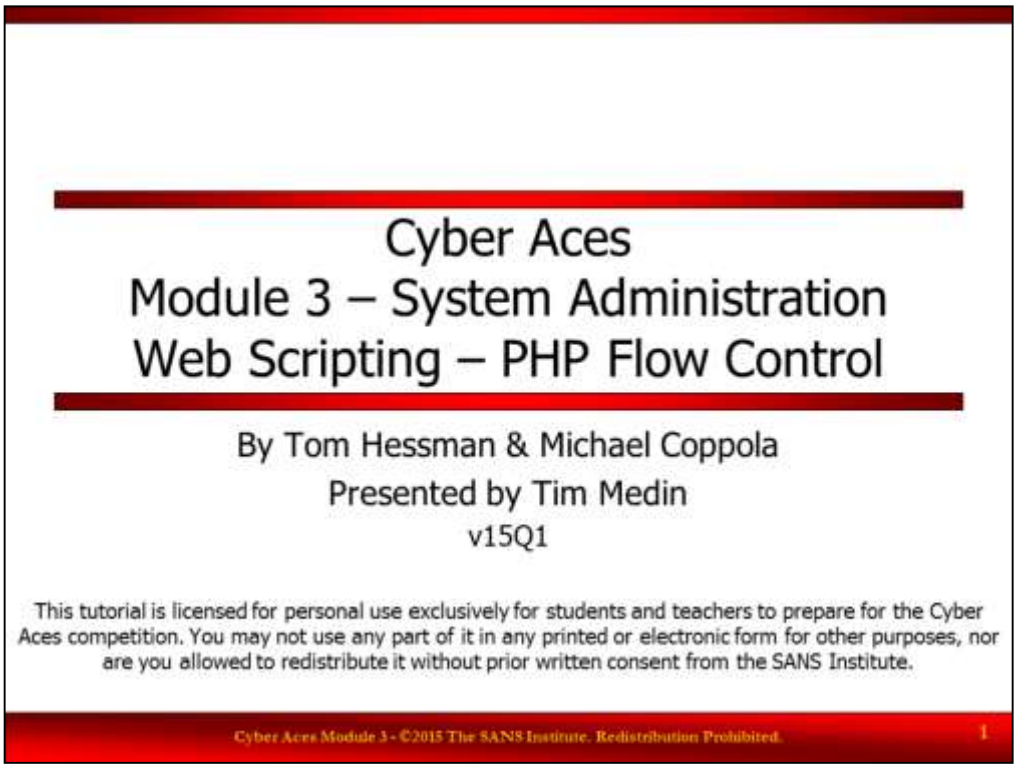
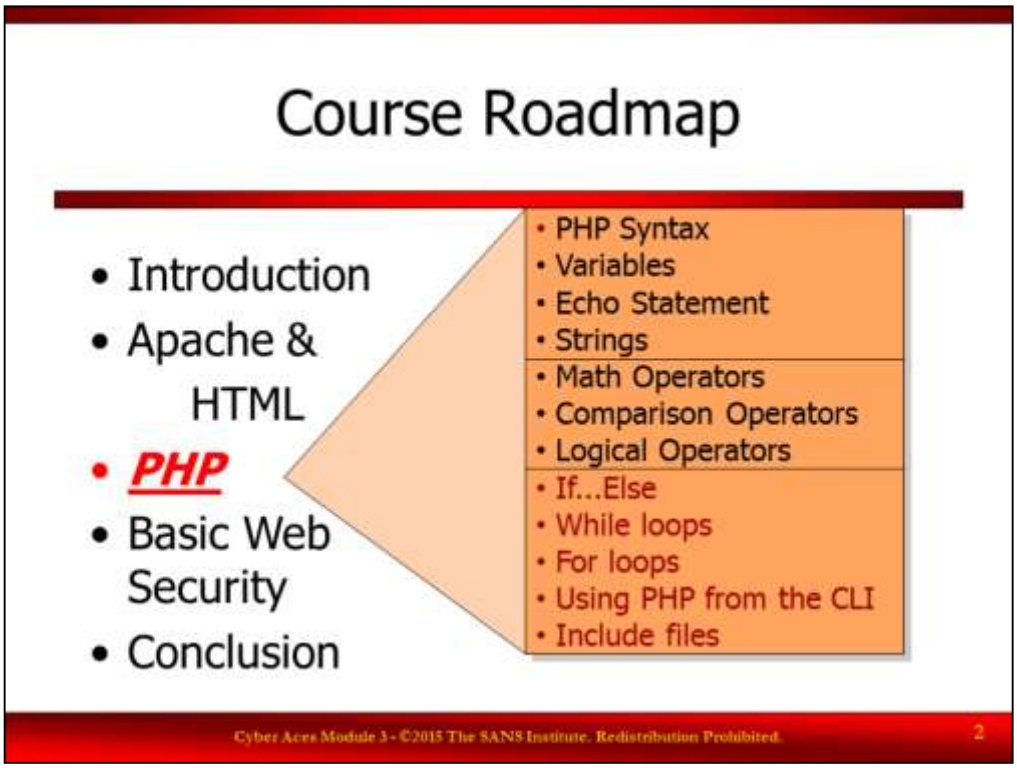
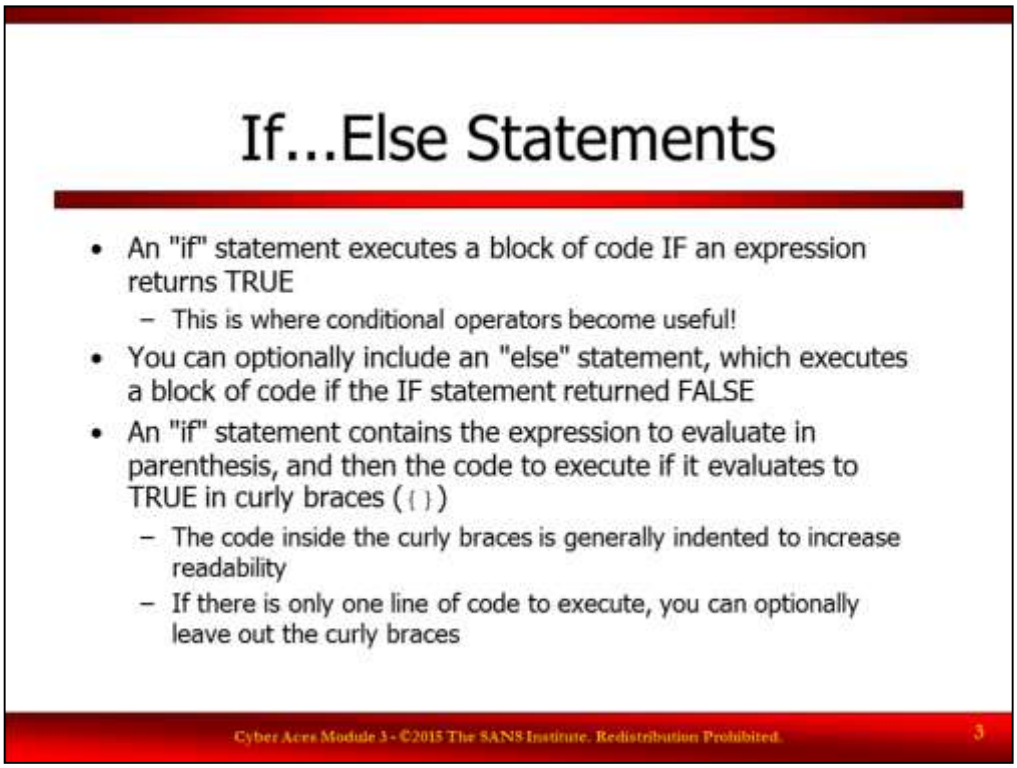


Welcome to Cyber Aces, Module 3! This module provides an introduction to the Apache Web Server, HTML, PHP, and basic web security.



Course Roadmap

In this section, we will continue to discuss PHP. Specifically, we'll discuss flow control in PHP.



If…Else Statements

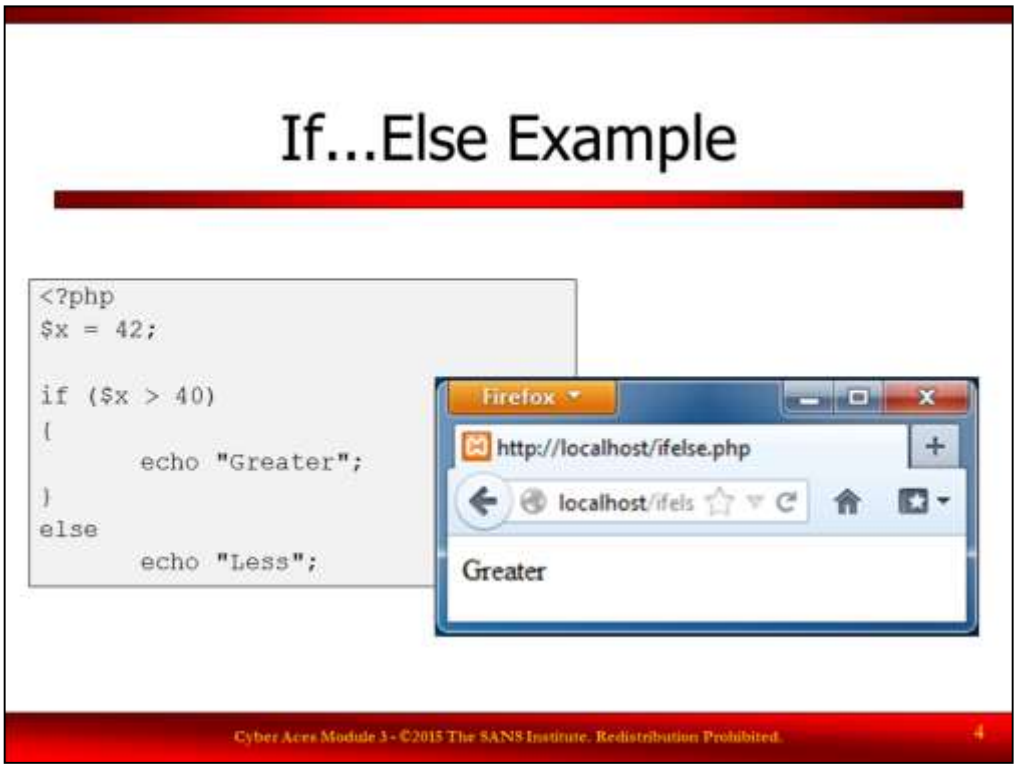
One of the most important constructs in any programming language is a conditional statement. The most common form of conditional statement is an "if" statement, which executes a block of code IF some condition is true. This is where the conditional operators become useful! You can optionally include an "else" statement after the "if" statement, to be executed only if the "if" statement returned false.

An "if" statement contains the expression to evaluate in parenthesis, and then the code to execute if it evaluates to TRUE in curly braces ({ }) starting on the next line. There can be as many lines of code as necessary within the curly braces. The code block inside the curly braces is generally indented (with the tab key, or a fixed number of spaces), in order to increase code readability (making it more obvious that it's part of a particular statement. Also, if there is only one line of code to execute, you can optionally leave out the curly braces.

