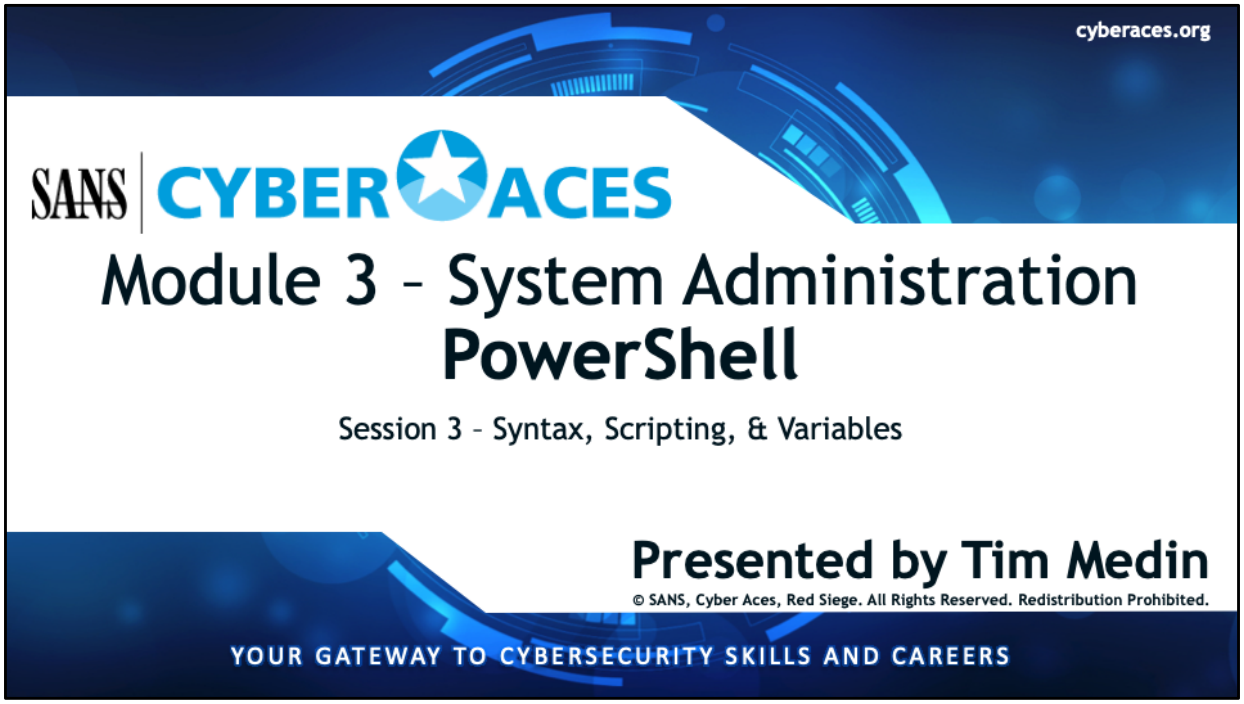
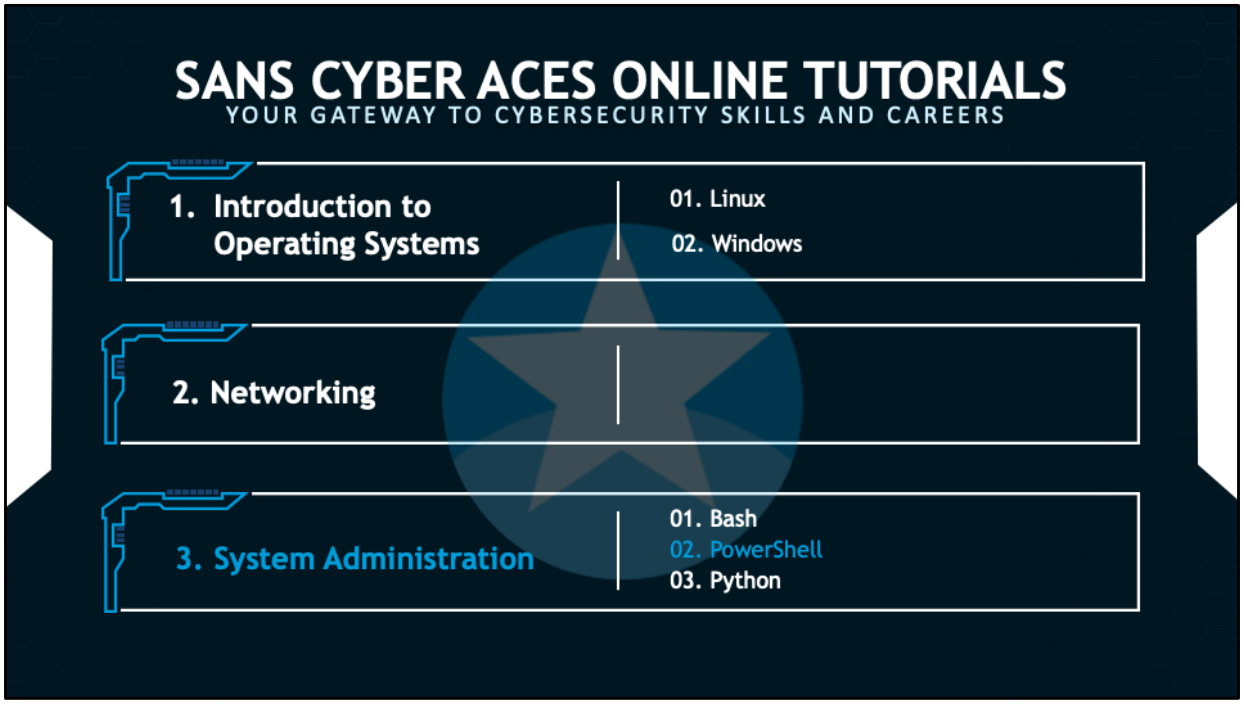


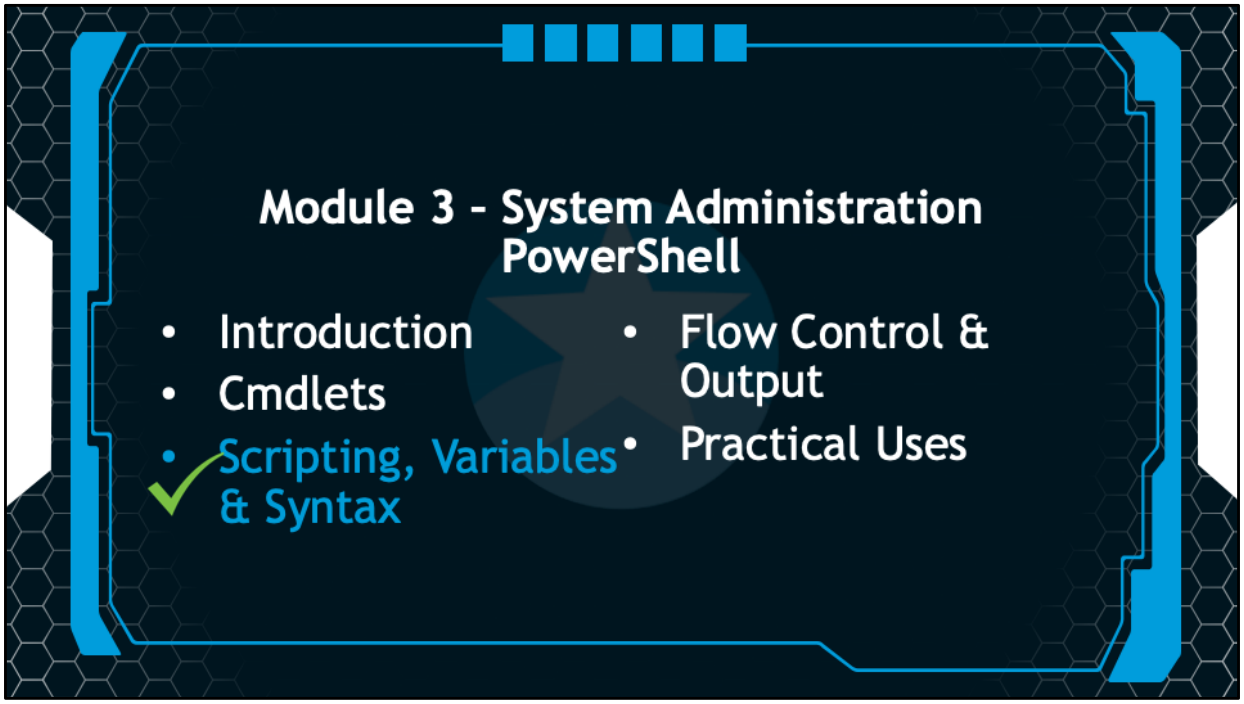
Welcome to Cyber Aces, Module 3! This module provides an introduction to the latest shell for Windows, PowerShell. In this session we'll discuss additional syntax as well as scripting and variables.



