Chapter 5

Functions and Files

Functions

- A **function** is a named set of statements that you can execute just by invoking the function name instead of retyping the statements.
- **Benefits:**
  - Reusability
  - Preventing errors
  - Faster and easier debugging
  - More meaningful program
- **An Example:**

```
function page_header() {
    print '<html><head><title>Welcome to my site</title></head>';
    print '<body bgcolor="#ffffff">';
}
```

Declaring and Calling Functions

- Function declaring syntax:

```
function functionName($arg1, $arg2,…)
   code to be executed;
}
```

```
function page_header() {
    print '<html><head><title>Welcome to my site</title></head>';
    print '<body bgcolor="#ffffff">';
}
```

- functionName follows the same rule as variable names

- **To call a function:**
  - Use the name of the function with arguments if needed.
  - For example:
    - *Page_header();*

```
function page_header() {
    print '<html><head><title>Welcome to my site</title></head>';
    print '<body bgcolor="#ffffff">';
}

page_header();
print "Welcome, $user";
page_footer();

function page_footer() {
    print '<hr>Thanks for visiting.';
    print '</body></html>';
}
```

- Functions can be defined before or after calling them.

Passing Arguments to Functions

- The input values supplied to a function are called **arguments**.
- Arguments add to the power of functions because they make functions **more flexible.**

```
function page_header2($color) {
    print '<html><head><title>Welcome to my site</title></head>';
    print '<body bgcolor="#' . $color . '">';
}
```

```
page_header2('cc00cc');
```

This sets `$color` to `cc00cc` inside `page_header2()`, so it prints:

```
<html><head><title>Welcome to my site</title></head><body bgcolor="#cc00cc">
```

Passing Arguments to Functions

- If you call the function without a value for the argument, **the PHP engine complains with a warning.**
- **Assign Default Values**

```php
function page_header3($color = 'cc3399') {
    print '<html><head><title>Welcome to my site</title></head>';
    print '<body bgcolor="#' . $color . '">';
}
```

```php
// One optional argument: it must be last
function page_header5($color, $title, $header = 'Welcome') {
    print '<html><head><title>Welcome to ' . $title . '</title></head>';
    print '<body bgcolor="#' . $color . '">';
    print "<h1>$header</h1>";
}
// Acceptable ways to call this function:
page_header5('66cc99','my wonderful page'); // uses default $header
page_header5('66cc99','my wonderful page','This page is great!'); // no defaults

// Two optional arguments: must be last two arguments
function page_header6($color, $title = 'the page', $header = 'Welcome') {
    print '<html><head><title>Welcome to ' . $title . '</title></head>';
    print '<body bgcolor="#' . $color . '">';
    print "<h1>$header</h1>";
}
// Acceptable ways to call this function:
page_header6('66cc99'); // uses default $title and $header
page_header6('66cc99','my wonderful page'); // uses default $header
page_header6('66cc99','my wonderful page','This page is great!'); // no defaults
```

- **Default values must be literals not variables**
- NOTE: all of the **optional arguments** must come **after any mandatory** arguments.

Returning Values from Functions

- To return values from functions you write, use the **return** keyword with **a value to return**.
- When a function is executing, as soon as it encounters the return keyword, it stops running and returns the associated value.

```
function restaurant_check($meal, $tax, $tip) {
    $tax_amount = $meal * ($tax / 100);
    $tip_amount = $meal * ($tip / 100);
    $total_amount = $meal + $tax_amount + $tip_amount;

    return $total_amount;
}
```

```
// Find the total cost of a $15.22 meal with 8.25% tax and a 15% tip
$total = restaurant_check(15.22, 8.25, 15);

print 'I only have $20 in cash, so...';
if ($total > 20) {
    print "I must pay with my credit card.";
} else {
    print "I can pay with cash.";
}
```

Returning an Array

- A function can return an array
- This way you can return multiple values

```
function restaurant_check2($meal, $tax, $tip) {
    $tax_amount  = $meal * ($tax / 100);
    $tip_amount  = $meal * ($tip / 100);
    $total_notip = $meal + $tax_amount;
    $total_tip   = $meal + $tax_amount + $tip_amount;

    return array($total_notip, $total_tip);
}
```

```
$totals = restaurant_check2(15.22, 8.25, 15);

if ($totals[0] < 20) {
    print 'The total without tip is less than $20.';
}

if ($totals[1] < 20) {
    print 'The total with tip is less than $20.';
}
```

Multiple Returns

- You can have more than one return statement in a function

```
function payment_method($cash_on_hand, $amount) {
    if ($amount > $cash_on_hand) {
        return 'credit card';
    } else {
        return 'cash';
    }
}
```

Using return value of functions in if-statements

The return value must not be a Boolean value.

