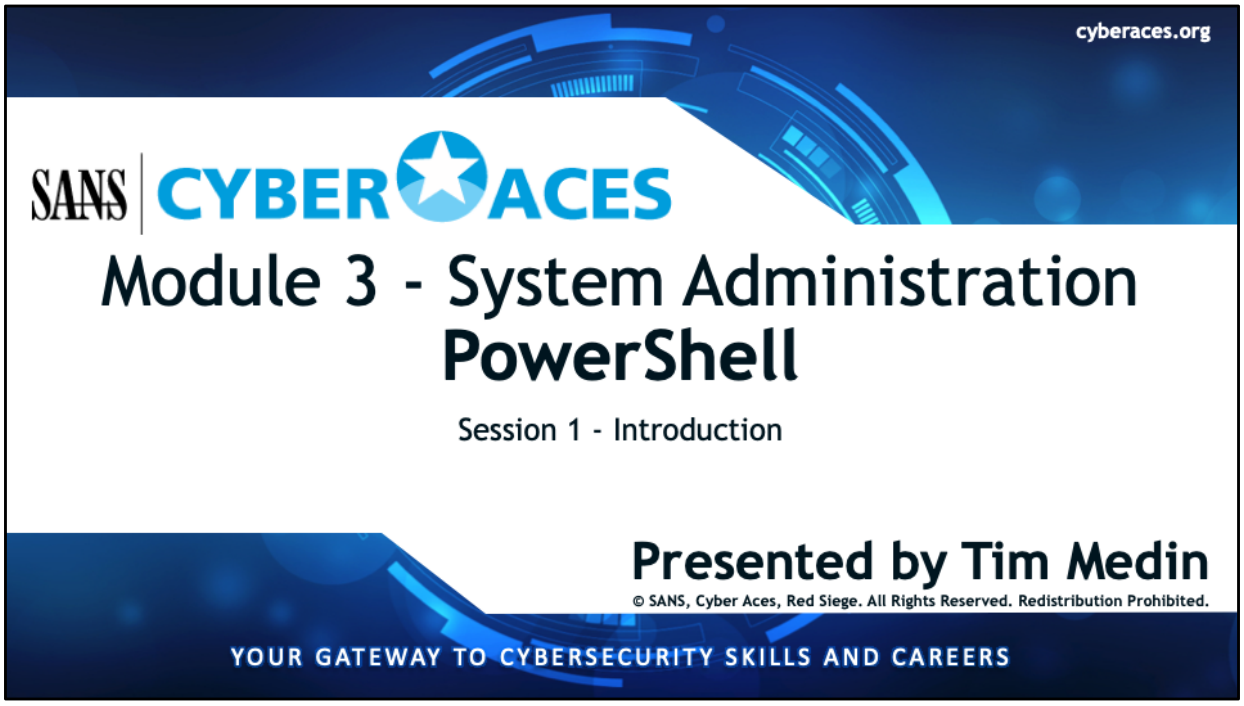
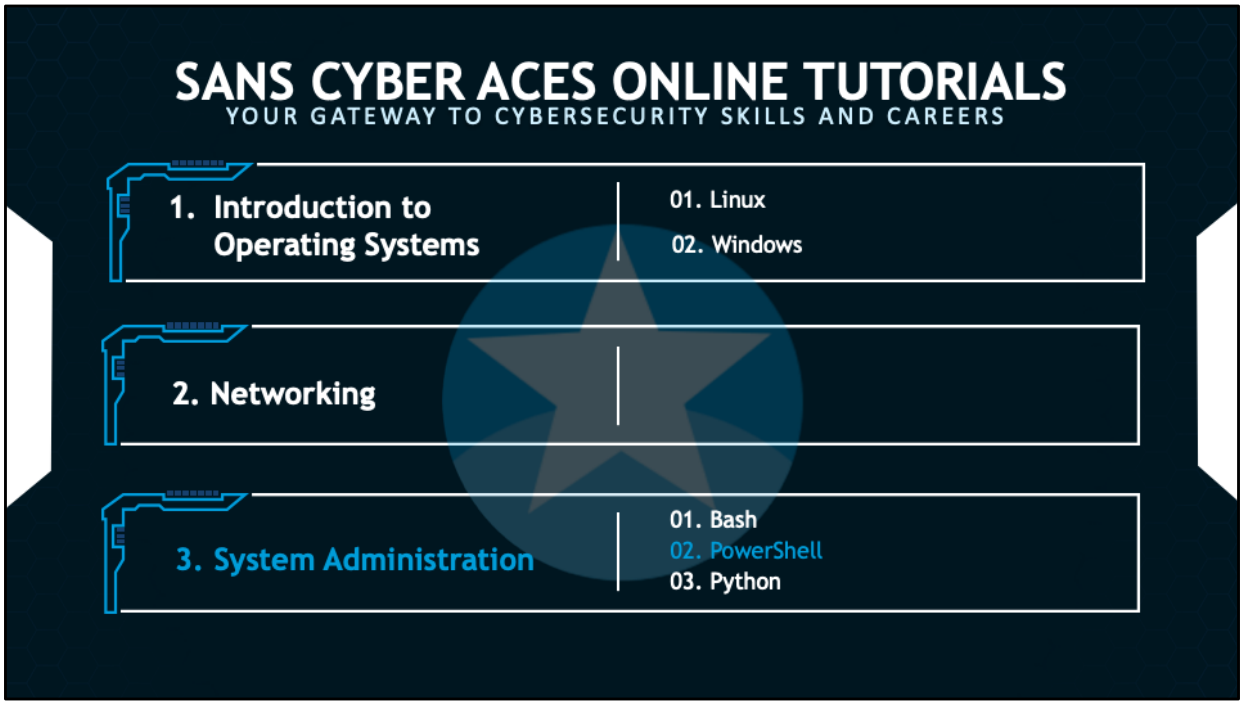


