Learning PHP

A GENTLE INTRODUCTION TO THE WEB'S MOST POPULAR LANGUAGE

Intro to PHP

- **Static Websites:** the content does not change and is fixed. The content is the same for all visitors. **E.g. personal websites**
- **Dynamic Websites:** pictures and contents are different for different visitors.**E.g. Amazon.com**

- PHP is a programming language for building **dynamic websites**
- PHP is a **server-side** language
  - **Example**: JavaScript is a **client-side** language
  - **Example**: ASP.NET is a **server-side** language
- PHP is free
- **OS X** and most **Linux** distributions come with PHP **already installed**.

Static Webpages

- PHP runs on the server not on the client

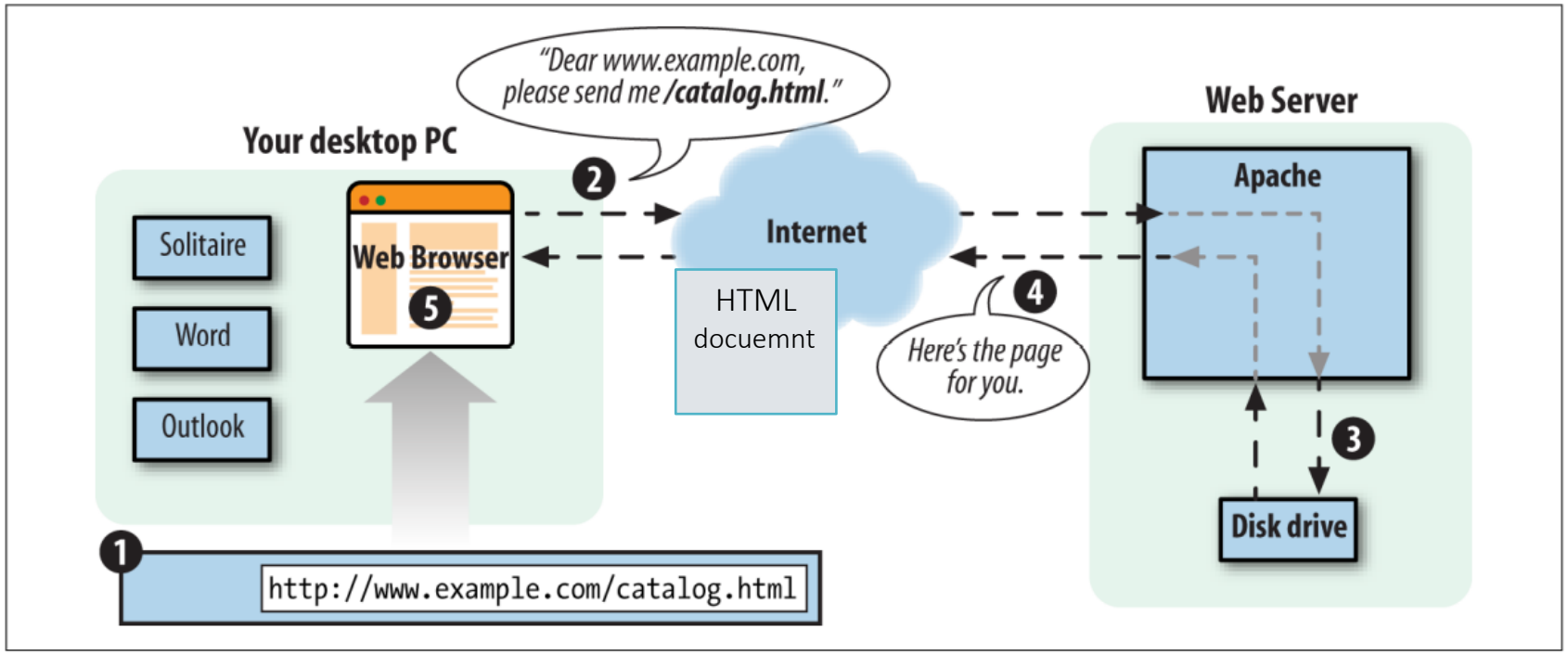


Figure 1-1. Client and server communication without PHP

Dynamic Webpages

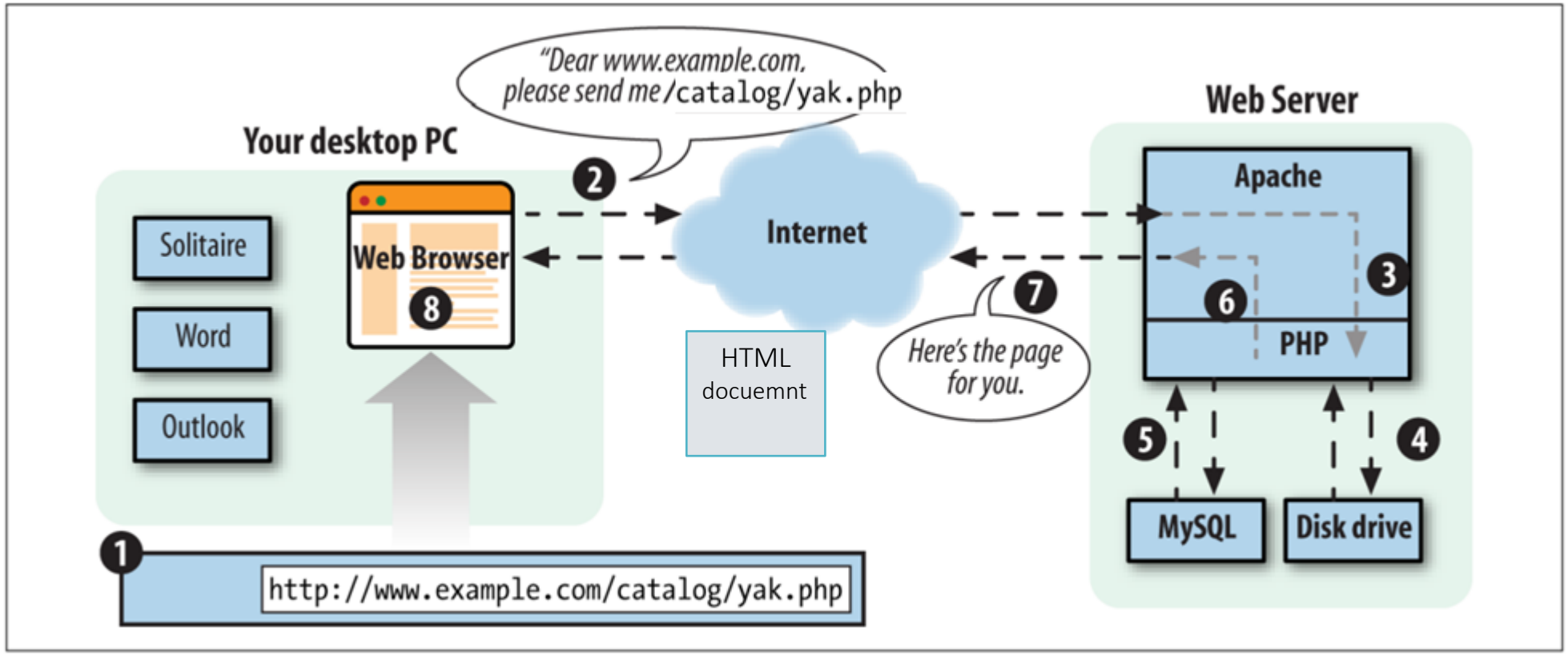


Figure 1-2. Client and server communication with PHP

PHP and PHP Engine

- PHP is a language
- PHP Engine is the software
  - Running on a Web Server
  - Understanding PHP language and executes the commands
  - For example, talking to DBMS, retrieving data and generating pages
- **PHP Engine** is written in the **C programming** language
- PHP works with a web server running on Windows, Mac OS X, Linux, and many other versions of Unix.
- PHP works on **Web Servers** such as Apache, nginx, MS IIS, or any web server that supports CGI standard.
- PHP works on many **DBMSs**: MySQL, PostgreSQL, Orcle, MS SQL Server, SQLite, Redis, and MongoDB
- PHP is used on more than 200 million different websites, including giants like **Facebook**, **Wikipedia**, and **Yahoo**

Basics of PHP

- It can be part of a HTML file
- It starts with **<?php** and ends with **?>**
- **PHP engine** executes only code between **<?php and ?>,** text out of them is ignored
- If there is no code at the end of the file, **?> end tag is optional**
- There can be multiple blocks of PHP code in an HTML file

```
<span>Five plus five is:<span>
<?php print 5 + 5; ?>
<p>
Four plus four is:
<?php
print 4 + 4;
?>
</p>
<img src="vacation.jpg" alt="My Vacation" />
```

Basics of PHP

- PHP is a case-sensitive language
  - **$_POST** and **$_post** are different
  - **But**, Language keywords (such as print) and function names are **not case-sensitive**.

Example 1-13. Keywords and function names are case-insensitive

```php
<?php
// These four lines all do the same thing
print number_format(320853904);
PRINT Number_Format(320853904);
Print number_format(320853904);
pRiNt NUMBER_FORMAT(320853904);
?>
```

Basics of PHP

Comments

1. are an essential part of any program. By **explaining in plain language** how the programs work, comments make programs much more understandable.
2. You can also disable a part of code for testing your program

- inline comments //
- Inline comments #
- multiline commensts /* */

Example 1-15. Multiline comments

```php
<?php
/* We're going to add a few things to the menu:
   - Smoked Fish Soup
   - Duck with Pea Shoots
   - Shark Fin Soup
*/
print 'Smoked Fish Soup, Duck with Pea Shoots, Shark Fin Soup ';
print 'Cost: 3.25 + 9.50 + 25.00';
```

Example 1-14. Single-line comments with // or #

```php
<?php
// This line is a comment
print "Smoked Fish Soup ";
print 'costs $3.25.';

# Add another dish to the menu
print 'Duck with Pea Shoots ';
print 'costs $9.50.';
// You can put // or # inside single-line comments
// Using // or # somewhere else on a line also starts a comment
print 'Shark Fin Soup'; // I hope it's good!
print 'costs $25.00!'; # This is getting expensive!

# Putting // or # inside a string doesn't start a comment
print 'http://www.example.com';
print 'http://www.example.com/menu.php#dinner';
?>
```

First Program: HELLO WORLD!

The .PHP file

```
<!DOCTYPE html>
<html>
<head>
    <title>PHP says hello</title>
</head>
<body>
    <b>
        <?php
        print "Hello, World!";
        ?>
    </b>
</body>
</html>
```

OUTPUT:

```
<!DOCTYPE html>
<html>
<head>
    <title>PHP says hello</title>
</head>
<body>
      <b>
            Hello, World!
      </b>
</body>
</html>
```

Another Example

.html File

```
<!DOCTYPE html>
<html>
<head>
    <meta charset="utf-8" />
    <title></title>
</head>
<body>
    <form method="POST" action="sayhello.php">
        Your Name: <input type="text" name="user" />
        <br />
        <button type="submit">Say Hello</button>
    </form>

</body>
</html>
```

.php file

```
<?php
print "Hello, ";
// Print what was submitted in the form parameter called 'user'
print $_POST['user'];
print "!";
?>
```

Basics of PHP

- Every program is composed of **statements**
- Statments end with **semi-colon** (;)
- You can write **multiple PHP statements** on the same line of a program as long as they are separated with a **semicolon**.
- You can put as many **blank lines between statements** as you want. The PHP engine ignores them.
- It is recommended to put **one statement on a line** and blank lines between statements only when it improves the readability.