The rules of evaluating all expression to Boolean is applied here.

```
if (restaurant_check(15.22, 8.25, 15) < 20) {
    print 'Less than $20, I can pay cash.';
} else {
    print 'Too expensive, I need my credit card.';
}
```

```
function complete_bill($meal, $tax, $tip, $cash_on_hand) {
    $tax_amount = $meal * ($tax / 100);
    $tip_amount = $meal * ($tip / 100);
    $total_amount = $meal + $tax_amount + $tip_amount;
    if ($total_amount > $cash_on_hand) {
        // The bill is more than we have
        return false;
    } else {
        // We can pay this amount
        return $total_amount;
    }
}

if ($total = complete_bill(15.22, 8.25, 15, 20)) {
    print "I'm happy to pay $total.";
} else {
    print "I don't have enough money. Shall I wash some dishes?";
}
```

```
function can_pay_cash($cash_on_hand, $amount) {
    if ($amount > $cash_on_hand) {
        return false;
    } else {
        return true;
    }
}

$total = restaurant_check(15.22,8.25,15);
if (can_pay_cash(20, $total)) {
    print "I can pay in cash.";
} else {
    print "Time for the credit card.";
}
```

Variable Scopes (Local & Global)

- Local Variables
- Local variables are created and destroyed inside a functions
- Variables inside a function are local, or **their scope is the body of function**

```
function countdown($top) {
    while ($top > 0) {
        print "$top..";
        $top--;
    }
    print "boom!\n";
}

$counter = 5;
countdown($counter);
print "Now, counter is $counter";
```

Example 5-9 prints:

```
5..4..3..2..1..boom!
Now, counter is 5
```


```
function countdown($counter) {
    while ($counter > 0) {
        print "$top..";
        $counter --;
    }
    print "boom!\n";
}

$counter = 5;
countdown($counter);
print "Now, counter is $counter";
```

Example 5-9 prints:

```
5..4..3..2..1..boom!
Now, counter is 5
```

Variable Scopes (Local & Global)

- Global Variables

- Global variables are defined out of all functions

```
$dinner = 'Curry Cuttlefish';

function vegetarian_dinner() {
    print "Dinner is $dinner, or ";
    $dinner = 'Sauteed Pea Shoots';
    print $dinner;
    print "\n";
}

function kosher_dinner() {
    print "Dinner is $dinner, or ";
    $dinner = 'Kung Pao Chicken';
    print $dinner;
    print "\n";
}

print "Vegetarian ";
vegetarian_dinner();
print "Kosher ";
kosher_dinner();
print "Regular dinner is $dinner";
```

Example 5-20 prints:

```
Vegetarian Dinner is , or Sauteed Pea Shoots
Kosher Dinner is , or Kung Pao Chicken
Regular dinner is Curry Cuttlefish
```

Access Globals in Functions

There are two ways to access a global variable from inside a function:

- Use of **a special array called $GLOBALS** (recommeneded)
- Use of **keyword global** in front of a local variable

```
$dinner = 'Curry Cuttlefish';

function macrobiotic_dinner() {
    $dinner = "Some Vegetables";
    print "Dinner is $dinner";
    // Succumb to the delights of the ocean
    print " but I'd rather have ";
    print $GLOBALS['dinner'];
    print "\n";
}

macrobiotic_dinner();
print "Regular dinner is: $dinner";
```

Example 5-21 prints:

```
Dinner is Some Vegetables but I'd rather have Curry Cuttlefish
Regular dinner is: Curry Cuttlefish
```

```
$dinner = 'Curry Cuttlefish';

function vegetarian_dinner() {
    global $dinner;
    print "Dinner was $dinner, but now it's ";
    $dinner = 'Sauteed Pea Shoots';
    print $dinner;
    print "\n";
}

print "Regular Dinner is $dinner.\n";
vegetarian_dinner();
print "Regular dinner is $dinner";
```

Example 5-23 prints:

```
Regular Dinner is Curry Cuttlefish.
Dinner was Curry Cuttlefish, but now it's Sauteed Pea Shoots
Regular dinner is Sauteed Pea Shoots
```

Enforcing Data Types

- By default, function arguments and return values **do not have constraint** on their **types** and **values**
- **Type declarations** are a way to express constraints on argument values.