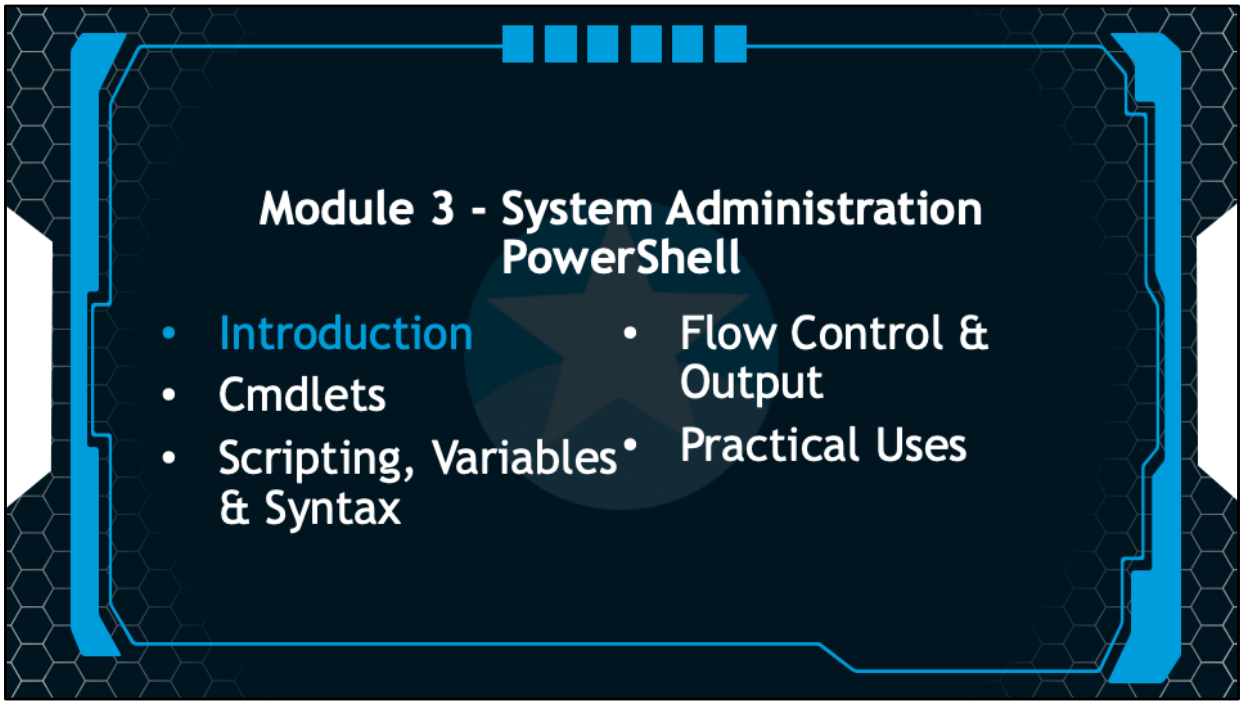
Welcome to Cyber Aces, Module 3! This module provides an introduction to the latest shell for Windows, PowerShell.



