

PHP

PHP: HYPERTEXT PREPROCESSOR

Myanmar IT Bootcamp

Course Outline

- Introduction
- Working with Text and Numbers
- Making Decision and Repeating Yourself
- Working with Arrays
- Functions and Files
- Working with Objects
- Working with Databases (MySQL)
- Cookies and Sessions
- Dates and Times
- Git

Introduction

- PHP is a computer scripting language.
- Originally designed for producing dynamic web pages
- Appeared in 1995
- PHP Group is responsible for the language, no formal specification
- Free software
- Runs on most operating systems and platforms
- URL:<http://www.php.net>

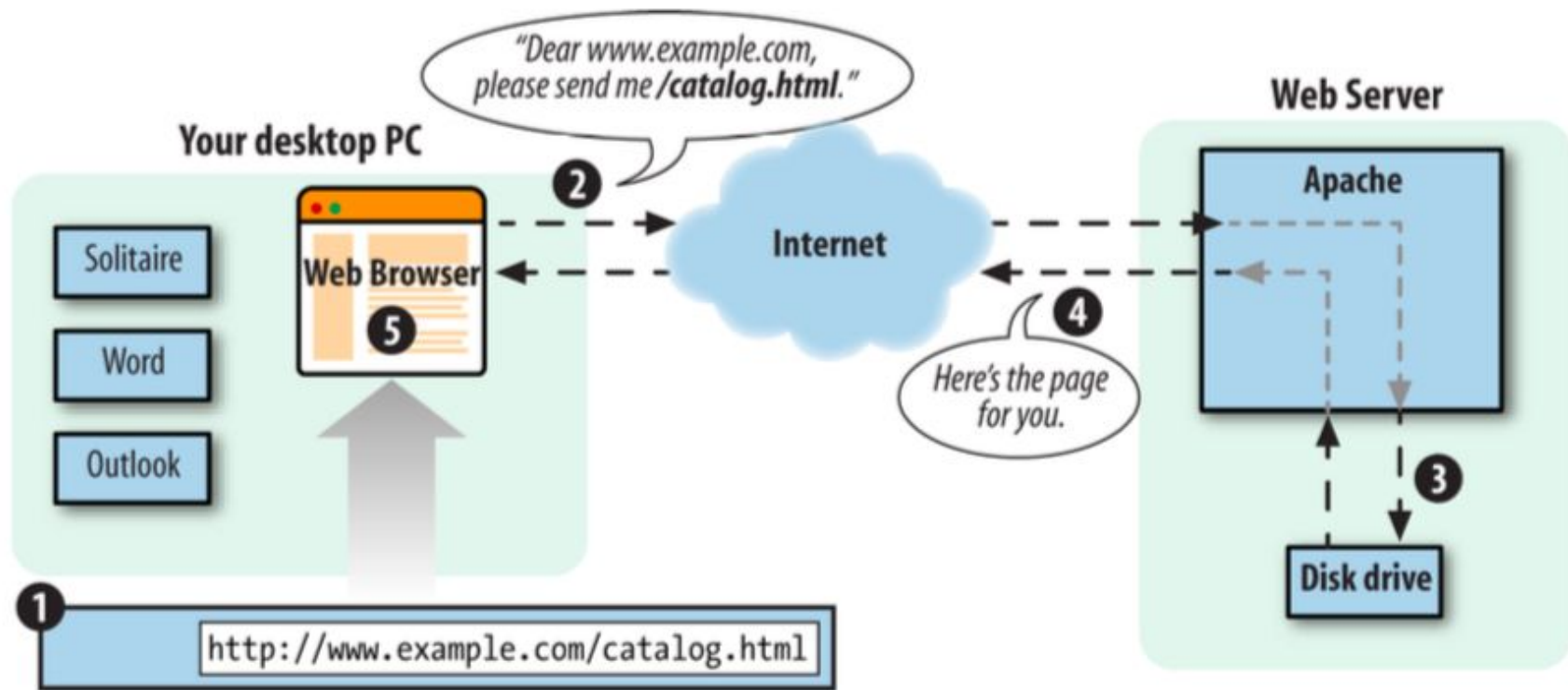


Figure 1-1. Client and server communication without PHP

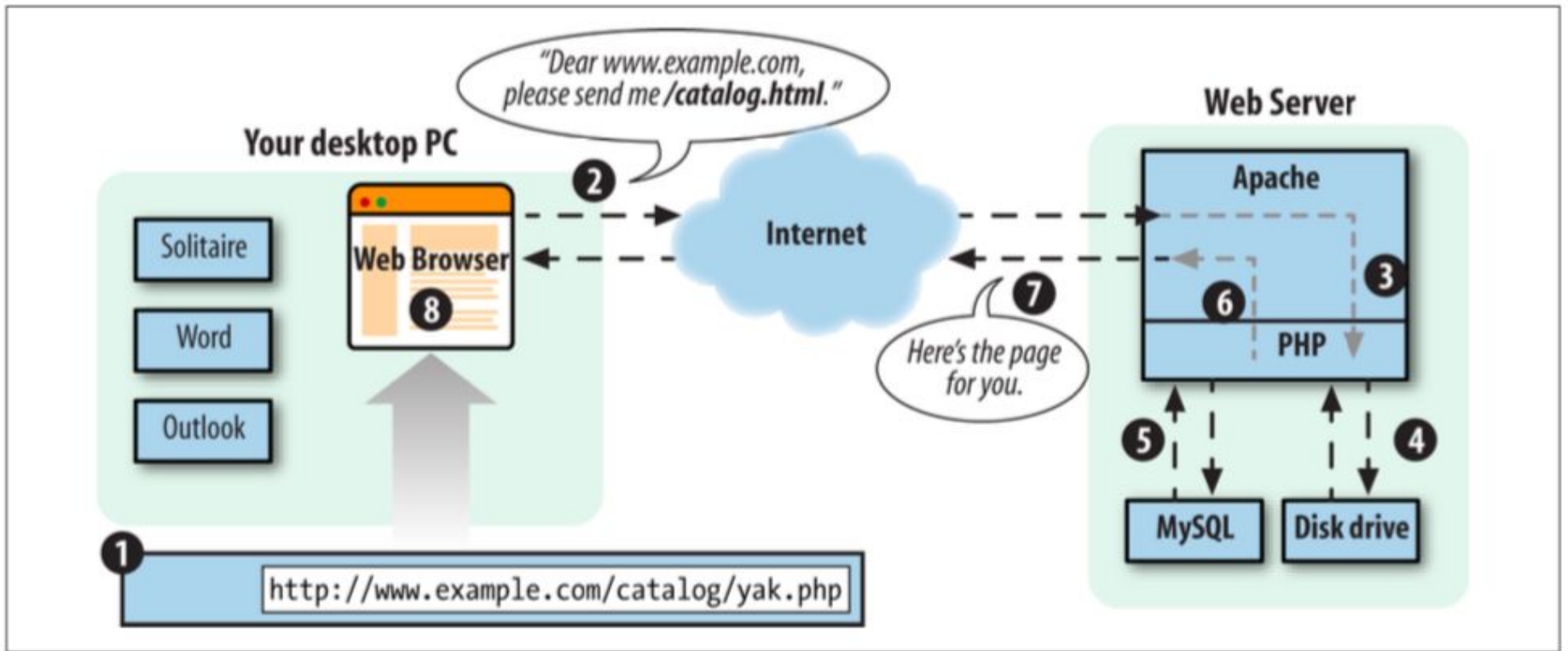


Figure 1-2. Client and server communication with PHP

- PHP's usage by a web server to create a response or document to send back to the browser
- PHP as a server-side language, meaning it runs on the web server (this is in contrast to a client-side language such as JavaScript that is run inside of a web browser)
- What you sign up for when you decide to use PHP: it's free (in terms of money and speech), cross-platform, popular, and designed for web programming
- How PHP programs that print information, process forms, and talk to a database appear
- Some basics of the structure of PHP programs, such as the PHP start and end tags (`<?php` and `?>`), whitespace, case-sensitivity, and comments

