
    $total = 0;
    foreach($numbers as $number) {
        $total += $number;
    }
    return $total;
}

$sum = add(5, 10, 15);          // $sum is 30
```


A function that averages one or more numbers

```
function average($x) {   // $x forces one argument
    $count = func_num_args();
    $total = 0;
    for($i = 0; $i < $count; $i++) {
        $total += func_get_arg($i);
    }
    return $total / $count;
}

$avg = average(75, 95, 100);   // $avg is 90
```

Using required parameters with a variable parameter list

```
function array_append(&$array, $x) {
    $values = func_get_args();  // Also contains $array
    array_shift($values);    // Removes $array from front
    foreach($values as $value) {
        $array[] = $value;
    }
}

$data = array('apples', 'oranges');
array_append($data, 'grapes', 'pears');
```


A library of functions (the cart.php file)

```
<?php
    // Add an item to the cart
    function cart_add_item(&$cart, $name, $cost,
                           $quantity) {
        $total = $cost * $quantity;
        $item = array(
            'name' => $name,
            'cost' => $cost,
            'qty'  => $quantity,
            'total' => $total
        );
        $cart[] = $item;
    }
```


A library of functions (the cart.php file) (cont.)

```
    // Update an item in the cart
    function cart_update_item(&$cart, $key, $quantity) {
        if (isset($cart[$key])) {
            if ($quantity <= 0) {
                unset($cart[$key]);
            } else {
                $cart[$key]['qty'] = $quantity;
                $total = $cart[$key]['cost'] *
                                 $cart[$key]['qty'];
                $cart[$key]['total'] = $total;
            }
        }
    }
```


A library of functions (the cart.php file) (cont.)

```
    // Get cart subtotal
    function cart_get_subtotal($cart) {
        $subtotal = 0;
        foreach ($cart as $item) {
            $subtotal += $item['total'];
        }
        $subtotal = round($subtotal, 2);
        $subtotal = number_format($subtotal, 2);
        return $subtotal;
    }
?>
```


Code that uses the library

```
// load the library
require_once('cart.php');

// create an array to store the cart
$cart = array();

// call methods from the library
cart_add_item($cart, 'Flute', 149.95, 1);
cart_update_item($cart, 0, 2);   // update first item
$subtotal = cart_get_subtotal($cart);

// display the result
echo 'This subtotal is $' . $subtotal;
```


Functions for working with the include path

`get_include_path()`

`set_include_path(`*`$path`*`)`

The default include path

Windows

`.;C:\xampp\php\PEAR`

Mac OS X

`.:/Applications/XAMPP/xamppfiles/lib/php`

Linux

`.:/opt/lampp/lib/php`

How to get the include path

```
$include_path = get_include_path();
```

How to set the include path

Windows

```
set_include_path($include_path .
                 ';C:\xampp\htdocs\book_apps\lib');
```

Mac OS X

```
set_include_path($include_path .
        ':/Applications/XAMPP/htdocs/book_apps/lib');
```

Linux

```
set_include_path($include_path .
                 ':/opt/lampp/htdocs/book_apps/lib');
```

How to include a file after the path has been set

```
require_once cart.php;
```


How to create a namespace in a file

Using the statement syntax

```
<?php
namespace cart;
    // Functions in cart namespace
?>
```

Using the brace syntax

```
<?php
namespace cart {
    // Functions in cart namespace
}
?>
```

How to create nested namespaces

```
<?php
namespace murach\cart {
    // Functions in murach\cart namespace
}
?>
```


How to use the functions in a namespace

Create a file that contains a namespace with one function

```
<?php
namespace murach\errors {
    function log($error) {
        echo '<p class="error">' . $error . '</p>';
    }
}
?>
```

Call a function that is stored in the namespace

```
// load the file that stores the namespace
require_once 'errors.php';

// call the log function
murach\errors\log('Invalid value');

// create an alias and use it to call the log function
use murach\errors as e;
e\log('Invalid value');
```


A variable function

```
$function = (mt_rand(0,1) == 1) ?
    'array_sum' : 'array_product';
$values = array(4, 9, 16);
$result = $function($values);    // 29 for array_sum, 576 for
array_product
```


A function that uses a callback

```
function validate($data, $functions) {
    $valid = true;
    foreach ($functions as $function) {
        $valid = $valid && $function($data);
    }
    return $valid;
}

function is_at_least_18($number) {
    return $number >= 18;
}

function is_less_than_62($number) {
    return $number < 62;
}

$age = 25;
$functions = array(
    'is_numeric', 'is_at_least_18', 'is_less_than_62');
$is_valid_age = validate($age, $functions);    // TRUE
```


Language constructs that can't be use in variable functions

- die
- eval
- list
- print
- echo
- include
- require
- unset
- empty
- include_once
- require_once
- exit
- isset
- return

How variable functions and callbacks work

- A *variable function* is a function name stored in a variable as a string. When PHP encounters a variable function, it evaluates the variable and attempts to call the function.
- To call a variable function, code the variable name followed by a set of parentheses. Within the parentheses, code the argument list for the function.
- You can use a variable function when the function isn't known until runtime.
- You can use a variable function in a function that uses a callback. A *callback* is a function that's passed as an argument to another function.
- You can't use the language constructs listed above with variable functions directly. However, you can use a wrapper function that calls one of these constructs in a variable function.

A function for sorting an array with a custom comparison function

