hackademy.ca

Growing Tech Literacy Through Community

Introduction to PHP and MySQL

Developed by Andrew Taylor
10/24/2013

Contents

License ... 3
A Note For Instructors ... 3
Intro ... 3
What is PHP? ... 4
Breakdown of Server-side and Scripting ... 4
PHP Files and "Hello World!" ... 4
Delimiters ... 5
Hello World! And Constructs ... 5
Comments ... 6
Functions, Parameters, and Return Values ... 6
PHP.net ... 6
Variables and Operators ... 7
Operators ... 7
Variables ... 7
Variable Challenge ... 8
Data Types ... 9
Integers ... 9
Strings ... 9
Booleans ... 9
Floating Point Numbers ... 9
Type Juggling ... 9
Conditional Statements ... 10
If ... 10
Elseif ... 11
Else ... 11
Conditional Challenge ... 12
User Defined Functions ... 12
Function Challenge ... 13

Variable Scope 13
Superglobal Variables 13
Arrays 14
Array Challenge 14
Loops 14
While 14
Do-while 15
For 15
Foreach 16
Final Challenge (part 1) 17
MySQL 17
Databases, Tables, Columns, and Rows 17
Query Types 18
Insert 18
Select 18
Update 18
Delete 18
MySQLi Extension 18
Open Database Connection 19
Execute Queries 19
Select 19
Insert, Update, and Delete 20
Final Challenge (part 2) 20

License

A Note For Instructors

This lesson plan is designed to provide an outline for the flow of the lesson and does not contain all the information to be taught. If it contained all the information we'd be publishing a novel, not teaching a condensed PHP and SQL course. Please stress the importance of questions, as student questions will assist with covering topics and understanding. I have tried to include the more important points, but it is up to you to make sure you're talking about each topic in more detail than is written about in here.

In addition, this course is designed to have the instructor typing code live on screen in-front of the students, with the students following along. Make sure you're typing the code examples so that they can get a visual of these concepts and try them out. Don't be afraid to take small detours to explain subjects and allow the students to participate.

Note: Please take notes! The more this course is taught, the better it will ultimately become. Please take notes on the back of each page to be passed on to the course developer. Notes should include any points or code that should be included in the lesson plan, points or code that are irrelevant and should be removed, time frames for each topic (i.e. how long did it take to cover the whole topic), and any other information that you think is relevant.

Intro

[Ensure all students have a authoring program open and can successfully connect to the FTP server and their web space]

Display attached slideshow showing course title.

Instructor and assistants introduce themselves. Provide a brief description about your job and your programming experience.

Allow each student to introduce themselves and why they want to take the course. i.e. what they hope to gain.

OUESTIONS! – Questions are very important to the learning process. A lot of material will be covered very quickly in this course, and it is important that you ask questions if you need clarification on a topic. Don't be afraid to put your hand up!

What is PHP?

PHP is a programming language that was originally designed in 1994 as a tool to help web developers manipulate submitted form data.

Wikipedia says that "PHP is a server-side scripting language designed for web development, but also used as a general-purpose programming language." By breaking that definition down, we can get a better understanding of how PHP is intended to be used.

Breakdown of Server-side and Scripting

"Server-side" – Is a term that comes from the client server model. A very basic description of the client server model is a structure of communication where the server contains data and the client requests and receives the data. Viewing a webpage is an example of this. Your web browser is the client and the web server contains the page you want to view (the data). When you browse to google.ca, your browser sends a request to Google's web server saying "Please show me this page". The web server parses the file and executes any code that needs to be executed, and then sends the output back to the client saying "Here is the page you asked for."

- So "server-side" means that the PHP code is executed on the server, and only the output is passed back to the client.

"Scripting" – This term is often used to describe programming languages that do not need to be compiled into binaries to be executed. PHP is a language that is parsed, and executed in real time by the interpreter or code engine. A simple way of thinking about it, is that (much like a TV show script) a PHP script is a list of instructions that tell the code engine what to do.

- So "scripting" means that the language does not need to be compiled.

There is a lot more to know about the history and capabilities of PHP, but this isn't a history lesson. If you would like to do more research on your own time, we will talk about some resources later in the lesson.

PHP Files and "Hello World!"

[Have students open a blank PHP file, and demonstrate live on screen]

Let's jump right in!

