Outline

In this lab, we'll discuss how PowerShell finds modules automatically and how to create a PowerShell profile.

Exercise 1 - Modify PSModulePath

PowerShell stores its search directories as an environment variable called `PSModulePath` with each path separated by a semicolon. Let's launch a new PowerShell session and take a look.

```
# Raw variable
$ENV:PSModulePath

# Variable split into individual paths
$ENV:PSModulePath.Split(';')
```

In order to add a path, all we have to do is append the variable with a semicolon and whichever directory we want to search. Let's use the directory for this lab as an example.

```
# Before running this, make sure your PowerShell working directory is the same
folder as the file you're currently reading!

#Search for modules containing "Shakespeare" in the name; no modules will be
returned
Get-Module -ListAvailable *Shakespeare*

# Modify PSModulePath to add our present working directory
$ENV:PSModulePath = $ENV:PSModulePath + ';' + $PWD

# Confirm current directory was added
$ENV:PSModulePath.Split(';')

#Search for modules containing "Shakespeare" in the name; our lab module should now
be returned
Get-Module -ListAvailable *Shakespeare*
```

Now we don't have to specify a path to import our module, we can simply specify the name when we run `Import-Module`. We can also run any of the functions in the module without having to manually import since PowerShell is already aware of the module and it's contents; this behavior is controlled by the $PSModuleAutoLoadingPreference automatic variable. The changes we've made to `PSModulePath` aren't permanent, though; if we close the window or open a new session, the additional path we provided is disposed.

In order to make the changes permanent, we can permanently modify the environment variable, but it's generally easier (and more portable) to use a PowerShell Profile.

If you still decide to modify your environment variable, you can do so with the examples below. Use with caution!

```
# !!! USE AT YOUR OWN RISK !!!
# These commands are intentionally commented out to prevent accidental execution;
supply your own values before proceeding!

# Modify PSModulePath for system scope
# [System.Environment]::SetEnvironmentVariable('PSModulePath', 'ENTER YOUR VALUES
HERE', 'Machine')

# Modify PSModulePath for user scope
# [System.Environment]::SetEnvironmentVariable('PSModulePath', 'ENTER YOUR VALUES
HERE', 'User')
```

Exercise 2 - Create a PowerShell Profile

The preferred way to add a directory to your `PSModulePath` variable is to create a PowerShell profile. A PowerShell profile is just a script that executes every time you launch a PowerShell session, and can contain anything you want, from functions to messages of the day. The location of your profile is stored as an automatic variable called `$PROFILE`. Let's open it in Notepad and add make some modifications. If the file doesn't already exist, you'll be prompted to created it.

```
notepad.exe $PROFILE
```

Now we can add our module directory to the profile, save and close.

```
# You'll have to modify this line with your own local path to the lab
$ENV:PSModulePath = $ENV:PSModulePath + ';' + "ENTER THE PATH TO THE LAB HERE"
```

Now every time we open PowerShell, our profile will update our `PSModulePath` for us automatically. That said, none of your currently open PowerShell sessions will automatically update after you make changes to your profile; you'll have to run the following if you wanted to do so:

```
. $PROFILE
```

When specifying a PowerShell module directory to search, you should specify the folder **one level above** the folder containing your module. So your directory structure should look like this:

```
Add_This_Path_As_Your_Search_Directory
  ∟ Module1
    ∟ en-US
    ∟ Private\
    ∟ Public\
    ∟ Module1.psd1
    ∟ Module1.psm1
  ∟ Module2
    ∟ en-US
    ∟ Private\
    ∟ Public\
    ∟ Module2.psd1
    ∟ Module2.psm1
```

A few things to note:

- PowerShell doesn't recursively search folders, so if you have modules that are several levels deeper in your folder structure, you'll have to manually specify them as well unless they're automatically loaded by another module.
- You can have as many modules under a directory as you like, so long as they each have a folder matching the actual name of the module.
- UNC paths are supported

The modifications we've made to our profile are only for our current user and host; there are actually several other profiles you can modify, but that falls outside the scope of this exercise. You can read the official documentation here and a great Microsoft blog entry on each of the different profiles here.