Further reading:

http://www.tizag.com/phpT/if.php

http://www.tizag.com/phpT/ifelse.php



If…Else Example

In this example, the variable `$x` is set to 42, and then the if statement checks to see if `$x` is greater than 40. Since 42 is greater than 40, it executes the code block immediately following the if statement, echoing the string "Greater". If `$x` was not greater than 40, it would have executed the code block immediately following the else statement, echoing the string "Less". Note that the curly braces are not necessary when there is only one line of code, as illustrated by the else statement.

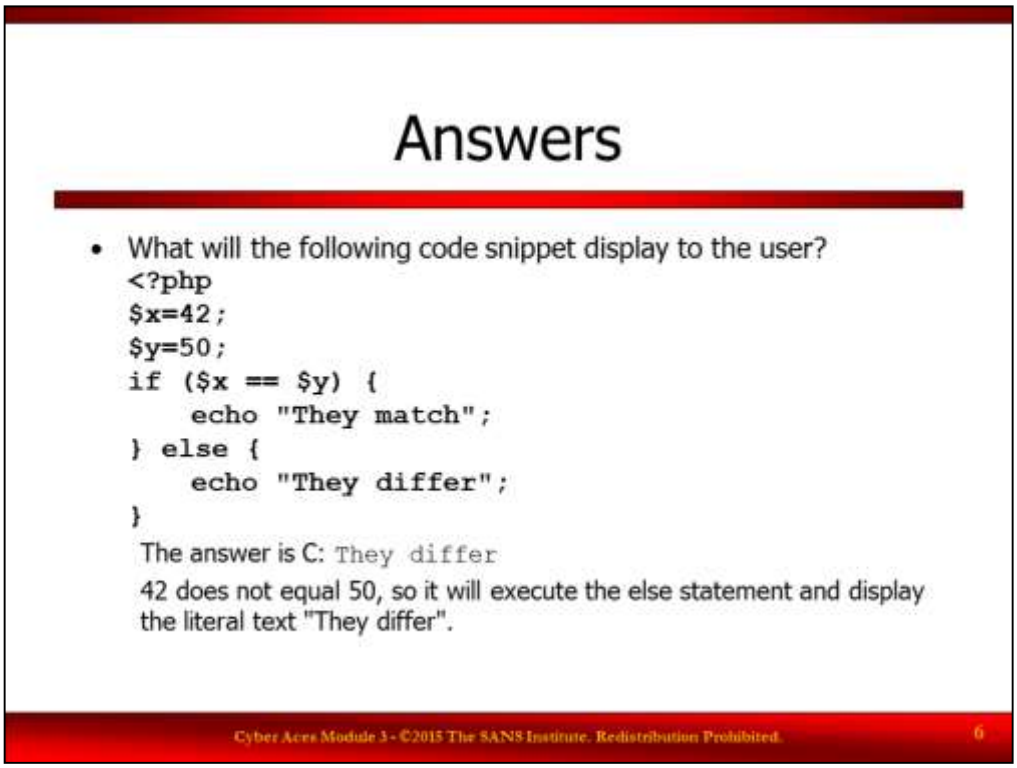
Review

- What will the following code snippet display to the user?

```
<?php
$x=42;
$y=50;
if ($x == $y) {
    echo "They match";
} else {
    echo "They differ";
}
```

a. 42
b. 50
c. They differ
d. They match

Review



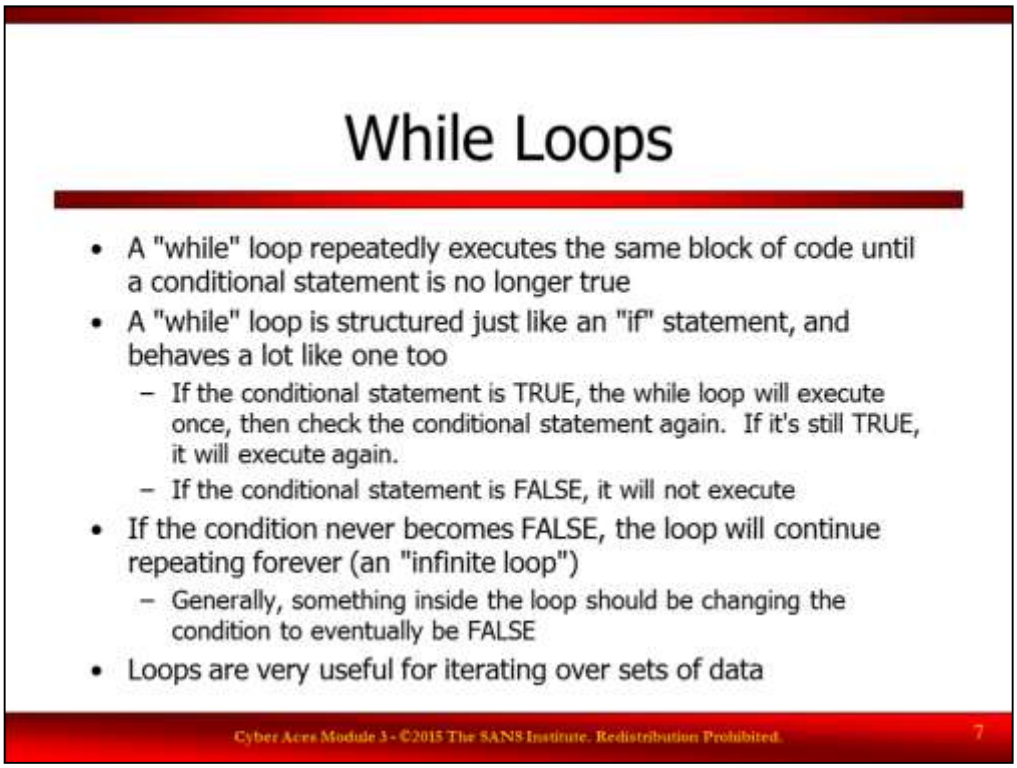
Answers

What will the following code snippet display to the user?

```
<?php
$x=42;
$y=50;
if ($x == $y) {
      echo "They match";
} else {
      echo "They differ";
}
```

The answer is C: `They differ`

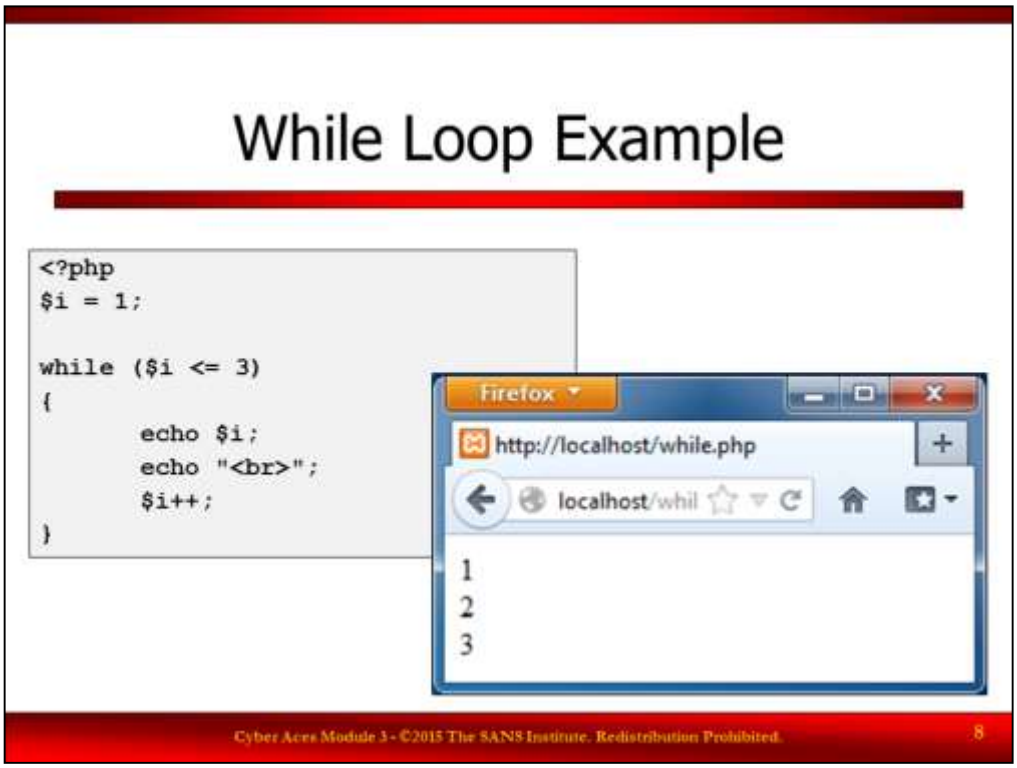
42 does not equal 50, so it will execute the else statement and display the literal text "They differ".



While Loops

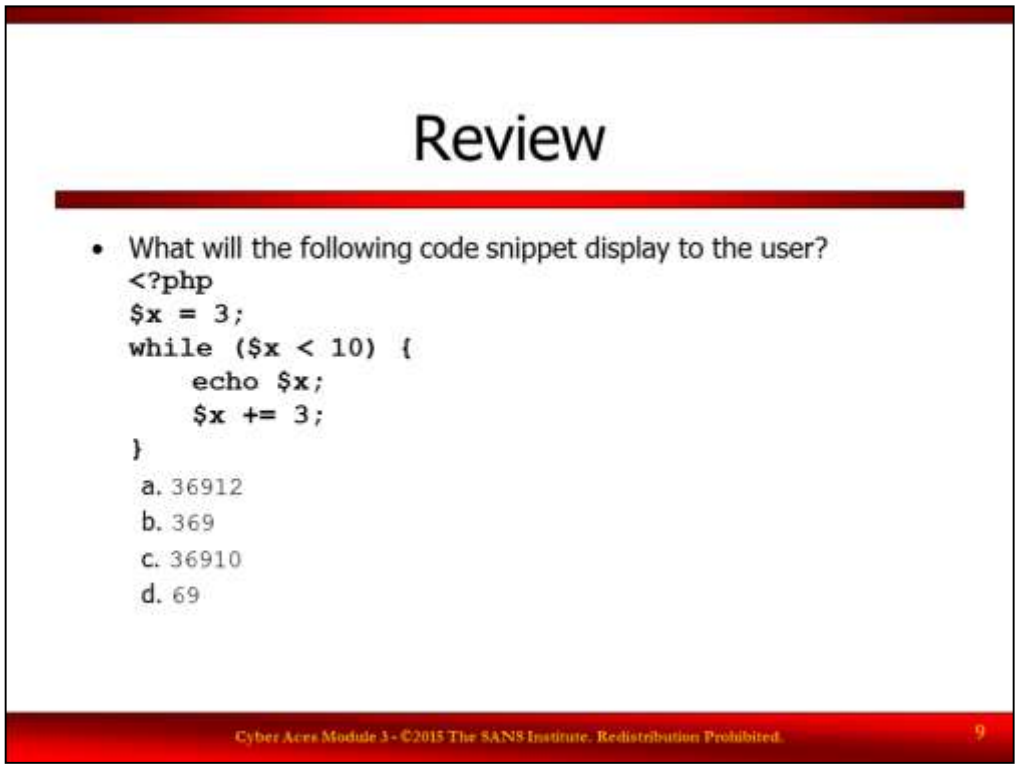
A "while" loop repeatedly executes the same block of code until a conditional statement is no longer true. A "while" loop looks a lot like an "if" statement, and behaves a lot like one too. If the conditional statement is never true, the "while" loop will never run. If the conditional statement is always true, the loop will never stop running (which is referred to as an "infinite loop"), so generally something in the body of the loop should be changing the condition such that it ends at some point.