Because PHP files are just "scripts" of instructions, they are essentially just text files that can be created, edited, and viewed with any text editor. Special applications like Adobe Dreamweaver have features built in like syntax highlighting or autocompleting function names or parameters. These applications can also assist with debugging by highlighting syntax errors in the code.

Delimiters

PHP is uniquely suited for web design, because it uses delimiters for its code. Any text outside the delimiters is passed through the interpreter untouched, while code within the delimiters is executed.

```
<?php

?>
```

Hello World! And Constructs

The first instruction we are going to learn is how to output text. To do that we are going to use the PHP construct *echo*. A construct is an element or command that is of the most basic nature. i.e. the language couldn't exist without it.

```
<?php
echo 'Hello World!';
?>
```

The semi-colon at the end of the line is very important. The semi-colon is placed at the end of every instruction to let the code-engine know that the instruction is complete that that a new instruction will be next.

Save the file with the extension .php and upload it to your web space. When you name PHP files you should keep it simple and short. DO NOT use spaces or special characters other than dash and underscore in your file names. It is very poor programming practice to have long complicated names or special characters.

Now view the file in a web browser. Congratulations! You've all successfully programmed in PHP.

So what just happened? Well, when you browsed to the file the web server passed it to the PHP code engine. The code engine parsed the file and executed all the code in between the delimiters; In this case, it output "hello world". Then the web server passed the output back to your browser.

```
<?php
echo 'Hello World!';
?> echo 'Hello World!';
```

So what happens if we had put that same instruction outside the delimiters?

It would have output the exact text, instead of executing the echo instruction. [Try it]

Comments

To add comments to your PHP files you can use one of three methods. You can comment out an entire line by placing a # as the first character in the line, you can comment out a line by placing // anywhere in the line, or you can comment out multiple lines by placing /* at the beginning of the comment and */ at the end of the comment.

Functions, Parameters, and Return Values

A major part of any programming language is functions. You'll also hear them referred to as methods. A function is a collection of instructions that preform a specific action. Basically, functions perform a specific function.

Functions can be built-in, come as part of an extension (plugin; an add-on to PHP that extends its functionality), or can be user-defined. Having the ability to create your own functions makes the programming possibilities virtually endless.

Let's use a function that takes uppercase text and turns it into lowercase text. In PHP, this function is `strtolower()`.

```
<?php
echo strtolower('Hello World!');
?>
```

The function `strtolower()` accepts one parameter, which is the string that you want to convert to lowercase. A parameter is a piece of information that gets passed to a function, and is placed inside the parenthesis (with multiple parameters separated by commas). In this case, the parameter was "Hello World!" which was passed to the function, manipulated, and then returned. Generally speaking, all functions will return a value of some kind, even if a return value is not specifically declared. The return value of this function is passed directly to the echo construct and thus is printed.

PHP.net

It is inevitable that you will forget the specific order or set of required parameters for functions, and there are several resources that can assist you. Some software will have tool-tips that pop up when it recognizes the function and tell you what parameters are needed, but it is unwise to rely on those. When in doubt a Google search will most likely get the results, but the best resource you have is PHP.net

This website contains the documentation for PHP along with tons of user-contributed content to help you. Let's go there and look up `strtolower()`.

[Conduct a brief walkthrough of PHP.net and search for the function. Highlight the examples, related functions, and user contributed content.]

** Use caution when referencing user-contributed code. This code may be incorrect, insecure, or use poor coding practices. It is better to use this code as a guideline to writing your own version.

As we proceed through this course, more and more of the code is going to come from your heads rather than being given to you by me. PHP.net is going to be a great resource for you to reference and help yourselves learn.

As previously mentioned, a lot of the history of PHP is also on the website. If you ever want to know more about the language and how it came to be, feel free to read up on it.

Variables and Operators

Operators

Let's look at a bit of math; after all PHP is a programming language. Basic arithmetic is accomplished with the usual symbols + - * and / . These are known as operators. Let's try them.

```
<?php
echo 5 + 5;
?>
```

There are tons of other operators for numbers, strings, to quickly add or subtract a number without having to type as much code.

[See handout for table of operators]

Variables

Easy as pie right? But up until now, we have only been outputting information. What if we wanted to store it temporarily to use in another part of the script? That's where variables come in. In PHP variables are stored in memory while the script is executed and are deleted once the script terminates.