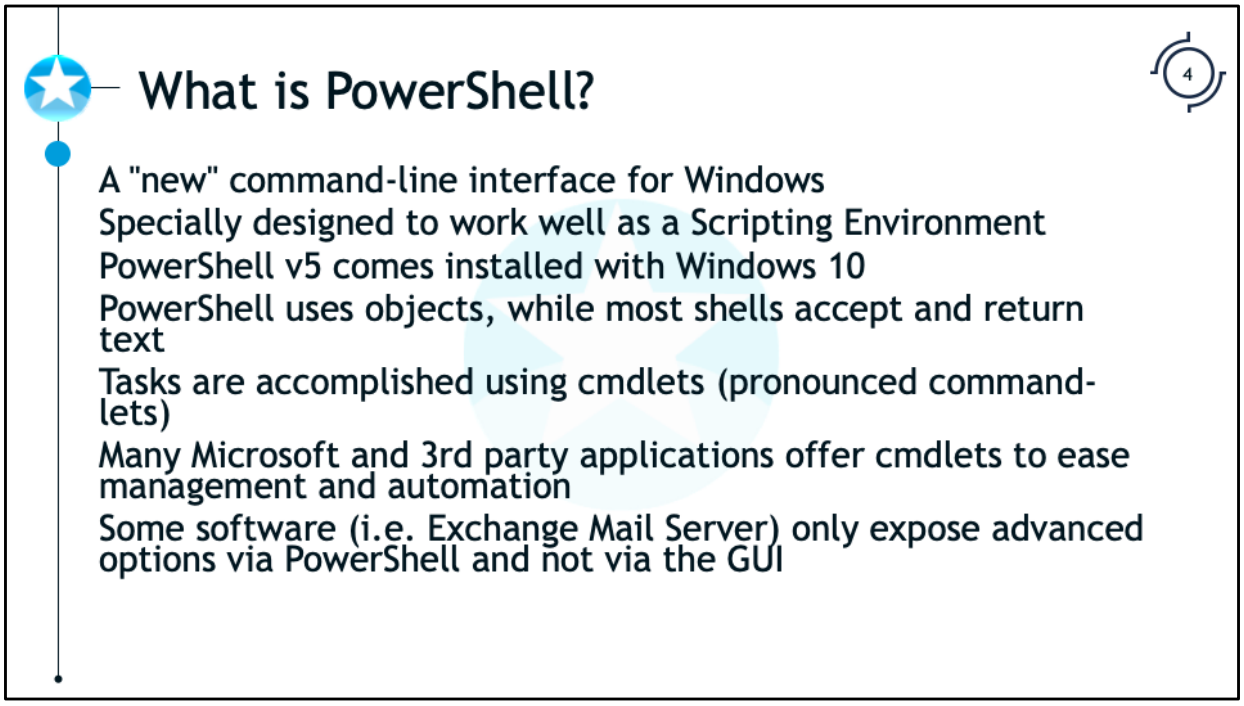
This training material was originally developed to help students, teachers, and mentors prepare for the Cyber Aces Online Competition. This module focuses on the basics of what an operating systems is as well as the two predominant OS's, Windows and Linux. In this session we will provide a walkthrough 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.