Further reading: http://www.tizag.com/phpT/whileloop.php



While Loop Example

This example uses a while loop to display all the numbers between 1 and 3, inclusive. It starts by setting the variable "`$i`" to "1", the first number to display (`$i` is the most commonly used counter/iterator variable in loops). The while statement is then evaluated, like an if statement, to see if 1 is less than or equal to 3. It is, so it executes the code block (echoing 1, an HTML line break, and then incrementing `$i` by 1). PHP then checks the while condition again. This time, `$i` is 2, which is still less than or equal to 3, so it executes the loop body again. Now `$i` is 3, which is still less than or equal to 3, so the loop body executes again. This time, `$i` is 4, which is not less than or equal to 3, so it stops executing the loop.



Review

What will the following code snippet display to the user?

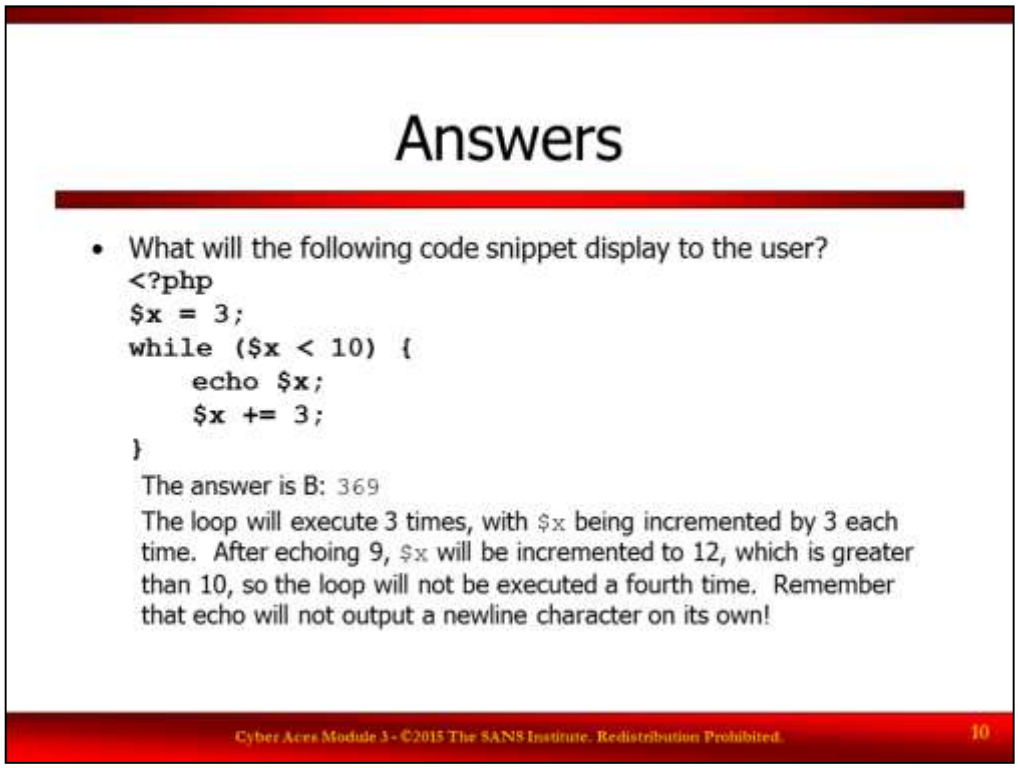
```
<?php
$x = 3;
while ($x < 10) {
      echo $x;
      $x += 3;
}
```

a. 36912

b. 369

c. 36910

d. 69



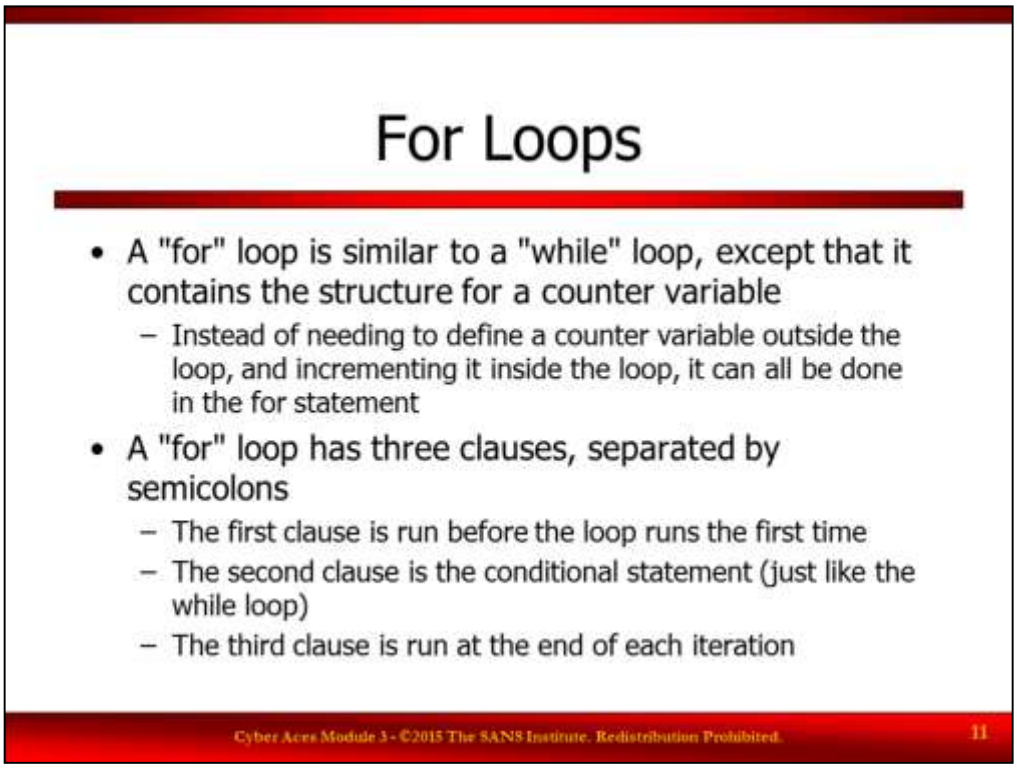
Answers

What will the following code snippet display to the user?

```
<?php
$x = 3;
while ($x < 10) {
      echo $x;
      $x += 3;
}
```

The answer is B: `369`

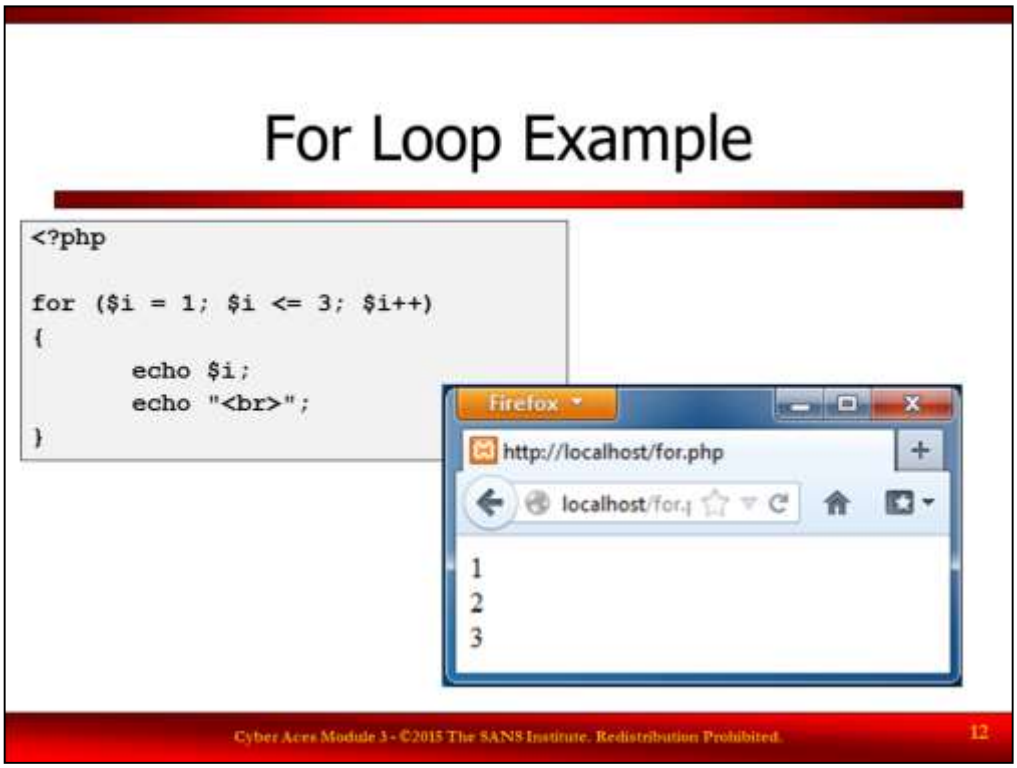
The loop will execute 3 times, with $x being incremented by 3 each time. After echoing 9, $x will be incremented to 12, which is greater than 10, so the loop will not be executed a fourth time. Remember that echo will not output a newline character on its own!



For Loops

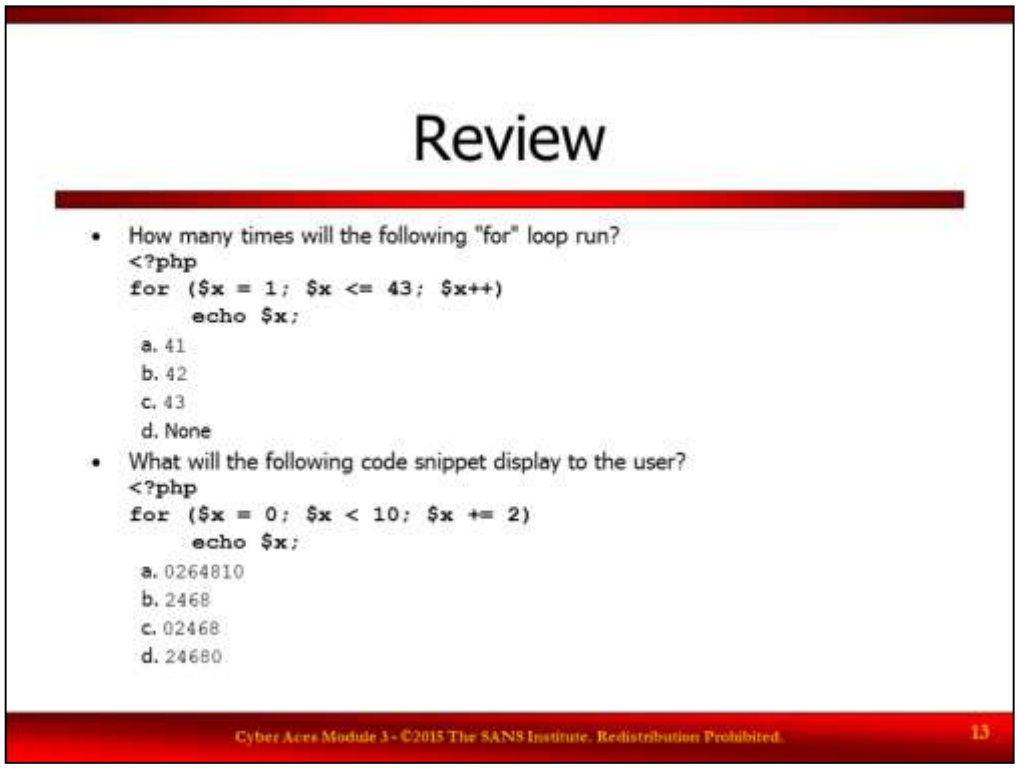
A "for" loop is similar to a "while" loop, except that it contains the structure for a counter variable in the loop itself. Therefore, instead of defining a separate counter variable before the loop and incrementing it in the body of the loop, it can all be done in one step. The first clause is run before the loop runs the first time, the second clause is the conditional statement (just like the while loop), and the third clause is run at the end of each iteration.

