Episode 2

Basic Syntax. PHP and HTML

Introduction

Welcome back. In this episode, we are going to learn about PHP syntax and how to better integrate PHP and HTML.
What do we mean by syntax? Well, the syntax of a language is the rules of how to correctly write that language. PHP borrows some of its syntaxes from languages like C and Perl, like curly braces for blocks of code (inspired by C) and something like $ dollar signs for variables which you don't see almost anywhere else but it comes from Perl.

Goals

In this section of the course your goals are:

- [ ] To learn the basic syntax of PHP.
- [ ] To learn how to use comments in your code.
- [ ] To learn why case sensitivity matters.
- [ ] To learn to different ways to integrate PHP with HTML.
- [ ] To learn about statement separation.

Basic Syntax

Every line of code that you write in PHP must be between the
opening tag: **<?php**
and the closing tag: **?>**
Everything in between those tags will be parsed by the PHP interpreter, and everything outside of the tags will be ignored.

Let's take a look at the following example:

```
<!DOCTYPE html>
<html>
<head>
    <meta charset="utf-8">
    <title>PHP Syntax</title>
</head>
<body>
    <?php
    echo "<h1>Hello World!</h1>";
    echo "<h2>My name is John</h2>";
    echo "<p>I am a master pianist.</p>";
    echo "I can solve a Rubik's cube in 35 seconds.";
    ?>
</body>
</html>
```

So far so good. If we open this code in our browser, everything looks good. Now let's add one more line to our code. Under the closing PHP tag let's add

```
    echo "<h1>Hello Again</h1>";
```

Let's take a look in the browser to see what happens. What we find is that the last line that we added has spread over three lines in our browser. What a disaster! Why did that happen? Let's find out. First, the PHP interpreter ignored that last line we added, so now it is up to the browser to render it as it sees fit. The last line that the PHP interpreter sent to the browser doesn't have a line break, so this is where the browser takes over adding **echo "** on the same line. Next, the browser finds an **h1** tag, and because the h1 tag is a block level tag the contents of it are rendered on their own line. Finally, after the closing h1 tag the browser finds **>"**, and so it sends it to the next line. So our disaster is now complete. One seemingly harmless line of code has turned into a three line disaster. So always remember to include your PHP code in between the opening and closing PHP tags. Now you know what can happen if you don't.

Comments in PHP

PHP supports several kinds of comments:

1. C style comments that begin with /* and end with */.
2. C++ style comments that start with // and go to the end of the line.
3. Unix shell-style comments that begin with # and go to the end of the line.

Comments are used mainly for two reasons:

1. To document your code. This is true for other developers that may read your code or just as important for yourself, because as your code grows more complex having a couple of lines of comments will help you remember what a particular piece of code does. It is more common than you think to take a look at some of your code a few months after you've written it and have no idea what it does.
2. To "comment out" sections of code that you are working on. Sometimes you need to add a few lines

of code to see how it affects the rest of what you've already written and commented out pieces of code in different parts of your file can help you know what each block of code does.

Let's look at an example:

```
<!DOCTYPE html>
<html>
<head>
    <meta charset="utf-8">
    <title>Comments</title>
</head>
<body>
    <?php
    /*
       The tradition of writing your first program in a
       new programming language as "hello, world" goes
       back to 1972.
    */
    echo "<h1>Hello World!</h1>"; /* One line comment */
    echo "<h2>My name is John</h2>"; // Yes, that is my name.
    echo "<p>I am a master pianist.</p>"; # That is because I practice all the time.
    echo "I can solve a Rubik's cube in 35 seconds.";
    // My best time in competition is 34.56 seconds.

    // Test the next two lines to see how the page looks
    echo "<h2>I also play soccer.</h2>";
    echo "<h2>I have three brothers and one sister.</h2>";
    ?>
</body>
</html>
```

Case Sensitivity in PHP

In PHP all variables are case-sensitive. For example, the following code

```
$car = "Toyota Camry";
$CAR = "Toyota Rav4";
$Car = "Toyota Sienna";
```

will create three different variables. If we output these three variables to the screen, this is what we'll see:

```
Toyota Camry
Toyota Rav4
Toyota Sienna
```