Originally codenamed Monad (or Microsoft Shell or MSH), it was designed as a new approach to managing Windows systems via the command line. PowerShell was originally a separate download for Windows XP, Vista, Windows Server 2003, and later for Window Server 2008 (R1), but it is not supported on Windows 2000 or older.

PowerShell version 2.0 was integrated into Windows 7 and was released at the same time as Windows 7. Windows Server 2008R2 also came with PowerShell v2.0 installed. Separate installs were made available for previous versions of Windows. The Windows 10 family includes PowerShell v5.

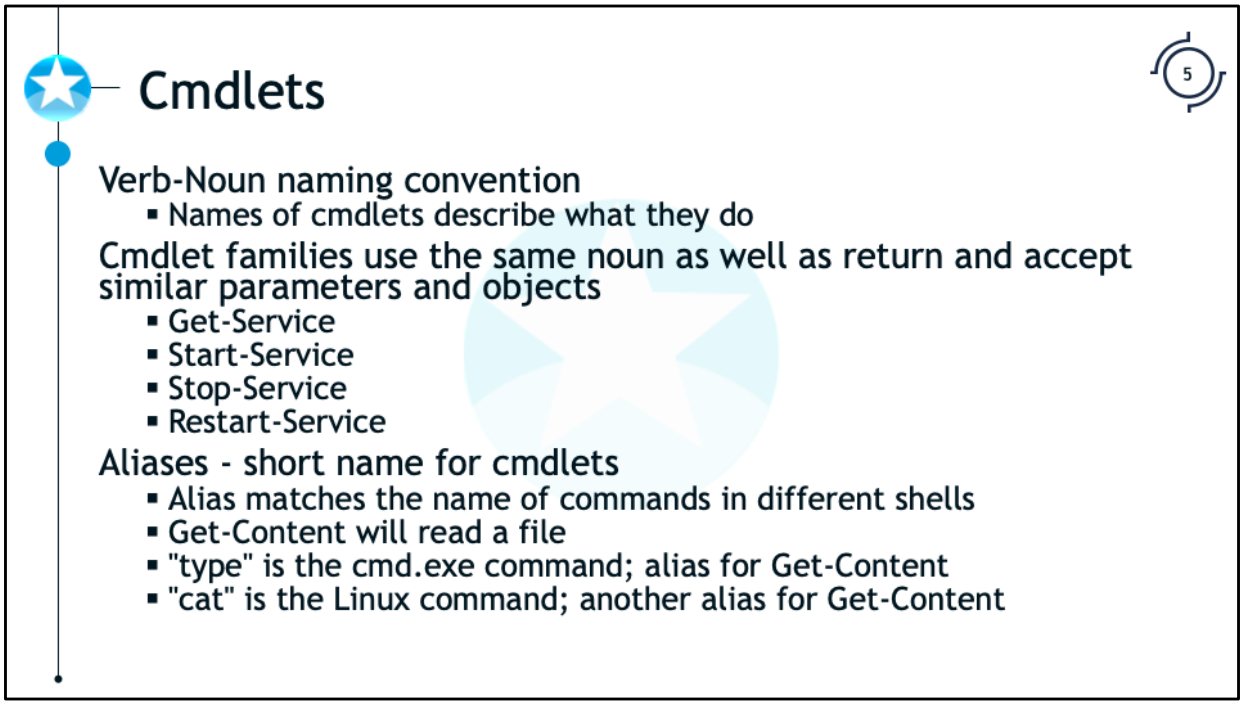
Prior to PowerShell, all major shells used text as input and output. As we'll see, the use of objects allows structured data to be used as input and output which allows for simpler manipulation of data via the command line.