Table 5-1. Type declarations

Declaration	Argument rule	Minimum PHP version
`array`	Must be an array	5.1.0
`bool`	Must be boolean: `true` or `false`	7.0.0
`callable`	Must be something representing a function or method that can be called[a]	5.4.0
`float`	Must be a floating-point number	7.0.0
`int`	Must be an integer	7.0.0.
`string`	Must be a string	7.0.0.
Name of a class	Must be an instance of that class (see Chapter 6 for more information about classes and instances).	5.0.0

[a] This can be a string containing a valid function name, a two-element array where the first element is an object instance and the second is a string holding a method name, or a few other things. See *http://www.php.net/language.types.callable* for all the details.

Running Code in Another File

- Examples of Type Declaration for arguments

```
function countdown(int $top) {
    while ($top > 0) {
        print "$top..";
        $top--;
    }
    print "boom!\n";
}

$counter = 5;
countdown($counter);
print "Now, counter is $counter";
```

- Type declataion for return value

Example 5-25. Declaring a return type

```
function restaurant_check($meal, $tax, $tip): float {
    $tax_amount = $meal * ($tax / 100);
    $tip_amount = $meal * ($tip / 100);
    $total_amount = $meal + $tax_amount + $tip_amount;

    return $total_amount;
}
```

Running Code in Another File

- The PHP code examples we've seen so far are mostly self-contained individual files
- The **require** directive tells the PHP engine to load code located in a different file
  - Great for reusing a code in many places

```php
<?php

function restaurant_check($meal, $tax, $tip) {
    $tax_amount = $meal * ($tax / 100);
    $tip_amount = $meal * ($tip / 100);
    $total_amount = $meal + $tax_amount + $tip_amount;

    return $total_amount;
}

function payment_method($cash_on_hand, $amount) {
    if ($amount > $cash_on_hand) {
        return 'credit card';
    } else {
        return 'cash';
    }
}

?>
```

This code is saved in a file named restaurant-functions.php

```php
require 'restaurant-functions.php';

/* $25 check, plus 8.875% tax, plus 20% tip */
$total_bill = restaurant_check(25, 8.875, 20);

/* I've got $30 */
$cash = 30;

print "I need to pay with " . payment_method($cash, $total_bill);
```

Running Code in Another File

- If the **require** statement **can't find the file** to load, or it **doesn't contain valid PHP code**, the PHP engine **stops** running your program.
- The **include** statement also **loads code** from another file, **but will keep going if there's a problem** with the loaded file.

Exercises

1. Write a function to return an HTML `<img />` tag. The function should accept a mandatory argument of the image URL and optional arguments for `alt` text, `height`, and `width`.
2. Modify the function in the previous exercise so that only the filename is passed to the function in the URL argument. Inside the function, prepend a global variable to the filename to make the full URL. For example, if you pass *photo.png* to the function, and the global variable contains `/images/`, then the `src` attribute of the returned `<img>` tag would be `/images/photo.png`. A function like this is an easy way to keep your image tags correct, even if the images move to a new path or server. Just change the global variable—for example, from `/images/` to `http://images.example.com/`.
3. Put your function from the previous exercise in one file. Then make another file that loads the first file and uses it to print out some `<img />` tags.
4. What does the following code print out?

```php
<?php

function restaurant_check($meal, $tax, $tip) {
    $tax_amount = $meal * ($tax / 100);
    $tip_amount = $meal * ($tip / 100);
    return $meal + $tax_amount + $tip_amount;
}

$cash_on_hand = 31;
$meal = 25;
$tax = 10;
$tip = 10;

while(($cost = restaurant_check($meal,$tax,$tip)) < $cash_on_hand) {
    $tip++;
    print "I can afford a tip of $tip% ($cost)\n";
}

?>
```

5. Web colors such as `#ffffff` and `#cc3399` are made by concatenating the hexadecimal color values for red, green, and blue. Write a function that accepts decimal red, green, and blue arguments and returns a string containing the appropriate color for use in a web page. For example, if the arguments are `255`, `0`, and `255`, then the returned string should be `#ff00ff`. You may find it helpful to use the built-in function `dechex()`, which is documented at *http://www.php.net/dechex*.