Let's jump right in. Pick a number between one and ten (inclusive) and write PHP code that outputs four lines of text, with each line including the number.

```
<?php
echo 'I am thinking of the number 10.<br>';
echo '10 is the number I am thinking of.<br>';
echo 'The number 10 shall be the number.<br>';
echo 'And the number of the thinking shall be 10.<br>';
?>
```

To make it easier to read when we output this, make sure to include an HTML line break so that the browser knows to put the next text on the next line. So if you have your four lines, each line has the number you were thinking of. If I now told you to think of a different number you would have to go through each line and replace the number with the new number. This is one of the things that variables come in handy with.

`$variablename = 'value';`

Variables are declared with a dollar sign and the variable name. Variable names follow the same scheme as other labels in PHP (alphanumeric and underscores). Think of variables like placeholders. Let's try.

```
<?php
$x = 10;
echo 'I am thinking of the number ' . $x . '.<br>';
echo $x . ' is the number I am thinking of.<br>';
echo 'The number ' . $x . ' shall be the number.<br>';
echo 'And the number of the thinking shall be ' . $x . '.<br>';
?>
```

When you view this page you'll see that it's the same as before with the variable instead of the actual number. Like I said before, variables are like placeholders so every time `$x` is used it gets substituted for the value of `$x`, which is 10.

Variables can be substituted for any value. For example, as a parameter for the `strtolower()` function.

Variable Challenge

Make a script that assigns a variable, and then modifies it before outputting it. The modification can be either mathematical or String based. For a challenge, include more than one variable and tie them together somehow.

Data Types

You've already heard me talk a little about integers and Strings, and it's time we talked about what those terms mean.

Integers

An integer is a whole number. -3, -2, -1, 1, 2, 3, 4, 5 … are examples of integers. An integer is typed literally and does not need quotes around it.

Strings

A String is text. A string is defined with quotations. String that are defined with double quotes will expand variables and special characters like newlines (\n).

Booleans

Booleans are the simplest data type. Booleans are true or false, and defined using those constants.

Floating Point Numbers

Also known as "floats", "doubles", or "real numbers". Floats are decimal or exponential numbers. We aren't going to be working with them today, but if you would like to know more about them you can read about them on the PHP.net website.

```
<?php
$foo = true; //Type Boolean bool(true)
$foo = 10; //Type Integer int(10)
$foo = "10"; //Type String String(2)
$foo = 3.123; //Type Float float(3.123)
?>
```

Type Juggling

PHP variables are scalar. This means that they do not need to have a data type explicitly defined, and that the data type of a variable will change depending on the context in which it is used. Let's play around with it a little.

```
<?php
$a = '0';
var_dump($a);
$a = $a + 2;
var_dump($a);
$a = 5 + "10 Little Piggies";
var_dump($a);
?>
```

[This is open time for students to experiment with type juggling. Demonstrate some example on the screen including combining Strings with integers]

Conditional Statements

If

A big part of logic in programming is executing certain code when a set of conditions are met. In PHP this is accomplished with an 'if' statement. An if statement is best described as "IF a specific condition proves true, execute this code".

```
<?php
if (5 == 5){
    echo 'Yes! 5 does equal 5.';
}
?>
```

In this example, the condition is that 5 must equal 5. And of course the answer is yes, or true. And when the condition evaluates as true, the code within the braces is executed. It is worthy to note that conditional statements always require a true condition to execute, even if the condition is that something must prove false. For example:

```
<?php
$foo = false;
if (!$foo){
    echo 'Yes! $foo was false.';
}
?>
```

In this example, the condition is that the variable `$foo` must be false. However, if `$foo` is false, the condition still proves true because it is true that `$foo` is false.

With if statements you can also have multiple conditions using the AND/OR operators.

```
<?php
$foo = false;
if (!$foo && 5 == 5 || 4 == 4){
    echo 'Yes! $foo was false AND either 5 equals 5 OR 4 equals 4.';
}
?>
```

It is very important to think about the logic of your conditions. If you have two conditions and use the AND operator, BOTH conditions must prove true. If you use the OR operator, AT LEAST one or both conditions must prove true. In this example, *$foo* must be false AND EITHER 5 must equal 5 OR 4 must equal 4.

Elseif

Sometimes you may have multiple groups of conditions with different code to execute for each group. You could write a second if statement, but if the second condition group doesn't matter as long as the first one is true, then an elseif is the way to go. An elseif follows an if, and is only evaluated if the preceding ifs and elseifs have not proved true.

```
<?php
$foo = true;
if (!$foo){
    echo '$foo was false!';
} elseif ($foo){
    echo '$foo was true!';
}
?>
```

An elseif is like an "otherwise if". If this condition proves true, execute this code, otherwise if this other condition proves true, execute this other code. You can have as many elseifs as you'd like, but remember that as soon as one of them proves true the rest of them are no longer tested. Logical operators can also be used in elseif statements.