PowerShell uses cmdlets (pronounced command-lets) to accomplish tasks and are very similar to *commands* used by other shells and operating systems. The cmdlets often expose more options than are available via the GUI (Graphical User Interface) and are generally the recommended approach for adjusting advanced features of many server packages that support PowerShell.

Read more about PowerShell:

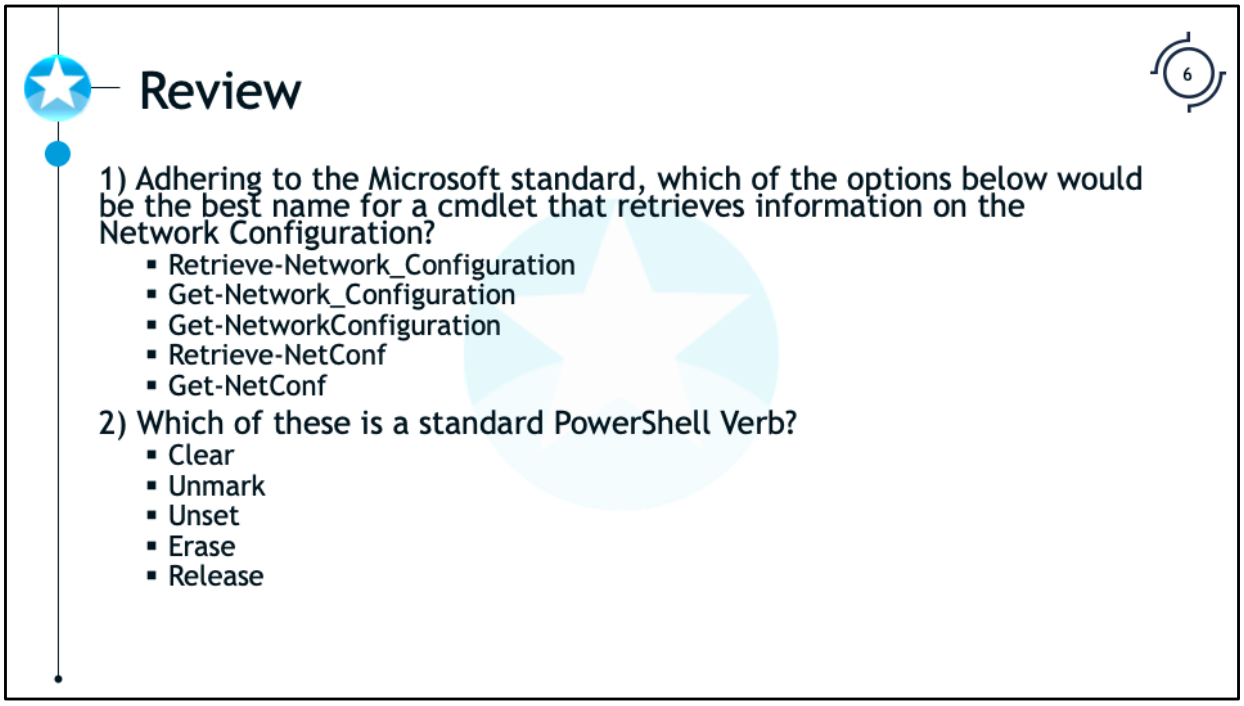
http://technet.microsoft.com/en-us/scriptcenter/powershell.aspx

https://en.wikipedia.org/wiki/Windows_PowerShell



PowerShell uses a verb-noun pair for cmdlet names. For example, Get-Date would "get" the current "date." The verbs are standardized by Microsoft (http://msdn.microsoft.com/en-us/library/ms714428(v=vs.85).aspx) to make memorization easier and to ensure consistent use of names. This standard ensures that cmdlet developers all use the verb "Add", instead of a seemingly random assortment of "Append", "Attach", "Concatenate", or "Insert".