Basic Rules of PHP Programs

(1) Start and End Tags

`<?php`

`echo "Hello World!";`

`?>`

The PHP source code inside each set of `<?php ?>` tags is processed by the PHP engine.

(2) End sentence with semicolon “ ; “

```
<?php
```

```
echo “Hello World!” ;
```

```
echo “ This is PHP Bootcamp” ;
```

```
$name=“Mg Mg”;
```

```
echo “ My name is $name “ ;
```

```
?>
```


(3) Comments

`<?php`

`// This is single line comment`

`# This is also single line comment`

`echo "Hello World";`

`?>`

`<?php`

`/*`

`This is`

`Multi line`

`comment`

`*/`

`echo "Hello World!" ;`

`?>`

Working with Text and Numbers

(1) String

```
$name="MgMg";    $address="Yangon" ;
```

```
echo " $name live in $address" ; echo " I live in \"Yangon\" " ;
```

```
echo ' $name live in $address ' ; echo ' I live in \'Yangon\' ' ;
```

Use a \ to escape in a string

Use \' to use single quote inside single quote or double quote inside double quote.

Can declare string using double quote or Single quote.

When we output (echo) => Single quote cannot interpret variable as double quote .

We can use trim() and strlen() functions to validate 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) {
```

```
echo "Please enter a zip code that is 5 characters long.";
```

```
}
```

The trim() function removes whitespace from the beginning and end of a string. Combined with strlen(), which tells you the length of a string,

Comparing strings case-insensitively

To compare strings without paying attention to case, use `strcasecmp()`. It compares two strings while ignoring differences in capitalization. If the two strings you provide to `strcasecmp()` are the same independent of any differences between upper- and lowercase letters, it returns 0.

```
if (strcasecmp($_POST['email'], 'president@whitehouse.gov') == 0) {  
    echo "Welcome back, US President."  
}  
  
echo strtolower('Beef, CHICKEN, Pork, duCK');  
echo strtoupper('Beef, CHICKEN, Pork, duCK');  
echo substr("Hello Bootcampers",5);    str_replace();
```

Numbers

Numbers in PHP are expressed using familiar notation, although you can't use commas or any other characters to group thousands. You don't have to do anything special to use a number with a decimal part as compared to an integer.

```
echo "56" ; echo " 56.3" ; echo " 56.30" ; echo " 0.774422" ;
```

Arithmetic Operators

+ , - , * , / , %

```
$first_number=45 ; $second_number=12;
```

```
$sub= $first_number-$second_number ; $sum=$first_number+$second_number;
```

```
$mul=$first_number*$second_number; $div=$first_number/$second_number;
```

Variable Name declaration

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 if you're using a character encoding such as UTF-8 for your program file.

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;
```

```
echo 'The tax is ' . $tax_amount;  
echo "<br>"; // this prints a line break  
echo 'The total cost is ' . $total_cost;  
echo "<br>"; // this prints a line break  
echo $email_address;
```

```
<?php
```

```
$val = 299.97888;
```

```
$txt = sprintf("%.2f", $val);
```

```
echo $txt;
```

```
?>
```

(1) Combined assignment and addition

// Add 3 the regular way

`$price = $price + 3;`

// Add 3 with the combined operator

`$price += 3;`

(2) Combined assignment and concatenation

`$username = 'james';`

`$domain = '@example.com';`

// Concatenate \$domain to the end of \$username the regular way

`$username = $username . $domain;`

// Concatenate with the combined operator

`$username .= $domain;`

(3) Incrementing and decrementing

// Add 1 to \$birthday

`$birthday = $birthday + 1;`

// Add another 1 to \$birthday

`++$birthday;`

// Subtract 1 from \$years_left

`$years_left = $years_left - 1;`

// Subtract another 1 from \$years_left

`--$years_left;`

- Defining strings in your programs in three different ways: with single quotes, with double quotes, and as a here document
- Escaping: what it is and what characters need to be escaped in each kind of string Validating a string by checking its length, removing leading and trailing white- space from it, or comparing it to another string
- Formatting a string with printf()
- Manipulating the case of a string with strtolower(), strtoupper(), or ucwords()
- Selecting part of a string with substr()
- Changing part of a string with str_replace()
- Defining numbers in your programs
- Doing math with numbers
- Storing values in variables
- Naming variables appropriately
- Using combined operators with variables
- Using increment and decrement operators with variables

Making Decisions and Repeating Yourself

With the `if()` construct, you can have statements in your program that are only run if certain conditions are true. This lets your program take different actions depending on the circumstances.

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

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

Example 3-2. Multiple statements in an if() code block

```
echo "This is always printed.";
```

```
if ($logged_in) {
```

```
    echo "Welcome aboard, trusted user.";
```

```
    echo 'This is only printed if $logged_in is true.';
```

```
}
```

```
echo "This is also always printed.";
```

Example 3-4. Using elseif()

```
if ($logged_in) {  
    echo "Welcome aboard, trusted user.";  
}  
elseif ($new_messages) {  
    echo "Dear stranger, there are new messages.";  
}  
elseif ($emergency) {  
    echo "Stranger, there are no new messages, but  
    there is an emergency.";  
}
```

Example 3-3. Using else with if()

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

Example 3-5. elseif() with else

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

Example 3-6. The equal operator

```
if ($new_messages == 10) {  
  
    echo "You have ten new messages."  
  
}  
  
if ($new_messages == $max_messages) {  
  
    echo "You have the maximum number of  
    messages."  
  
}  
  
if ($dinner == 'Braised Scallops') {  
  
    echo "Yum! I love seafood."  
  
}
```

Example 3-7. The not-equal operator

```
if ($new_messages != 10) {  
  
    echo "You don't have ten new messages."  
  
}  
  
if ($dinner != 'Braised Scallops') {  
  
    echo "I guess we're out of scallops."  
  
}
```

Example 3-8. Less-than and greater-than (or equal to)

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

Example 3-9. Comparing floating-point numbers

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

Example 3-14. Using the negation operator

```
if ($finished == false) {  
  
    echo 'Not done yet!';  
  
}
```

```
if (! $finished) {  
  
    echo 'Not done yet!';  
  
}
```