Example 1-9. This PHP is too cramped

```
<?php print "Hello"; print " World!"; ?>
```

Example 1-10. This PHP is too sprawling

```
<?php

print "Hello";


print " World!";

?>
```

Example 1-11. This PHP is just right

```
<?php
print "Hello";
print " World!";
?>
```

Learning PHP

Data: Working with Text and Numbers

Data Types

- **String**: a sequence of bytes (represented by charaters)
- **String can contain**
  - Letters a-z, A-Z
  - Numbers 0-9
  - Punctuation . ; ? ! , ( ) :
  - Spaces
  - Tabs
  - Or any other character
- **Examples**:
  - Gtf46gx!#ddfgfg
  - Life is beautiful

Defining Strings

- Surround the string with **single-quote** or **double-quote**
  - There are differences between using ‘ and “
- If you want to include a single quote inside a string, put a backslash (\) before it
- Word processors often change straight quotes like ' and " into **curly quotes** like ‘, ’, “, and ”. The PHP engine only understands straight quotes as string delimiters.

```
print 'I would like a bowl of soup.';
print 'chicken';
print '06520';
print '"I am eating dinner," he growled.';
```

```
print 'We\'ll each have a bowl of soup.';
```

Ecape Sequence

- Backslash (\) is a escape character in PHP (like C,C++,C#,JavaScript, Java)
- inside single-quoted strings, **backslash** and **single quote** are only special characters. Everything else is treated literally.

```
print 'Use a \\ to escape in a string';
```

This prints:

```
Use a \ to escape in a string
```

- Double-Quoted Strings
  - If you use " for strings, there are more special characters
  - Example:
    - Print 'Hi $user' *this prints Hi $user*
    - Print "Hi $user" *this prints Hi John if John is the current user*

Table 2-1. Special characters in double-quoted strings

Character	Meaning
\n	Newline (ASCII 10)
\r	Carriage return (ASCII 13)
\t	Tab (ASCII 9)
\\	\
\$	$
\"	"
\0 .. \777	Octal (base 8) number
\x0 .. \xFF	Hexadecimal (base 16) number

HERE document

- You can define strings with the *here document* syntax, specialy *HTML code string*
- ***Start** with <<< and a delimiter word, **end** with the same word used at start*
- delimiters can contain
  - **letters**, **numbers**, and the **underscore** character.
  - The **first character** of the delimiter must be a letter or underscore.
  - For **readability**, recommended writing uppercase
- The end delimiter must be **alone on its line**.
- The delimiter can't be **indented** and **no whitespace**, comments, or other characters are allowed after it, only **semi-colon**

Example 2-1. Here document

```
<<<HTMLBLOCK
<html>
<head><title>Menu</title></head>
<body bgcolor="#fffed9">
<h1>Dinner</h1>
<ul>
  <li> Beef Chow-Fun
  <li> Sauteed Pea Shoots
  <li> Soy Sauce Noodles
  </ul>
</body>
</html>
HTMLBLOCK
```

HERE Document

- **Note**: **print** command must be used,
- but single or double quote not needed

Example 2-2. Printing a here document

```
print <<<HTMLBLOCK
<html>
<head><title>Menu</title></head>
<body bgcolor="#fffed9">
<h1>Dinner</h1>
<ul>
  <li> Beef Chow-Fun
  <li> Sauteed Pea Shoots
  <li> Soy Sauce Noodles
  </ul>
</body>
</html>
HTMLBLOCK;
```

Combining Strings

- Use a . (period) to combine (concatenate) strings

```
print 'bread' . 'fruit';
print "It's a beautiful day " . 'in the neighborhood.';
print "The price is: " . '$3.95';
print 'Inky' . 'Pinky' . 'Blinky' . 'Clyde';
```

The combined strings print as:

```
breadfruit
It's a beautiful day in the neighborhood.
The price is: $3.95
InkyPinkyBlinkyClyde
```

PHP String functions

- trim() function removes whitespace from the beginning and end of a string.
- strlen() function tells you the length of a string

Example 2-3. Checking the length of a trimmed string

```
// $_POST['zipcode'] holds the value of the submitted form parameter
// "zipcode"
$zipcode = trim($_POST['zipcode']);
// Now $zipcode holds that value, with any leading or trailing spaces
// removed
$zip_length = strlen($zipcode);
// Complain if the zip code is not 5 characters long
if ($zip_length != 5) {
    print "Please enter a zip code that is 5 characters long.";
}
```

PHP String functions

- Use == to cpmrae two strings

```
if ($_POST['email'] == 'president@whitehouse.gov') {
    print "Welcome, US President.";
}
```

- strcasecmp() function compares two strings without considering their cases (upper or lower case)

```
if (strcasecmp($_POST['email'], 'president@whitehouse.gov') == 0) {
    print "Welcome back, US President.";
}
```

- strtolower() and strtoupper() changes a string to all-lowercase or all-uppercase versions, respectively.

Example 2-10. Changing case

```
print strtolower('Beef, CHICKEN, Pork, duCK');
print strtoupper('Beef, CHICKEN, Pork, duCK');
```

Example 2-10 prints:

```
beef, chicken, pork, duck
BEEF, CHICKEN, PORK, DUCK
```

PHP String Functions

- The **ucwords()** function uppercases the first letter of each word in a string.

```
print ucwords(strtolower('JOHN FRANKENHEIMER'));
```

Example 2-11 prints:

```
John Frankenheimer
```

- **substr()** function, you can extract just part of a string.
  - 0 to 30 means first 30 characters
  - If you put negative number, it starts from the end
    - *-4, 4 means last 4 character from the end of the string*

Example 2-12. Truncating a string with substr()

```
// Grab the first 30 bytes of $_POST['comments']
print substr($_POST['comments'], 0, 30);
// Add an ellipsis
print '...';
```

If the submitted form parameter comments is:

```
The Fresh Fish with Rice Noodle was delicious, but I didn't like the Beef Tripe.
```

Example 2-12 prints:

```
The Fresh Fish with Rice Noodl...
```

PHP String Functions

- str_replace() function changes parts of a string. It looks for a substring and replaces the substring with a new string.

Example 2-14. Using str_replace()

```
$html = '<span class="{class}">Fried Bean Curd<span>
<span class="{class}">Oil-Soaked Fish</span>';

print str_replace('{class}',$my_class,$html);
```

If `$my_class` has been set to `lunch`, then Example 2-14 prints:

```
<span class="lunch">Fried Bean Curd<span>
<span class="lunch">Oil-Soaked Fish</span>
```

Numbers

Numbers

- No special notation needed
- Numbers:
  - **Floating-point**: 34.6, 7.567, 0.765, -8.543
  - **Integers:** 356, 1876, 21, 0, -76

- **NOTE:** Integers are stored **precisely** but floating point numbers may not
- 47 is stored exactly 47 but **46.3 could be stored as 46.299999**

Example 2-15. Numbers

```
print 56;
print 56.3;
print 56.30;
print 0.774422;
print 16777.216;
print 0;
print -213;
print 1298317;
print -9912111;
print -12.52222;
print 0.00;
```

Arithmetic Operators

You can use arithmetic operators (+, -, *, /) as you did in primary school

New operators:

- ** for exponentation
- % for modulus division (returing remainder of a devision)

```
print 17 % 3;
```

This prints:

```
2
```

Example 2-16. Math operations

```
print 2 + 2;
print 17 - 3.5;
print 10 / 3;
print 6 * 9;
```

The output of Example 2-16 is:

```
4
13.5
3.3333333333333
54
```

Order of Operators

- Like in Math, operators are executed in order
- Example:
  - 3+4*2 → 11
- Unless, you use brackets (parentheses) to change the order
  - (3+4)*2 → 14
  - 3+(4*2) → 11

- Learn more about **PHP Operator Precedence :** PHP: Operator Precedence - Manual

Variables

Variables

- **Variables** hold the data in the memory of computer while your program uses and manipulates it
- In PHP, variables are **denoted by a $** followed by the variable's name.
- **To assign a value to a variable**, use an **equals sign (=).** This is known as the **assignment** operator.

Here are a few examples:

```
$plates = 5;
$dinner = 'Beef Chow-Fun';
$cost_of_dinner = 8.95;
$cost_of_lunch = $cost_of_dinner;
```

Assignment works with here documents as well:

```
$page_header = <<<HTML_HEADER
<html>
<head><title>Menu</title></head>
<body bgcolor="#fffed9">
<h1>Dinner</h1>
HTML_HEADER;

$page_footer = <<<HTML_FOOTER
</body>
</html>
HTML_FOOTER;
```

Variable Names

- **Variable names may only include:**
  - Uppercase or lowercase Basic Latin letters (A-Z and a-z)
  - Digits (0-9)
  - Underscore (_)
  - Any non-Basic Latin character (such as ç), if you're using a character encoding such as UTF-8 for your program file
  - Additionally, the **first character** of a variable name is not allowed to be a digit.

Table 2-2. Allowable variable names

`$size`
`$drinkSize`
`$SUPER_BIG_DRINK`
`$_d_r_i_n_k_y`
`$drink4you2`
`$напиток`
`$သောက်စရာ`
`$🄳🅁🄸🄽🄺`
`$😀`

- **NOTE**:Even though you can use special charcaters, but **NOT RECOMMENDED**

- **Variable names are case-sensitive.**
  - $dinner, $Dinner, and $DINNER are separate and distinct
  - **Avoid using variables names with same letters but differ in cases**

Table 2-3. Disallowed variable names

Variable name	Flaw
`$2hot4u`	Begins with a number
`$drink-size`	Unacceptable character: -
`$drinkmaster@example.com`	Unacceptable characters: @ and .
`$drink!lots`	Unacceptable character: !
`$drink+dinner`	Unacceptable character: +

Arithmetic Operation on Variables

- Arithmetic and string operators work on variables containing numbers or strings just like they do on literal numbers or strings.

Example 2-17. Operating on variables

```
$price = 3.95;
$tax_rate = 0.08;
$tax_amount = $price * $tax_rate;
$total_cost = $price + $tax_amount;

$username = 'james';
$domain = '@example.com';
$email_address = $username . $domain;

print 'The tax is ' . $tax_amount;
print "\n"; // this prints a line break
print 'The total cost is ' .$total_cost;
print "\n"; // this prints a line break
print $email_address;
```

Example 2-17 prints:

```
The tax is 0.316
The total cost is 4.266
james@example.com
```

Putting Variables Inside Strings

- Frequently, you print the values of variables combined with other text
  - **Example:** display an HTML table with calculated values in the cells
- Use double-quoted strings and variable inside them

Example 2-22. Interpolating in a here document

```
$page_title = 'Menu';
$meat = 'pork';
$vegetable = 'bean sprout';
print <<<MENU
<html>
<head><title>$page_title</title></head>
<body>
<ul>
<li> Barbecued $meat
<li> Sliced $meat
<li> Braised $meat with $vegetable
</ul>
</body>
</html>
MENU;
```

Example 2-22 prints:

```
<html>
<head><title>Menu</title></head>
<body>
<ul>
<li> Barbecued pork
<li> Sliced pork
<li> Braised pork with bean sprout
</ul>
</body>
</html>
```

```
$email = 'jacob@example.com';
print "Send replies to: $email";
```

Example 2-21 prints:

```
Send replies to: jacob@example.com
```

Putting Variables Inside Strings using {}

- You could surround the variable with curly braces to remove the confusion

Example 2-23. Interpolating with curly braces

```
$preparation = 'Braise';
$meat = 'Beef';
print "{$preparation}d $meat with Vegetables";
```

Example 2-23 prints:

```
Braised Beef with Vegetables
```

- If you could put {} around $preparation, it becomes $preparationd which is not the variable

Exercises

Exercises

Find the errors in this PHP program:

```
<? php
print 'How are you?';
print 'I'm fine.';
??>
```

Write a PHP program that computes the total cost of this restaurant meal: two hamburgers at $4.95 each, one chocolate milkshake at $1.95, and one cola at 85 cents. The sales tax rate is 7.5%, and you left a pre-tax tip of 16%.