Further reading: http://www.tizag.com/phpT/forloop.php



Foor Loop Example

This example uses a for loop to display all the numbers between 1 and 3, inclusive. Note that it uses the same structure as the while loop example, but adapted to a for loop. It starts by setting the variable "`$i`" to 1, the first number to display ($i is the most commonly used counter/iterator variable in loops). The conditional statement is then evaluated, like an if statement, to see if 1 is less than or equal to 3. It is, so it executes the code block (echoing 1 and an HTML line break), and then increments `$i` by 1. PHP then checks the while condition again. This time, `$i` is 2, which is still less than or equal to 3, so it executes the loop body again. Now `$i` is 3, which is still less than or equal to 3, so the loop body executes again. This time, `$i` is 4, which is not less than or equal to 3, so it stops executing the loop.



Review

How many times will the following "for" loop run?

```
<?php
for ($x = 1; $x <= 43; $x++)
      echo $x;
```

a. 41

b. 42

c. 43

d. None

What will the following code snippet display to the user?

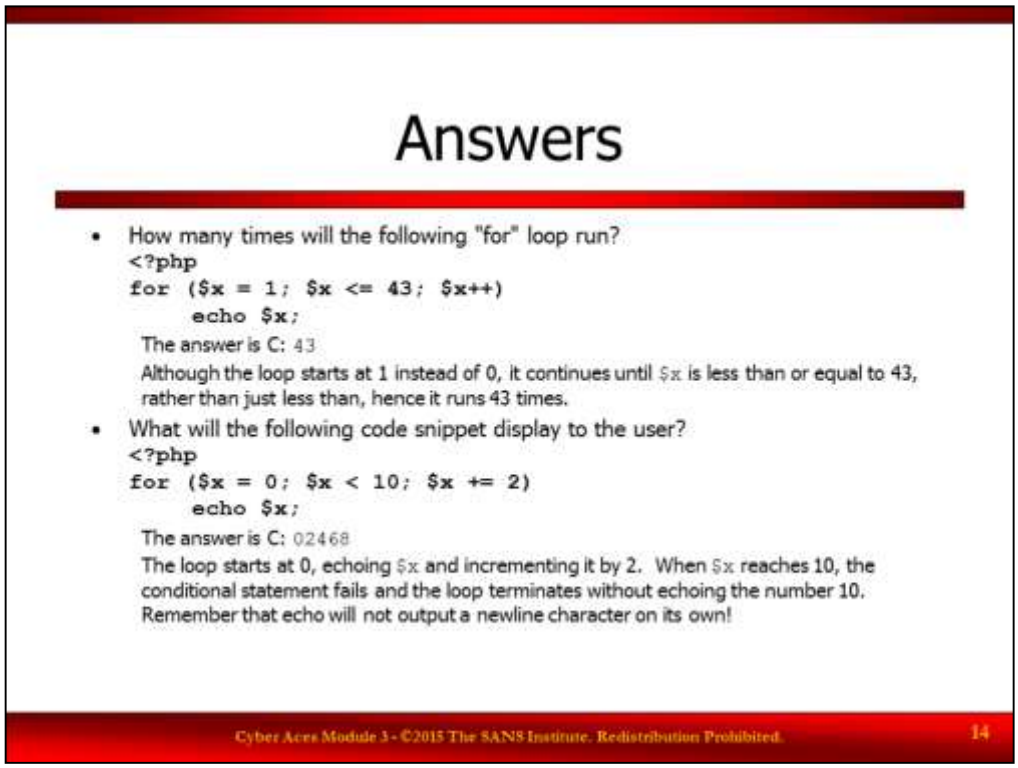
```
<?php
for ($x = 0; $x < 10; $x += 2)
      echo $x;
```

a. 0264810

b. 2468

c. 02468

d. 24680



Answers

How many times will the following "for" loop run?

```
<?php
for ($x = 1; $x <= 43; $x++)
      echo $x;
```

The answer is C: `43`

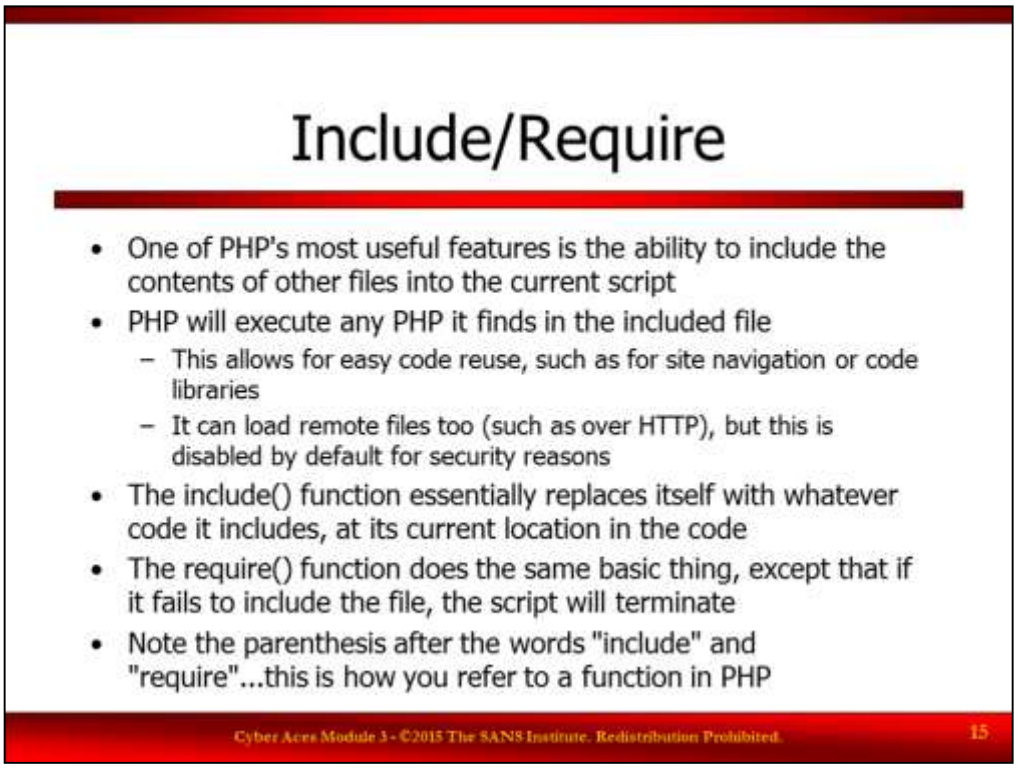
Although the loop starts at 1 instead of 0, it continues until $x is less than or equal to 43, rather than just less than, hence it runs 43 times.

What will the following code snippet display to the user?

```
<?php
for ($x = 0; $x < 10; $x += 2)
      echo $x;
```

The answer is C: `02468`

The loop starts at 0, echoing $x and incrementing it by 2. When $x reaches 10, the conditional statement fails and the loop terminates without echoing the number 10. Remember that echo will not output a newline character on its own!



Include/Require

One very useful PHP feature is the ability to include the contents of other files into the current script. On a website, for example, you could put the code that generates the header or the site navigation into a single file, and then have every other page include that file so that you don't have to keep copying it and maintaining separate copies of the same thing. "Include files" are also useful for storing a library of custom PHP code that you plan to reuse. Depending on how PHP is configured, "include()" can also pull in remote files, such as from a website over HTTP, but newer versions of PHP disable this functionality by default for security reasons.

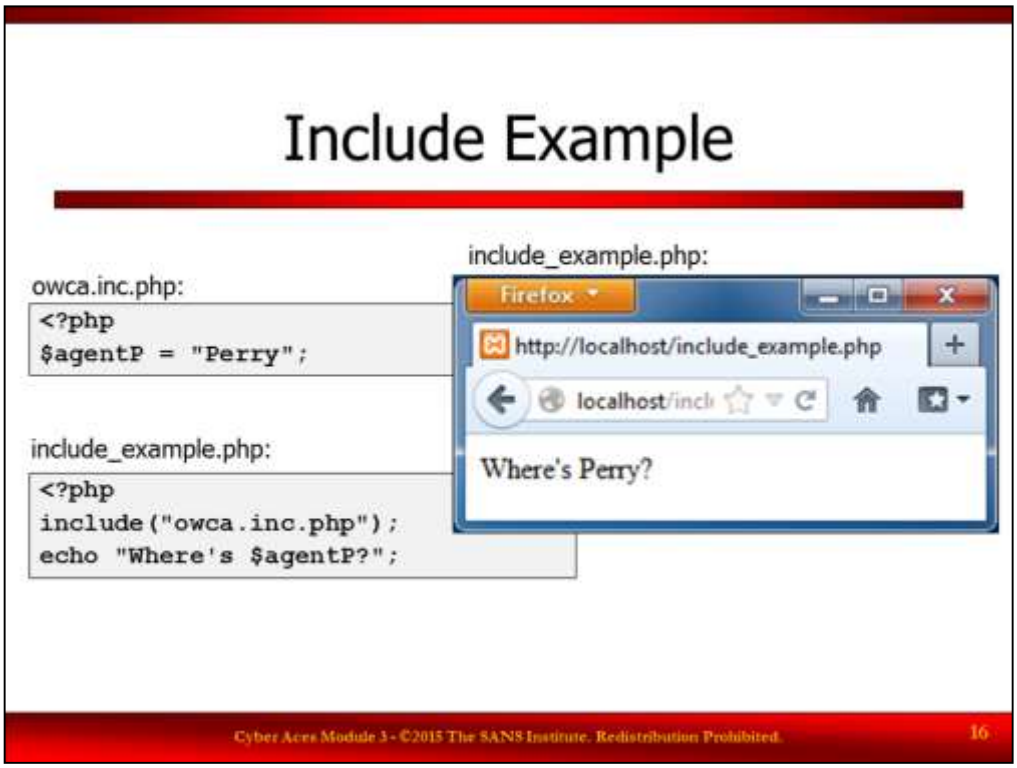
"Include()" also has a complementary function, "require()". The main difference between "include()" and "require()" is that if "include()" fails to pull in the file for some reason, the rest of the script will still be executed, whereas if "require()" fails to pull in the file the script will terminate. If the code pulled in by the included file is critical to the rest of your script, then "require()" should be used instead.