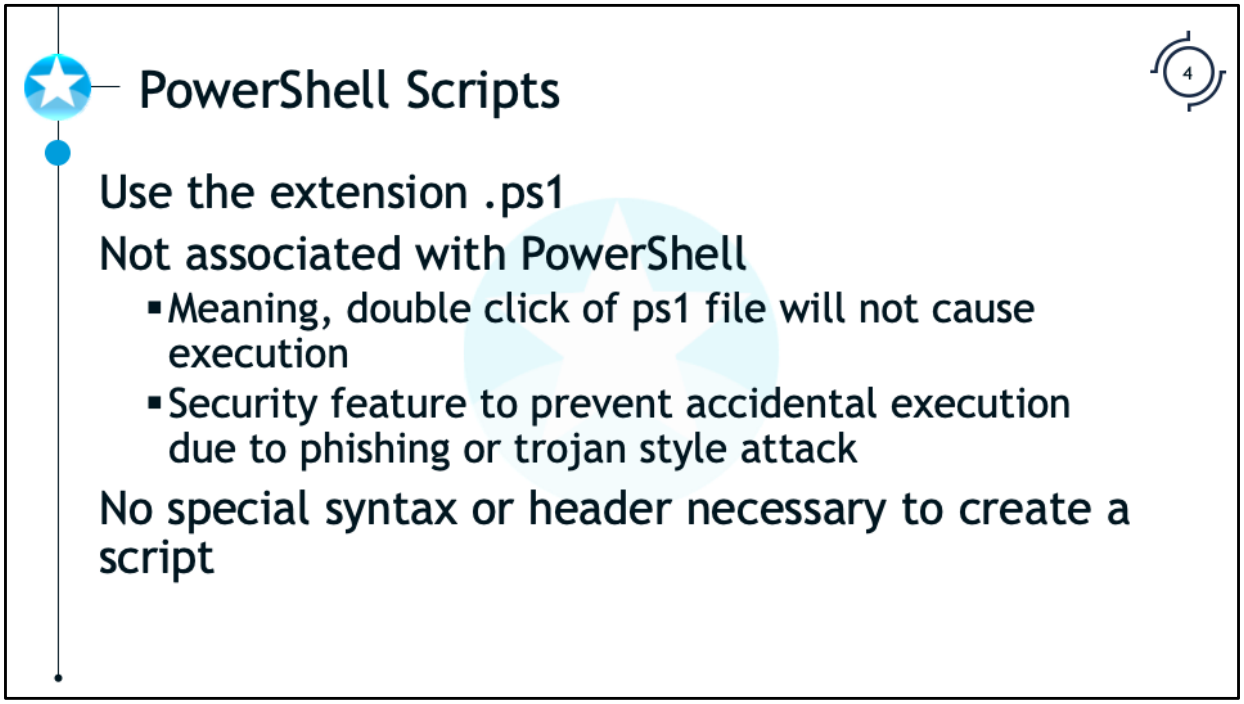
This training material was originally developed to help students, teachers, and mentors prepare for the Cyber Aces Online Competition. This module focuses on the basics of what an operating systems is as well as the two predominant OS's, Windows and Linux. In this session we will provide a walkthrough of the installation a Windows VM using VMware Fusion (MacOS) and VMware Player (Windows & Linux). These sessions include hands-on labs, but before we begin those labs we need to install the operating systems used in those labs. We will be using VMware to virtualize these operating systems. You can use other virtualization technologies if you like, but instruction for their setup and use are not included in this training.

The three modules of Cyber Aces Online are Operating Systems, Networking, and System Administration.

For more information about the Cyber Aces program, please visit the Cyber Aces website at https://CyberAces.org/.



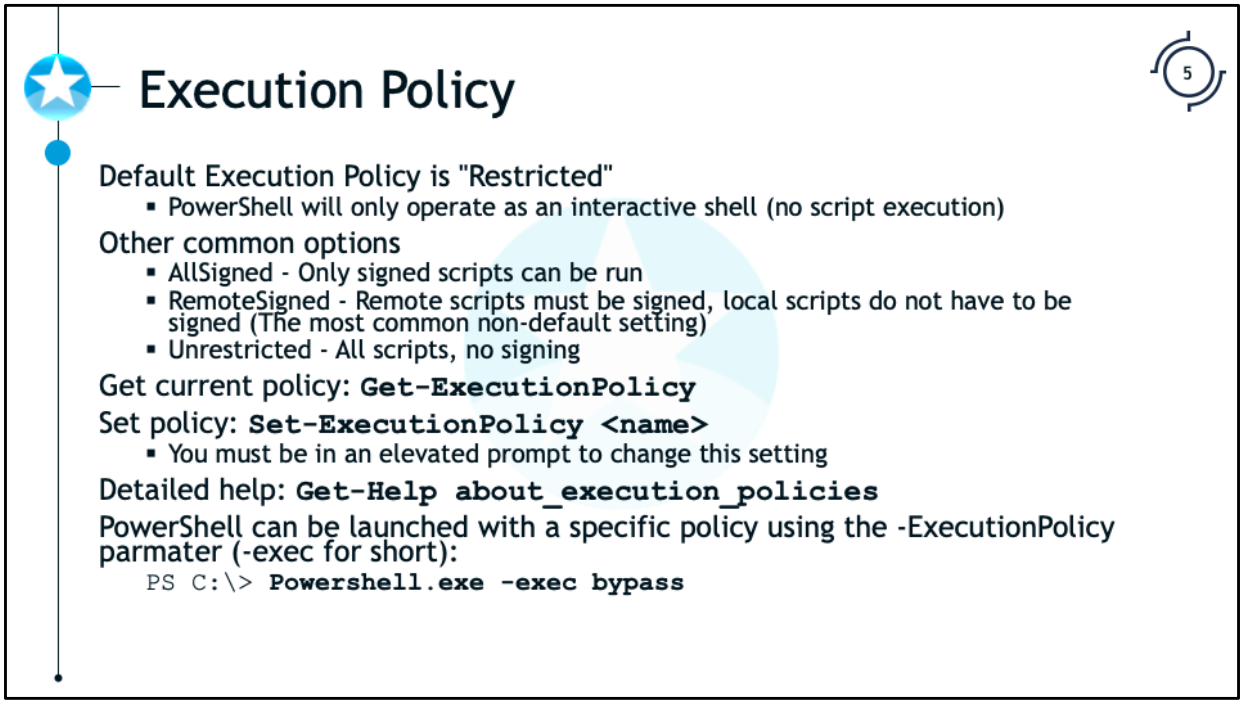
Is this section, you will be introduced to PowerShell and some basic syntax.



As with any shell, you can write scripts to automate common tasks, and this can make life a lot easier. Scripts can make boring and repetitive tasks much easier and quicker. Why ever do the same thing twice?

These scripts have the extension ".ps1", and do not require any special headers. The syntax of variables and commands in scripts is the same as that used on the command line. However, there are a few security features surrounding the execution of these script files.

The first security feature is, by default, double clicking on a ".ps1" file will not execute the script, but rather will open it in a text editor. This prevents the inadvertent execution of script files. To manually execute a script, it must be run from the command line.



The second security feature is that, by default, no scripts can be run. The default "ExecutionPolicy" is "Restricted." In this mode, PowerShell only operates as an interactive shell.

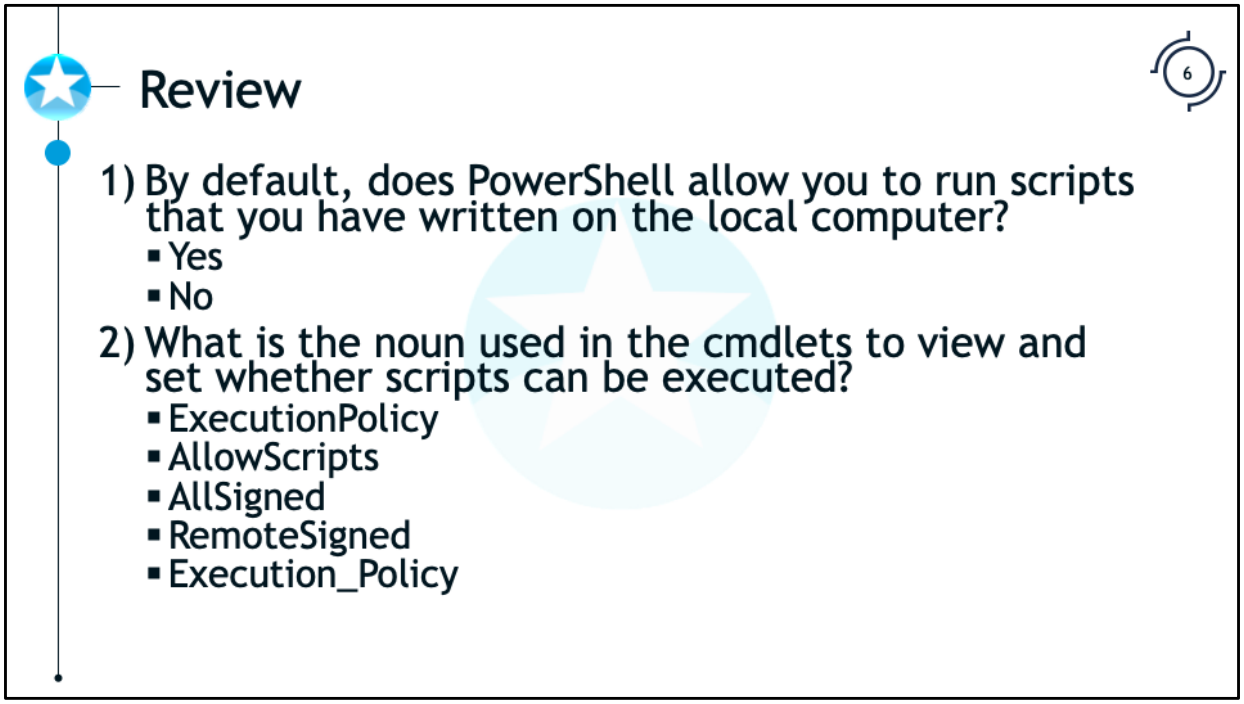
If you need to run scripts, the most secure setting is "AllSigned." With this setting, scripts can run, but all scripts and configuration files must be signed by a trusted publisher. Even scripts written on the local computer must be signed, and that can make writing and debugging scripts difficult. Because this setting can be a pain, the most common setting is "RemoteSigned." It is the same as "AllSigned," except locally written scripts do not have to be signed. With "RemoteSigned", any scripts or configuration files downloaded from the Internet, e-mail, or IM still must be signed. Use the "Set-ExecutionPolicy" cmdlet to change this setting.

```
PS C:\> Set-ExecutionPolicy AllSigned
PS C:\> Set-ExecutionPolicy RemoteSigned
```