```
usort($array, $function)
```


How to create and use an anonymous function

A custom sorting function

```
$compare_function = function($left, $right) {
    $l = (float) $left;
    $r = (float) $right;
    if ($l < $r) return -1;
    if ($l > $r) return 1;
    return 0;
};
```

Code that tests the custom sorting function

```
$a = 3;
$b = 5;
$result = $compare_function($a, $b);  // -1
```

Code that uses the custom sorting function

```
$values = array(5, 2, 4, 1, 3);
usort($values, $compare_function);    // 1, 2, 3, 4, 5
```


The spaceship operator

Operator	Description
<=>	Returns -1 if the left operand is less than the right operand, returns 0 if the two operands are equal, and returns 1 if the left operand is greater than the right operand.

A custom sorting function that uses the spaceship operator and type declarations

```
$compare_function = function(float $left,
                             float $right) {
    return $left <=> $right;
};
```


An array of arrays

```
$employees = array (
    array('name' => 'Ray',  'id' => 5685),
    array('name' => 'Mike', 'id' => 4302),
    array('name' => 'Anne', 'id' => 3674),
    array('name' => 'Pren', 'id' => 1527),
    array('name' => 'Joel', 'id' => 6256)
);
```


A function to sort the array by any column

```
function array_compare_factory($sort_key) {
    return function ($left, $right) use ($sort_key) {
        if ($left[$sort_key] < $right[$sort_key]) {
            return -1;
        } else if ($left[$sort_key] >
                   $right[$sort_key]) {
            return 1;
        } else {
            return 0;
        }
    };
}
```

The function with the spaceship operator (PHP 7 and later)

```
function array_compare_factory($sort_key) {
    return function ($left, $right) use ($sort_key) {
        return $left[$sort_key] <=> $right[$sort_key];
    };
}
```

Code that sorts the array by the name column

```
$sort_by_name = array_compare_factory('name');
usort($employees, $sort_by_name);
```

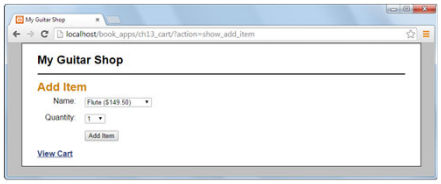
Code that sorts the array by the id column

```
$sort_by_id = array_compare_factory('id');
usort($employees, $sort_by_id);
```

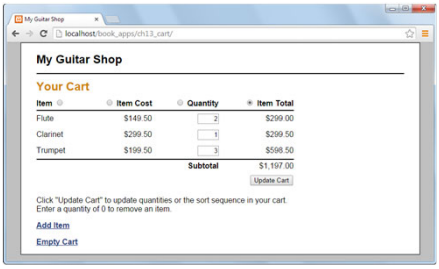

How closures work

- A *closure* is an inner function that has access to the outer function's variables. To create a closure, code a use clause in the inner function.
- To allow the inner function to change the outer function's variable, use the reference operator (&) in the use clause.
- The outer function's variables are available after it has finished executing as long as there is a reference to the inner function.
- The inner function is an anonymous function that is returned by the outer function or stored in a parameter that was passed by reference. You can store it in a variable and call it as a variable function like you would an anonymous function.

The Add Item page



The Cart page



The cart.php file

```
<?php
namespace cart {

    // Add an item to the cart
    function add_item($key, $quantity) {
        // Same code as cart application for chapter 12
    }

    // Update an item in the cart
    function update_item($key, $quantity) {
        // Same code as cart application for chapter 12
    }

    // Get cart subtotal
    function get_subtotal () {
        // Same code as cart application for chapter 12
    }
```


The cart.php file (continued)

```
    // Get a function for sorting the cart by key
    function compare_factory($sort_key) {
        return function($left, $right) use ($sort_key) {
            if ($left[$sort_key] == $right[$sort_key]) {
                return 0;
            } else if ($left[$sort_key] <
                       $right[$sort_key]) {
                return -1;
            } else {
                return 1;
            }
        };
    }

    // Sort the cart by the specified key
    function sort($sort_key) {
        $compare_function = compare_factory($sort_key);
        uasort($_SESSION['cart13'], $compare_function);
    }
}
?>
```

The index.php file

```
<?php
// Start session management
session_start();

// Create a cart array if needed
if (empty($_SESSION['cart13'])) { $_SESSION['cart13'] = array(); }