Note the parenthesis after the words "include" and "require". When referring to a function, it is required to put parenthesis after the name to indicate that it is a function, since you have to use parenthesis when calling the function in your PHP code.

Further reading:

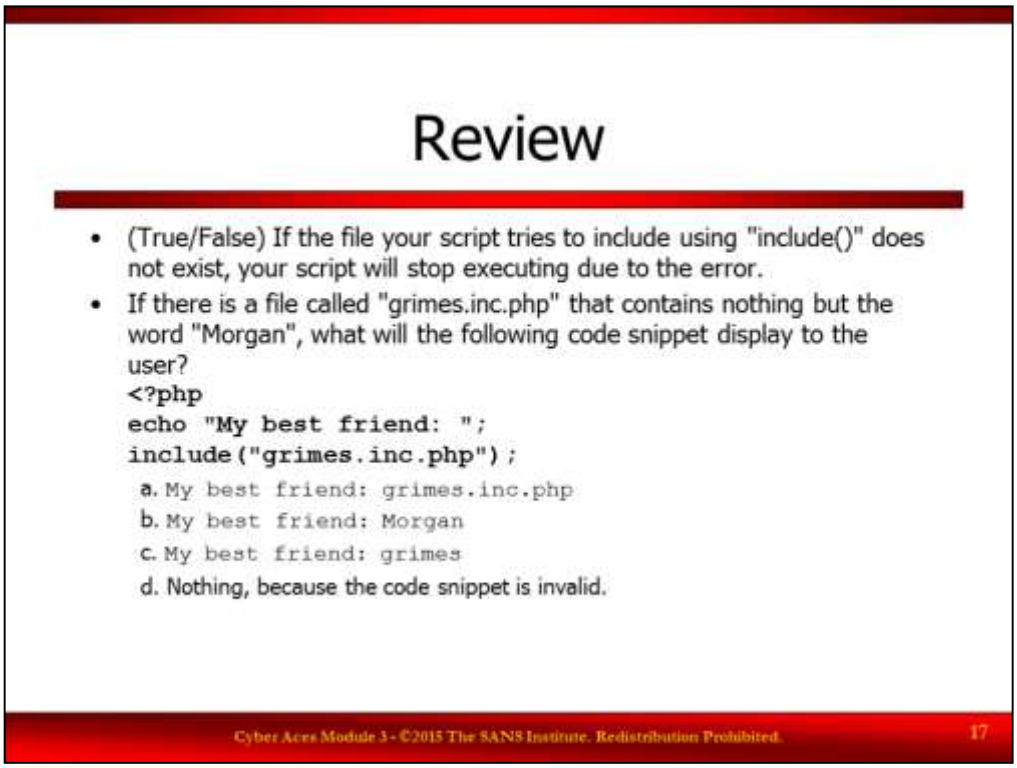
http://www.tizag.com/phpT/include.php

http://www.tizag.com/phpT/require.php



Include Example

This example has two separate files: owca.inc.php and include_example.php. owca.inc.php just sets the variable "`$agentP`" to the value "Perry". include_example.php includes owca.inc.php, and then echoes a string containing that variable. The Firefox screenshot shows what include_example.php looks like when it is executed. If owca.inc.php wasn't there, the code would have an error because that variable would not be defined.



Review

(True/False) If the file your script tries to include using "include()" does not exist, your script will stop executing due to the error.

If there is a file called "grimes.inc.php" that contains nothing but the word "Morgan", what will the following code snippet display to the user?

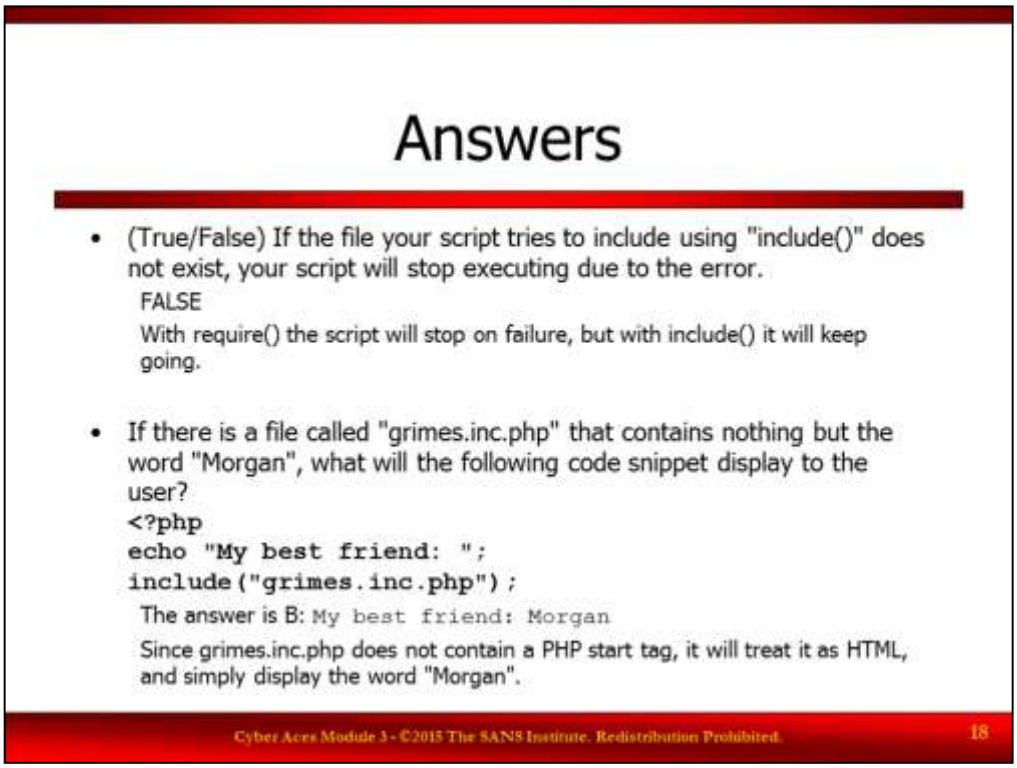
```
<?php
echo "My best friend: ";
include("grimes.inc.php");
```

a. `My best friend: grimes.inc.php`

b. `My best friend: Morgan`

c. `My best friend: grimes`

d. Nothing, because the code snippet is invalid.



Answers

(True/False) If the file your script tries to include using "include()" does not exist, your script will stop executing due to the error.

FALSE

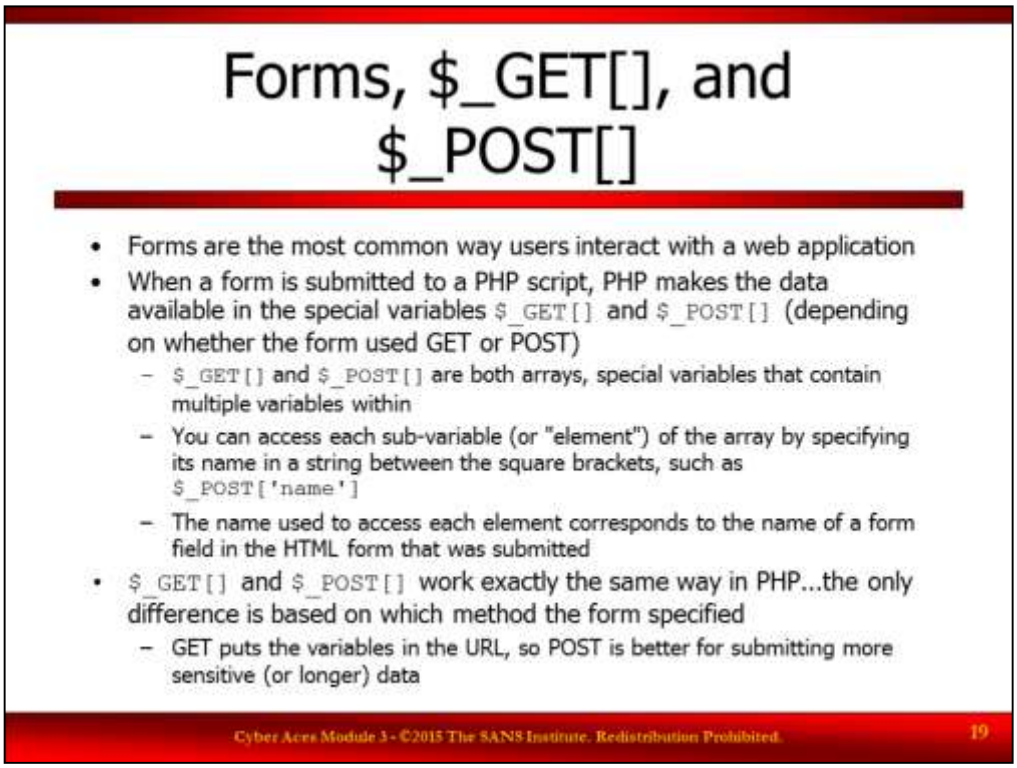
With require() the script will stop on failure, but with include() it will keep going.

If there is a file called "grimes.inc.php" that contains nothing but the word "Morgan", what will the following code snippet display to the user?

```
<?php
echo "My best friend: ";
include("grimes.inc.php");
```

The answer is B: `My best friend: Morgan`

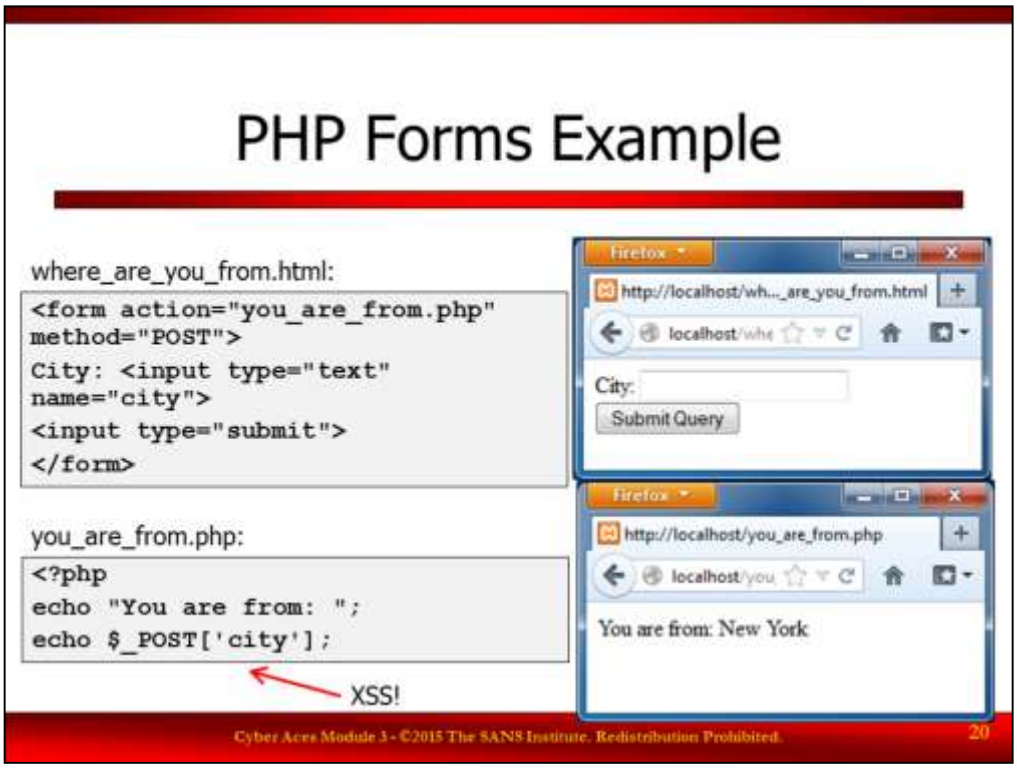
Since grimes.inc.php does not contain a PHP start tag, it will treat it as HTML, and simply display the word "Morgan".