Write a PHP program that sets the variable `$first_name` to your first name and `$last_name` to your last name. Print out a string containing your first and last name separated by a space. Also print out the length of that string.

Write a PHP program that uses the increment operator (++) and the combined multiplication operator (*=) to print out the numbers from 1 to 5 and powers of 2 from 2 (2^1) to 32 (2^5).

Chapter 3

Logic: Making Decisions and Repetition

Boolean Expressions

- **Every expression** in a PHP program can be evaluated to **true or false**.

- All **non-zero integers or floating-point** numbers evaluated to **true**
- **zero (0 or 0.0)** is evaluated to **false**
- **All non-empty strings** are evaluated to **true**
- **Empty string,** or **string containing zero ('0') is** evaluated to **false**
- A variable containing **null** is evaluated to **false**

Evaluated to False	Evaluated to True
$v1 = 0; or $v1 = 0.0;	$v1 = 7; or $v1 = 5.78;
$v2 = '';	$v2 = 'Saeed';
$v3 = '0';	$v3 = true;
$v4 = null;	
$v5 = false;	

If – Statement

Example 3-1. Making a decision with if()

```
if ($logged_in) {
   print "Welcome aboard, trusted user.";
}
```

```
if ($logged_in) {
    print "Welcome aboard, trusted user.";
} else {
    print "Howdy, stranger.";
}
```

```
print "This is always printed.";
if ($logged_in) {
    print "Welcome aboard, trusted user.";
    print 'This is only printed if $logged_in is true.';
}
print "This is also always printed.";
```

elseif – Statement

```php
if ($logged_in) {
    // This runs if $logged_in is true
    print "Welcome aboard, trusted user.";
} elseif ($new_messages) {
    // This runs if $logged_in is false but $new_messages is true
    print "Dear stranger, there are new messages.";
} elseif ($emergency) {
    // This runs if $logged_in and $new_messages are false
    // but $emergency is true
    print "Stranger, there are no new messages, but there is an emergency.";
}
```

```php
if ($logged_in) {
    // This runs if $logged_in is true
    print "Welcome aboard, trusted user.";
} elseif ($new_messages) {
    // This runs if $logged_in is false but $new_messages is true
    print "Dear stranger, there are new messages.";
} elseif ($emergency) {
    // This runs if $logged_in and $new_messages are false
    // but $emergency is true
    print "Stranger, there are no new messages, but there is an emergency.";
} else {
    // This runs if $logged_in, $new_messages, and
    // $emergency are all false
    print "I don't know you, you have no messages, and there's no emergency.";
}
```

Comparison Operators

==, != , <, >, <=, >=

```
if ($new_messages == 12) {
    print "It seems you now have twelve new messages.";
}
```

```
if ($new_messages != 10) {
    print "You don't have ten new messages.";
}
```

```
if ($age > 17) {
    print "You are old enough to download the movie.";
}
if ($age >= 65) {
    print "You are old enough for a discount.";
}
if ($celsius_temp <= 0) {
    print "Uh-oh, your pipes may freeze.";
}
if ($kelvin_temp < 20.3) {
    print "Your hydrogen is a liquid or a solid now.";
}
```

Assignment vs. Comparison

- Be careful not to use = when you mean ==

```
if ($new_messages = 12) {
    print "It seems you now have twelve new messages.";
}
```

- One way to avoid using = instead of == is to put the variable on the right side of the comparison

```
if (12 == $new_messages) {
    print "You have twelve new messages.";
}
```

Floating-point numbers comparison

- Floating-point number may be stored slightly different
  - For example: 50.0 is stored as 50.000000002
- To compare two floating-point numbers, check whether the two numbers differ by less than some acceptably small threshold

```
if(abs($price_1 - $price_2) < 0.00001) {
    print '$price_1 and $price_2 are equal.';
} else {
    print '$price_1 and $price_2 are not equal.';
}
```

Comparison Operators - Strings

- The <, <=, ==, >=, and > operators can be used with **numbers** or **strings**.

```
if ($word < 'baa') {
    print "Your word isn't cookie.";
}
if ($word >= 'zoo') {
    print "Your word could be zoo or zymurgy, but not zone.";
}
```

- When the PHP engine sees strings containing **only numbers**, it **converts them to numbers** for the comparison.

```
// These values are compared using dictionary order
if ("x54321"> "x5678") {
    print 'The string "x54321" is greater than the string "x5678".';
} else {
    print 'The string "x54321" is not greater than the string "x5678".'
}

// These values are compared using numeric order
if ("54321" > "5678") {
    print 'The string "54321" is greater than the string "5678".';
} else {
    print 'The string "54321" is not greater than the string "5678".';
}
```

Comparison Operators - Strings

- the PHP engine converts the **string 6 pack to the number 6**, and then compares it to the number 55 using numeric order. Since 6 is less than 55, the less than test returns true.

```
// These values are compared using numeric order
if ('6 pack' < 55) {
    print 'The string "6 pack" is less than the number 55.';
} else {
    print 'The string "6 pack" is not less than the number 55.';
}
```

- **strcmp()** compares string using dictionary order without any converting to numbers

```
$x = strcmp("x54321","x5678");
if ($x > 0) {
    print 'The string "x54321" is greater than the string "x5678".';
} elseif ($x < 0) {
    print 'The string "x54321" is less than the string "x5678".';
}
```

spaceship operator (<=>)

- The **spaceship operator (<=>)** does comparison **similar to strcmp(),** but for **any data type**.
  - It evaluates to a **negative** number when its left-hand operand is **less than** the righthand operand,
  - a **positive** number when the righthand operand is **bigger**,
  - and **0** when they are **equal**.

```
// $a is a negative number since 1 is less than 12.7
$a = 1 <=> 12.7;

// $b is a positive number since "c" comes after "b"
$b = "charlie" <=> "bob";

// Comparing numeric strings works like < and >, not like strcmp()
$x = '6 pack' <=> '55 card stud';
if ($x > 0) {
    print 'The string "6 pack" is greater than the string "55 card stud".';
} elseif ($x < 0) {
    print 'The string "6 pack" is less than the string "55 card stud".';
}

// Comparing numeric strings works like < and >, not like strcmp()
$x ='6 pack' <=> 55;
if ($x > 0) {
    print 'The string "6 pack" is greater than the number 55.';
} elseif ($x < 0) {
    print 'The string "6 pack" is less than the number 55.';
}
```

Negation (!)

- To negate a truth value, use !
- Putting ! before an expression is like testing to see whether the expression equals false.

```
// The entire test expression ($finished == false)
// is true if $finished is false
if ($finished == false) {
    print 'Not done yet!';
}

// The entire test expression (! $finished)
// is true if $finished is false
if (! $finished) {
    print 'Not done yet!';
}
```

Combining multiple expressions with && and ||

- The logical AND operator (&&)
- The logical OR operator (||)

```
if (($age >= 13) && ($age < 65)) {
   print "You are too old for a kid's discount and too young for the senior's
discount.";
}

if (($meal == 'breakfast') || ($dessert == 'souffle')) {
   print "Time to eat some eggs.";
}
```

Repetition (Loops)

For-loop, while-loop, do-while

- for()
- while()
- do-while()

```
print '<select name="people">';
for ($i = 1; $i <= 10; $i++) {
    print "<option>$i</option>\n";
}
print '</select>';
```

```
$i = 1;
print '<select name="people">';
while ($i <= 10) {
    print "<option>$i</option>\n";
    $i++;
}
print '</select>';
```

Example 3-17 prints:

```
<select name="people"><option>1</option>
<option>2</option>
<option>3</option>
<option>4</option>
<option>5</option>
<option>6</option>
<option>7</option>
<option>8</option>
<option>9</option>
<option>10</option>
</select>
```

For-loop

- You can combine multiple expressions in the initialization expression and the iteration expression of a for() loop by separating each of the individual expressions with a comma.

```
print '<select name="doughnuts">';
for ($min = 1, $max = 10; $min < 50; $min += 10, $max += 10) {
    print "<option>$min - $max</option>\n";
}
print '</select>';
```

Each time through the loop, `$min` and `$max` are each incremented by 10. Example 3-19 prints:

```
<select name="doughnuts"><option>1 - 10</option>
<option>11 - 20</option>
<option>21 - 30</option>
<option>31 - 40</option>
<option>41 - 50</option>
</select>
```

Exercises

1. Without using a PHP program to evaluate them, determine whether each of these expressions is `true` or `false`:
   a. `100.00 - 100`
   b. `"zero"`
   c. `"false"`
   d. `0 + "true"`
   e. `0.000`
   f. `"0.0"`
   g. `strcmp("false","False")`
   h. `0 <=> "0"`
2. Without running it through the PHP engine, figure out what this program prints:

```php
$age = 12;
$shoe_size = 13;
if ($age > $shoe_size) {
    print "Message 1.";
} elseif (($shoe_size++) && ($age > 20)) {
    print "Message 2.";
} else {
    print "Message 3.";
}
print "Age: $age. Shoe Size: $shoe_size";
```

3. Use `while()` to print a table of Fahrenheit and Celsius temperature equivalents from –50 degrees F to 50 degrees F in 5-degree increments. On the Fahrenheit temperature scale, water freezes at 32 degrees and boils at 212 degrees. On the Celsius scale, water freezes at 0 degrees and boils at 100 degrees. So, to convert from Fahrenheit to Celsius, you subtract 32 from the temperature, multiply by 5, and divide by 9. To convert from Celsius to Fahrenheit, you multiply by 9, divide by 5, and then add 32.
4. Modify your answer to Exercise 3 to use `for()` instead of `while()`.

Chapter 4

Arrays

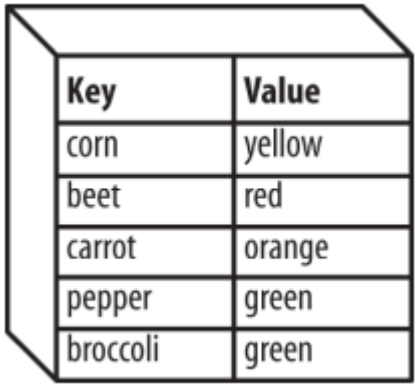
Arrays

- Arrays are **collections of related values**, such as the data submitted from a form, the names of students in a class, or the populations of a list of cities.
- An array is made up of **elements**.
- **Any string** or **number** value can be an array element **key**, such as corn, 4, -36, or Salt Baked Squid.

Creating an **Associative Array**

Associative Arrays: Each element has a **key** and a **value**.