Example 3-15. The negation operator

```
if (! strcmp($first_name,$last_name)) {  
  
    echo “$first_name and $last_name are equal.”;  
  
}
```


Example 3-16. Logical operators

```
if (($age >= 13) && ($age < 65)) {  
  
    echo "You are too old for a kid's discount and too young for the senior's  
discount."  
  
}  
  
if (($meal == 'breakfast') || ($dessert == 'souffle')) {  
  
    echo "Time to eat some eggs."  
  
}
```

Repeating Yourself

- (1) For Loop (Use when we know exact loop count)
- (2) While Loop (Use when we want to run on a condition)
- (3) do while loop (Use to skip first time and check on second time)

Every Loop has 3 steps :

(1) Initialize (\$i=1) (2) Condition Check (\$i<10) (3) Increment/decrement (\$i++)

- (1) For Loop

```
for($i=1;$i<10;$i++){  
  
    echo "<h3>Hello $i </h3>"  
  
}
```

Add using for loop

```
$sum=0;  
  
for($num=1;$num<=100;$num++){  
    $sum+=$num;  
  
}  
  
echo "Sum of 1 to 100 is $sum" ;
```

Multiply using for loop

```
$mul=1;  
  
for($num=1;$num<=10;$num++){  
    $mul*=$num;  
  
}  
  
echo "Multiplication of 1 to 10 is $mul" ;
```

Add using while loop

```
$sum=0; $num=1;
```

```
while($num<11){  
    $sum+=$num;  
  
    $num++;  
  
}
```

```
echo "Sum of 1 to 100 is $sum" ;
```

Multiply using while loop

```
$mul=1; $num=2;
```

```
while($num<11){  
    $mul*=$num;  
  
    $num++;  
  
}
```

```
echo "Multiplication of 1 to 10 is $mul" ;
```

Add using do while loop

```
$sum=0; $num=1;
```

```
do{  
    $sum+=$num;  
  
    $num++;
```

```
}while($num<11);
```

```
echo "Sum of 1 to 100 is $sum" ;
```

Multiply using do while loop

```
$mul=1; $num=2;
```

```
do{  
    $mul*=$num;  
  
    $num++;
```

```
}while($num<11);
```

```
echo "Multiplication of 1 to 10 is $mul" ;
```

Escape from Loop using “break” and “continue”

“break” is use to exit from loop and “continue” is use to skip one time in loop.

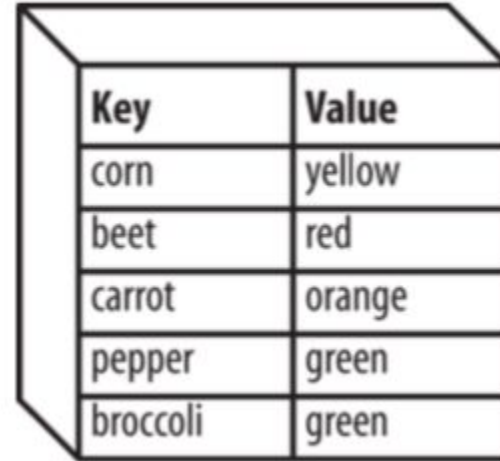
```
for($num=1;$num<10;$num++){  
    if($num==3){break;} // if($num==3){continue;}  
    echo "<br>$num";  
}
```

Working with 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. Each element has a key and a value.

To create an array, use the `array()` language construct. Specify a comma-delimited list of key/value pairs, with the key and the value separated by `=>`.



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

Creating an Array

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

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


Creating a Numeric Array

PHP provides some shortcuts for working with arrays that have only numbers as keys. 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.

```
1. $dinner = array(  
2.     'Sweet Corn and Asparagus',  
3.     'Lemon Chicken',  
4.     'Braised Bamboo Fungus'  
5. );
```

```
echo "I want $dinner[0] and $dinner[1]."; echo count($dinner); // size of array
```

Adding elements with []

1. *// Create \$lunch array with two elements*
2. *// This sets \$lunch[0]*
3. *\$lunch[] = 'Dried Mushrooms in Brown Sauce'; // This sets \$lunch[1]*
4. *\$lunch[] = 'Pineapple and Yu Fungus';*
5. *// Create \$dinner with three elements*
6. *\$dinner = array('Sweet Corn and Asparagus', 'Lemon Chicken', 'Braised Bamboo Fungus');*
7. *// Add an element to the end of \$dinner*
8. *// This sets \$dinner[3]*
9. *\$dinner[] = 'Flank Skin with Spiced Flavor';*

Looping through Array (foreach loop)

```
1. $meal = array(  
2.     'breakfast' => 'Walnut Bun',  
3.     'lunch' => 'Cashew Nuts and White Mushrooms',  
4.     'snack' => 'Dried Mulberries',  
5.     'dinner' => 'Eggplant with Chili Sauce'  
6. );  
7. foreach($meal as $key=>$value){  
8.     echo "<br> $meal => $value ";  
9. }
```

Modifying an array with foreach()

```
1. $meals = array( 'Fried Rice' => 1,
2.               'Pork Curry'=> 4.95,
3.               'Chicken Curry'=> 3.00,
4.               'Mutton Curry'=> 6.50);
5. foreach ($meals as $dish => $price) {
6. // $price = $price * 2 does NOT work
7. $meals[$dish] = $meals[$dish] * 2;
8. }
9. // Iterate over the array again and print the changed values
10. foreach ($meals as $dish => $price) {
11.     echo "<br> $dish => $price ";
12. }
```

Search Array elements using functions

If you're looking for a specific element in an array, you don't need to iterate through the entire array to find it. There are more efficient ways to locate a particular element. To check for an element with a certain key, use ***array_key_exists()***

The ***in_array()*** function returns true if it finds an element with the given value. It is case-sensitive when it compares strings. The ***array_search()*** function is similar to ***in_array()***, but if it finds an element, it returns the element key instead of true.

```
1. $meals = array(  
2.     'Walnut Bun' => 1,  
3.     'Cashew Nuts and White Mushrooms' => 4.95,  
4.     'Dried Mulberries' => 3.00,  
5.     'Eggplant with Chili Sauce' => 6.50,  
6.     'Shrimp Puffs' => 0  
7. ); // Shrimp Puffs are free!  
8.  
9. if (array_key_exists('Shrimp Puffs',$meals)) {  
10.     echo "Yes, we have Shrimp Puffs";  
11. }  
12. if (array_key_exists('Steak Sandwich',$meals)) {  
13.     echo "We have a Steak Sandwich";  
14. }
```

```
1. $books = array(  
2.     "The Eater's Guide to Chinese Characters",  
3.     'How to Cook and Eat in Chinese'  
4. );  
5. if (array_key_exists(1, $books)) {  
6.     echo "Element 1 is How to Cook and Eat in Chinese";  
7. }  
8. if (in_array(3, $meals)) {  
9.     echo 'There is a $3 item.';  
10. }  
11. if (in_array('How to Cook and Eat in Chinese', $books)) {  
12.     echo "We have How to Cook and Eat in Chinese";  
13. }
```

```
1. $dish = array_search(6.50, $meals);
2. if ($dish) {
3.     echo "$dish costs \$6.50";
4. }
5. Modifying Array
6. $dishes['Beef Chow Foon'] = 12;
7. $dishes['Beef Chow Foon']++;
8. $dishes['Roast Duck'] = 3;
9. $dishes['total'] = $dishes['Beef Chow Foon'] + $dishes['Roast Duck'];
10. if ($dishes['total'] > 15) { echo "You ate a lot: ";
11. }
12. echo 'You ate ' . $dishes['Beef Chow Foon'] . ' dishes of Beef Chow Foon.';
```


Array Unset , Implode , Explode

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

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

When you want to print all of the values in an array at once, the quickest way is to use the implode() function.