Forms, $_GET[], and $_POST[]

The most common way that a user interacts with a web application is through a web form. When a form is submitted to a PHP script, PHP makes the data available through the special variables "`$_GET[]`" and "`$_POST[]`", depending on which method was used to submit the form. "`$_GET[]`" and "`$_POST[]`" are both arrays, a special type of variable that can hold multiple variables inside it. You can access each variable (or "element") of the array by specifying its name in a string between the square brackets, such as `$_POST['name']`. The name used to access each element corresponds to the name of a form field in the HTML form that was submitted. "`$_GET[]`" and "`$_POST[]`" both work exactly the same way; the only difference is in choosing which to use based on the HTTP method used to submit the form.

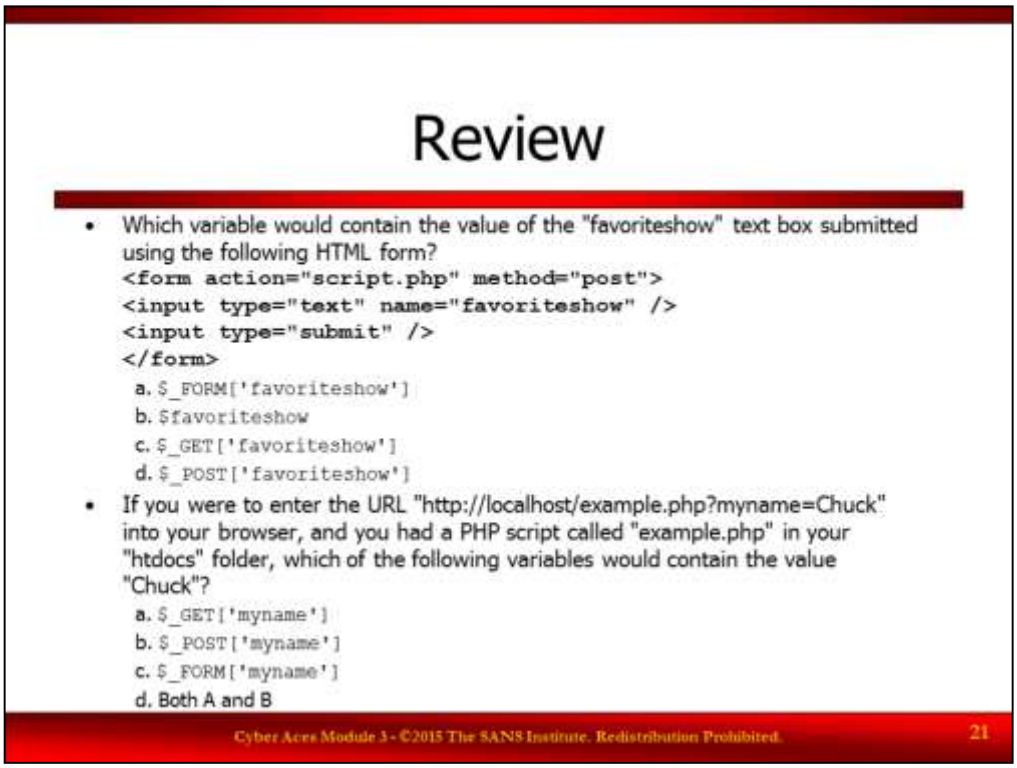
Further reading: http://www.tizag.com/phpT/forms.php



PHP Forms Example

This example has two files: an HTML file containing a web form, and a PHP file that processes its results. The form in where_are_you_from.html defines an action of "you_are_from.php", meaning that's where it submits its data. It defines "POST" as its method. It then defines a text box called "city", and a submit button. When a user fills out that text box and clicks the submit button, the data is sent to the PHP script at you_are_from.php. That script displays the text "You are from: ", and then displays the contents of the `$_POST['city']` variable (which contains the contents of the "city" textbox from the form). In the example above, a user submitted the form with the text "New York".

Note that the example above is vulnerable to a class of attack called Cross Site Scripting (XSS), because it does not filter or encode the user input before sending it back to the user. We will learn more about XSS in the Web Security section.



Which variable would contain the value of the "favoriteshow" text box submitted using the following HTML form?

```
<form action="script.php" method="post">
<input type="text" name="favoriteshow" />
<input type="submit" />
</form>
```

a. `$_FORM['favoriteshow']`

b. `$favoriteshow`

c. `$_GET['favoriteshow']`

d. `$_POST['favoriteshow']`

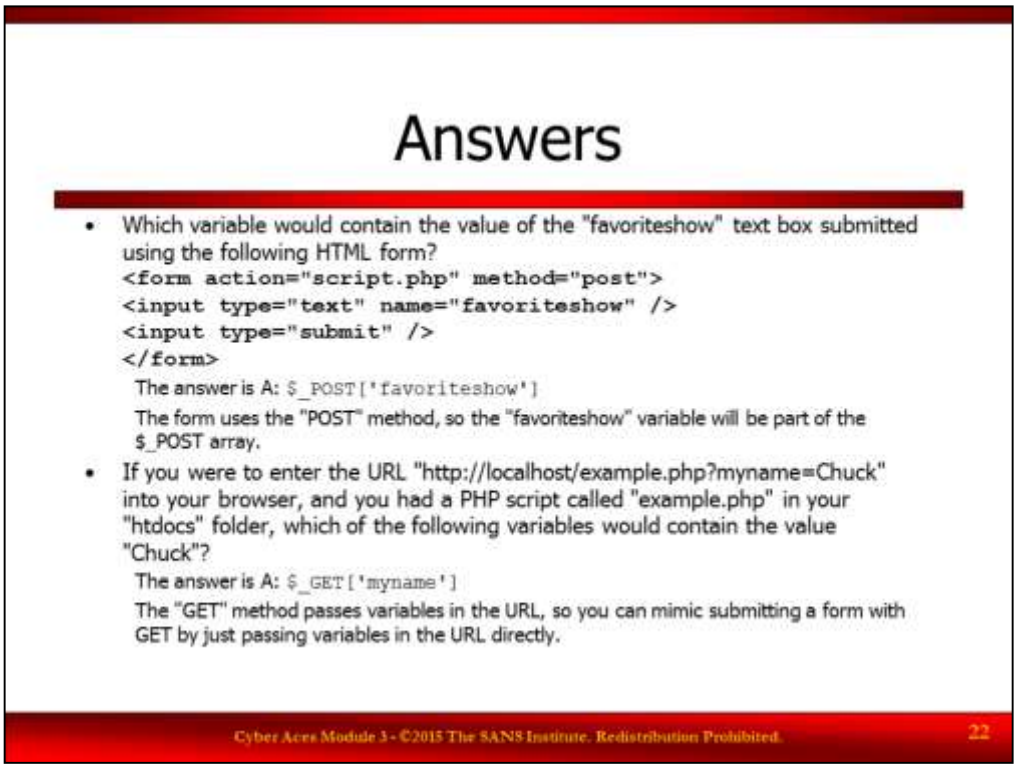
If you were to enter the URL "http://localhost/example.php?myname=Chuck" into your browser, and you had a PHP script called "example.php" in your "htdocs" folder, which of the following variables would contain the value "Chuck"?

a. `$_GET['myname']`

b. `$_POST['myname']`

c. `$_FORM['myname']`

d. Both A and B



Answers

Which variable would contain the value of the "favoriteshow" text box submitted using the following HTML form?

```
<form action="script.php" method="post">
<input type="text" name="favoriteshow" />
<input type="submit" />
</form>
```

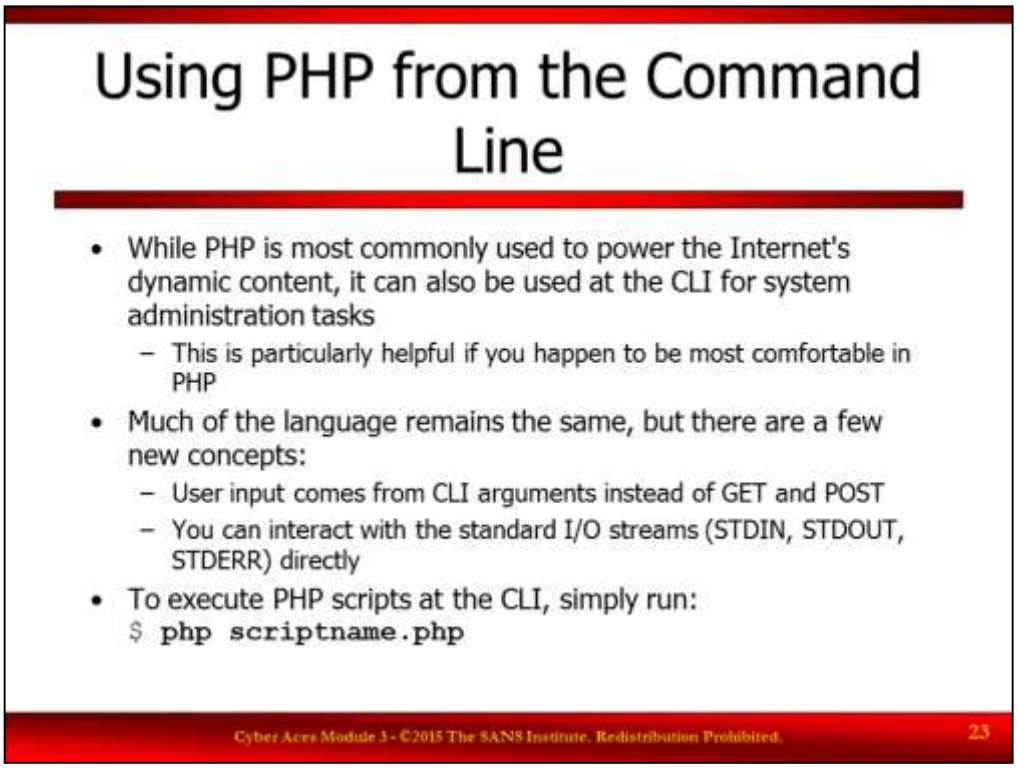
The answer is A: `$_POST['favoriteshow']`

The form uses the "POST" method, so the "favoriteshow" variable will be part of the $_POST array.

If you were to enter the URL "http://localhost/example.php?myname=Chuck" into your browser, and you had a PHP script called "example.php" in your "htdocs" folder, which of the following variables would contain the value "Chuck"?