- **For example**, an array holding information about the colors of vegetables has vegetable names for keys and colors for values
- Arrays and other non-scalar values **can't be keys**, but they can be element values.
- **Scalar values** are simple values such as 6, 98.76, "Saeed". Arrays or objects are non-scalar values



Key	Value
corn	yellow
beet	red
carrot	orange
pepper	green
broccoli	green

```
$vegetables['corn'] = 'yellow';
$vegetables['beet'] = 'red';
$vegetables['carrot'] = 'orange';
```

```
$vegetables = array('corn' => 'yellow',
                    'beet' => 'red',
                    'carrot' => 'orange');

$dinner = array(0 => 'Sweet Corn and Asparagus',
                1 => 'Lemon Chicken',
                2 => 'Braised Bamboo Fungus');

$computers = array('trs-80' => 'Radio Shack',
                   2600 => 'Atari',
                   'Adam' => 'Coleco');
```

Shortcut to Create Arrays

```php
$vegetables = ['corn' => 'yellow', 'beet' => 'red', 'carrot' => 'orange'];

$dinner = [0 => 'Sweet Corn and Asparagus',
           1 => 'Lemon Chicken',
           2 => 'Braised Bamboo Fungus'];

$computers = ['trs-80' => 'Radio Shack', 2600 => 'Atari', 'Adam' => 'Coleco'];
```

Creating an array element by element

```php
// An array called $vegetables with string keys
$vegetables['corn'] = 'yellow';
$vegetables['beet'] = 'red';
$vegetables['carrot'] = 'orange';

// An array called $dinner with numeric keys
$dinner[0] = 'Sweet Corn and Asparagus';
$dinner[1] = 'Lemon Chicken';
$dinner[2] = 'Braised Bamboo Fungus';

// An array called $computers with numeric and string keys
$computers['trs-80'] = 'Radio Shack';
$computers[2600] = 'Atari';
$computers['Adam'] = 'Coleco';
```

Creating a **Numeric Array**

- If you create an array with [] or array() by specifying only a list of values instead of key/value pairs, the PHP engine automatically assigns a numeric key to each value.
- The keys start at 0 and increase by one for each element.

```php
$dinner = array('Sweet Corn and Asparagus',
                'Lemon Chicken',
                'Braised Bamboo Fungus');
print "I want $dinner[0] and $dinner[1].";
```

- PHP automatically uses **incrementing numbers for array keys** when you create an array or add elements to an array with the **empty brackets**

```php
// Create $lunch array with two elements
// This sets $lunch[0]
$lunch[] = 'Dried Mushrooms in Brown Sauce';
// This sets $lunch[1]
$lunch[] = 'Pineapple and Yu Fungus';

// Create $dinner with three elements
$dinner = array('Sweet Corn and Asparagus', 'Lemon Chicken',
                'Braised Bamboo Fungus');
// Add an element to the end of $dinner
// This sets $dinner[3]
$dinner[] = 'Flank Skin with Spiced Flavor';
```

Finding the Size of an Array

- The **count()** function tells you the number of elements in an array
- An **empty array** (an array with no elements in it), count() **returns 0.**
- An **empty array** also evaluates to **false** in an if() test expression.
  - For example: **$x = []; if($x) print "$x is an empty array"**

```
$dinner = array('Sweet Corn and Asparagus',
                'Lemon Chicken',
                'Braised Bamboo Fungus');

$dishes = count($dinner);

print "There are $dishes things for dinner.";
```

Example 4-7 prints:

```
There are 3 things for dinner.
```

Looping Through Arrays - **foreach**

- The **foreach()** construct lets you run a code block once for each element in an array.

```
$meal = array('breakfast' => 'Walnut Bun',
              'lunch' => 'Cashew Nuts and White Mushrooms',
              'snack' => 'Dried Mulberries',
              'dinner' => 'Eggplant with Chili Sauce');
print "<table>\n";
foreach ($meal as $key => $value) {
    print "<tr><td>$key</td><td>$value</td></tr>\n";
}
print '</table>';
```

Example 4-8 prints:

```
<table>
<tr><td>breakfast</td><td>Walnut Bun</td></tr>
<tr><td>lunch</td><td>Cashew Nuts and White Mushrooms</td></tr>
<tr><td>snack</td><td>Dried Mulberries</td></tr>
<tr><td>dinner</td><td>Eggplant with Chili Sauce</td></tr>
</table>
```

Looping Through Arrays - **foreach**

- Inside the foreach(), changing the values of $key and $value **doesn't affect the elements** in the actual array.
- If you want to change the array element values, use the **$key variable as an index** into the array.

```php
foreach ($meals as $dish => $price) {
    // $price = $price * 2 does NOT work
    $meals[$dish] = $meals[$dish] * 2;
}

// Iterate over the array again and print the changed values
foreach ($meals as $dish => $price) {
    printf("The new price of %s is \$%.2f.\n",$dish,$price);
}
```

Example 4-10 prints:

```
The new price of Walnut Bun is $2.00.
The new price of Cashew Nuts and White Mushrooms is $9.90.
The new price of Dried Mulberries is $6.00.
The new price of Eggplant with Chili Sauce is $13.00.
```

Looping Through Arrays - **foreach**

Example 4-11. Using foreach() with numeric arrays

```
$dinner = array('Sweet Corn and Asparagus',
                'Lemon Chicken',
                'Braised Bamboo Fungus');
foreach ($dinner as $dish) {
    print "You can eat: $dish\n";
}
```

Example 4-11 prints:

```
You can eat: Sweet Corn and Asparagus
You can eat: Lemon Chicken
You can eat: Braised Bamboo Fungus
```

Looping Through Arrays - **foreach**

- If elements of a numeric array were added in a different order than how their keys would usually be ordered, this could produce **unexpected results**.

```
$letters[0] = 'A';
$letters[1] = 'B';
$letters[3] = 'D';
$letters[2] = 'C';

foreach ($letters as $letter) {
    print $letter;
}
```

Example 4-14 prints:

```
ABDC
```

```
for ($i = 0, $num_letters = count($letters); $i < $num_letters; $i++) {
    print $letters[$i];
}
```

This prints:

```
ABCD
```

Locate a particular **key** in an array

- To check for an element with a certain **key**, use **array_key_exists()**

```
$meals = array('Walnut Bun' => 1,
               'Cashew Nuts and White Mushrooms' => 4.95,
               'Dried Mulberries' => 3.00,
               'Eggplant with Chili Sauce' => 6.50,
               'Shrimp Puffs' => 0); // Shrimp Puffs are free!
$books = array("The Eater's Guide to Chinese Characters",
               'How to Cook and Eat in Chinese');

// This is true
if (array_key_exists('Shrimp Puffs',$meals)) {
    print "Yes, we have Shrimp Puffs";
}
// This is false

 if (array_key_exists('Steak Sandwich',$meals)) {
     print "We have a Steak Sandwich";
 }
 // This is true
 if (array_key_exists(1, $books)) {
     print "Element 1 is How to Cook and Eat in Chinese";
 }
```

Locate a particular **value** in an array

- To check for an element with a particular **value**, use **in_array()**,

```
$meals = array('Walnut Bun' => 1,
               'Cashew Nuts and White Mushrooms' => 4.95,
               'Dried Mulberries' => 3.00,
               'Eggplant with Chili Sauce' => 6.50,
               'Shrimp Puffs' => 0);
$books = array("The Eater's Guide to Chinese Characters",
               'How to Cook and Eat in Chinese');

// This is true: key Dried Mulberries has value 3.00
if (in_array(3, $meals)) {
  print 'There is a $3 item.';
}
// This is true
if (in_array('How to Cook and Eat in Chinese', $books)) {
  print "We have How to Cook and Eat in Chinese";
}
// This is false: in_array() is case-sensitive
if (in_array("the eater's guide to chinese characters", $books)) {
  print "We have the Eater's Guide to Chinese Characters.";
}
```

Locate a particular **value** in an array

- The **array_search()** function, if it finds an element, it **returns the element key** instead of true.

```
$meals = array('Walnut Bun' => 1,
               'Cashew Nuts and White Mushrooms' => 4.95,
               'Dried Mulberries' => 3.00,
               'Eggplant with Chili Sauce' => 6.50,
               'Shrimp Puffs' => 0);

$dish = array_search(6.50, $meals);
if ($dish) {
    print "$dish costs \$6.50";
}
```

Example 4-17 prints:

```
Eggplant with Chili Sauce costs $6.50
```

Modifying Arrays

- You can operate on individual array elements just like regular scalar variables, using arithmetic, logical, and other operators.

```
$dishes['Beef Chow Foon'] = 12;
$dishes['Beef Chow Foon']++;
$dishes['Roast Duck'] = 3;

$dishes['total'] = $dishes['Beef Chow Foon'] + $dishes['Roast Duck'];

if ($dishes['total'] > 15) {
    print "You ate a lot: ";
}

print 'You ate ' . $dishes['Beef Chow Foon'] . ' dishes of Beef Chow Foon.';
```

Example 4-18 prints:

```
You ate a lot: You ate 13 dishes of Beef Chow Foon.
```

Using element arrays in a string

- **Inside strings, don't put quotes** around the element key

```
$meals['breakfast'] = 'Walnut Bun';
$meals['lunch'] = 'Eggplant with Chili Sauce';
$amounts = array(3, 6);

print "For breakfast, I'd like $meals[breakfast] and for lunch,\n";
print "I'd like $meals[lunch]. I want $amounts[0] at breakfast and\n";
print "$amounts[1] at lunch.";
```

- If you have an array key that has **whitespace or other punctuation** in it, interpolate it **with curly braces**

```
$meals['Walnut Bun'] = '$3.95';
$hosts['www.example.com'] = 'website';

print "A Walnut Bun costs {$meals['Walnut Bun']}.\n";
print "www.example.com is a {$hosts['www.example.com']}.";
```

Example 4-20 prints:

```
A Walnut Bun costs $3.95.
www.example.com is a website.
```

Remove elements by **unset()**

- To remove an element from an array, use **unset()**

```php
unset($dishes['Roast Duck']);
```

- Removing an element with unset() is **different** than just setting the element value to **0 or the empty** string.

Print all of the values in an array

- Use the **implode()** function to print all of the values in an array at once

```php
$dimsum = array('Chicken Bun','Stuffed Duck Web','Turnip Cake');
$menu = implode(', ', $dimsum);
print $menu;
```

Example 4-21 prints:

```
Chicken Bun, Stuffed Duck Web, Turnip Cake
```

To implode an array with no delimiter, use the empty string as the first argument to `implode()`:

```php
$letters = array('A','B','C','D');
print implode('',$letters);
```

This prints:

```
ABCD
```

Creating Multidimensional Arrays

- Use the array() construct or the [] to create **arrays that have more arrays as element values**

```
$meals = array('breakfast' => ['Walnut Bun','Coffee'],
               'lunch'     => ['Cashew Nuts', 'White Mushrooms'],
               'snack'     => ['Dried Mulberries','Salted Sesame Crab']);

$lunches = [ ['Chicken','Eggplant','Rice'],
             ['Beef','Scallions','Noodles'],
             ['Eggplant','Tofu'] ];
```

```
$flavors = array('Japanese' => array('hot' => 'wasabi',
                                     'salty' => 'soy sauce'),
                 'Chinese'  => array('hot' => 'mustard',
                                     'pepper-salty' => 'prickly ash'));
```

Accessing Multidimensional Arrays

```
print $meals['lunch'][1];              // White Mushrooms
print $meals['snack'][0];              // Dried Mulberries
print $lunches[0][0];                  // Chicken
print $lunches[2][1];                  // Tofu
print $flavors['Japanese']['salty']; // soy sauce
print $flavors['Chinese']['hot'];    // mustard
```

Iterating through a multidimensional array

```
$flavors = array('Japanese' => array('hot' => 'wasabi',
                                     'salty' => 'soy sauce'),
                 'Chinese'  => array('hot' => 'mustard',
                                     'pepper-salty' => 'prickly ash'));

// $culture is the key and $culture_flavors is the value (an array)
foreach ($flavors as $culture => $culture_flavors) {
    // $flavor is the key and $example is the value
    foreach ($culture_flavors as $flavor => $example) {
        print "A $culture $flavor flavor is $example.\n";
    }
}
```

Example 4-31 prints:

```
A Japanese hot flavor is wasabi.
A Japanese salty flavor is soy sauce.
A Chinese hot flavor is mustard.
A Chinese pepper-salty flavor is prickly ash.
```

Exercises

1. According to the US Census Bureau, the 10 largest American cities (by population) in 2010 were as follows:

   - New York, NY (8,175,133 people)
   - Los Angeles, CA (3,792,621)
   - Chicago, IL (2,695,598)
   - Houston, TX (2,100,263)
   - Philadelphia, PA (1,526,006)
   - Phoenix, AZ (1,445,632)
   - San Antonio, TX (1,327,407)
   - San Diego, CA (1,307,402)
   - Dallas, TX (1,197,816)
   - San Jose, CA (945,942)

   Define an array (or arrays) that holds this information about locations and populations. Print a table of locations and population information that includes the total population in all 10 cities.