Making a string from an array with implode()

2. `$dimsum = array('Chicken Bun','Stuffed Duck Web','Turnip Cake');`
3. `$menu = implode(' ', $dimsum);`
4. `echo $menu;`

Explode

1. The counterpart to implode() is called explode(). It breaks a string apart into an array. The delimiter argument to explode() is the string it should look for to separate array elements.
2. Turning a string into an array with explode()
3. `$fish = 'Bass, Carp, Pike, Flounder';`
4. `$fish_list = explode(', ', $fish);`
5. `echo "The second fish is $fish_list[1]";`

Sorting Array

`sort()` / `rsort()` => The `sort()` function sorts an array by its element values. It should only be used on numeric arrays, because it resets the keys of the array when it sorts.

`ksort()` / `krsort()` => While `sort()` and `asort()` sort arrays by element value, you can also sort arrays by key with `ksort()`. This keeps key/value pairs together, but orders them by key.

`asort()` / `arsort()` => To sort an associative array by element value, use `asort()`. This keeps keys together with their values.

```
1. $dinner = array(  
2.     'Sweet Corn and Asparagus',  
3.     'Lemon Chicken',  
4.     'Braised Bamboo Fungus'  
5. );  
6. $meal = array(  
7.     'breakfast' => 'Walnut Bun',  
8.     'lunch' => 'Cashew Nuts and White Mushrooms',  
9.     'snack' => 'Dried Mulberries',  
10.    'dinner' => 'Eggplant with Chili Sauce'  
11. );  
12. sort($dinner) ; asort($meal) ; ksort($meal);
```

Multidimensional Array

The value of an array element can be another array.

Use the `array()` construct or the `[]` short array syntax to create arrays that have more arrays as element values.

```
1. $meals = array(  
2.   'breakfast' => ['Walnut Bun','Coffee'],  
3.   'lunch' => ['Cashew Nuts', 'White Mushrooms'],  
4.   'snack'    => ['Dried Mulberries','Salted Sesame Crab']  
5. );
```

```
1. $lunches = [  
2.     ['Chicken','Eggplant','Rice'],  
3.     ['Beef','Scallions','Noodles'],  
4.     ['Eggplant','Tofu']  
5. ];  
6. $flavors = array(  
7.     'Japanese' => array('hot' => 'wasabi', 'salty' => 'soy sauce'),  
8.     'Chinese' => array('hot' => 'mustard', 'pepper-salty' => 'prickly ash')  
9. );  
10. Modifying array elements  
11. $lunches[1][0]="Pork"; $flavours['japanese']['hot']='chilli';
```

Accessing multidimensional array elements

1. `echo $meals['lunch'][1];`
2. `echo $meals['snack'][0];`
3. `echo $lunches[0][0];`
4. `echo $lunches[2][1];`
5. `echo $flavors['Japanese']['salty']; // soy sauce`
6. `echo $flavors['Chinese']['hot']; // mustard`
7. `foreach ($flavors as $culture => $culture_flavors) {`
8. `foreach ($culture_flavors as $flavor => $example) {`
9. `echo "A $culture $flavor flavor is $example.\n";`
10. `}`
11. `}`

Functions and Files

A function is a named set of statements that you can execute just by invoking the function name instead of retyping the statements. This saves time and prevents errors. Plus, functions make it easier to use code that other people have written.

When you call a function, you can hand it some values with which to operate. These values are called arguments.

The function gives you back a value. This value is called the return value. You can use the return value of a function like any other value or variable.

1. Defining functions before or after calling them
2. *function page_header() {*
3. *echo '<html><head><title>Welcome to my site</title></head>';*
4. *echo '<body bgcolor="#ffffff">';*
5. *}*
6. *page_header();*
7. *echo "Welcome, \$user";*
8. *page_footer();*
9. *function page_footer() {*
10. *echo '<hr>Thanks for visiting.';*
11. *echo '</body></html>';*
12. *}*

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.

Declaring a function with an argument

```
1. function page_header2($color) {  
2.   echo '<html><head><title>Welcome to my site</title></head>';  
3.   echo '<body bgcolor="#' . $color . "'>';  
4. }  
5. page_header2("AABBCC");
```

Specifying a default value

```
1. function page_header3($color = 'cc3399') {  
2. echo '<html><head><title>Welcome to my site</title></head>';  
3. echo '<body bgcolor="#' . $color . "'>';  
4. }  
5. page_header3();
```

Defining a two-argument function

```
1. function page_header4($color, $title) {  
2. echo '<html><head><title>Welcome to ' . $title . '</title></head>';  
3. echo '<body bgcolor="#' . $color . "'>';  
4. }  
5. page_header4("9955AA", "Hello PHP");
```

Multiple optional arguments

```
1. function page_header5($color, $title, $header = 'Welcome') {  
2.   echo '<html><head><title>Welcome to ' . $title . '</title></head>';  
3.   echo '<body bgcolor="#' . $color . "'>';  
4.   echo "<h1>$header</h1>";  
5. }
```

// 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

```
1. function page_header6($color, $title = 'the page', $header = 'Welcome') {  
2.   echo '<html><head><title>Welcome to ' . $title . '</title></head>';  
3.   echo '<body bgcolor="#' . $color . "'>';  
4.   echo "<h1>$header</h1>";  
5. }
```

// 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

Functions can also compute a value, called the return value, which can be used later in a program. To capture the return value of a function, assign the function call to a variable.

Capturing a return value

```
$number_to_display = number_format(321442019);
```

```
echo "The population of the US is about: $number_to_display";
```

1. *Returning a value from a function*
2. *function restaurant_check(\$meal, \$tax, \$tip) {*
3. *\$tax_amount = \$meal * (\$tax / 100);*
4. *\$tip_amount = \$meal * (\$tip / 100);*
5. *\$total_amount = \$meal + \$tax_amount + \$tip_amount;*
6. *return \$total_amount;*
7. *}*
8. *\$total = restaurant_check(15.22, 8.25, 15);*
9. *echo 'I only have \$20 in cash, so...';*
10. *if (\$total > 20) {*
11. *echo "I must pay with my credit card.";*
12. *}else{*
13. *echo "I can pay with cash.";*
14. *}*

