CSC 360 — Lab Assignment #8
Spring 2015 — Due: April 3, 2015

The following are the end-of-chapter Labs from Chapter 19 through Chapter 24 of our textbook. Answer each question or describe the PowerShell command(s) necessary to accomplish the task described.

Chapter 19: *Input and output*

NOTE: For this lab, you'll need any computer running PowerShell v3.

1. Use Write-Output to display the result of 100 multiplied by 10.

2. Use Write-Host to display the result of 100 multiplied by 10.

3. Prompt the user to enter a name, and then display that name in yellow text.

4. Prompt the user to enter a name, and then display that name only if it's longer than 5 characters. Do this all in a single line—don't use a variable.

Chapter 20: *Sessions: remote control with less work*

NOTE: For this lab, you'll need a Windows Server 2008 R2 or Windows Server 2012 computer running PowerShell v3. If you've access only to a client computer (running Windows 7 or Windows 8), you won't be able to complete steps 6 through 9 of this lab.

To complete this lab, you'll want to have two computers: one to remote from, and another to remote to. If you only have one computer, use its computer name to remote to it. You should get a similar experience that way.

1. Close all open sessions in your shell.

2. Establish a session to a remote computer. Save the session in a variable named `$session`.

3. Use the `$session` variable to establish a one-to-one remote shell session with the remote computer. Display a list of processes, and then exit.

4. Use the `$session` variable with `Invoke-Command` to get a list of services from the remote computer.

5. Use `Get-PSSession` and `Invoke-Command` to get a list of the 20 most recent Security event log entries from the remote computer.

6. Use `Invoke-Command` and your `$session` variable to load the `ServerManager` module on the remote computer.

7. Import the `ServerManager` module's commands from the remote computer to your computer. Add the prefix "rem" to the imported commands' nouns.

8. Run the imported `Get-WindowsFeature` command.

9. Close the session that's in your `$session` variable.

NOTE: Thanks to a new feature in PowerShell v3, you could also accomplish steps 6 and 7 with a single step, by using the Import-Module command. Feel free to review the help for this command and see if you can figure out how to use it to import a module from a remote computer.

Chapter 21: *You call this scripting?*

NOTE: For this lab, you'll need any computer running PowerShell v3.

The following command is for you to add to a script. You should first identify any elements that should be parameterized, such as the computer name. Your final script should define the parameter, and you should create comment-based help within the script. Run your script to test it, and use the Help command to make sure your comment-based help works properly. Don't forget to read the help files referenced within this chapter for more information.

Here's the command:

```
Get-WmiObject Win32_LogicalDisk -comp "localhost" -filter "drivetype=3" |
Where { $_.FreeSpace / $_.Size -lt .1 } |
Select -Property DeviceID,FreeSpace,Size
```

Here's a hint: There are at least two pieces of information that will need to be parameterized. This command is intended to list all drives that have less than a given amount of free disk space. Obviously, you won't always want to target localhost, and you might not want 10% (that is, .1) to be your free space threshold. You could also choose to parameterize the drive type (which is 3, here), but for this lab leave that hardcoded with the value 3.

Chapter 22: *Improving your parameterized script*

NOTE: For this lab, you'll need any computer running PowerShell v3.

This lab is going to require you to recall some of what you learned in chapter 21, because you'll be taking the following command, parameterizing it, and turning it into a script—just like you did for the lab in chapter 21. But this time we also want you to make the -computerName parameter mandatory and give it a hostname alias. Have your script display verbose output before and after it runs this command, too. Remember, you have to parameterize the computer name—but that's the only thing you have to parameterize in this case.

Be sure to run the command as-is before you start modifying it, to make sure it works on your system.

```
get-wmiobject win32_networkadapter -computername localhost |
where { $_.PhysicalAdapter } |
select MACAddress,AdapterType,DeviceID,Name,Speed
```

To reiterate, here's your complete task list:

- Make sure the command runs as-is before modifying it.
- Parameterize the computer name.
- Make the computer name parameter mandatory.
- Give the computer name parameter an alias, hostname.
- Add comment-based help with at least one example of how to use the script.
- Add verbose output before and after the modified command.
- Save the script as Get-PhysicalAdapters.ps1.

Attach a printed copy of your completed script and sample output.

Chapter 23: *Advanced remoting configuration*

> NOTE: For this lab, you'll need a Windows 8 or Windows Server 2012 computer running PowerShell v3.

Create a remoting endpoint named TestPoint on your local computer. Configure the endpoint so that the SmbShare module is loaded automatically, but so that only the Get-SmbShare cmdlet is visible from that module. Also ensure that key cmdlets like Exit-PSSession are available, but no other core PowerShell cmdlets can be used. Don't worry about specifying special endpoint permissions or designating a "run as" credential.

Test your endpoint by connecting to it using Enter-PSSession (specify localhost as the computer name, and TestPoint as the configuration name). When connected, run Get-Command to ensure that only the designated handful of commands can be seen.

Note that this lab might only be possible on Windows 8, Windows Server 2012, and later versions of Windows—the SmbShare module didn't ship with earlier versions of Windows.

Chapter 24: *Using regular expressions to parse text files*

> NOTE: For this lab, you'll need any computer running PowerShell v3.

Make no mistake about it, regular expressions can make your head spin, so don't try to create complex regexes right off the bat—start simple. Here are a few exercises to ease you into it. Use regular expressions and operators to complete the following:

1. Get all files in your Windows directory that have a two-digit number as part of the name.

2. Find all processes running on your computer that are from Microsoft, and display the process ID, name, and company name. Hint: pipe Get-Process to Get-Member to discover property names.

3. In the Windows Update log, usually found in C:\Windows, you want to display only the lines where the agent began installing files. You may need to open the file in Notepad to figure out what string you need to select.