2. Modify your solution to the previous exercise so that the rows in the result table are ordered by population. Then modify your solution so that the rows are ordered by city name.
3. Modify your solution to the first exercise so that the table also contains rows that hold state population totals for each state represented in the list of cities.

Exercises

4. For each of the following kinds of information, state how you would store it in an array and then give sample code that creates such an array with a few elements. For example, for the first item, you might say, "An associative array whose key is the student's name and whose value is an associative array of grade and ID number," as in the following:

```
$students = [ 'James D. McCawley' => [ 'grade' => 'A+','id' => 271231 ],
              'Buwei Yang Chao' => [ 'grade' => 'A', 'id' => 818211] ];
```

   a. The grades and ID numbers of students in a class
   b. How many of each item in a store inventory are in stock
   c. School lunches for a week: the different parts of each meal (entrée, side dish, drink, etc.) and the cost for each day
   d. The names of people in your family
   e. The names, ages, and relationship to you of people in your family

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();*

Functions can be defined before or after calling them.

```
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>';
}
```

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.**

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

```php
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 . '">';
}
```

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

```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
```

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

```
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

```php
$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
```

```php
$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 have **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*.

Working with Objects.

Chapter 6

Object-Oriented Programming

- OOP helps to organize better your code
- Objects are great for making reusable code
- An object is a structure that combines data with the logic that operates on it
- Objects provide an organizational structure for grouping related variables and functions together.

OOP Terms

- **Class:** A template that describes the variables and functions for a kind/group of objects. (e.g. Entree)
- **Method:** A function defined in a class. (e.g. hasIngredient)
- **Property:** A variable defined in a class. (e.g. $name)
- **Instance:** An individual usage of a class. If you create three instances of a class, each of these instances is based on the same class, they differ by property values. The methods in each instance contain the same instructions, but may produce different results because they have different property values. Creating a new instance of a class is called "instantiating an object."
- **Constructor:** A special method that is automatically run when an object is instantiated. Usually, constructors set up object properties and do other housekeeping that makes the object ready for use.
- **Static method:** A special kind of method that can be called without instantiating a class.

```php
<?php
class Foo {
    public $aMemberVar = 'aMemberVar Member Variable';
    public $aFuncName = 'aMemberFunc';


    function aMemberFunc() {
        print 'Inside `aMemberFunc()`';
    }
}

$foo = new Foo;
?>
```

this keyword

- this keyword is used inside a class, generally withing the member functions, to access non-static members of a class (variables or functions) for the current object.

```php
<?php
    class Person {
        // first name of person
        private $name;

        // public function to set value for name (setter method)
        public function setName($name) {
            $this->name = $name;
        }

        // public function to get value of name (getter method)
        public function getName() {
            return $this->name;
        }
    }

    // creating class object
    $john = new Person();

    // calling the public function to set fname
    $john->setName("John Wick");

    // getting the value of the name variable
    echo "My name is " . $john->getName();

?>
```

Class and Object

- Defining a class (Entree)
- Then creating two objects/instances (soup and sandwich)
- Use -> to access properties or methods of an object

```php
class Entree {
    public $name;
    public $ingredients = array();

    public function hasIngredient($ingredient) {
        return in_array($ingredient, $this->ingredients);
    }
}
```

```php
// Create an instance and assign it to $soup
$soup = new Entree;
// Set $soup's properties
$soup->name = 'Chicken Soup';
$soup->ingredients = array('chicken', 'water');

// Create a separate instance and assign it to $sandwich
$sandwich = new Entree;
// Set $sandwich's properties
$sandwich->name = 'Chicken Sandwich';
$sandwich->ingredients = array('chicken', 'bread');


foreach (['chicken','lemon','bread','water'] as $ing) {
    if ($soup->hasIngredient($ing)) {
        print "Soup contains $ing.\n";
    }
    if ($sandwich->hasIngredient($ing)) {
        print "Sandwich contains $ing.\n";
    }
}
```

Static Methods

- Add static keyword before function to make a method static.
- To call a static method, you put :: between the class name and the method name instead of ->

```
class Entree {
    public $name;
    public $ingredients = array();

    public function hasIngredient($ingredient) {
        return in_array($ingredient, $this->ingredients);
    }

    public static function getSizes() {
        return array('small','medium','large');
    }
}
```

```
$sizes = Entree::getSizes();
```

Constructors

- A constructor is a special method, which is run when the object is created.
- Constructors typically handle setup and housekeeping tasks that make the object ready to use
- In PHP, the constructor method of a class is always called __construct()
- The input parameters can be different name than the properties.
- Inside a constructor, the $this keyword refers to the specific object instance being constructed.
- the constructor function doesn't return a value

```php
class Entree {
    public $name;
    public $ingredients = array();

    public function __construct($name, $ingredients) {
        $this->name = $name;
        $this->ingredients = $ingredients;
    }

    public function hasIngredient($ingredient) {
        return in_array($ingredient, $this->ingredients);
    }
}
```

Constructors and Creating a New Object

To pass arguments to the constructor, treat the class name like a function name when you invoke the new operator.

```php
// Some soup with name and ingredients
$soup = new Entree('Chicken Soup', array('chicken', 'water'));

// A sandwich with name and ingredients
$sandwich = new Entree('Chicken Sandwich', array('chicken', 'bread'));
```

Extending an Object

- A subclass (AKA child class) inherits all the methods and properties of an existing class (the parent class), but then can change them or add its own.
- It's as if you retyped the definition of Entree inside the definition of ComboMeal, but you get that without actually typing

```
class ComboMeal extends Entree {

    public function hasIngredient($ingredient) {
        foreach ($this->ingredients as $entree) {
            if ($entree->hasIngredient($ingredient)) {
                return true;
            }
        }
        return false;
    }
}
```

Using a Subclass

.

```
// Some soup with name and ingredients
$soup = new Entree('Chicken Soup', array('chicken', 'water'));

// A sandwich with name and ingredients
$sandwich = new Entree('Chicken Sandwich', array('chicken', 'bread'));

// A combo meal
$combo = new ComboMeal('Soup + Sandwich', array($soup, $sandwich));

foreach (['chicken','water','pickles'] as $ing) {
    if ($combo->hasIngredient($ing)) {
        print "Something in the combo contains $ing.\n";
    }
}
```

Constructor for a Subclass

- We can add a constructor for a subclass. We must call the parent constructor **explicitly**
- parent::__construct() refers to the constructor in the parent class.

```php
public function __construct($name, $entrees) {
    parent::__construct($name, $entrees);
    foreach ($entrees as $entree) {
        if (! $entree instanceof Entree) {
            throw new Exception('Elements of $entrees must be Entree objects');
        }
    }
}
```

Property and Method Visibility

- Visibility of the properties or methods can be:
  - Public
  - Private
  - Protected

- The public vidibility means all other classes can access the properties and methods.
- The private visibility prevents any code outside the class from accessing the property.
- The protected visibility means that the only subclass code can access the property

```php
class Entree {
    private $name;
    protected $ingredients = array();

    /* Since $name is private, this provides a way to read it */
    public function getName() {
        return $this->name;
    }

    public function __construct($name, $ingredients) {
        if (! is_array($ingredients)) {
            throw new Exception('$ingredients must be an array');
        }
        $this->name = $name;
        $this->ingredients = $ingredients;
    }
```

Property and Method Visibility

- getName() method is public and provides access to a private property.
- This kind of method is called an accessor. (getter)
- We can have a modifier (setter) method too, to change(set) the value of a private property.

```php
class Entree {
    private $name;
    protected $ingredients = array();

    /* Since $name is private, this provides a way to read it */
    public function getName() {
        return $this->name;
    }

    public function __construct($name, $ingredients) {
        if (! is_array($ingredients)) {
            throw new Exception('$ingredients must be an array');
        }
        $this->name = $name;
        $this->ingredients = $ingredients;
    }
```

Exercises

1. Create a class called `Ingredient`. Each instance of this class represents a single ingredient. The instance should keep track of an ingredient's name and its cost.
2. Add a method to your `IngredientCost` class that changes the cost of an ingredient.
3. Make a subclass of the `Entree` class used in this chapter that accepts `Ingredient` objects instead of string ingredient names to specify the ingredients. Give your `Entree` subclass a method that returns the total cost of the entrée.
4. Put your `Ingredient` class into its own namespace and modify your other code that uses `IngredientCost` to work properly.

Working with Files

CHAPTER 9

Working with files

- A web application is typically uses a database to store data
- But using plain text files is also beneficial
  - For example:
    - *storing an HTML template in a file and reading it and printing by PHP*
    - *reading and writing the CSV (commaseparated value) files*

Reading an **Entire** Files

- To read the contents of a file into a string, use file_get_contents()

```
<html>
<head><title>{page_title}</title></head>
<body bgcolor="{color}">

<h1>Hello, {name}</h1>

</body>
</html>
```

```
// Load the template file from the previous example
$page = file_get_contents('page-template.html');

// Insert the title of the page
$page = str_replace('{page_title}', 'Welcome', $page);

// Make the page blue in the afternoon and
// green in the morning
if (date('H')>= 12 ) {
    $page = str_replace('{color}', 'blue', $page);
} else {
    $page = str_replace('{color}', 'green', $page);
}

// Take the username from a previously saved session
// variable
$page = str_replace('{name}', $_SESSION['username'], $page);

// Print the results
print $page;
```

Writing a **Entire** File

- To write the content of a string into a file use file_put_contents()

```php
// Write the results to page.html
file_put_contents('page.html', $page);
```

- The file_get_contents() and file_put_contents() functions are fine when you want to work with an entire file at once.

file() function

- You can use the file() function to access each line of a file.
- The file() function returns an array. The array elements are a string containg one line of the file, newline included
- Very convenient, but problematic with very large files.