// Create a table of products
$products = array();
$products['MMS-1754'] =
    array('name' => 'Flute', 'cost' => '149.50');
$products['MMS-6289'] =
    array('name' => 'Trumpet', 'cost' => '199.50');
$products['MMS-3408'] =
    array('name' => 'Clarinet', 'cost' => '299.50');

// Include cart functions
require_once('cart.php');
```


The index.php file (continued)

```
// Get the sort key
$sort_key = filter_input(INPUT_POST, 'sortkey');
if ($sort_key === NULL) { $sort_key = 'name'; }

// Get the action to perform
$action = filter_input(INPUT_POST, 'action');
if ($action === NULL) {
    $action = filter_input(INPUT_GET, 'action');
    if ($action === NULL) {
        $action = 'show_add_item';
    }
}
```


The index.php file (continued)

```
// Add or update cart as needed
switch($action) {
    case 'add':
        $key = filter_input(INPUT_POST, 'productkey');
        $quantity  = filter_input(INPUT_POST, 'itemqty');
        cart\add_item($key, $quantity);
        include('cart_view.php');
        break;
    case 'update':
        $new_qty_list = filter_input(INPUT_POST, 'newqty',
                        FILTER_DEFAULT, FILTER_REQUIRE_ARRAY);
        foreach($new_qty_list as $key => $qty) {
            if ($_SESSION['cart13'][$key]['qty'] != $qty) {
                cart\update_item($key, $qty);
            }
        }
        cart\sort($sort_key);
        include('cart_view.php');
        break;
    case 'show_cart':
        cart\sort($sort_key);
        include('cart_view.php');
        break;
```


The index.php file (continued)

```
    case 'show_add_item':
        include('add_item_view.php');
        break;
    case 'empty_cart':
        unset($_SESSION['cart13']);
        include('cart_view.php');
        break;
}
?>
```


The cart_view.php file

```
<!DOCTYPE html>
<html>
<head>
    <title>My Guitar Shop</title>
    <link rel="stylesheet" type="text/css" href="main.css">
</head>
<body>
    <header>
        <h1>My Guitar Shop</h1>
    </header>
    <main>
        <h1>Your Cart</h1>
        <?php if (empty($_SESSION['cart13']) ||
                  count($_SESSION['cart13']) == 0) : ?>
            <p>There are no items in your cart.</p>
        <?php else: ?>
```


The cart_view.php file

```
        <form action="." method="post">
            <input type="hidden" name="action" value="update">
            <table>
                <tr id="cart_header">
                    <th class="left">
                        Item <input type="radio"
                        <?php if ($sort_key == 'name') : ?>
                            checked
                        <?php endif; ?>
                        name="sortkey" value="name"></th>
                    <th class="right">
                        <input type="radio"
                        <?php if ($sort_key == 'cost') : ?>
                            checked
                        <?php endif; ?>
                            name="sortkey" value="cost">
                        Item Cost</th>
```

The cart_view.php file (continued)

```
                <th class="right" >
                    <input type="radio"
                    <?php if ($sort_key == 'qty') : ?>
                        checked
                    <?php endif; ?>
                        name="sortkey" value="qty">
                    Quantity</th>
                <th class="right">
                    <input type="radio"
                    <?php if ($sort_key == 'total') : ?>
                      checked
                    <?php endif; ?>
                        name="sortkey" value="total">
                    Item Total</th>
            </tr>
            <?php foreach( $_SESSION['cart13'] as $key =>
                           $item ) :
                $cost  = number_format($item['cost'],  2);
                $total = number_format($item['total'], 2);
            ?>
            <tr>
                <td>
                    <?php echo $item['name']; ?>
                </td>
```


The cart_view.php file (continued)

```
                <td class="right">
                        $<?php echo $cost; ?>
                </td>
                <td class="right">
                    <input type="text" class="cart_qty"
                        name="newqty[<?php echo $key; ?>]"
                        value="<?php echo $item['qty']; ?>">
                </td>
                <td class="right">
                    $<?php echo $total; ?>
                </td>
            </tr>
            <?php endforeach; ?>
            <tr id="cart_footer">
                <td colspan="3"><b>Subtotal</b></td>
                <td>$<?php echo cart\get_subtotal(); ?></td>
            </tr>
            <tr>
                <td colspan="4" class="right">
                    <input type="submit" value="Update
                      Cart">
                </td>
            </tr>
```


The cart_view.php file

```
            </table>
            <p>Click "Update Cart" to update quantities or the
               sort sequence in your cart.<br>
               Enter a quantity of 0 to remove an item.
            </p>
        </form>
    <?php endif; ?>
    <p><a href=".?action=show_add_item">Add Item</a></p>
    <p><a href=".?action=empty_cart">Empty Cart</a></p>
</main>
</body>
</html>
```