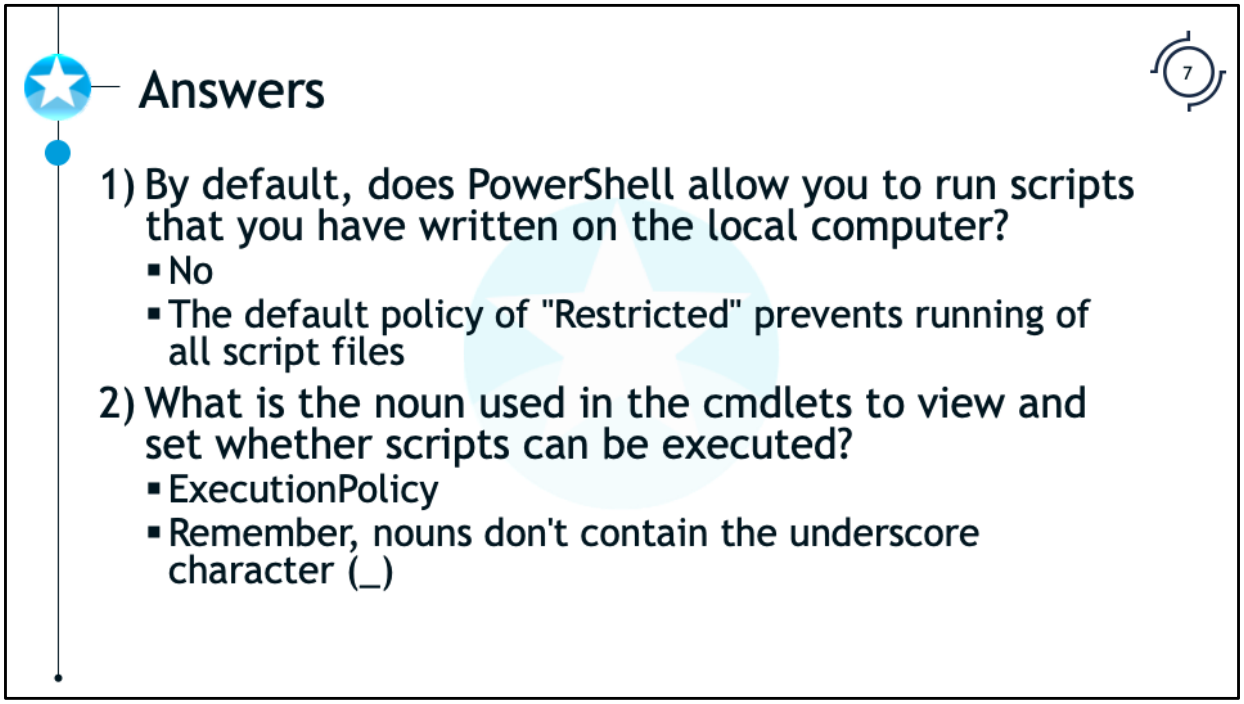
These commands can be run to change the execution policy to "AllSigned" or "RemoteSigned" respectively. Please note that you may need to run PowerShell with elevated permissions to use the "Set-ExecutionPolicy" cmdlet!

PowerShell can be executed with a specific policy using the -ExecutionPolicy parameter (-exec for short):

```
PS C:\> powershell.exe -exec bypass
```



1) By default, does PowerShell allow you to run scripts that you have written on the local computer?
   - Yes
   - No
2) What is the noun used in the cmdlets to view and set whether scripts can be executed?
   - ExecutionPolicy
   - AllowScripts
   - AllSigned
   - RemoteSigned
   - Execution_Policy



1) By default, does PowerShell allow you to run scripts that you have written on the local computer?

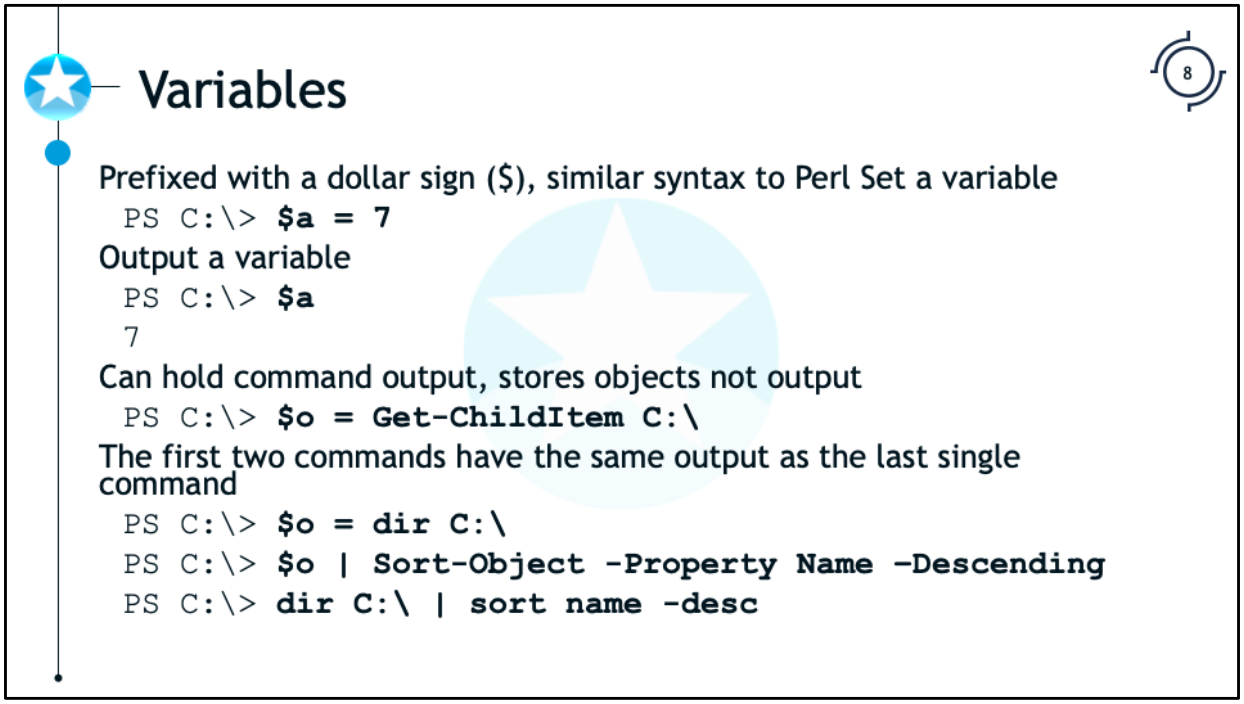
   No

   The default policy of "Restricted" prevents running of all script files

2) What is the noun used in the cmdlets to view and set whether scripts can be executed?

   ExecutionPolicy

   Remember, nouns don't contain the underscore character (_)



Variables are useful for storing data that you want to use later. Variables in PowerShell are preceded by a dollar sign ("$"), so we could use the following to store the number 7 in the variable "$a":

```
PS C:\> $a = 7
```

We can than output the variable "$a" just by typing it on the command line.