1. *Returning an array from a function*
2. *function restaurant_check2(\$meal, \$tax, \$tip) {*
3. *\$tax_amount = \$meal * (\$tax / 100);*
4. *\$tip_amount = \$meal * (\$tip / 100);*
5. *\$total_notip = \$meal + \$tax_amount;*
6. *\$total_tip = \$meal + \$tax_amount + \$tip_amount;*
7. *return array(\$total_notip, \$total_tip);*
8. *}*
9. *\$totals = restaurant_check2(15.22, 8.25, 15);*
10. *if (\$totals[0] < 20) {*
11. *echo 'The total without tip is less than \$20.';*
12. *}*
13. *if (\$totals[1] < 20) {*
14. *echo 'The total with tip is less than \$20.';*
15. *}*

Multiple return statements in a function

```
1.  function payment_method($cash_on_hand, $amount) {  
2.    if ($amount > $cash_on_hand) {  
3.      return 'credit card';  
4.    }else{  
5.      return 'cash';  
6.    }  
7.  }  
8.  $total = restaurant_check(15.22, 8.25, 15);  
9.  $method = payment_method(20, $total);  
10. echo 'I will pay with ' . $method;
```

Functions that return true or false

```
1.  function can_pay_cash($cash_on_hand, $amount) {  
2.    if ($amount > $cash_on_hand) {  
3.      return false;  
4.    }else{  
5.      return true;  
6.    }  
7.  }  
8.  $total = restaurant_check(15.22,8.25,15);  
9.  if (can_pay_cash(20, $total)) {  
10.    echo "I can pay in cash.";   
11.  }else{  
12.    echo "Time for the credit card.";   
13.  }
```

1. *Assignment and function call inside a test expression*
2. *function complete_bill(\$meal, \$tax, \$tip, \$cash_on_hand) {*
3. *\$tax_amount = \$meal * (\$tax / 100);*
4. *\$tip_amount = \$meal * (\$tip / 100);*
5. *\$total_amount = \$meal + \$tax_amount + \$tip_amount;*
6. *if (\$total_amount > \$cash_on_hand) {*
7. *return false;*
8. *}else{*
9. *return \$total_amount;*
10. *}}*
11. *if (\$total = complete_bill(15.22, 8.25, 15, 20)) {*
12. *echo "I'm happy to pay \$total.";*
13. *}else{*
14. *echo "I don't have enough money. Shall I wash some dishes?";*
15. *}*

Understanding Variable Scope

Variables defined outside of a function are called global variables. They exist in one scope. Variables defined inside of a function are called local variables. Each function has its own scope.

Variable scope

```
1. $dinner = 'Curry Cuttlefish';  
2. function vegetarian_dinner() {  
3.   echo "Dinner is $dinner, or ";  
4.   $dinner = 'Sauteed Pea Shoots';  
5.   echo $dinner;  
6.   }
```

1. *function kosher_dinner() {*
2. *echo "Dinner is \$dinner, or ";*
3. *\$dinner = 'Kung Pao Chicken';*
4. *echo \$dinner;*
5. *}*
6. *echo "Vegetarian ";*
7. *vegetarian_dinner();*
8. *echo "Kosher ";*
9. *kosher_dinner();*
10. *echo "Regular dinner is \$dinner";*

There are two ways to access a global variable from inside a function. The most straightforward is to look for them in a special array called `$GLOBALS`.

1. *The `$GLOBALS` array*
2. *`$dinner = 'Curry Cuttlefish';`*
3. *`function macrobiotic_dinner() {`*
4. *`$dinner = "Some Vegetables";`*
5. *`echo "Dinner is $dinner";`*
6. *`echo $GLOBALS['dinner'];`*
7. *`}`*
8. *`macrobiotic_dinner();`*
9. *`echo "Regular dinner is: $dinner";`*

The \$GLOBALS array can also modify global variables.

1. *Modifying a variable with \$GLOBALS*
2. *\$dinner = 'Curry Cuttlefish';*
3. *function hungry_dinner() {*
4. *\$GLOBALS['dinner'] .= ' and Deep-Fried Taro';*
5. *}*
6. *echo "Regular dinner is \$dinner";*
7. *hungry_dinner();*
8. *echo "Hungry dinner is \$dinner";*

The second way to access a global variable inside a function is to use the global keyword.

1. *The global keyword*
2. *\$dinner = 'Curry Cuttlefish';*
3. *function vegetarian_dinner() {*
4. *global \$dinner;*
5. *echo "Dinner was \$dinner, but now it's ";*
6. *\$dinner = 'Sauteed Pea Shoots';*
7. *echo \$dinner;*
8. *}*
9. *echo "Regular Dinner is \$dinner.";*
10. *vegetarian_dinner();*
11. *echo "Regular dinner is \$dinner";*

Enforcing Rules on Arguments and Return Values

Type declarations are a way to express constraints on argument values. These tell the PHP engine what kind of value is allowed for an argument so it can warn you when the wrong kind is provided.

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

1. *Declaring an argument type*
2. *function countdown(int \$stop) {*
3. *while (\$stop > 0) {*
4. *echo "\$stop..";*
5. *\$stop--;*
6. *}*
7. *echo "boom!
";*
8. *}*
9. *\$counter = 5;*
10. *countdown(\$counter); // countdown("grunt");*
11. *echo "Now, counter is \$counter";*

PHP 7 also supports type declarations for the kind of value a function returns. To enforce checking of the return type of a function, put a : after the) that closes the argument list, and then the return type declaration.

Declaring a return type

```
1. function restaurant_check($meal, $tax, $tip): float {  
2.     $tax_amount = $meal * ($tax / 100);  
3.     $tip_amount = $meal * ($tip / 100);  
4.     $total_amount = $meal + $tax_amount + $tip_amount;  
5.     return $total_amount;  
6. }
```

Running Code in Another File

```
1. <?php
2.     function restaurant_check($meal, $tax, $tip) {
3.         $tax_amount = $meal * ($tax / 100);
4.         $tip_amount = $meal * ($tip / 100);
5.         $total_amount = $meal + $tax_amount + $tip_amount;
6.         return $total_amount;
7.     }
8.     function payment_method($cash_on_hand, $amount) {
9.         if ($amount > $cash_on_hand) {
10.            return 'credit card';
11.        }else{
12.            return 'cash';
13.        } }
14. ?>
```

1. *Referencing a separate file*
2. *require 'restaurant-functions.php';*
3. */* \$25 check, plus 8.875% tax, plus 20% tip */*
4. *\$total_bill = restaurant_check(25, 8.875, 20); /* I've got \$30 */*
5. *\$cash = 30;*
6. *echo "I need to pay with " . payment_method(\$cash, \$total_bill);*

Files Read and Write PHP

There are many other ways of reading and writing data to files in PHP. However, `file_get_contents()` and `file_put_contents()` will address almost all your basic needs without adding unnecessary complication.

The only time you might face a problem with `file_get_contents()` is when the file you are reading is very large—like 2GB or more in size. This is because `file_get_contents()` loads the whole file into memory at once, and there is a good chance of running out of memory with such large files. In that case, you will have to rely on functions like `fgets()` and `fread()` to read a small part of the file at once.