Else

In a situation where you want to execute different code if none of the conditions prove true, PHP gives us the else statement. The else statement is the last statement in the if "stack" and acts like a catchall. If none of the conditions prove true, the code within the else statement will execute, but if any of the conditions do prove true, the else statement is skipped.

```
<?php
$foo = true;
if (!$foo){
    echo '$foo was false!';
} elseif ($foo){
    echo '$foo was true!';
} else {
    echo '$foo was neither true nor false!';
}
?>
```

While elseif acts like "otherwise if", else acts like just "otherwise". If this condition proves true, execute this code, otherwise if this condition proves true, execute this other code, otherwise execute that.

Conditional Challenge

Create a series of conditions using if, elseif, and else. For an extra challenge include variables. For a real challenge, try to validate data (i.e. confirm that a variable meets specific data requirements like being a number).

User Defined Functions

Let's go back to functions. I mentioned earlier that functions can be user-defined. To define a function we simply write 'function' followed by the function name. Function names follow the same rules as other labels in PHP. A valid function name starts with a letter or underscore, followed by any number of letters, numbers, or underscores. Following the function name we have two parentheses. Inside those parentheses is where we would define any parameters for our function. We remember from our earlier discussion on functions; parameters are pieces of information that get passed to functions. For our example we will use one parameter `$text`.

```
<?php
function showText($text){
    echo $text;
}
?>
```

When we save and run this code, nothing happens. This is because we didn't call the function. Functions do not execute unless they're called by name. You can call user-defined functions the same way as built-in functions.

```
<?php
function showText($text){
    echo $text;
}
showText('Hello World!');
?>
```

That's better. Now our function is declared and called. From a technical perspective, a function does not have to be defined before it is called (unless it is defined conditionally using an 'if' statement) but it is good programming practice to define functions before they are called for easier readability.

Let's try something else. Let's have the function return it to be output.

```
<?php
function showText($text){
    return $text;
}
echo showText('Hello World!');
?>
```

Now, instead of outputting the text from within the function, we return the text and output the returned value.

Function Challenge

Create a function that is passed information, does something with it, and returns it. For an extra challenge, try adding a second parameter or figuring out how to set a default value for a parameter. For a bigger challenge, try creating a function that validates data and use the function as a condition.

Variable Scope

Something very important to note is the scope of a variable. The scope of a variable is basically where it can be accessed from. Most variables can only be accessed from within their script and any script that is included in that script, which is to say they only have a single scope. Function parameters are only accessible from within their functions and variables defined outside a function are not available inside (unless passed through a parameter). Using the global keyword within a function will allow direct access to variables outside the function.

Superglobal Variables

In PHP there are a few variables that are referred to as "superglobal" variables. These variables are available in every script and from within every function. These variables are:

`$_GET` - Contains the HTTP GET variables passed to the script.

`$_POST` - Contains the HTTP POST variables passed to the script.

`$_COOKIE` - Contains the HTTP COOKIE variables passed to the script.

`$_SERVER` - Contains information about the script, server, and client.

`$_SESSION` - This is the PHP session variable that is generated when you run the `session_start()` function. The values of this variable are stored in the /tmp directory on the server and are set manually at first, but are loaded automatically afterwards, until the session is expired.

$GLOBALS - This is the alternative to the globals keyword. This super global contains every variable that is currently defined.

Arrays

Arrays are a special data type that contains an index of data. An easy way to think about it is that arrays are a list of variables contained within a single variable. In PHP arrays can be indexed numerically or with a String. Arrays that have a String index are called associative arrays, but are also referred to as "hash tables" by some. A single array can have both numerical and String keys at the same time.

```
<?php
$a = array('value1', 'value2', 'value3'); //Index starts at (int)0
echo $a[0];
$a[0] = 'new value1'; //Only sets index (int)0
$a['0'] = 'Different than (int)0'; //String key of '0' is different than (int)0
$b = array('key1' => 'value1',
           'key2' => 'value2',
           'key3' => 'value3'
     );
$b[] = 'blank'; //Append to the array.  Takes next available (int)key
print_r($a);
print_r($b);
?>
```

Arrays are indexed with integers starting with 0 by default. If you set only String keys and then append to the array it will take the next available integer value, which would be 0. If you set integer key 5 and then append it will take the next available integer key, which it will assume is 6.