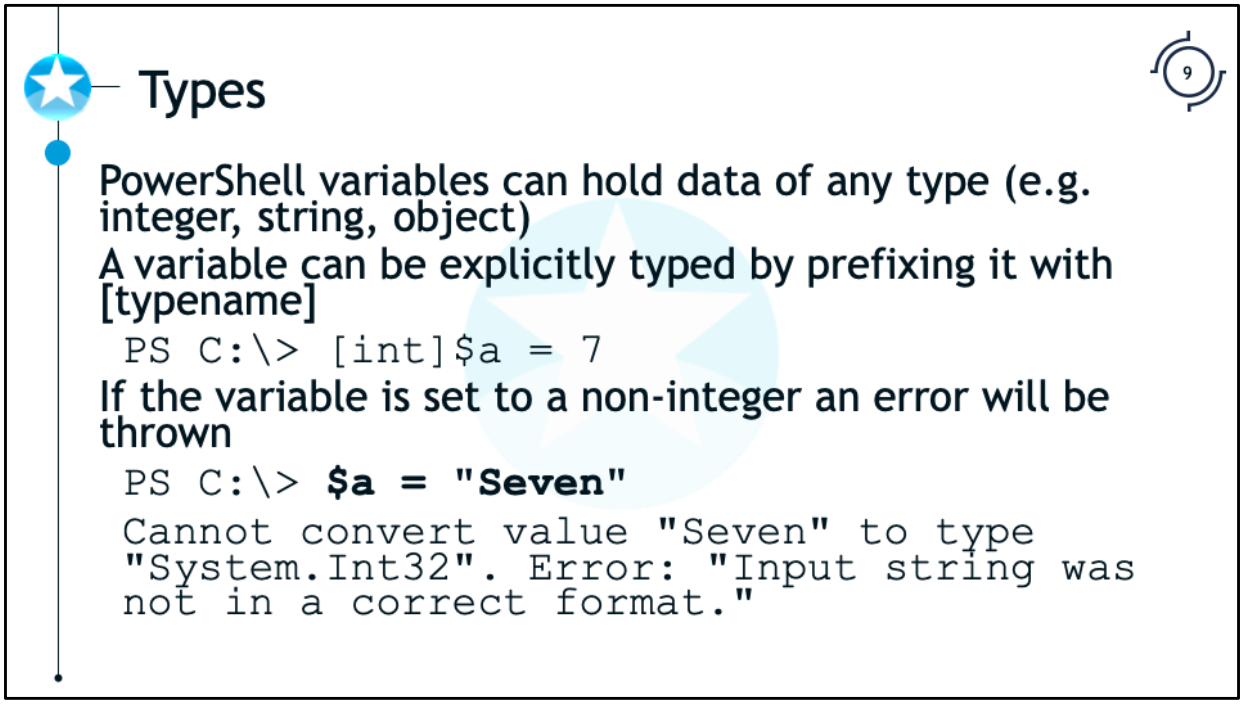
```
PS C:\> $a
7
```

Variables can store collections of objects, such as the output of a directory listing.

```
PS C:\> $o = Get-ChildItem
PS C:\> $o

Directory: C:\

Mode        LastWriteTime         Length Name
d-r--       1/2/2011 1:27 PM      Program Files
d-r--       12/8/2010 8:56 AM     Users
d----       12/31/2010 11:50 AM   Windows
-a---       6/10/2009 4:42 PM     24 autoexec.bat
-a---       6/10/2009 4:42 PM     10 config.sys
```



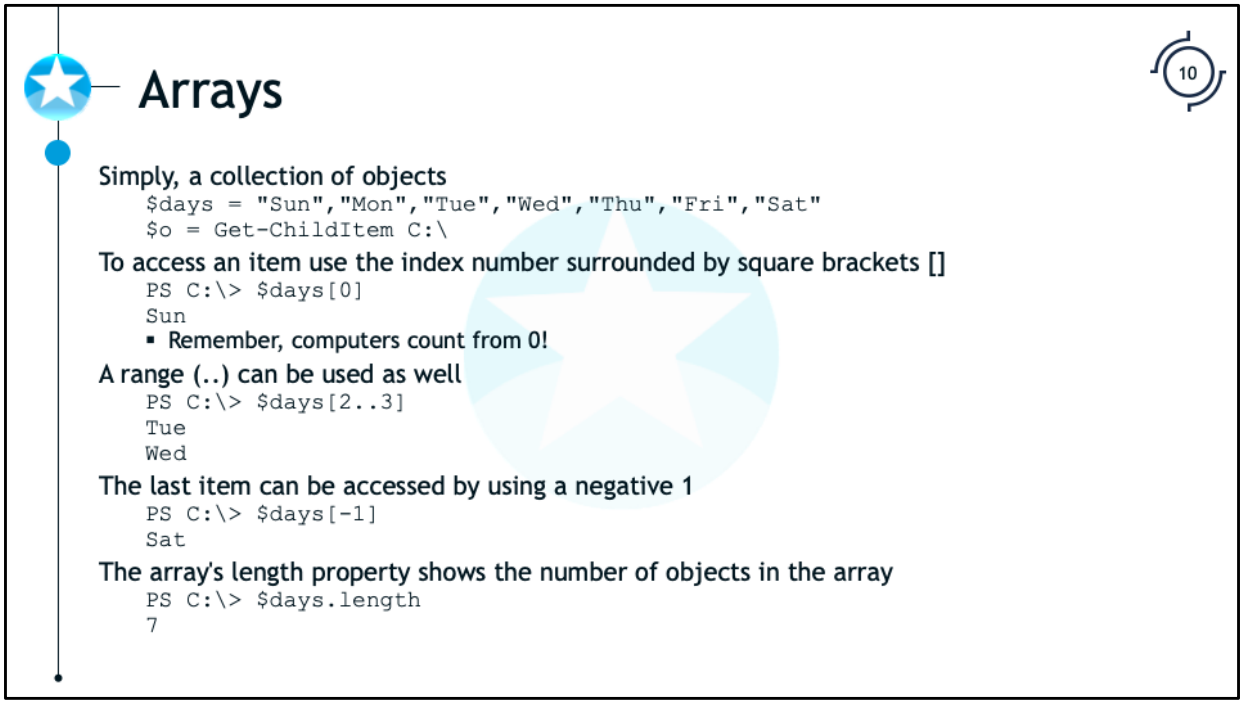
We can also explicitly cast a variable to be of a certain type by using "[type]" so that it will only store values of the given type. Here "$a" is declared as an integer and set to "7". Let's see what happens when "$a" is set to an invalid value.

```
PS C:\> [int]$a = 7

PS C:\> $a = "Seven"

Cannot convert value "Seven" to type "System.Int32".
Error: "Input string was not in a correct format."
```

PowerShell throws an error since the string "Seven" is not an integer.



The most common mistake with arrays is the "Off-by-One" error, when the programmer forgets that the index is base 0 and accesses the wrong item in an array or attempts to access the last item in an array and uses a non-existent index number.

Arrays are just a collection of objects. They can be created manually:

```
PS C:\> $days= "Sun","Mon","Tue","Wed","Thu","Fri","Sat"
```

...or they can be from the output of another command:

```
PS C:\> $files = Get-ChildItem C:\
```

Both the "$days" and "$files" variables are arrays. We can access an item in the array using square brackets ([]). PowerShell arrays are base zero, meaning the first item is 0.

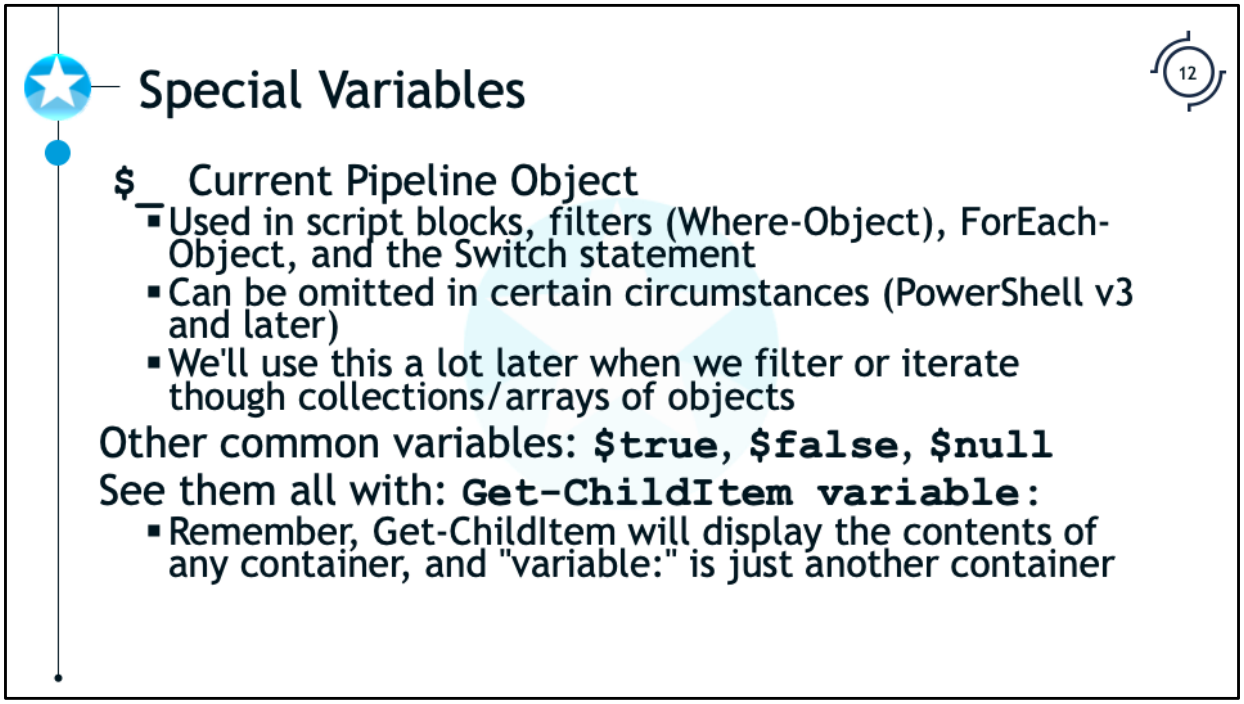
```
PS C:\> $days[0]
Sun
```

Multiple items can be accessed like this:

```
PS C:\> $days[1..3]
Mon
Tue
Wed
PS C:\> $days[2,4,6]
Tue
Thu
Sat
```