1. Read File using fread() function
2. <?php
3. \$myfile = fopen("studentlist.txt", "r") or die("Unable to open file!");
4. echo fread(\$myfile,filesize("studentlist.txt"));
5. fclose(\$myfile);
6. ?>

Read each line of File using feof() function and fgets() function

1. <?php
2. \$myfile = fopen("studentlist.txt", "r") or die("Unable to open file!");
3. while(!feof(\$myfile)) {
4. echo fgets(\$myfile) . "
";
5. }
6. fclose(\$myfile);
7. ?>

Write File using fwrite() function

1. <?php
2. \$myfile = fopen("studentlist.txt", "w") or die("Unable to open file!");
3. \$txt = "Mg Mg \n";
4. fwrite(\$myfile, \$txt);
5. \$txt = "Ko Ko \n";
6. fwrite(\$myfile, \$txt);
7. fclose(\$myfile);
8. ?>

We use “w” to write new file and “a” to append existing file.

```
$myfile = fopen("studentlist.txt", "a") or die("Unable to open file!");
```


Use file_get_contents() and file_put_contents()

1. <?php
2. \$file = 'people.txt';
3. // Open the file to get existing content
4. \$current = file_get_contents(\$file);
5. // Append a new person to the file
6. \$current .= "John Smith\n";
7. // Write the contents back to the file
8. file_put_contents(\$file, \$current);
9. ?>
10. Add student information using JSON file.

PHP OOP

- *Class* -A template or recipe that describes the variables and functions for a kind of object.
- *Method* -A function defined in a class.
- *Property* - A variable defined in a class.
- *Instance* -An individual usage of a class.
- *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. Static methods don't depend on the property values of a particular instance.

- **Parent class** – A class that is inherited from by another class. This is also called a base class or super class.
- **Child Class** – A class that inherits from another class. This is also called a subclass or derived class.
- **Polymorphism** – Same function can be used for different purposes. For example function name will remain same but it make take different number of arguments and can do different task.
- **Overloading** – a type of polymorphism in which some or all of operators have different implementations depending on the types of their arguments. Similarly functions can also be overloaded with different implementation.
- **Data Abstraction** – Any representation of data in which the implementation details are hidden .
- **Encapsulation** – refers to a concept where we encapsulate all the data and member functions together to form an object.

Sample Class Definition

```
1.  <?php
2.  class Student{
3.  public $name;
4.  public $hobbies= array();
5.  public function hobbyshow($hobby){
6.  return in_array($hobby, $this->hobbies);
7.  }
8.  }
9.  $student1=new Student;
10. $student1->name="Mg Mg";
11. $student1->hobbies=array('Reading','Football');
12. $studenthobbies=$student1->hobbyshow("Reading");
13. var_dump($studenthobbies);
14. ?>
```

Static Method

- Classes can also contain static methods.
- These methods cannot use the `$this` variable since they do not get run in the context of a specific object instance, but on the class itself.
- Static methods are useful for behavior that is relevant to what the class is for, but not to any one object.
- Example

```
public static function getHobbies() {  
return array('Music','Guitar','Piano');  
}
```

- *Calling a static method*
`$totalhobbies= Student::getHobbies();`

Constructors

- A class can have a special method, called a *constructor*, which is run when the object is created.
- Constructors typically handle setup and housekeeping tasks that make the object ready to use.
- By passing those values to the constructor, we avoid having to set the properties after the object is created.
- In PHP, the constructor method of a class is always called `__construct()`.

Constructor

```
public function __construct($name, $hobbies) {
```

```
$this->name = $name;
```

```
$this->hobbies= $hobbies;
```

```
}
```

```
$student2= new Student('Aye Aye', array('Beauty','Shopping'));
```

Exceptions

- Constructors are great for verifying that supplied arguments are the right type or otherwise appropriate.
- But they need a way to complain if there is a problem.
- This is where an *exception* comes in.
- An exception is a special object that can be used to indicate that something exceptional has happened.
- Creating an exception interrupts the PHP engine and sets it on a different code path.

```
if (!is_array($hobbies)) {  
throw new Exception('$hobbies must be an array'); }
```


Try Catch Exception

- To handle an exception yourself, do two things:
- 1. Put the code that might throw an exception inside a try block.
- 2. Put a catch block after the potentially exception-throwing code in order to handle the problem.

```
try {  
    your code  
} catch (Exception $e) {  
    echo $e->getMessage();  
}
```

Extending the Object

- One of the aspects of objects that make them so helpful for organizing your code is the notion of *subclassing*, which lets you reuse a class while adding some custom functionality.
- A subclass (sometimes called a *child class*) starts with all the methods and properties of an existing class (the *parent class*), but then can change them or add its own.

- We use keyword **extends**

```
class Child extends Parent {  
<definition body>  
}
```

- For Parent Constructor use `parent::__construct(pro1,pro2);`

```
1. class BootcampStudent extends Student{
2.   public $subjects=array();
3.   public function __construct($name,$hobbies,$subjects){
4.     parent::__construct($name,$hobbies);
5.     $this->subjects=$subjects;
6.   }
7.   public function showSubjects(){
8.     return $this->subjects;
9.   }
10. }
```

Function Overriding

- Function definitions in child classes override definitions with the same name in parent classes. In a child class, we can modify the definition of a function inherited from parent class.
- In the following example getPrice and getTitle functions are overridden to return some values.

```
public function hobbyshow($hobby){  
    return array_search($hobby, $this->hobbies);  
}
```

Property and Method Visibility

- We prevent this problem by changing the *visibility* of the properties.
- Instead of public, we can label them as private or protected.
- These other visibility settings don't change what code inside the class can do—it can always read or write its own properties.
- The private visibility prevents any code outside the class from accessing the property.
- By designating a member private, you limit its accessibility to the class in which it is declared. The private member cannot be referred to from classes that inherit the class in which it is declared and cannot be accessed from outside the class.
- A class member can be made private by using **private** keyword in front of the member.