Access the data within an array by referencing it's key within square brackets after the array name. [Reiterate that superglobal variables are arrays and are accessed as such.]

Array Challenge

Create two arrays using different methods. For an extra challenge, find functions to sort arrays and use them.

Loops

Often times there is a requirement to repeat a certain section of code a number of times before carrying on. Rather than typing out that same section of code over and over again, we can use loops to do it for us.

While

The while loop is the most basic of loops. The while loops checks for a condition, and as long as the condition proves true it will continue to execute the code within its braces. The condition check happens at the start of each loop

iteration, which means the while loop will only execute if the condition proves true at the beginning. It also means that if the condition changes from true to false in the middle of an iteration, the code will continue to execute until the end of that iteration.

```
<?php
$i = 0;
while ($i <= 10){
    echo $i . '<br>';
    $i++;
}
?>
```

Do-while

Do-while loops are basically the same thing as while loops. The exception being that a do-while checks the condition at the end of a loop iteration. This means that the code within the braces is executed once before the condition is checked.

```
<?php
$i = 0;
do {
      echo $i . '<br>';
      $i++;
} while ($i < 0);
?>
```

This code will only execute once. Once the condition is checked it will prove false and not execute again. Do-while loops also support the conditional break statement. The break statement will end execution of the loop code and proceed with the rest of the script.

For

For loops are one of the most complex loop types. For loops evaluate/execute three expressions as they run, and can be used any number of ways.

The first expression is evaluated/executed once at the beginning of the for loop. The second expression is evaluated at the beginning of each loop iteration. If it evaluates to true, the loop continues; otherwise the loop ends. The third expression is evaluated/executed at the end of each loop iteration. Each of the expression groups can have multiple expressions separated by a comma, or they can be left empty. If the second expression is left empty it defaults to true and the loop will run indefinitely.

For loops also support the break statement, to end execution at any point during the iteration.

```php
<?php
/* example 1 */
for ($i = 1; $i <= 10; $i++){
    echo $i;
}

/* example 2 */
for ($i = 1; ; $i++){
    if ($i > 10) {
        break;
    }
    echo $i;
}

/* example 3 */
$i = 1;
for (; ; ) {
    if ($i > 10){
        break;
    }
    echo $i;
    $i++;
}

/* example 4 */
for ($i = 1, $j = 0; $i <= 10; $j += $i, print $i, $i++);
?>
```

Each of these examples will output the numbers 1 through 10.

Foreach

A foreach loop is specifically purposed to loop through array and object values. An error will be generated if you try to run a foreach loop that references a non-array or object. It steps through each value of an array, and executes the code in its braces. The two syntaxes provide you with access to both the array key and the value, depending on which you use.

```php
<?php
$i = array('value1', 'value 2', 'value3');
foreach ($i as $value){
    echo $value . '<br>';
}
?>
```

A foreach loop uses an array's internal pointer to iterate through each value, so changing the pointer during the loop could have unexpected results. In this example, only the value of each array index is available. The following shows the alternate syntax allowing access to the key as well.

```
<?php
$i = array('value1', 'value 2', 'value3');
print_r($i);
foreach ($i as $key => $value){
    $i[$key] = 'newvalue' . $key;
}
print_r($i);
?>
```

Final Challenge (part 1)

Create a PHP file that accepts data from an instructor created form, validates it, and outputs it in an organized format.

MySQL

MySQL is the world's most popular open-source database engine. You will encounter it on nearly every web host and it's uses far exceed that of just website applications. MySQL can be used to stone information for any application. MySQL is its own application, completely separate from PHP. It offers a username/password authentication system to access databases, ensuring that they aren't open to the world. It also allows certain users to be given specific permissions on a database; meaning that you can restrict what certain user accounts can do (i.e. only view data, or only add data).

Databases, Tables, Columns, and Rows

A database is a really simple concept. Think of it like a collection of information organized into tables. Each table is divided into columns and rows, and each row is a data entry.

[Use phpMyAdmin to provide a more visual demonstration of structure.]

Where each column and row meet is called a cell. Each cell can have its own set of data restrictions. So far we have been taking about storing information in variables in PHP, which do not need a type defined. In MySQL, each column can only contain a specific data type, and in most cases has a maximum length or size. This makes it very important that information that is being stored in the database is checked first to make sure that it meets the data type requirements.