The last item in an array can be accessed by counting backwards:

```
PS C:\> $days[-1]
Sat
```



The Current Pipeline object ($_) is used a lot in PowerShell. It is used when iterating over a number of objects or for filtering with the Where-Object cmdlet. We'll use this variable quite a bit towards the end of this module. The use of this variable is not required in PowerShell version 3 which ships with Windows 8 and later.

In PowerShell v1 and v2, the Where-Object command is used similar to this:

```
PS C:\> Get-Process | where {$_.CPU -gt 25}
```

…but in v3 and later it can be shortened to this:

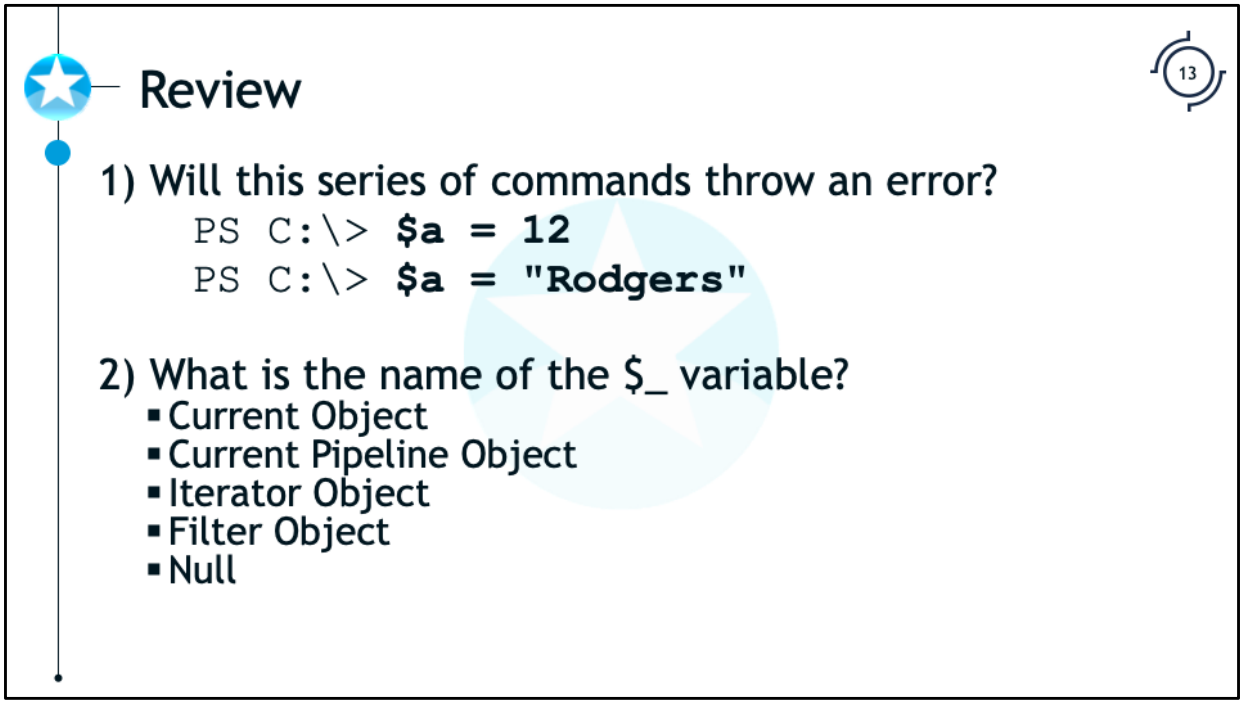
```
PS C:\> Get-Process | where CPU -gt 25
```

The Get-ChildItem (aliases of ls or dir) can be used to view the variables currently in use:

```
PS C:\> Get-ChildItem variable:
```

```
PS C:\> dir variable:
```

```
PS C:\> ls variable:
```

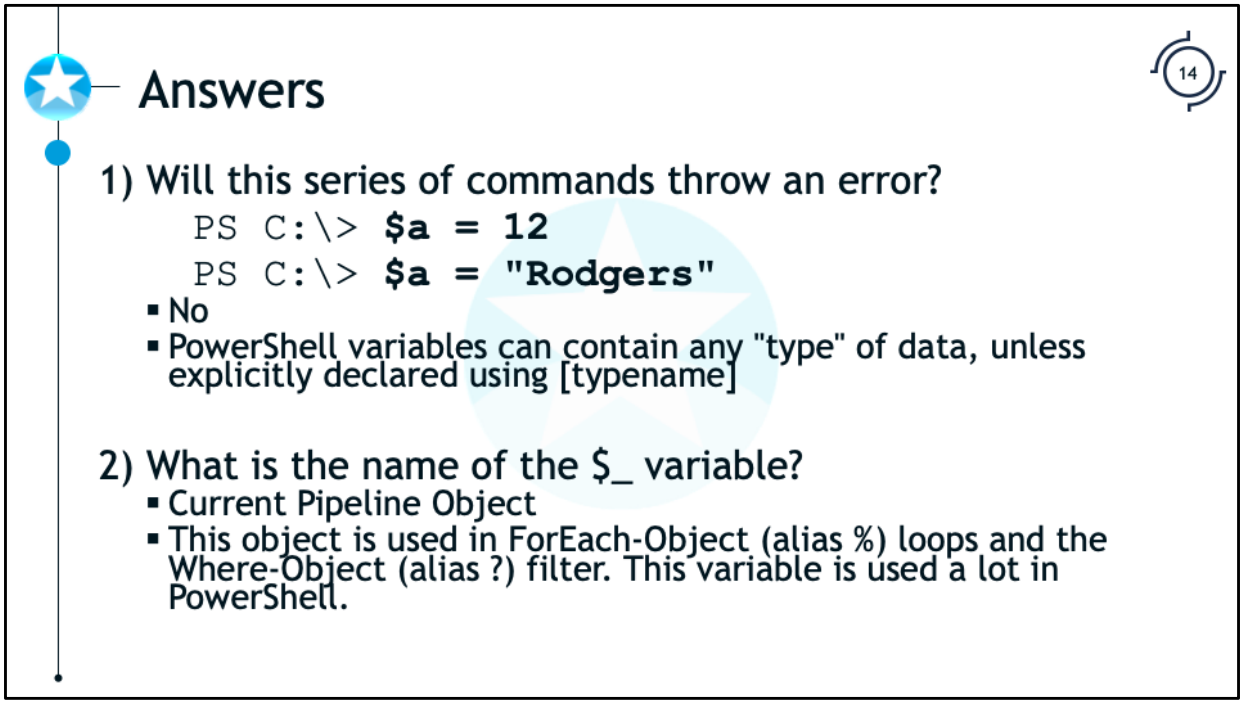


1) Will this series of commands throw an error?

```
PS C:\> $a = 12
PS C:\> $a = "Rodgers"
```

2) What is the name of the $_ variable?

Current Object
Current Pipeline Object
Iterator Object
Filter Object
Null



1) Will this series of commands throw an error?

