Pits Of Doom Lesson 3: Mixing Things Up

In the last lesson we covered how to make a form, check to see if a button has been pressed, and display a message according to whether the user found the treasure or not. Now it's time to take our very simple navigation page to the next level: random treasure location.

In our final game, the position of the treasure will be pre-determined by data in our database. However, we won't know what position the treasure is in as the user is navigating their character around the game because this information will be loaded as they play.

In our next lesson we're going to apply a more advanced concept to figuring out if the treasure is in the correct location. Instead of always knowing where the treasure is, we're going to make the treasure start in a random direction each time the page is loaded. That way we have to check every time to figure out where the treasure is, and if the player will fall into a pit or not.

Lesson Concepts

Functions

In programming, a function is a great way to break bigger tasks into smaller more manageable pieces, or reduce the need of having to write the same piece of code over and over again. In PHP the syntax for a function is very simple:

```
<?php
function myfunction ($parameter1, $parameter2)
{
echo $parameter1 . " " . $parameter2;
}
?>
```

This is a really simple function called myfunction. It takes two parameters and simply echoes them on the screen whenever you call it. The periods you see in the echo statement are a way to concatenate or join the two variables together with a space in between them.

Calling a function is just as easy as making one. Anywhere you want myfunction to print something on the screen all you have to do is:

```
<?php
echo "Hey there, here is what myfunction has to say: ";
myfunction("Hello", "World!");
?>
```

If you were to run this you'd see: Hey there, here is what myfunction has to say: Hello World! Not only can a function print to the screen, it can return values you can then store into variables. Let's take a look at this function:

```
<?php
function Mod5 ($number)
{
return $number % 5;
}
?>
```

This function takes any number you give it and mods it by 5. If you aren't familiar with the modulus operator, all you need to know is that it returns the remainder from the division, and not the number you'd get if you divided $number by 5.

Now let's see what how we'd use a function that returns a value instead of printing it on the screen:

```
<?php
//declare our function
function Mod5 ($number)
{
return $number % 5;
}
if ($_POST['number']) //they've entered a number
{
echo "Your number is: " . $_POST['number'] . "<br/>";
$number = myfunction($_POST['number']); //we store the modulous of what they entered back into the number variable
echo "Your number % 5 is: $number";
}
else //show them a form to enter a number
{
?>
<form action="#" method="post">
Please Enter a Number: <input type="text" name="number"><br/>
<center><input type="submit" name="submit" value="submit" /></center>
</form>
<?php
} //don't forget to close our ELSE statement
?>
```

In this example we've taken the number a user enters into a text box and then display the number back to them along with the remainder of that number when it's divided by 5.

One of the great things about PHP is that it comes with a huge collection of pre-defined functions you can use when you program. You can take a look at all the functions by going to php.net. The one function I want to touch on in this lesson is the random function.

As you can probably tell, you don't always need to know all the little specifics that go on inside of a function as long as it always gives you the results you're expecting. The same goes with the random function. I could go into details on how it takes the current time on your computer and does a bunch of mathematical stuff to it — but that's just plain boring!

The short of it is, the rand function takes 2 parameters. The first parameter is the lowest number you want it to generate when it runs. The second number is the highest number you want it to generate when it runs. So, if you wanted a random number between 1 and 4, including the numbers 1 and 4 you'd do this:

```
<?php
echo "My random number is: " . rand(1, 4);
?>
```

View the page with this php code. Now refresh the page. You'll notice the number is constantly changing.

One other function you'll need to know is the unset() function. This function is great, you pass it a variable or an object (don't worry, we'll get to those later) and it sets the value of that function to null. Think of it as taking a jar of cookies, opening the lid, and dumping all the cookies on the ground. That's essentially what the unset() function does.

Congratulations, you now know how the rand and unset functions work. You'll need both of these for this lesson!

Sessions

Now one of the biggest problems we run into is how to keep track of a user's past responses. Or how to keep track of anything about the user from one page to another. That's where sessions come in. Sessions are a way to track information about the user as they go from one page to another. Eventually we'll be using information from the database to track a lot of the moves a user is making with their characters, but we'll still need to know who the member is as they go from page to page. For right now, we'll use sessions to keep track of the direction the user picked. And now, instead of letting the user continue to pick a direction after they've fallen into the pit we're going to give them a game over message until they decide to reset the game and try again.

Sessions are very easy to use. They work almost exactly like the $_POST and $_GET variables you can use to access information about what someone has entered on to a page. However, there's a few major differences:

1. You can set the values of the $_SESSION variable
2. Session variables remain with the user until you tell them to expire, they don't change page by page.
3. Session variables can be set to expire so they don't last forever, or you can set them so they never expire and DO last forever.

Using sessions now will get you familiar with them before the programming becomes more complicated.

If you've done anything with cookies before then sessions will be a breeze for you. Sessions are a good alternative to cookies because if a browser has cookie functionality turned off you will still be able to keep information about a user from one page to another. If that went right over your head don't worry, it's not essential to this lesson.

There's one thing you have to keep in mind about sessions. In order to use a session you have to call a special function at the top of every php page, BEFORE YOU DO ANY KIND OF OUTPUT!! This last is important. If you try to output anything before calling the session_start() function you will get an error.

Now let's setup a simple session. Create two pages, one called welcome.php and the other called main.php. Welcome.php is an example of a web page you might see that greets you and asks you what your name is. Main.php is an example of a page you might look at after you've visited the welcome.php page.

In welcome.php we have this:

```
<?php
session_start(); //the first thing we do is start our sessions
if ($_POST['name']) //they entered their name in the form
{
$_SESSION['name'] = $_POST['name'];
//now we're going automatically to redirect them our next page
//using a function called header() that's built into PHP
header("Location: main.php");
exit;
}
?>
<form action="#" method="post">
Enter your name for our first session variable! <input type="text" name="name" />
<center><input type="submit" name="submit" value="All Done!"></center>
</form>
```

Now, inside of main.php we're going to put this:

```
<?php
session_start(); //we have to start our session again to get the data we saved before
echo "Welcome to the main menu " . $_SESSION['name'] . "!";
?>
```

In the main.php file we pull out the information we stored in the session variable for name and print it back out on the screen. Nifty! Now we know, no matter which page this user goes to from now on, as long as we call session_start() we'll be able to access their name from the variables $_SESSION['name'].

Congratulations! You should now understand the basic concept of sessions.

Summary

In this lesson we learned about how to make our own function and talked about how PHP has it's own predefined functions we can use to make our life a whole lot easier. I talked about how to make a function print something on the screen versus a function that returns a value. Functions that return a value can also be printed on the screen, but you'd have to put an echo in front of the function call to do that.

Next we talked about sessions. Sessions are a way to store data as the user moves from page to page around your website. Without sessions we wouldn't be able to tell who was who as multiple people will eventually load this game and try to play it. Right now we'll be using sessions to keep track of a game being over or not, but eventually we'll only use sessions to keep track of which user a person is as they play.

Game Files

character.php
character2.php (no source code)
Note: these are text files so you can copy/paste all the source code.

Lesson Reasoning

In this lesson character.php now has the treasure starting at a random location. If the user picks the incorrect location they get a game over message instead of being able to pick a direction again. In order to play again the user has to click on a reset button to start the game all over again. I've introduced functions, sessions, and a more abstract checking.

Once we start pulling information from the database we won't know anything about the character's current location at runtime other then their x,y,z coordinates and the map they're on. Information about items in that position will all have to be tested each time the character tries to move somewhere. By introducing the treasure in random locations we start to get into the habit of checking information about our location as we select a particular direction.

Find lesson 4 at http://design1online.com!