The answer is A: `$_GET['myname']`

The "GET" method passes variables in the URL, so you can mimic submitting a form with GET by just passing variables in the URL directly.



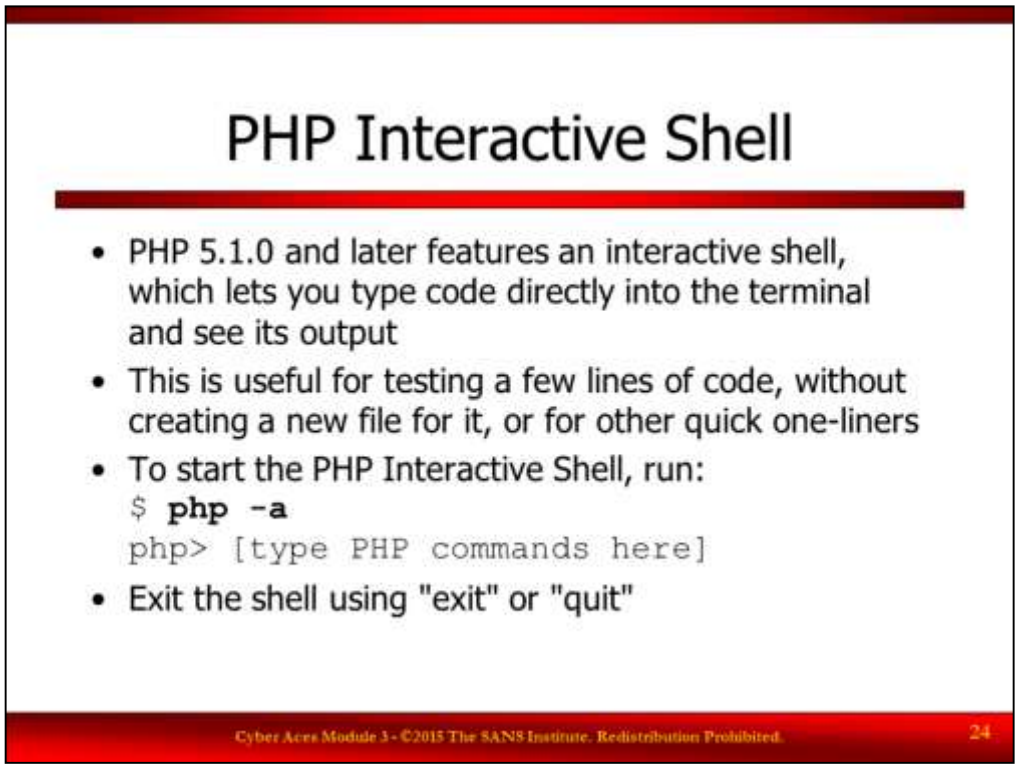
Using PHP from the Command Line

While PHP is the underlying language for much of the Internet's dynamic web content, it can also be used at the command line interface (CLI) for system administration-related tasks. Much of the language remains the same when using it at the command line, but a few new concepts come into play. For one, user input is taken using CLI arguments instead of GET and POST variables. Also, you are granted the ability to interface with the I/O standard streams STDIN (standard input), STDOUT (standard output), and STDERR (standard error), which lets you prompt the user for input while the program is running, providing "stateful" interaction.

To execute PHP scripts at the CLI, simply run:

```
$ php scriptname.php
```

The tutorial at http://www.developertutorials.com/tutorials/php/simple-system-maintenance-with-php-cli-8-01-17-950/ provides a broad overview of using PHP at the CLI and writing your first CLI script.

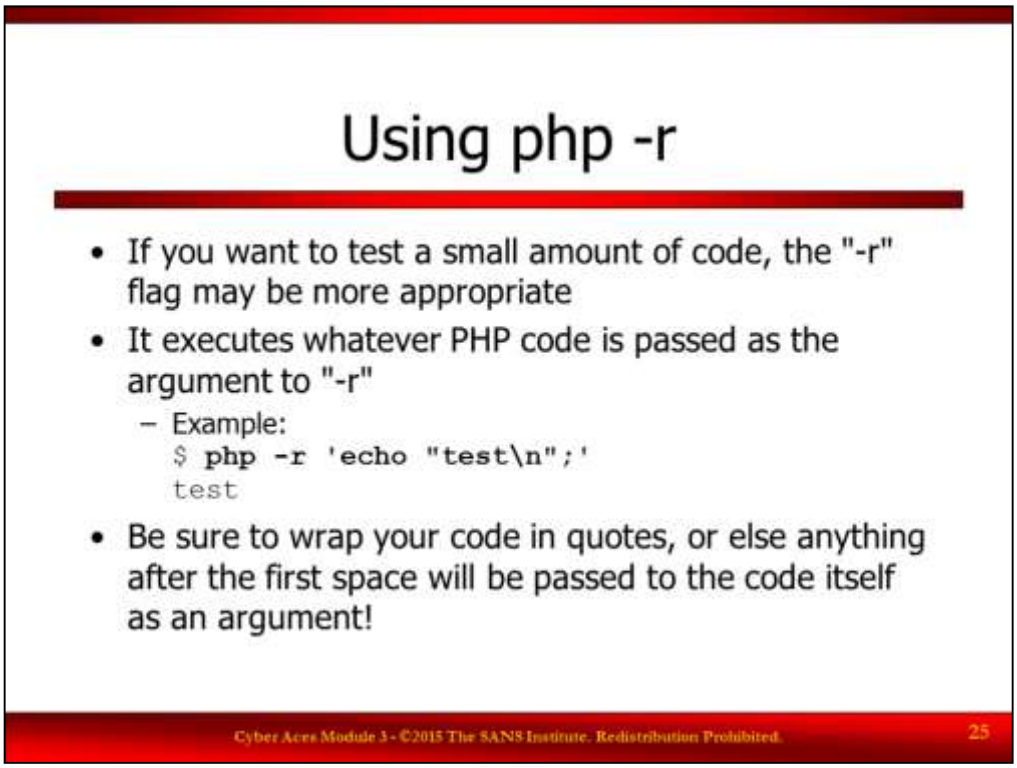


PHP Interactive Shell

Another neat feature of PHP at the CLI is the interactive shell. It is a fairly recent feature, so make sure you have version 5.1.0 or later if you want to use it! If you run "`php -a`", you are dropped into a `php>` shell, which lets you type code directly into the terminal and see the output of its execution. This is useful if you are just testing a few lines of code and don't want to create an entirely new file for it. Here is a very simple example of how to use it:

```
$ php -a
Interactive shell
php > echo 'testing';
testing
php > echo time();
1293835948
php >
```

You may exit from the PHP shell by typing "exit" or "quit".

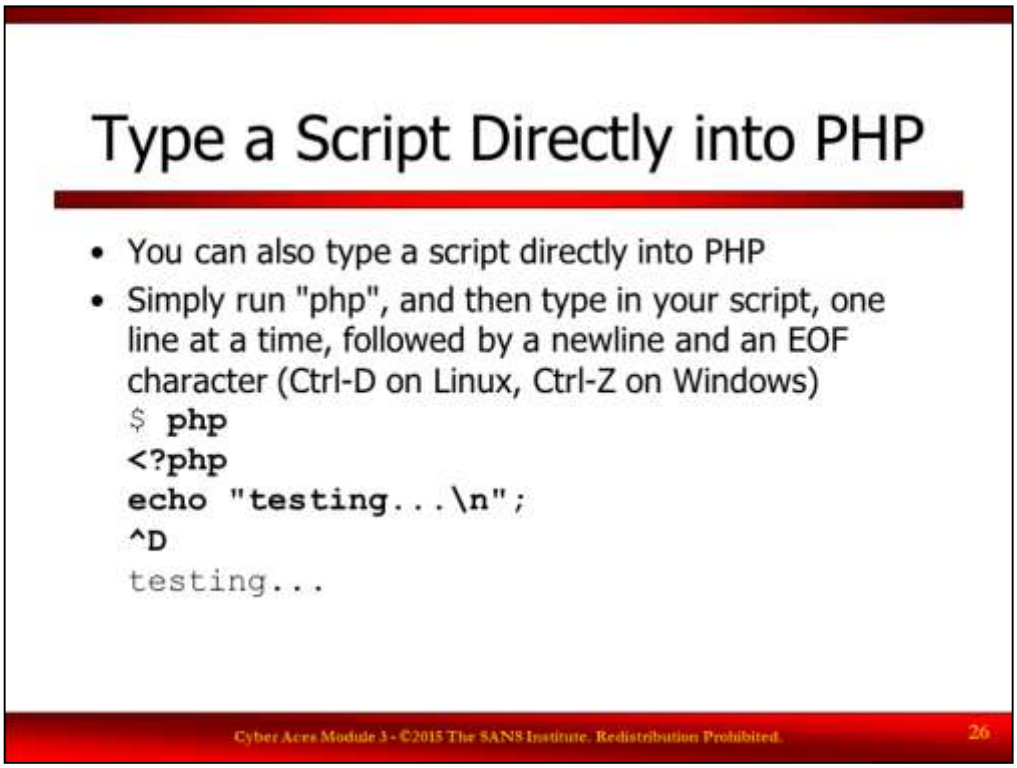


Using "php -r"

If you want to test a small amount of code without having to enter the PHP shell, then the "-r" flag may be more appropriate. It also lets you execute PHP code without having to create a new file for it:

```
$ php -r 'echo "test\n";'
test
```

Be sure to keep in mind that you need to wrap your code in quotes! If you don't wrap it in quotes, like if you were to do: **`php -r echo "test\n";`**, then PHP will think that `echo` is your code and pass `"test\n";` as the first argument.