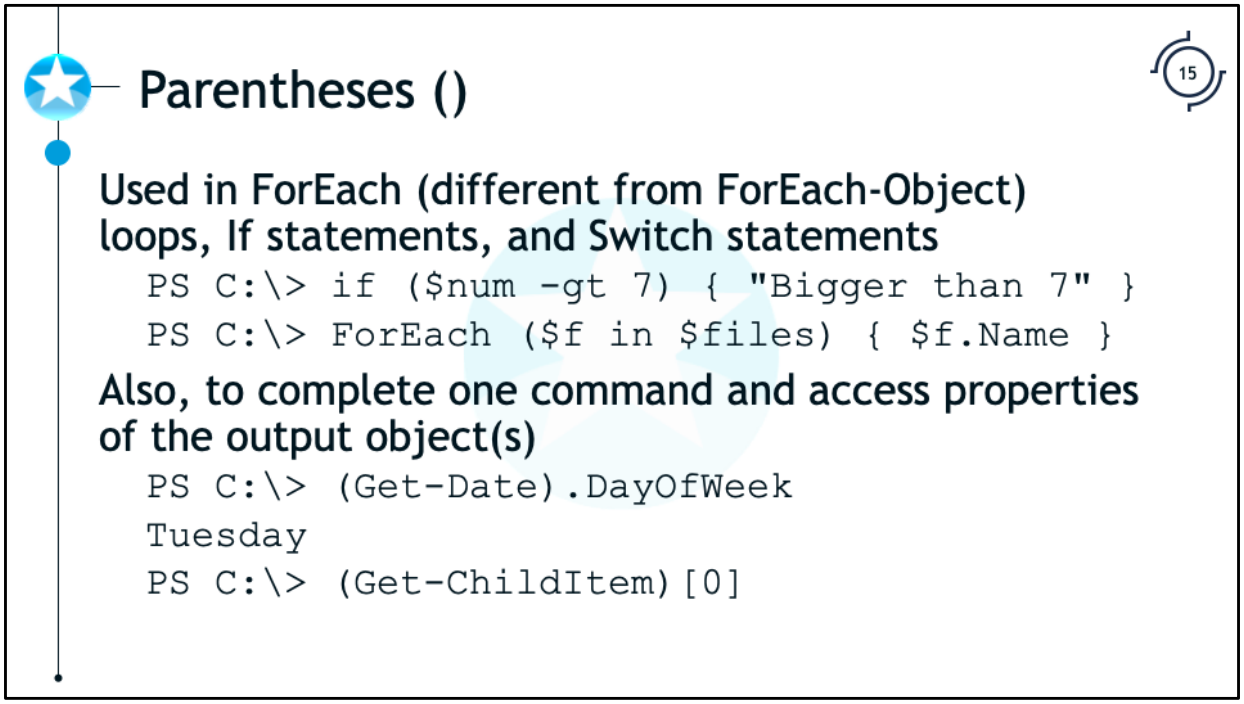
```
PS C:\> $a = 12
PS C:\> $a = "Rodgers"
```

No, PowerShell variables can contain any "type" of data, unless explicitly declared using [typename]

2) What is the name of the $_ variable?

Current Pipeline Object

This object is used in ForEach-Object (alias %) loops and the Where-Object (alias ?) filter. This variable is used a lot in PowerShell.

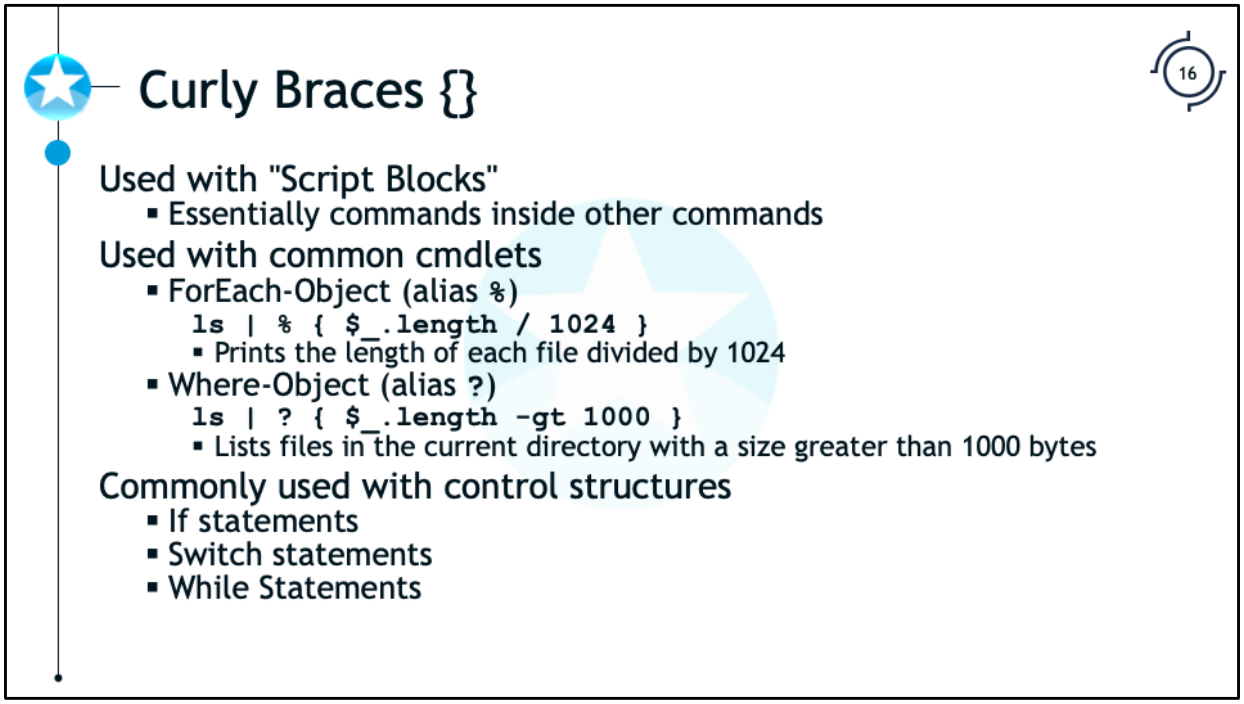


If a cmdlet returns output and you would like to access a property, method, or item then wrap it in parenthesis and then use the operator in question.

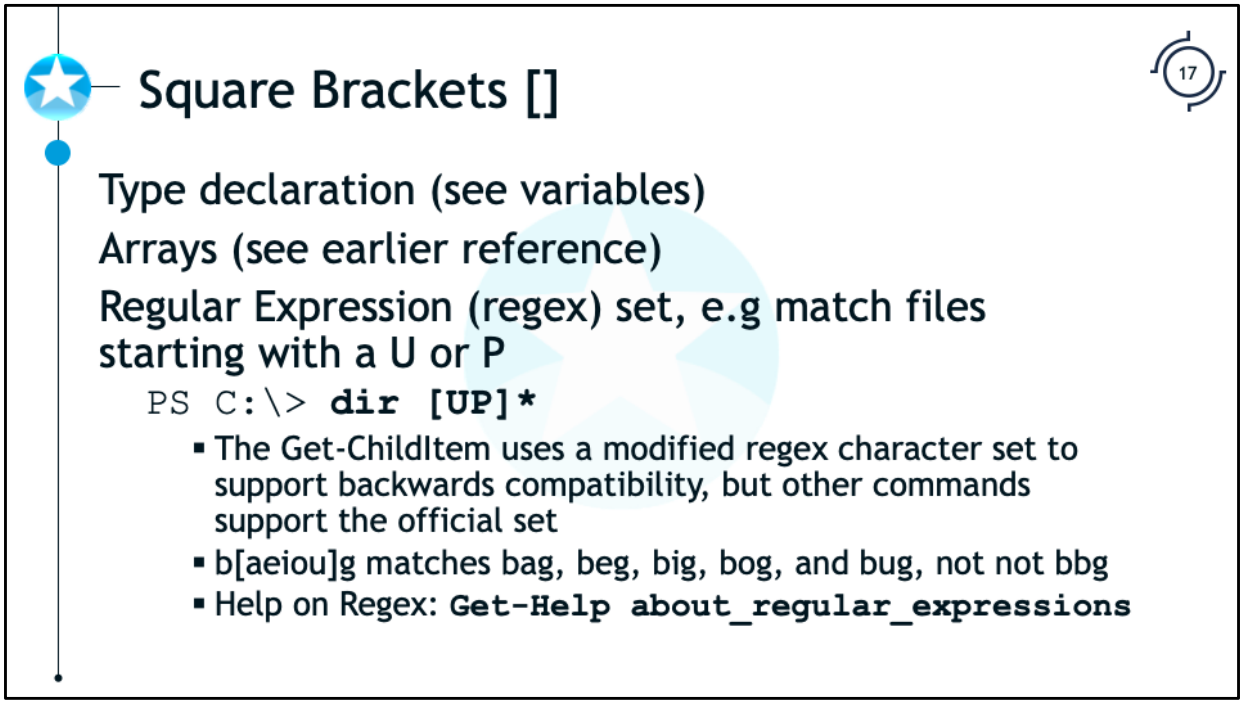
As previously mentioned, the output of cmdlets are objects. Objects have properties that we may need to access. For example, the Get-Date cmdlet returns a Date Object that has a DayOfWeek property. To get the day of the week we need to get the current date and then access that property.

```
PS C:\> (Get-Date).DayOfWeek
Tuesday
```

Similarly, the Get-ChildItem command will return a list of objects and to access the first object (technically the 0th) we need to wrap it in parenthesis to finish the command and then access the first item in the array.



Curly braces are used with "script blocks," which are essentially commands inside of commands. This is most commonly used with "Where-Object", "ForEach-Object", and control structures like the "If", "While", and "Switch" statements.



Square brackets have a number of uses in PowerShell: Regular Expressions, Type Declaration, and accessing an item in an array.