Query Types

When you run a command on the SQL database it's called a query. There are many different kinds of queries for creating new databases and tables, and much more, but for today we are going to focus on the ones that manipulate data within a database table that has already been created. The query syntax can become very complicated so it will be explained as simply as possible.

Insert

Self-explanatory really. An insert query is used to insert data into a table. You can use it to insert one or multiple rows at one time. The syntax for a basic insert query is [type]:

> INSERT INTO tbl_name (column1, column2, column3) VALUES (1, 2, 3), (4, 5, 6), (7, 8, 9);

Note that the proper syntax for SQL queries has a semi-colon at the end of each instruction, just like PHP, but when executing one query at a time it is not mandatory.

Select

This is the most commonly used SQL query. A select query searches the table for a specific set of conditions and selects only those rows that match to return. The syntax for a basic select query is [type]:

> SELECT column1, column2 FROM tbl_name WHERE column1 = 1;

Note the paralanguage structure of the query.

Update

An update query updates an existing row with new information. The basic syntax is [type]:

> UPDATE tbl_name SET column1 = 1, column2 = 3, column3 = 5 WHERE column1 = 1;

Delete

The delete query will delete an entire row from the table. The delete query only works on rows within tables and cannot delete tables or databases themselves. The basic syntax is [type]:

> DELETE FROM tbl_name WHERE column1 = 1;

MySQLi Extension

To connect to a MySQL database in PHP there are many tools and extensions available, but the most common one with the latest PHP version is MySQLi. MySQLi is a PHP extension that comes packaged with most builds of PHP. We haven't talked about objects much, but we will be using this extension as an object; although it can also be used procedurally.

Open Database Connection

The first step is to connect to the database. Using the MySQLi extension we are going to accomplish this by creating a new MySQLi object. Think of an object in PHP as an actual object. Each object has a set of functions it's designed to do.

```
<?php
$db = new mysqli('localhost', 'username', 'password', 'database_name');
?>
```

This code will create a new MySQLi object using the database server, username, password, and the actual database name. It's important to remember that the database server field is relative to the location PHP is being executed. For example, if PHP and MySQL are on the same server then use localhost, whereas if you use a remote MySQL server that is different from your web server, use that server's IP address.

Execute Queries

Now that we have our object containing the functions of the MySQLi class, we can start running queries on the database. Executing a query is as simple as calling the query function.

```
<?php
$db = new mysqli('localhost', 'username', 'password', 'database_name');
$db->query('SQL COMMAND');
?>
```

Select

A select query is unique because it is requesting data, and therefor needs to return something. Interestingly, when you execute a select query using MySQLi it will return a reference to the location where the data can be found, which you then run another function on to extract the data from the database. But to capture that reference we have to capture the return of the function and assign it to a variable.

```
<?php
$db = new mysqli('localhost', 'username', 'password', 'database_name');
$res = $db->query('SELECT * FROM people WHERE id <= 5');
?>
```

The result variable becomes a result object, containing the reference and the functions to review and retrieve the data. For example, we can use the result to check the number of rows that were found, and as long as it's more than 1, to fetch the data.

```php
<?php
$db = new mysqli('localhost', 'username', 'password', 'database_name');
$res = $db->query("SELECT * FROM people WHERE id <= 5");
if ($res->num_rows > 1){
      while ($row = $res->fetch_array(MYSQL_NUM)){
            $people[] = $row;
      }
}
?>
```

The while loop is the method of retrieving multiple rows from a return. Using the while loop in this way advances the internal counter of the result which keeps track of the query. Each time the while loop advances the internal counter does as well, making it stop once there are no longer any rows to use. This example uses the `fetch_array()` function which returns the current row as an array. You can either have it use a numerical index or an associative index. The associative index uses the column names as the key values when the array is returned.

This is done using the two constants MYSQL_NUM and MYSQL_ASSOC. Constants are like variables but aren't referenced the same way, and aren't designed to have their values changed.

A select query will return a false value if it was not successfully executed.

Insert, Update, and Delete

For insert, update, and delete queries, there is no data to return and thus the query needs only to be executed. The query function will return a true value as long as the query was successful, or a false if they were not.

Final Challenge (part 2)

Using the same form as part 1, store the validated form data in a database table, then display the table contents.