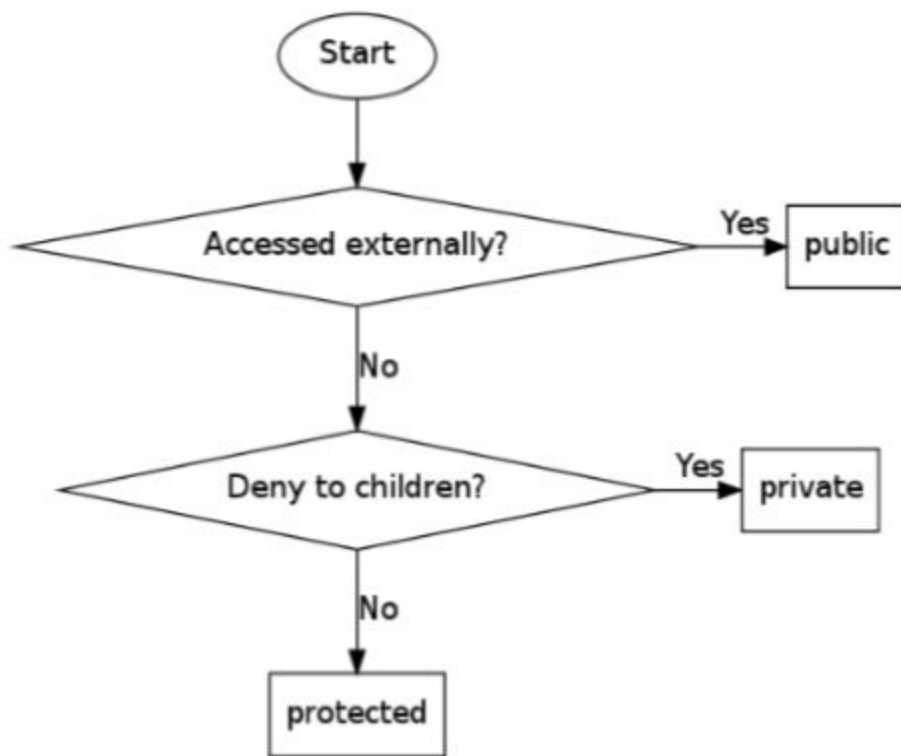
Public Members

- Unless you specify otherwise, properties and methods of a class are public. That is to say, they may be accessed in three possible situations –
- From outside the class in which it is declared
- From within the class in which it is declared
- From within another class that implements the class in which it is declared
- Till now we have seen all members as public members. If you wish to limit the accessibility of the members of a class then you define class members as **private** or **protected**.

Protected members

- A protected property or method is accessible in the class in which it is declared, as well as in classes that extend that class.
- Protected members are not available outside of those two kinds of classes.
- A class member can be made protected by using **protected** keyword in front of the member.
- **Private** property can access from child class

Choosing The Visibility



Namespaces

- Beginning with version 5.4, the PHP engine lets you organize your code into *name- spaces*.
- Namespaces provide a way to group related code and ensure that names of classes that you've written don't collide with identically named classes written by someone else.
- Think of a namespace as a container that can hold class definitions or other namespaces. It's a syntactic convenience, rather than providing new functionality. When you see the namespace keyword or some backslashes in what appears to be a class name, you've encountered a PHP namespace.

Working with Database MySQL

<http://myanmaritc.com/download/mysql.pdf>

MySQL is open-source software released under the GNU General Public License so it should come as no surprise there are also alternative forks available. Two popular forks are MariaDB, a community-maintained “enhanced, drop-in replacement” for MySQL, and Percona Server, a drop-in maintained by the consulting firm Percona LLC. The differences between MySQL, MariaDB, and Percona are mostly imperceptible to the casual user.

Storing Data

Data stored in a relational database is organized into tables. A database table organizes data in a grid-like fashion, where each entry forms a row and each column identifies a specific value in the entry.

Country	Gold	Silver	Bronze	Total
Russia	13	11	9	33
United States	9	7	12	28
Norway	11	5	10	26
Canada	10	10	5	25
Netherlands	8	7	9	24

Creating Table

```
1. CREATE TABLE employee (  
2.     employee_id INTEGER UNSIGNED NOT NULL AUTO_INCREMENT,  
3.     last_name VARCHAR(30) NOT NULL,  
4.     first_name VARCHAR(30) NOT NULL,  
5.     email VARCHAR(100) NOT NULL,  
6.     hire_date DATE NOT NULL,  
7.     notes MEDIUMTEXT,  
8.     PRIMARY KEY (employee_id),  
9.     INDEX (last_name),  
10.    UNIQUE (email)  
11. )  
12. ENGINE=InnoDB;
```

```
1. CREATE TABLE address (  
2.     employee_id INTEGER UNSIGNED NOT NULL,  
3.     address VARCHAR(50) NOT NULL,  
4.     city VARCHAR(30) NOT NULL,  
5.     state CHAR(2) NOT NULL,  
6.     postcode CHAR(5) NOT NULL,  
7.     FOREIGN KEY (employee_id)  
8.         REFERENCES employee (employee_id)  
9. )  
10. ENGINE=InnoDB;  
  
11. DESCRIBE employee;  
12. SHOW CREATE TABLE employee;
```

Integer Data Type

Data Type	Storage Used (Bytes)	Min. Signed	Max. Signed	Min. Unsigned	Max. Unsigned
TINYINT	1	-128	127	0	255
SMALLINT	2	-32,768	32,767	0	65,535
MEDIUMINT	3	-8,388,608	8,388,607	0	6,777,215
INTEGER	4	-2,147,483,648	2,147,483,647	0	4,294,967,295
BIGINT	8	-9,223,372,036,854,775,808	9,223,372,036,854,775,807	0	18,446,744,073,709,551,615

String Data Type

Data Type	Storage Used (Bytes)	Maximum
CHAR(<i>m</i>)	$m \times$ maximum-size character in the character set	255 chars
VARCHAR(<i>m</i>)	up to 2 bytes + $m \times$ maximum-size character in the character set	65,535 chars
TEXT	size of string in bytes + 2	65,535 chars
TINYTEXT	size of string in bytes + 1	255 chars
MEDIUMTEXT	size of string in bytes + 3	16,777,215 chars
LONGTEXT	size of string in bytes + 4	4,294,967,295 chars
BINARY(<i>m</i>)	m	255 bytes
VARBINARY(<i>m</i>)	up to 2 bytes + m	65,535 bytes

String Data Type 2

Data Type	Storage Used (Bytes)	Maximum
BLOB	size of string in bytes + 2	65,535 bytes
TINYBLOB	size of string in bytes + 1	255 bytes
MEDIUMBLOB	size of string in bytes + 3	16,777,215 bytes
LOB	size of string in bytes + 4	4,294,967,295 bytes
ENUM	up to 2 bytes	65,535 values
SET	up to 8 bytes	64 members

Temporary Data Type

Data Type	Storage Used (Bytes)	Minimum	Maximum
DATETIME	8	1000-01-01 00:00:00	9999-12-31 23:59:59
TIMESTAMP	4	1970-01-01 00:00:01 UTC	2038-01-19 03:14:07 UTC
DATE	3	1000-01-01	9999-12-31
TIME	3	-838:59:59	838:59:59
YEAR	1	1901	2155

Storage Engine

Feature	InnoDB	MyISAM	Archive	Black-hole	CSV	Federated	Merge	Memory	NDB
Storage limit	64TB	256TB	No Limit	N/A	File system	N/A	No Limit	RAM	384EB
ACID	Yes	No	No	No	No	?	No	No	Yes
Foreign keys enforced	Yes	No	No	No	No	?	No	No	Yes
Indexes	Yes	Yes	No	N/A	No	No	Yes	Yes	Yes
Concurrent inserts	Yes	Yes	Yes	Yes	Yes	?	Yes	No	Yes
UNIQUE enforced	Yes	Yes	No	No	No	?	No	Yes	Yes

Insert Data into Table

1. INSERT INTO employee
2. (first_name, last_name, email, hire_date)
3. VALUES
4. ('Yan Myoe', 'Aung', 'yanmyoe@gmail.com', '2019-7-15');

5. INSERT INTO employee
6. VALUES ('Mg Yan, 'Aung', mgyan@gmail.com', '2019-8-5');

7. SELECT * FROM employee;

Using Transaction

- A transaction is a set of statements that are treated as a group.
- The `START TRANSACTION` statement begins a transaction and MySQL considers any statements that follow to be part of that transaction.
- The transaction ends with either a `COMMIT` or `ROLLBACK` statement.
- `COMMIT` finalizes the transaction, making the changes official, and `ROLLBACK` rolls back the transaction, discarding any changes.