Example 9-5. people.txt for Example 9-4

```
alice@example.com|Alice Liddell
bandersnatch@example.org|Bandersnatch Gardner
charles@milk.example.com|Charlie Tenniel
dodgson@turtle.example.com|Lewis Humbert
```

```php
foreach (file('people.txt') as $line) {
    $line = trim($line);
    $info = explode('|', $line);
    print '<li><a href="mailto:' . $info[0] . '">' . $info[1] ."</li>\n";
}
```

```
<li><a href="mailto:alice@example.com">Alice Liddell</li>
<li><a href="mailto:bandersnatch@example.org">Bandersnatch Gardner</li>
<li><a href="mailto:charles@milk.example.com">Charlie Tenniel</li>
<li><a href="mailto:dodgson@turtle.example.com">Lewis Humbert</li>
```

Understanding File Permissions

- To read or write a file, the PHP engine must have permission from the OS to do so
- Every file is assigned some permissions
- Every user or an application has an account and is assigned some permissions
- Web servers containing PHP Engine has an account with specific permissons too
- Web server's account the privileges are more limited than the users privileges
- The web server (and the PHP engine) need to be able to read all of website files, but shouldn't be able to change them.

fopen(), feof(), and fgets() functions

- fopen() opens file or URL and binds it to a stream

```
$fh = fopen('people.txt','rb');
while ((! feof($fh)) && ($line = fgets($fh))) {
    $line = trim($line);
    $info = explode('|', $line);
    print '<li><a href="mailto:' . $info[0] . '">' . $info[1] ."</li>\n";
}
fclose($fh);
```

- The mode parameter specifies the type of access you require to the stream. It may be any of values on the rigth:

File modes

- Use 'b' to force binary mode, which will not translate your data.
- To use it, specify 'b' as the last character of the mode parameter, for example: **wb** or **rb**

- R: read
- W: write
- A: append
- X: exist
- C: create
- +: read+write

- PHP: fopen - Manual

Mode	Allowable actions	Position bookmark starting point	Clear contents?	If the file doesn't exist?
`rb`	Reading	Beginning of file	No	Issue a warning, return `false`.
`rb+`	Reading, Writing	Beginning of file	No	Issue a warning, return `false`.
`wb`	Writing	Beginning of file	Yes	Try to create it.
`wb+`	Reading, writing	Beginning of file	Yes	Try to create it.
`ab`	Writing	End of file	No	Try to create it.
`ab+`	Reading, writing	End of file	No	Try to create it.
`xb`	Writing	Beginning of file	No	Try to create it; if the file does exist, issue a warning and return `false`.
`xb+`	Reading, writing	Beginning of file	No	Try to create it; if the file does exist, issue a warning and return `false`.
`cb`	Writing	Beginning of file	No	Try to create it.
`cb+`	Reading, writing	Beginning of file	No	Try to create it.

fopen(), feof(), fgets(), and fclose() functions

- fgets() — Gets a line or a length byte from file pointer
- fgets(resource $handle, int $length = ?): string|false
- The file pointer must be valid, and must point to a file successfully opened by fopen()

- feof() — Tests for end-of-file on a file pointer

```
feof(resource $stream): bool
```

- Returns true if the file pointer is at EOF or an error occurs; otherwise returns false.

```php
$fh = fopen('people.txt','rb');
while ((! feof($fh)) && ($line = fgets($fh))) {
    $line = trim($line);
    $info = explode('|', $line);
    print '<li><a href="mailto:' . $info[0] . '">' . $info[1] ."</li>\n";
}
fclose($fh);
```

fwrite()

- Once you've opened a file in a mode that allows writing, use the fwrite() function to write something to the file.
- The fwrite() function doesn't automatically add a newline to the end of the string you write.

```php
<?php
$filename = 'test.txt';
$somecontent = "Add this to the file\n";

// Let's make sure the file exists and is writable first.
if (is_writable($filename)) {

    // In our example we're opening $filename in append mode.
    // The file pointer is at the bottom of the file hence
    // that's where $somecontent will go when we fwrite() it.
    if (!$handle = fopen($filename, 'a')) {
         echo "Cannot open file ($filename)";
         exit;
    }

    // Write $somecontent to our opened file.
    if (fwrite($handle, $somecontent) === FALSE) {
        echo "Cannot write to file ($filename)";
        exit;
    }

    echo "Success, wrote ($somecontent) to file ($filename)";

    fclose($handle);

} else {
    echo "The file $filename is not writable";
}
?>
```

Working with CSV Files

- To read a line of a CSV file, use fgetcsv()
- It reads a line from the CSV file and returns an array containing each field in the line.

```
fgetcsv(
    resource $stream,
    int $length = 0,
    string $separator = ",",
    string $enclosure = '"',
    string $escape = "\\"
): array
```

```php
<?php
$row = 1;
if (($handle = fopen("test.csv", "r")) !== FALSE) {
    while (($data = fgetcsv($handle, 1000, ",")) !== FALSE) {
        $num = count($data);
        echo "<p> $num fields in line $row: <br /></p>\n";
        $row++;
        for ($c=0; $c < $num; $c++) {
            echo $data[$c] . "<br />\n";
        }
    }
    fclose($handle);
}
?>
```

- The fputcsv() function takes a file handle and an array of values as arguments and writes those values, formatted as proper CSV, to the file.

Checking for Errors

- To check whether a file or directory exists, use file_exists()

```
if (file_exists('/usr/local/htdocs/index.html')) {
    print "Index file is there.";
} else {
    print "No index file in /usr/local/htdocs.";
}
```

- To determine whether your program has permission to read or write a particular file, use is_readable() or is_writeable()

```
$template_file = 'page-template.html';
if (is_readable($template_file)) {
    $template = file_get_contents($template_file);
} else {
    print "Can't read template file.";
}
```

Checking for Errors

To write robust file-handling code, you should check the return value of each file-related function.

```
$fh = fopen('people.txt','rb');
if (! $fh) {
    print "Error opening people.txt: $php_errormsg";
} else {
    while (! feof($fh)) {
        $line = fgets($fh);
        if ($line !== false) {
            $line = trim($line);
            $info = explode('|', $line);
            print '<li><a href="mailto:' . $info[0] . '">' . $info[1] ."</li>\n";
        }
    }
    if (! fclose($fh)) {
        print "Error closing people.txt: $php_errormsg";
    }
}
```

Excercises

1. Outside of the PHP engine, create a new template file in the style of Example 9-1. Write a program that uses `file_get_contents()` and `file_put_contents()` to read the HTML template file, substitute values for the template variables, and save the new page to a separate file.
2. Outside of the PHP engine, create a file that contains some email addresses, one per line. Make sure a few of the addresses appear more than once in the file. Call that file *addresses.txt*. Then, write a PHP program that reads each line in *addresses.txt* and counts how many times each address appears. For each distinct address in *addresses.txt*, your program should write a line to another file, *addresses-count.txt*. Each line in *addresses-count.txt* should consist of the number of times an address appears in *addresses.txt*, a comma, and the email address. Write the lines to *addresses-count.txt* in sorted order from the address that occurs the most times in *addresses.txt* to the address that occurs the fewest times in *addresses.txt*.

Excercises

3. Display a CSV file as an HTML table. If you don't have a CSV file (or spreadsheet program) handy, use the data from Example 9-9.
4. Write a PHP program that displays a form that asks a user for the name of a file underneath the web server's document root directory. If that file exists on the server, is readable, and is underneath the web server's document root directory, then display the contents of the file. For example, if the user enters *article.html*, display the file *article.html* in the document root directory. If the user enters *catalog/show.php*, display the file *show.php* in the directory *catalog* under the document root directory. Table 7-1 tells you how to find the web server's document root directory.
5. Modify your solution to the previous exercise so that the program displays only files whose names end in *.html*. Letting users look at the PHP source code of any page on your site can be dangerous if those pages have sensitive information in them such as database usernames and passwords.

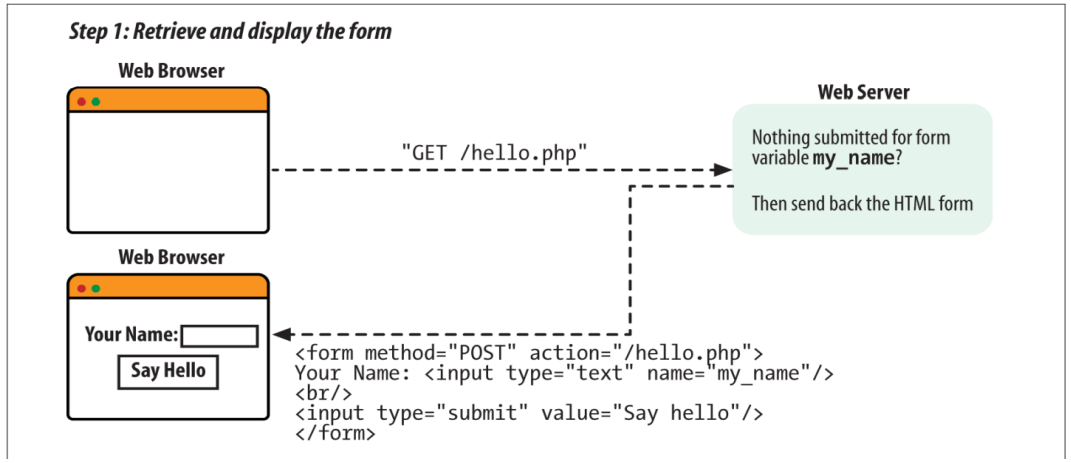
Web Forms

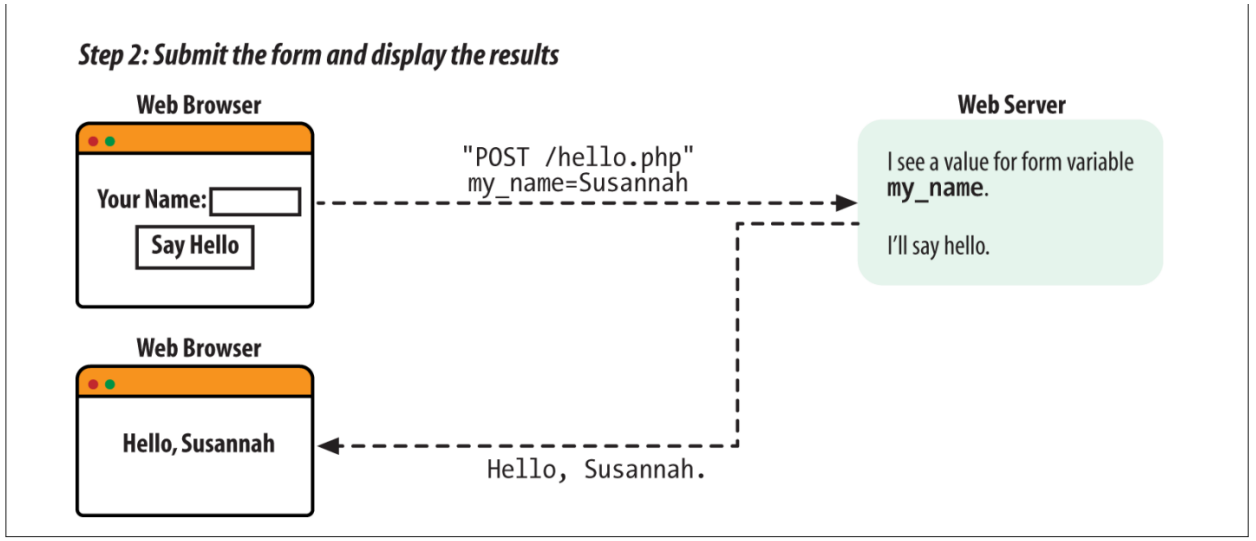
Chapter 7

Web Forms

- Form processing is an essential component of almost any web application.
- Users communicate with the server via web forms, Examples:
  - signing up for a new account
  - searching in a website
- Web Forms are Implemented in 2 steps:
  1. Using HTML, we construct a form with appropriate elements in it, such as text boxes, checkboxes, and buttons.
  2. Writing Server-side scripts (e.g. PHP) to process the user submitted information.

Review: Client-Server Communication





Useful Server Variables

Element	Example	Description
QUERY_STRING	category=kitchen&price=5	The part of the URL after the question mark where the URL parameters live. The example query string shown is for the URL *http://www.example.com/catalog/store.php?category=kitchen&price=5*.
PATH_INFO	/browse	Extra path information tacked onto the end of the URL after a slash. This is a way to pass information to a script without using the query string. The example PATH_INFO shown is for the URL *http://www.example.com/catalog/store.php/browse*.
SERVER_NAME	www.example.com	The name of the website on which the PHP engine is running. If the web server hosts many different virtual domains, this is the name of the particular virtual domain that is being accessed.
DOCUMENT_ROOT	/usr/local/htdocs	The directory on the web server computer that holds the documents available on the website. If the document root is */usr/local/htdocs* for the website *http://www.example.com*, then a request for *http://www.example.com/catalog/store.php* corresponds to the file */usr/local/htdocs/catalog/store.php*.
REMOTE_ADDR	175.56.28.3	The IP address of the user making the request to your web server.