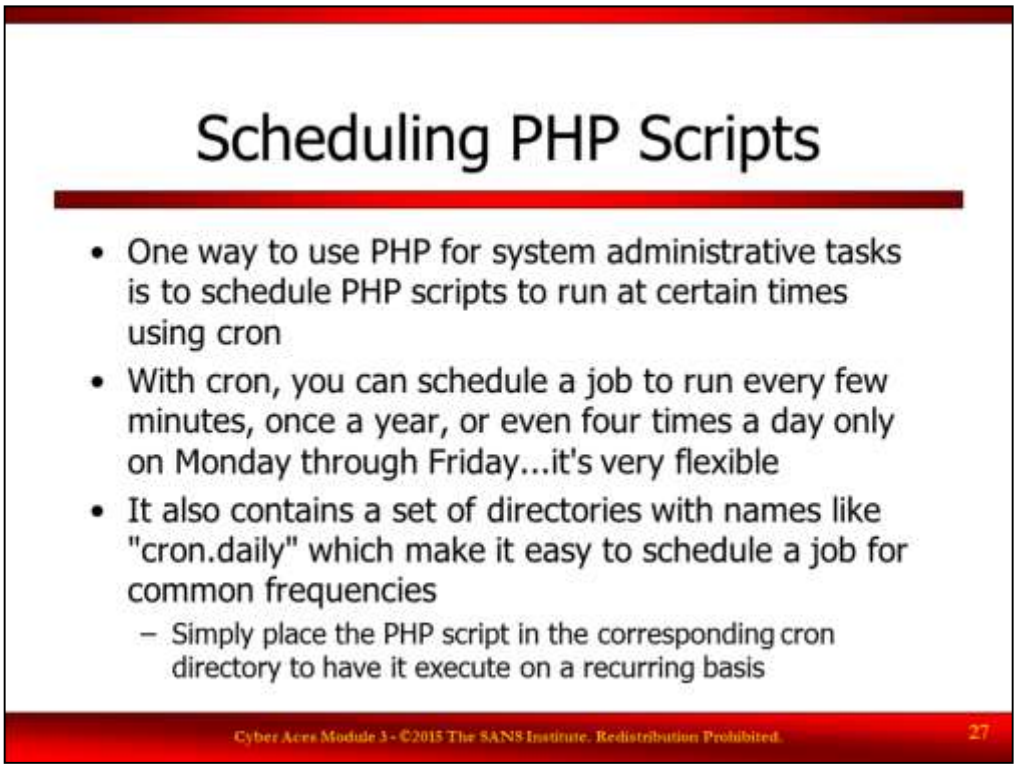


Type a Script Directly into PHP

Additionally, it is possible to execute PHP code directly from the command line without use of the "-a" flag, the "-r" flag, or any flag for that matter! By simply executing the command "php", the PHP interpreter lets you type code directly into the terminal without having to use the "`php>`" shell. Once you are done, input the EOF character by pressing Ctrl+D (or Ctrl+Z on Windows), and the PHP code that you entered will be run:

```
$ php
<?php
echo "testing...\n";
^D
Testing...
```

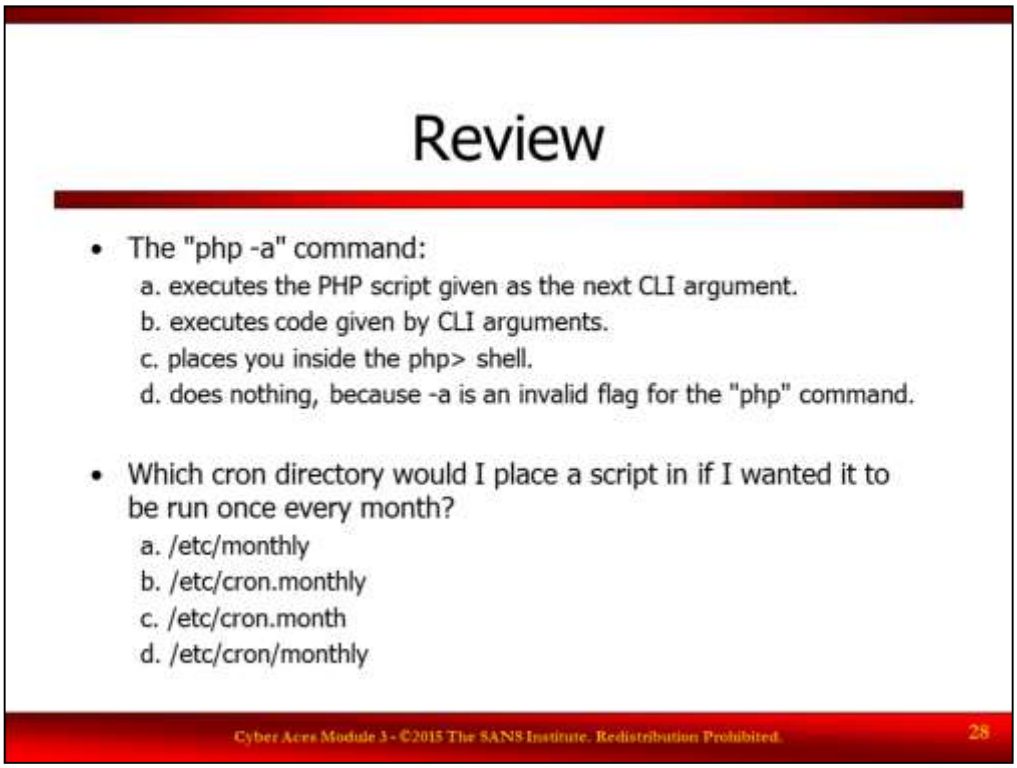
The PHP code you see above was typed in, and "`^D`" denotes where the EOF character was inputted. The text "testing..." is the output from the code. By knowing the behavior of "php", we can do all sorts of cool stuff with it. For instance, we now know that the command "`cat script.php | php`" acts the same as "`php script.php`". This works because "cat" would normally print the contents of script.php to the screen, but we redirect its output to "php" using the pipe "|", which executes it.



Scheduling PHP Scripts

Earlier, we mentioned that since you can run PHP scripts from the command line, you could use it for system administration-related tasks. If you wish to run a particular script on a regular, scheduled basis, then one particularly useful tool to look at is "cron". By setting up a "cronjob" (as they are nicknamed), sysadmins can tell a script to run every few minutes, once a year, or even four times a day only on Monday through Friday. The tutorial at http://www.ehow.com/how_5106582_use-cron.html demonstrates simple scheduling using cron and offers a great starting point for how to use the tool effectively.

Cron also offers a quick and easy way to schedule tasks through the use of the "cron.hourly", "cron.daily", "cron.weekly", and "cron.monthly" directories located in /etc. Scripts that are placed in these directories are run once an hour, once a day, once a week, or once a month, respectively. A more in-depth guide to using cron is available at http://www.unixgeeks.org/security/newbie/unix/cron-1.html. Although the guide was written in 1999, the core functionality of cron has changed very little over that time period.



Review

The "php -a" command:

a. executes the PHP script given as the next CLI argument.

b. executes code given by CLI arguments.

c. places you inside the php> shell.

d. does nothing, because -a is an invalid flag for the "php" command.

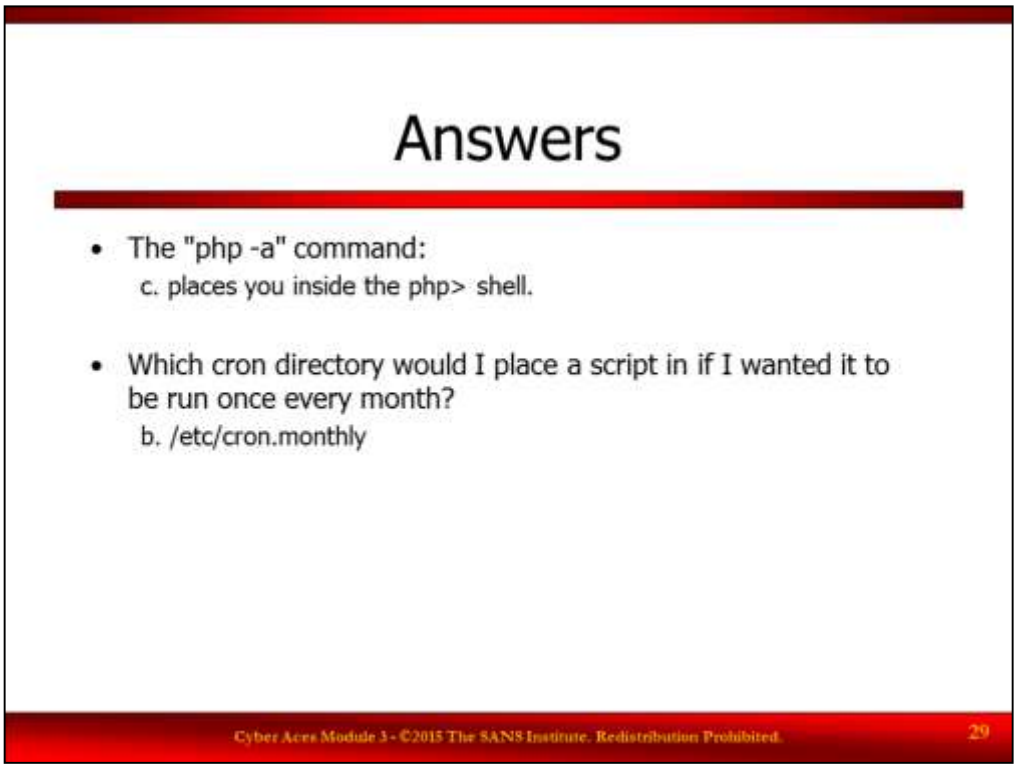
Which cron directory would I place a script in if I wanted it to be run once every month?

a. /etc/monthly

b. /etc/cron.monthly

c. /etc/cron.month

d. /etc/cron/monthly



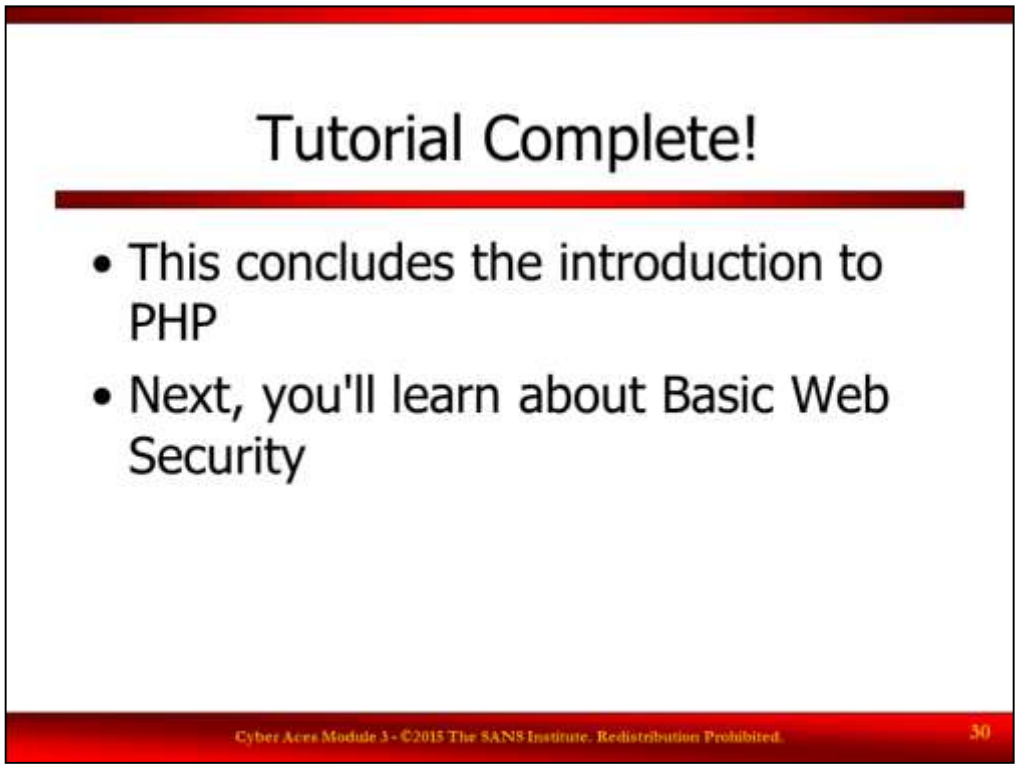
Answers

The "php -a" command:

c. places you inside the php> shell.

Which cron directory would I place a script in if I wanted it to be run once every month?

b. /etc/cron.monthly



This concludes the first section of Module 3. We've learned the basics of PHP. Next, you'll learn about basic web security.