Regular expressions, also called "regex", allow for very flexible and specific searching. This is a very deep subject; in fact, multiple books have been written on just Regular Expressions. We won't discuss this in depth, but suffice it to say that the values inside the brackets are part of the search set. For example, the set of "[a-eh]" includes "a", "b", "c", "d", "e", and "h". In a practical example, the command below will get all files and folders beginning with "U" or "P".

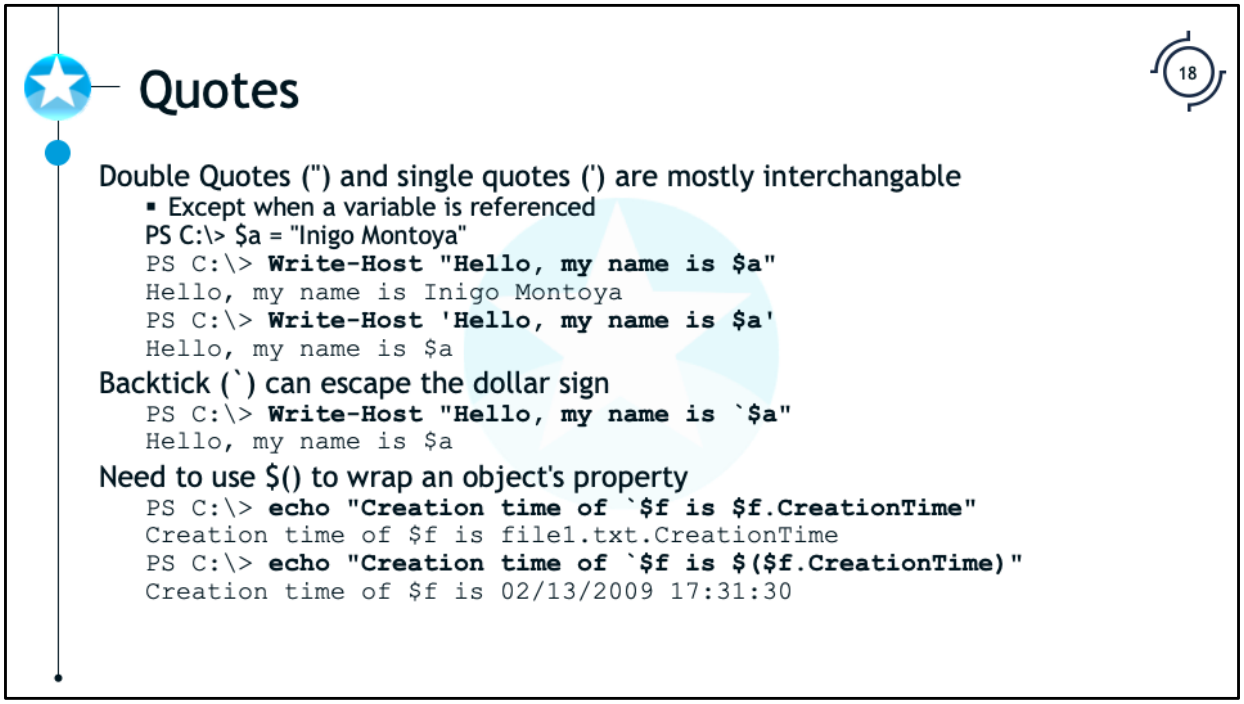
```
PS C:\> Get-ChildItem [UP]*


Directory: C:\


Mode    LastWriteTime        Length Name
d-r--   1/2/2011 1:27 PM     Program Files
d-r--   12/8/2010 8:56 AM    Users
```

What a nifty way to search.

Square brackets are also used to cast an object or declare a variable with a specific type (as shown in the "Variables" section). Also, the square brackets are used to access items in an array (see the "Arrays" slide).



For the most part, the single and double quotes are nearly interchangeable. The only difference is PowerShell tries to expand text inside double quotes, but not inside single quotes. Here is an example:

```
PS C:\> $a = "Inigo Montoya"
PS C:\> Write-Host "Hello, my name is $a"
Hello, my name is Inigo Montoya
PS C:\> Write-Host 'Hello, my name is $a'
Hello, my name is $a
```

You can accomplish the same thing with double quotes, but you need to use the backtick as a delimiter to escape the dollar sign:

```
PS C:\> Write-Host "Hello, my name is `$a"
Hello, my name is $a
```

If you want to use a variable inside quotes, you simply use the variable; but what if you want to access a property or a specific array index of that variable? Let's start with the basic version of the command by just accessing the variable.

```
PS C:\> $a = Get-Item Windows
PS C:\> echo "The variable `$a contains $a"
```

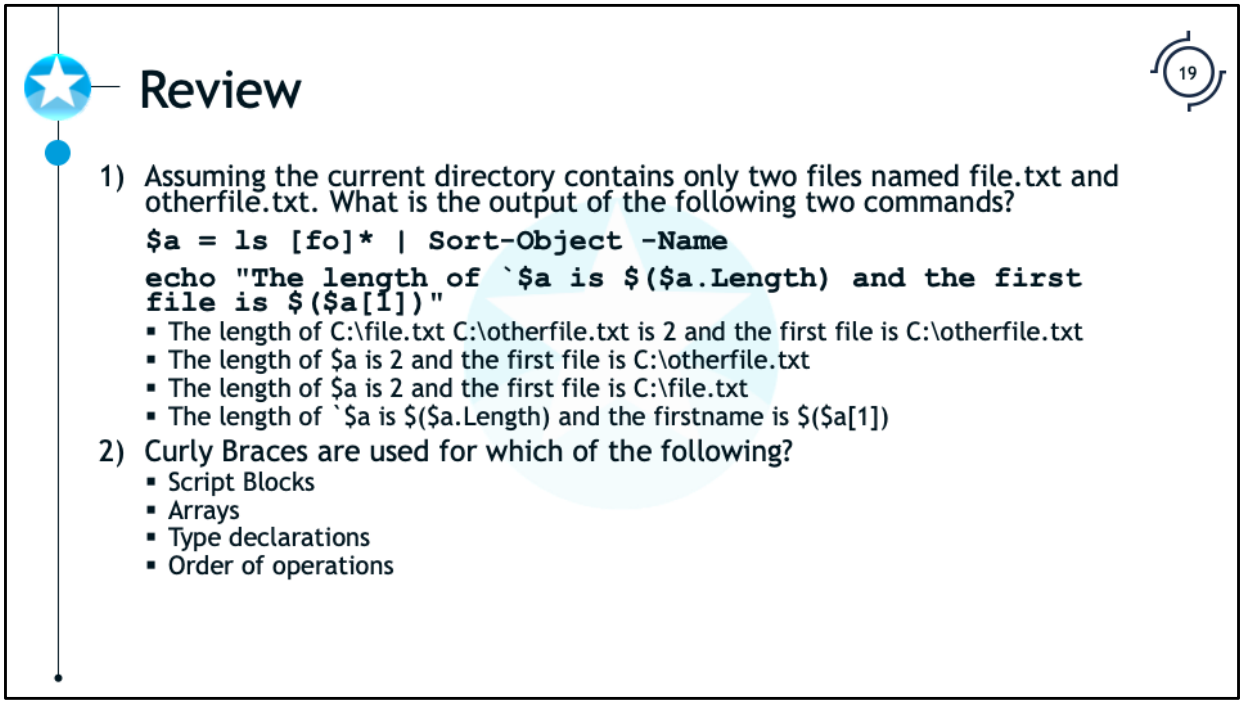
The variable $a contains C:\Windows

Now, what if we try to access the "CreationTime" property of the object "$a"?

```
PS C:\> echo "The creation time of `$a is
$a.CreationTime"
The creation time of $a is C:\Windows.CreationTime
```

Uh oh, that doesn't work! To access a property in a string, we need to wrap it in "$( )", called the sub-expression operator. Here is the right way of doing the same thing.

```
PS C:\> echo "The creation time of `$a is
$($a.CreationTime)"
The creation time of $a is 02/13/2009 17:31:30
```