Useful Server Variables

Element	Example	Description
REMOTE_HOST	pool0560.cvx.dialup.verizon.net	If your web server is configured to translate user IP addresses into hostnames, this is the hostname of the user making the request to your web server. Because this address-to-name translation is relatively expensive (in terms of computational time), most web servers do not do it.
HTTP_REFERER[a]	http://shop.oreilly.com/product/0636920029335.do	If someone clicked on a link to reach the current URL, HTTP_REFERER contains the URL of the page that contained the link. This value can be faked, so don't use it as your sole criterion for giving access to private web pages. It can, however, be useful for finding out who's linking to you.
HTTP_USER_AGENT	Mozilla/5.0 (Macintosh; Intel Mac OS X 10.10; rv:37.0) Gecko/20100101 Firefox/37.0	The web browser that retrieved the page. The example value is the signature of Firefox 37 running on OS X. Like with HTTP_REFERER, this value can be faked, but is useful for analysis.

[a] The correct spelling is HTTP_REFERRER. But it was misspelled in an early Internet specification document, so you frequently see the three-R version when web programming.

Accessing Form Parameters

- At every request, the PHP engine sets up some auto-global arrays that contain the values of parameters submitted in a form or passed in the URL.
- URL and form parameters from GET method forms are put into $_GET.
- Form parameters from POST method forms are put into $_POST.
- For Example:
  - The URL http://www.example.com/catalog.php?product_id=21&category=fryingpan puts two values into $_GET:
    - *$_GET['product_id'] is set to 21*
    - *$_GET['category'] is set to fryingpan*
  - the same values could be put into $_POST if the form method was post

```
<form method="POST" action="catalog.php">
<input type="text" name="product_id">
<select name="category">
<option value="ovenmitt">Pot Holder</option>
<option value="fryingpan">Frying Pan</option>
<option value="torch">Kitchen Torch</option>
</select>
<input type="submit" name="submit">
</form>
```

null coalesce operator (??)

- The coalesce, or ?? operator returns the result of its first operand if it exists and is not NULL, or else its second operand

```php
// Fetches the request parameter user and results in 'nobody' if it doesn't exist
$username = $_GET['user'] ?? 'nobody';
// equivalent to: $username = isset($_GET['user']) ? $_GET['user'] : 'nobody';
```

- Null coalesce introduced in PHP 7.0, for older versions use ternary operator (? :) and isset()

  - It is equivalent to: `$username = isset($_GET['user']) ? $_GET['user'] : 'nobody';`

- OR

```php
if (isset($_POST['product_id'])) {
   print $_POST['product_id'];
   }
```

Element with multiple values

- A form element with multiple values its name must end with []

```
<form method="POST" action="eat.php">
<select name="lunch[]" multiple>
<option value="pork">BBQ Pork Bun</option>
```

- If the form above is submitted with 2 values selected, then $_POST['lunch'] becomes a two-element array

```
<?php
if (isset($_POST['lunch'])) {
    foreach ($_POST['lunch'] as $choice) {
        print "You want a $choice bun. <br/>";
    }
}
?>
```

Form Processing with Functions

```
// Logic to do the right thing based on
// the request method
if ($_SERVER['REQUEST_METHOD'] == 'POST') {
    process_form();
} else {
    show_form();
}

// Do something when the form is submitted
function process_form() {
    print "Hello, ". $_POST['my_name'];
}

// Display the form
function show_form() {
    print<<<_HTML_
<form method="POST" action="$_SERVER[PHP_SELF]">
Your name: <input type="text" name="my_name">
<br/>
<input type="submit" value="Say Hello">
</form>
_HTML_;
}
```

Validating Data

```php
// Logic to do the right thing based on
// the request method
if ($_SERVER['REQUEST_METHOD'] == 'POST') {
    if (validate_form()) {
        process_form();
    } else {
        show_form();
    }
} else {
    show_form();
}
```

```php
// Check the form data
function validate_form() {
    // Is my_name at least 3 characters long?
    if (strlen($_POST['my_name']) < 3) {
        return false;
    } else {
        return true;
    }
}
```

Validating Data

- Ideally, when validation fails, you should explain the error to the user, if appropriate, redisplay the errornous value entered in the form element.

```php
// Logic to do the right thing based on
// the request method
if ($_SERVER['REQUEST_METHOD'] == 'POST') {
    // If validate_form() returns errors, pass them to show_form()
    if ($form_errors = validate_form()) {
        show_form($form_errors);
    } else {
        process_form();
    }
} else {
    show_form();
}
```

```php
// Check the form data
function validate_form() {
    // Start with an empty array of error messages
    $errors = array();

    // Add an error message if the name is too short
    if (strlen($_POST['my_name']) < 3) {
        $errors[  ] = 'Your name must be at least 3 letters long.';
    }

    // Return the (possibly empty) array of error messages
    return $errors;
}
```

```php
// Display the form
function show_form($errors = ) {

   // If some errors were passed in, print them out
   if ($errors) {
       print 'Please correct these errors: <ul><li>';
       print implode('</li><li>', $errors);
       print '</li></ul>';
   }
```

Validating Data

- The code in previous slide takes advantage of the fact that an empty array evaluates to false.
- Required Elements
- To check for required elements, check the element's length with strlen()

```php
if (strlen($_POST['email']) == 0) {
    $errors[] = "You must enter an email address.";
}
```

- Note:
- A if (! $_POST['quantity']) treats a value that evaluates to false as an error. Using strlen() lets users enter a value such as 0 into a required element.

Validating Numeric Values

- use filter_input() function with an appropriate filter, to check for numeric of string
  - The FILTER_VALIDATE_INT checks for integers
  - The FILTER_VALIDATE_FLOAT checks for floating-point numbers

```php
$ok = filter_input(INPUT_POST, 'age', FILTER_VALIDATE_INT);
if (is_null($ok) || ($ok === false)) {
    $errors[] = 'Please enter a valid age.';
}
```

- INPUT_POST is a predefined constant which refers to the elements of $_POST array.

- If the specified input element is valid, filter_input() returns the value.
- If the specified input element is missing, it returns null.

Validating Strings

- Use trim() to remove leading and trailing whitespaces.
- You can combine it with strlen() to disallow an entry of just whitespace characters.

```php
if (strlen(trim($_POST['name'])) == 0) {
    $errors[] = "Your name is required.";
}
```

Comprehensive Validation

```php
function validate_form() {
    $errors = array();
    $input = array();

    $input['age'] = filter_input(INPUT_POST, 'age', FILTER_VALIDATE_INT);
    if (is_null($input['age']) || ($input['age'] === false)) {
        $errors[] = 'Please enter a valid age.';
    }

    $input['price'] = filter_input(INPUT_POST, 'price', FILTER_VALIDATE_FLOAT);
    if (is_null($input['price']) || ($input['price'] === false)) {
        $errors[] = 'Please enter a valid price.';
    }

    // Use the null coalesce operator in case $_POST['name'] isn't set
    $input['name'] = trim($_POST['name'] ?? '');
    if (strlen($input['name']) == 0) {
        $errors[] = "Your name is required.";
    }

    return array($errors, $input);
}
```

Comprehensive Validation

Example 7-14. Handling errors and modified input data

```php
// Logic to do the right thing based on the request method
if ($_SERVER['REQUEST_METHOD'] == 'POST') {
    // If validate_form() returns errors, pass them to show_form()
    list($form_errors, $input) = validate_form();
    if ($form_errors) {
        show_form($form_errors);
    } else {
        process_form($input);
    }
} else {
    show_form();
}
```

- list($form_errors, $input) puts the first element of that returned array into the $form_errors variable and the second element into $input.
- Makes the code more readable

Number Range

- To check whether an integer falls within a certain range, use the min_range and max_range options of the FILTER_VALIDATE_INT filter.

```php
$input['age'] = filter_input(INPUT_POST, 'age', FILTER_VALIDATE_INT,
                             array('options' => array('min_range' => 18,
                                                      'max_range' => 65)));
```

- The FILTER_VALIDATE_FLOAT filter doesn't support the min_range and max_range options, so you need to do the comparisons yourself

```php
$input['price'] = filter_input(INPUT_POST, 'price', FILTER_VALIDATE_FLOAT);
if (is_null($input['price']) || ($input['price'] === false) ||
    ($input['price'] < 10.00) || ($input['price'] > 50.00)) {
    $errors[] = 'Please enter a valid price between $10 and $50.';
}
```

Email Addresses

- The FILTER_VALIDATE_EMAIL filter checks strings against the rules for valid email addresse syntax

```php
$input['email'] = filter_input(INPUT_POST, 'email', FILTER_VALIDATE_EMAIL);
if (! $input['email']) {
    $errors[] = 'Please enter a valid email address';
}
```

<select> Menu

- Although a user can't submit an off-menu value using common browsers, BUT an attacker can construct a request containing any arbitrary value without using a browser.
- To simplify display and validation of <select> menus:
  - put the menu choices in an array.
  - Then, iterate through that array to display the <select> menu inside the show_form() function.

```php
$sweets = array('Sesame Seed Puff','Coconut Milk Gelatin Square',
                'Brown Sugar Cake','Sweet Rice and Meat');

function generate_options($options) {
    $html = '';
    foreach ($options as $option) {
        $html .= "<option>$option</option>\n";
    }
    return $html;
}
```

```php
// Display the form
function show_form() {
    $sweets = generate_options($GLOBALS['sweets']);
    print<<<_HTML_
<form method="post" action="$_SERVER[PHP_SELF]">
Your Order: <select name="order">
$sweets
</select>
<br/>
<input type="submit" value="Order">
</form>
_HTML_;
}
```

<select> Menu

Inside `validate_form()`, use the array of `<select>` menu options like this:

```
$input['order'] = $_POST['order'];
if (! in_array($input['order'], $GLOBALS['sweets'])) {
    $errors[] = 'Please choose a valid order.';
}
```

HTML and JavaScript

- Submitted form data that contains HTML or JavaScript can cause big problems.
- Use strip_tags() to remove HTML tags from an input data

Example 7-21. Stripping HTML tags from a string

```
// Remove HTML from comments
$comments = strip_tags($_POST['comments']);
// Now it's OK to print $comments
print $comments;
```

If `$_POST['comments']` contains

```
I
<b>love</b> sweet <div
class="fancy">rice</div> &
tea.
```

then Example 7-21 prints:

```
I love sweet rice & tea.
```

HTML and JavaScript

- The strip_tags() behaves poorly with mismatched < and > characters.
- For example, it turns I <3 Monkeys to I because of < character

- Encoding instead of stripping the tags often gives better results.
- Converts the special characters to HTML entities
- (, &, and ")
  - < to &let;
  - > to >
  - & to &
  - " to &quote;

Example 7-22. Encoding HTML entities in a string

```
$comments = htmlentities($_POST['comments']);
// Now it's OK to print $comments
print $comments;
```

If `$_POST['comments']` contains

```
I
<b>love</b> sweet <div
class="fancy">rice</div> &
tea
```

then Example 7-22 prints:

```
I &lt;b&gt;love&lt;/b&gt; sweet &lt;div class=&quot;fancy
&quot;&gt;rice&lt;/div&gt; &amp; tea.
```