In contrast, all keywords, functions and user-defined functions are not case-sensitive. Let's take a look at the following code:

```
echo "I am not case sensitive.";
Echo "This is also valid code.";
ECHO "Yet another way to use echo.";
```

The difference between the two previous examples is that while in the first example three different variables were created, in the last case there is only one “echo.” These are also good examples of how not to write code. Imagine the confusion of having the three variables $car, $Car, and $CAR in your code. It would be easy to use one for the other and have an error in your code that would be very difficult to find. As for the example that uses the echo statement, even though all three ways are valid, there are code writing conventions that specify how to write them.
In PHP the general rules are the following:

1. Use **lowerCaseCamel** for variable names:
   $monthlyRent = 1500;
   $firstName = John;
2. Use **lowerCaseCamel** for function names:
   function addDigits($digitOne, $digitTwo) { ... };
   function writeMsg( ) { ... };
3. Use **UpperCaseCamel** for class names:
   class CreateUser;
   class ImportantPerson;

Although these rules are widely adopted, they may vary slightly. For example, a large company with several teams of developers may have its code writing conventions, such as writing variable names with underscores: **variable_name**. However, it is a good idea for you to adopt a widely used way of writing code.

PHP and HTML

Besides being able to embed PHP in HTML in a single chunk of code like we’ve done before, we can also code PHP in different places within HTML. Let’s look at the following code:

```
<!DOCTYPE html>
<html>
<head>
    <meta charset="utf-8">
    <title>Embedding PHP</title>
</head>
<body>
    <?php
    echo "<p>We can embed PHP in HTML in more than one way!</p>";
    echo "<p>One way is placing all the PHP in a single place</p>";
    ?>
    <p><?php echo "We can also do this!";?></p>
</body>
</html>
```

Notice the difference in the two ways of embedding PHP. Firstly, the HTML is created using PHP, whereas in the second way the HTML structure is in place just waiting for content sent by PHP to be placed in it. A third way of embedding PHP is to code it **above** the doctype declaration. This way is used to receive and process information from a different file, manipulate this data in some way and then place the data within the body of the HTML portion. This will become more apparent in the next episode when we learn to declare and use variables.

Statement Separation

One very quick but important note about how PHP knows when you finish one statement and start another. Consider the following two pieces of code:

```
    echo "Statement One";echo "Statement Two";
```

```
    echo "Statement One";
    echo "Statement Two";
```

As far as PHP is concerned, there is no difference between the two, but the second part is much easier to read by a person. So how does PHP know when one statement ends, and the other begins? The answer is the semicolon. Whenever PHP encounters a semicolon, it will mark that as the end of a sentence.

Debugging Code

Examine the following code. Find and fix the errors.

```
<!DOCTYPE html>
<html>
<head>
    <meta charset="utf-8">
    <title>Debugging Episode 2</title>
</head>
<body>
    <?php
    // Find the error in the next line
    echo "<p>In the next episode I will learn about PHP types.</p>"
    # The following line also has an error
    echo "<p>I will also learn about variables.</p>;
    /* This comment has an error
    ?>
    <p><? echo "What is wrong with this?" ></p>
</body>
</html>
```

Lab Exercises

The following exercises are meant to help you practice your coding skills. They need to be submitted for you to get a certification. As stated before coding is not a spectator sport, the more you practice (and make mistakes) the more you will learn.

Starting with this basic HTML:

```
<!DOCTYPE html>
<html>
<head>
    <meta charset="utf-8">
    <title>Lab Exercise 2</title>
</head>
<body>
    <h2> </h2>
    <p> </p>
</body>
</html>
```

1. Add PHP code inside the body:

- On the first line create an **h2** tag using PHP and add content in it.
- On the second line create a **p** tag using PHP and add content to it.
- On the third line add a sentence without creating any HTML tags.

2. Use PHP to add content inside the given **h2** tags.
3. Use PHP to add content inside the given **p** tags.

When you are done, save your file with a .php extension, and when you open in the browser it should look similar to this:
Image of Completed Lab

These were some of the main syntax rules that must be followed when coding PHP. As we move along in the course, you will learn more syntax about more specific parts of PHP like functions and arrays. In the next episode, we will learn about data types and variables. See you then.