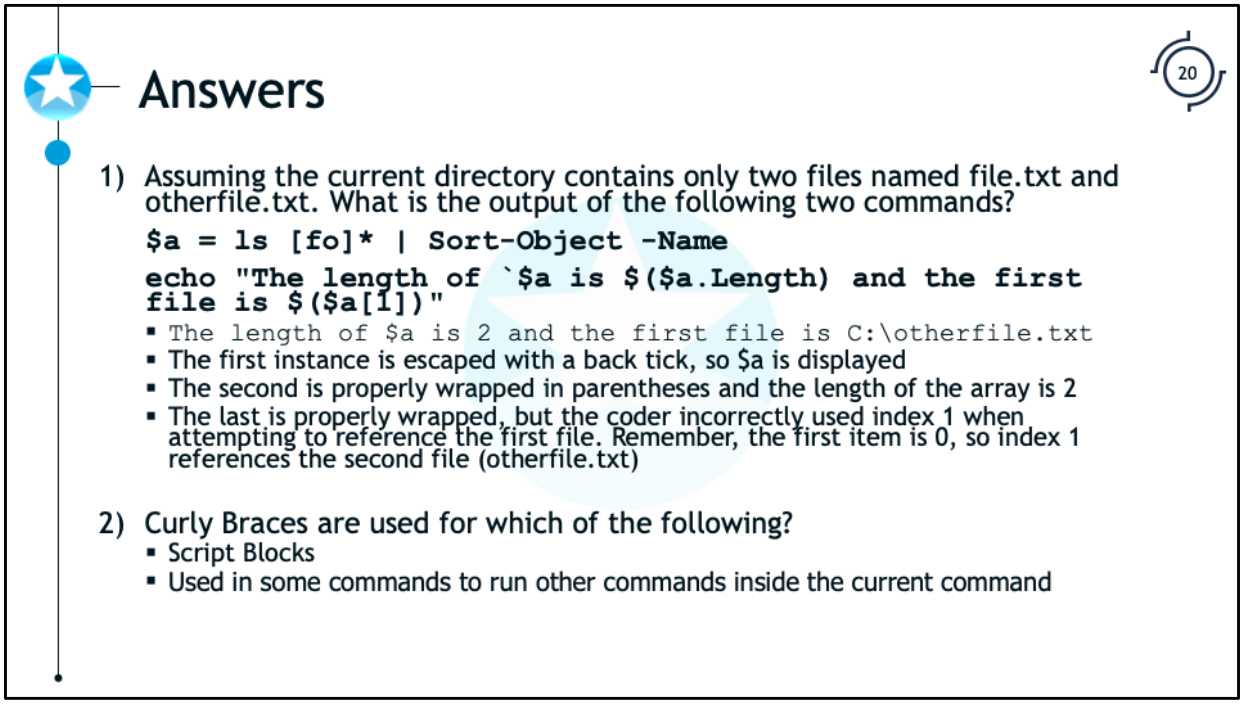
1) Assuming the current directory contains only two files named file.txt and otherfile.txt. What is the output of the following two commands?

```
$a = ls [fo]* | Sort-Object -Name
echo "The length of `$a is $($a.Length) and the first file is $($a[1])"
```

a. The length of C:\file.txt C:\otherfile.txt is 2 and the first file is C:\otherfile.txt
b. The length of $a is 2 and the first file is C:\otherfile.txt
c. The length of $a is 2 and the first file is C:\file.txt
d. The length of `$a is $($a.Length) and the firstname is $($a[1])

2) Curly Braces are used for which of the following?

Script Blocks
Arrays
Type declarations
Order of operations



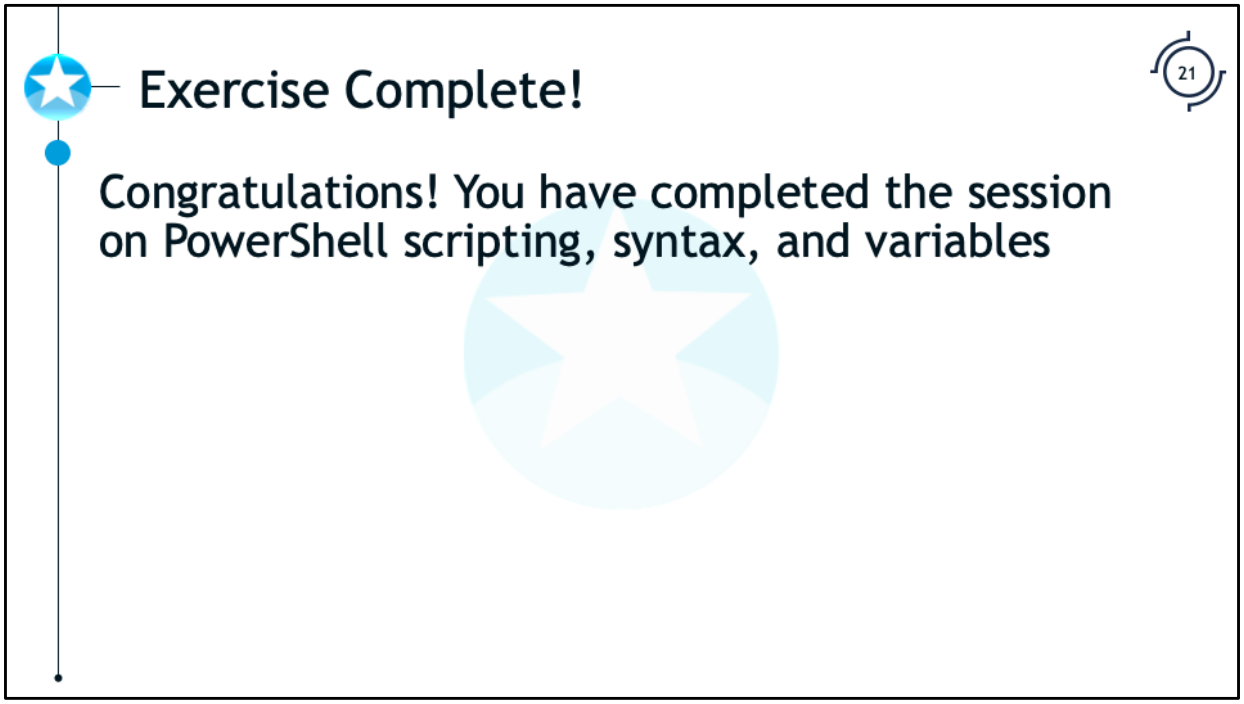
1) Assuming the current directory contains only two files named file.txt and otherfile.txt. What is the output of the following two commands?

```
$a = ls [fo]* | Sort-Object -Name
echo "The length of `$a is $($a.Length) and the first file is $($a[1])"
```

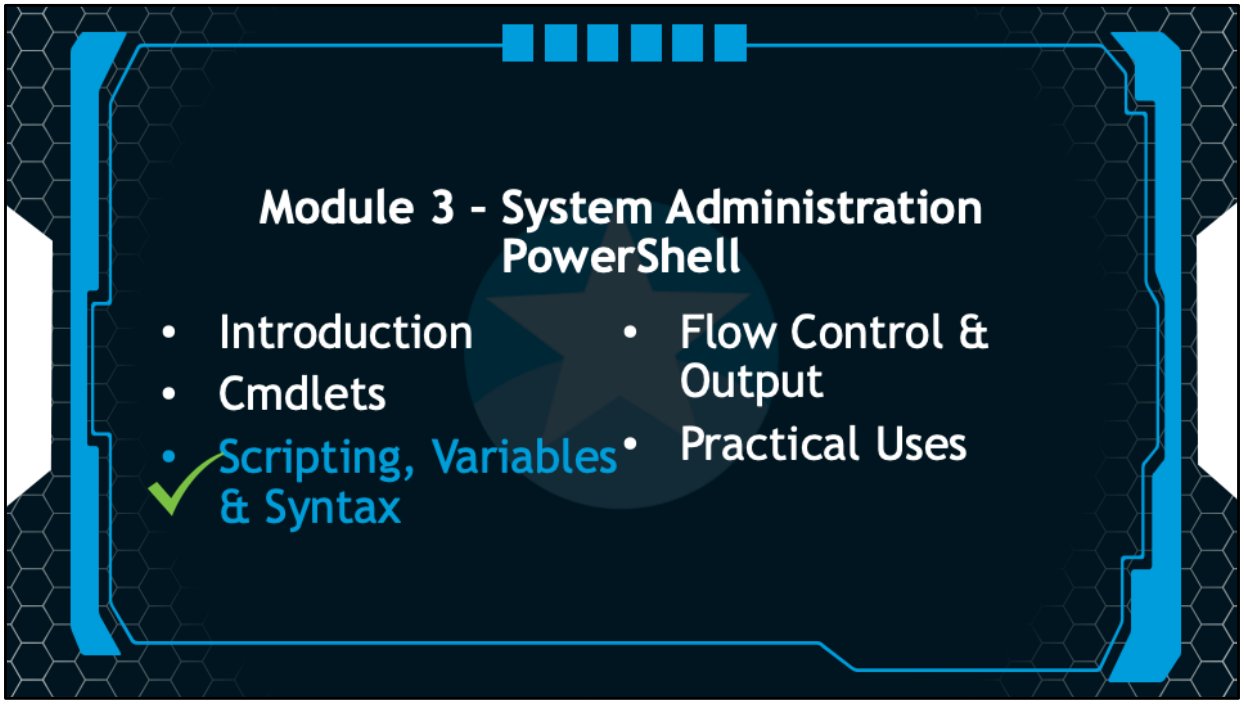
a. `The length of C:\file.txt C:\otherfile.txt is 2 and the first file is C:\otherfile.txt`
b. `The length of $a is 2 and the first file is C:\otherfile.txt`
c. `The length of $a is 2 and the first file is C:\file.txt`
d. `The length of `$a is $($a.Length) and the firstname is $($a[1])`

2) Curly Braces are used for which of the following?

Script Blocks
Arrays
Type declarations
Order of operations



Exercise Complete



This portion intentionally left blank.