Exercises

1. What does `$_POST` look like when the following form is submitted with the third option in the `Braised Noodles` menu selected, the first and last options in the `Sweet` menu selected, and 4 entered into the text box?

```
<form method="POST" action="order.php">
Braised Noodles with: <select name="noodle">
<option>crab meat</option>
<option>mushroom</option>
<option>barbecued pork</option>
<option>shredded ginger and green onion</option>
</select>
<br/>
Sweet: <select name="sweet[]" multiple>
<option value="puff"> Sesame Seed Puff
<option value="square"> Coconut Milk Gelatin Square
<option value="cake"> Brown Sugar Cake
<option value="ricemeat"> Sweet Rice and Meat
</select>
<br/>
Sweet Quantity: <input type="text" name="sweet_q">
<br/>
<input type="submit" name="submit" value="Order">
</form>
```

Exercises

2. Write a `process_form()` function that prints out all submitted form parameters and their values. You can assume that form parameters have only scalar values.
3. Write a program that does basic arithmetic. Display a form with text box inputs for two operands and a `<select>` menu to choose an operation: addition, subtraction, multiplication, or division. Validate the inputs to make sure that they are numeric and appropriate for the chosen operation. The processing function should display the operands, the operator, and the result. For example, if the operands are 4 and 2 and the operation is multiplication, the processing function should display something like `4 * 2 = 8`.
4. Write a program that displays, validates, and processes a form for entering information about a package to be shipped. The form should contain inputs for the from and to addresses for the package, dimensions of the package, and weight of the package. The validation should check (at least) that the package weighs no more than 150 pounds and that no dimension of the package is more than 36 inches. You can assume that the addresses entered on the form are both US addresses, but you should check that a valid state and a zip code with valid syntax are entered. The processing function in your program should print out the information about the package in an organized, formatted report.
5. (Optional) Modify your `process_form()` function that enumerates all submitted form parameters and their values so that it correctly handles submitted form parameters that have array values. Remember, those array values could themselves contain arrays.

Databases

Chapter 8

Why Databases?

- Why not storing data in simple files? Why databases?
- There are three big reasons why using databases:
  - **Convenience**: if data is in a file, you must read, search to find data, change it and write back data!
  - **Simultaneous access:** if 2 users change same data at the same time, which change get the final effect?
  - **Security:** we cannot control access to part of a data file, but in the database we can

Organizing Data in a Database

- Data in your database is organized in tables
- Each table has rows and columns (fields)

ID	Name	Price	Is spicy?
1	Fried Bean Curd	5.50	0
2	Braised Sea Cucumber	9.95	0
3	Walnut Bun	1.00	0
4	Eggplant with Chili Sauce	6.50	1

- Structured Query Language (SQL) is a language used to ask questions of and give instructions to the database program.
- SQL is case-insensitive for its keywords
- But sensitive (by default) for the string values

Connecting to a Database Program

- To establish a connection to a database program, create a new PDO object.
- You pass the PDO constructor a string that describes the database you are connecting to
- It returns an object that you use in the rest of your program to exchange data with the database program.

```
$db = new PDO('mysql:host=db.example.com;dbname=restaurant','penguin','top^hat');
```

- The first argument is called a data source name (DSN):
  - It begins with what kind of database program to connect to (e.g. mysql)
  - then has a colon :
  - then some key=value pairs separated by semicolon providing information about how to connect.
- If the database connection needs a username and password, these are passed as the second and third arguments

- PDO support many DBMSs but if it not, you will get could not find driver message when creating a PDO object

The charset option, available with some database programs, specifies how the database program should handle non-English characters.

Table 8-1. PDO DSN prefixes and options

Database program	DSN prefix	DSN options	Notes
MySQL	`mysql`	`host`, `port`, `dbname`, `unix_socket`, `charset`	`unix_socket` is for local MySQL connections. Use it or `host` and `port`, but not both.
PostgreSQL	`pgsql`	`host`, `port`, `dbname`, `user`, `password`, others	The whole connection string is passed to an internal PostgreSQL connection function, so you can use any of the options listed in the PostgreSQL documentation.
Oracle	`oci`	`dbname`, `charset`	The value of `dbname` should either be an Oracle Instant Client connection URI of the form `//hostname:port/database` or an address name defined in your *tnsnames.ora* file.
SQLite	`sqlite`	None	After the prefix, the entire DSN must be either a path to an SQLite database file, or the string `:memory:` to use a temporary in-memory database.
ODBC	`odbc`	`DSN`, `UID`, `PWD`	The value for the `DSN` key inside the DSN string should either be a name defined in your ODBC catalog or a full ODBC connection string.
MS SQL Server or Sybase	`mssql`, `sybase`, `dblib`	`host`, `dbname`, `charset`, `appname`	The `appname` value is a string that the database program uses to describe your connection in its statistics. The `mssql` prefix is for when the PHP engine is using Microsoft's SQL Server libraries; the `sybase` prefix is for when the engine is using Sybase CT-Lib libraries; the `dblib` prefix is for when the engine is using the FreeTDS libraries.

PDO

- If all goes well with new PDO(), it returns an object that you use to interact with the database.
- If there is a problem connecting, it throws a PDOException exception. Make sure to catch exceptions so you can verify that the connection succeeded before going forward in your program.

```php
try {
    $db = new PDO('mysql:host=localhost;dbname=restaurant','penguin','top^hat');
    // Do some stuff with $db here
} catch (PDOException $e) {
    print "Couldn't connect to the database: " . $e->getMessage();
}
```

Creating a Table

- First you must create a table. This is usually a one-time operation.

```
try {
    $db = new PDO('sqlite:/tmp/restaurant.db');
    $db->setAttribute(PDO::ATTR_ERRMODE, PDO::ERRMODE_EXCEPTION);
    $q = $db->exec("CREATE TABLE dishes (
        dish_id INT,
        dish_name VARCHAR(255),
        price DECIMAL(4,2),
        is_spicy INT
)");
} catch (PDOException $e) {
    print "Couldn't create table: " . $e->getMessage();
}
```

```
CREATE TABLE dishes (
    dish_id INTEGER PRIMARY KEY,
    dish_name VARCHAR(255),
    price DECIMAL(4,2),
    is_spicy INT
)
```

Column type	Description
VARCHAR(*length*)	A variable-length string up to *length* characters long
INT	An integer
BLOB[a]	Up to 64 KB of string or binary data
DECIMAL(*total_digits*,*decimal_places*)	A decimal number with a total of *total_digits* digits and *decimal_places* digits after the decimal point
DATETIME[b]	A date and time, such as `1975-03-10 19:45:03` or `2038-01-18 22:14:07`

[a] PostgreSQL calls this BYTEA instead of BLOB.

[b] Oracle calls this DATE instead of DATETIME.

- setAttribute() ensures that PDO throws exceptions if there are problems with queries, not just a problem when connecting.

Dropping a table

- The opposite of CREATE TABLE is DROP TABLE.

```
DROP TABLE dishes
```

- NOTE: It removes a table and the data in it from a database

Putting Data into the Database

- To put some data into the database, pass an INSERT statement to the object's exec() method

```
try {
    $db = new PDO('sqlite:/tmp/restaurant.db');
    $db->setAttribute(PDO::ATTR_ERRMODE, PDO::ERRMODE_EXCEPTION);
    $affectedRows = $db->exec("INSERT INTO dishes (dish_name, price, is_spicy)
                               VALUES ('Sesame Seed Puff', 2.50, 0)");
} catch (PDOException $e) {
    print "Couldn't insert a row: " . $e->getMessage();
}
```

- The exec() method returns the number of rows affected by the SQL statement. In this case, it returns 1

- If something goes wrong with INSERT, an exception is thrown.

PDO error modes

- PDO has 3 error modes:
  - Silent
  - Warning
  - Exception
- The silent mode is the default.
- The exception error mode is activated by $db->setAttribute(PDO::ATTR_ERRMODE, PDO::ERRMODE_EXCEPTION)
- The other two error modes require you to check the return values from your PDO function calls
  - If there is an error, use additional PDO methods to find information about the error.

- The warning mode is activated by setting the PDO::ATTR_ERRMODE attribute to PDO::ERRMODE_WARNING

errorinfo()

Like other PDO methods, if exec() fails at its task, it returns false. (Question: why === instead of ==?)

```
if (false === $result) {
    $error = $db->errorInfo();
    print "Couldn't insert!\n";
    print "SQL Error={$error[0]}, DB Error={$error[1]}, Message={$error[2]}\n";
}
```

errorInfo() returns a three-element array with error information.

- The **first** element is an SQLSTATE error code. These are error codes that are mostly standardized across different database programs. In this case, HY000 is a catch-all for general errors.
- The **second** element is an error code specific to the particular database program in use.
- The **third** element is a textual message describing the error.

Warnign mode

- In this mode, functions behave as they do in silent mode—no exceptions, returning false on error—but the PHP engine also generates a warning-level error message.
- Depending on how you've configured error handling, this message may get displayed on screen or in a log file.

```php
// The constructor always throws an exception if it fails
try {
    $db = new PDO('sqlite:/tmp/restaurant.db');
} catch (PDOException $e) {
    print "Couldn't connect: " . $e->getMessage();
}
$db->setAttribute(PDO::ATTR_ERRMODE, PDO::ERRMODE_WARNING);
$result = $db->exec("INSERT INTO dishes (dish_size, dish_name, price, is_spicy)
                        VALUES ('large', 'Sesame Seed Puff', 2.50, 0)");
if (false === $result) {
    $error = $db->errorInfo();
    print "Couldn't insert!\n";
    print "SQL Error={$error[0]}, DB Error={$error[1]}, Message={$error[2]}\n";
}
```

Update Data in a Table

```
try {
    $db = new PDO('sqlite:/tmp/restaurant.db');
    $db->setAttribute(PDO::ATTR_ERRMODE, PDO::ERRMODE_EXCEPTION);
    // Eggplant with Chili Sauce is spicy
    // If we don't care how many rows are affected,
    // there's no need to keep the return value from exec()
    $db->exec("UPDATE dishes SET is_spicy = 1
                WHERE dish_name = 'Eggplant with Chili Sauce'");
    // Lobster with Chili Sauce is spicy and pricy
    $db->exec("UPDATE dishes SET is_spicy = 1, price=price * 2
               WHERE dish_name = 'Lobster with Chili Sauce'");
} catch (PDOException $e) {
    print "Couldn't insert a row: " . $e->getMessage();
}
```

Delete Data From a Table

- emember that exec() returns the number of rows changed or removed by an UPDATE or DELETE statement.
- Use the return value to find out how many rows that query affected.

```php
try {
    $db = new PDO('sqlite:/tmp/restaurant.db');
    $db->setAttribute(PDO::ATTR_ERRMODE, PDO::ERRMODE_EXCEPTION);
    // remove expensive dishes
    if ($make_things_cheaper) {
        $db->exec("DELETE FROM dishes WHERE price > 19.95");
    } else {
        // or, remove all dishes
        $db->exec("DELETE FROM dishes");
    }
} catch (PDOException $e) {
    print "Couldn't delete rows: " . $e->getMessage();
}
```

Inserting Form Data Safely

- Using unsanitized form data in SQL queries can cause a problem, called an "SQL injection attack."

```
$db->exec("INSERT INTO dishes (dish_name)
           VALUES ('$_POST[new_dish_name]')");
```

- If the submitted value for new_dish_name is reasonable, such as Fried Bean Curd, then the query succeeds.
- A query with an apostrophe in it causes a problem.
  - If the submitted value for new_dish_name is *General Tso's Chicken*, DBMS thinks that the apostrophe between Tso and s ends the string, so the s Chicken' after the second single quote is an unwanted syntax error.