Rollback Example

1. START TRANSACTION;
2. INSERT INTO employee
3. (employee_id, last_name, first_name, email, hire_date)
4. VALUES
5. (42, 'Khin', 'Aye Chan', 'ayechan@gmail.com', '2019-04-02');
6. SELECT * FROM employee;
7. ROLLBACK;
8. SELECT * FROM employee;

Commit Example

1. START TRANSACTION;
2. INSERT INTO employee
3. (employee_id, last_name, first_name, email, hire_date)
4. VALUES
5. (42, 'Chan Khin' , 'Aye', 'ayechan@gmail.com', '2019-04-02');
- 6.
7. INSERT INTO address
8. (employee_id, address, city, state, postcode)
9. VALUES
10. (42, '227 Insein Road', 'Yangon', 'Ygn', '97052');
11. COMMIT;

Ordering Results

When the order of rows in the result is important, we can use an ORDER BY clause in the SELECT statement and MySQL will sort the rows before returning them to us. Reissue the SELECT statement, but this time add ORDER BY to sort the results.

```
SELECT last_name, first_name FROM actor ORDER BY last_name;
```

```
SELECT last_name, first_name FROM actor ORDER BY last_name DESC;
```

Managing the Number of Returned Rows

```
SELECT last_name, first_name FROM actor ORDER BY last_name,  
first_name DESC LIMIT 5;
```

Where Clause

The WHERE clause gives us the ability to specify conditions that a row must match for MySQL to include it in the results. For example, we can retrieve only the rows with a column value greater or less than some arbitrary value we provide. Let's look at some data from the film table:

```
SELECT title, length, rating FROM film WHERE length < 60 ORDER BY title;
```

```
SELECT title, length, rating FROM film WHERE length >= 60 AND length <= 120  
ORDER BY title;
```

```
SELECT title, length FROM film WHERE length = 60 OR length = 120 OR length  
= 180 ORDER BY title;
```

```
SELECT title, length, rating FROM film WHERE title LIKE '%GAME%'  
  
ORDER BY title;
```


Combine Result using “UNION”

It's also possible to combine the results from multiple SELECT statements using UNION. Because each additional set is appended to the result, each statement must return the same number of columns.

```
SELECT title FROM film WHERE title LIKE 'A%'
```

```
UNION
```

```
SELECT title FROM film WHERE title LIKE 'Z%';
```

Aggregate Functions and Grouping

Aside from filtering and sorting, MySQL also has the ability to group and condense rows and summarize their data. Aggregate functions like COUNT(), SUM(), and MAX() perform their calculation using all of the column's values from the matching rows to come up with a single summary value.

```
SELECT MAX(amount) FROM payment;
```

The COUNT() function simply counts the number of rows it passes over. For this reason, it's quite common to use COUNT() to determine the size of a table.

```
SELECT COUNT(payment_id) FROM payment;
```

Suppose we want to find the top 10 customers who've spent the most money renting movies. We could issue several SELECT statements to retrieve the necessary data and perform the calculations manually, but it's faster and easier to ask MySQL to do the work for us.

The GROUP BY clause batches rows with the same customer_id values into groups. Then, the SUM() function adds up the values and returns the sum for each group. By gathering the rows into a different batch for each customer and summing their payment amounts, we can easily work out how much each customer has spent. Sorting the results and returning the 10 highest values shows us who our most profitable customers are.

```
SELECT customer_id, SUM(amount) AS amt FROM payment GROUP BY  
customer_id ORDER BY amt DESC LIMIT 10;
```

Updating and Deleting Data

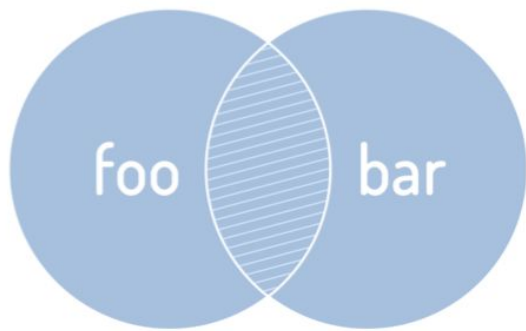
```
UPDATE customer SET last_name = 'DAY-WEBB', last_update =  
last_update WHERE customer_id = 245;
```

```
DELETE FROM customer WHERE customer_id = 245;
```

Working with Multiple Tables

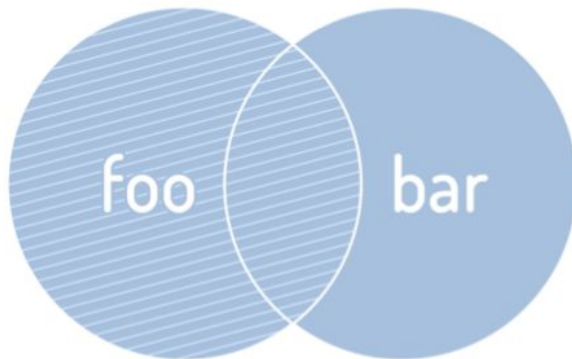
The three join types available to us with MySQL are INNER JOIN, LEFT OUTER JOIN, and RIGHT OUTER JOIN. The default is INNER JOIN, so when JOIN appears by itself (as it did in the earlier statements) MySQL interprets it as an INNER JOIN. It's important to know how each type behaves because if the desired rows aren't captured by the join in the first place then they can't possibly appear in the final result.

The two OUTER JOINS return the same rows as INNER JOIN, but also include the unmatched rows from one table or the other (the outer parts of the intersecting circles). LEFT OUTER JOIN includes the unmatched rows from the table that appears to the left of the JOIN keyword. In the result, the columns defined by the other table contain NULL.



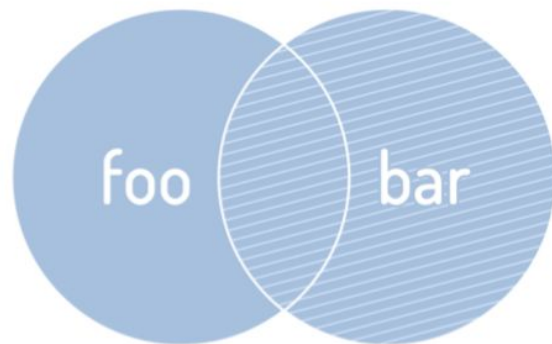
foo JOIN bar
foo INNER JOIN bar

Figure 4.1. INNER JOIN behavior illustrated as a Venn diagram.



foo LEFT OUTER JOIN bar

Figure 4.2. LEFT OUTER JOIN behavior illustrated as a Venn diagram.



foo RIGHT OUTER JOIN bar

Figure 4.3. RIGHT OUTER JOIN behavior illustrated as a Venn diagram.

Cookie and Sessions

A cookie identifies a particular web client to the web server and PHP engine. Each time a web client makes a request, it sends the cookie along with the request. The engine reads the cookie and figures out that a particular request is coming from the same web client that made previous requests, which were accompanied by the same cookie.