Families of commands are grouped by nouns. It is quickly apparent that "Get-Service", "Start-Service", "Stop-Service", and "Restart-Service" are all related. As such, these cmdlets accept a similar set of parameters and return a similar set of objects.



1) Adhering to the Microsoft standard, which of the options below would be the best name for a cmdlet that retrieves information on the Network Configuration?

Retrieve-Network_Configuration

Get-Network_Configuration

Get-NetworkConfiguration

Retrieve-NetConf

Get-NetConf

2) Which of these is a standard PowerShell Verb?

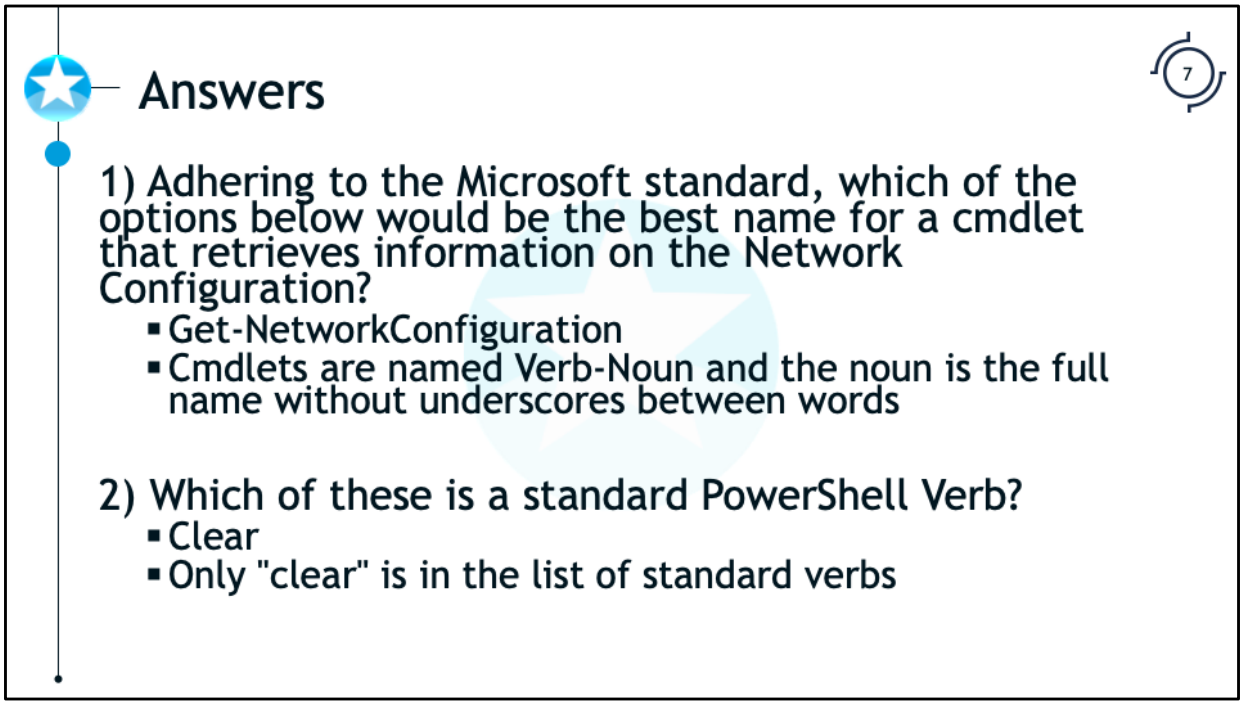
Clear

Unmark

Unset

Erase

Release



1) Adhering to the Microsoft standard, which of the options below would be the best name for a cmdlet that retrieves information on the Network Configuration?

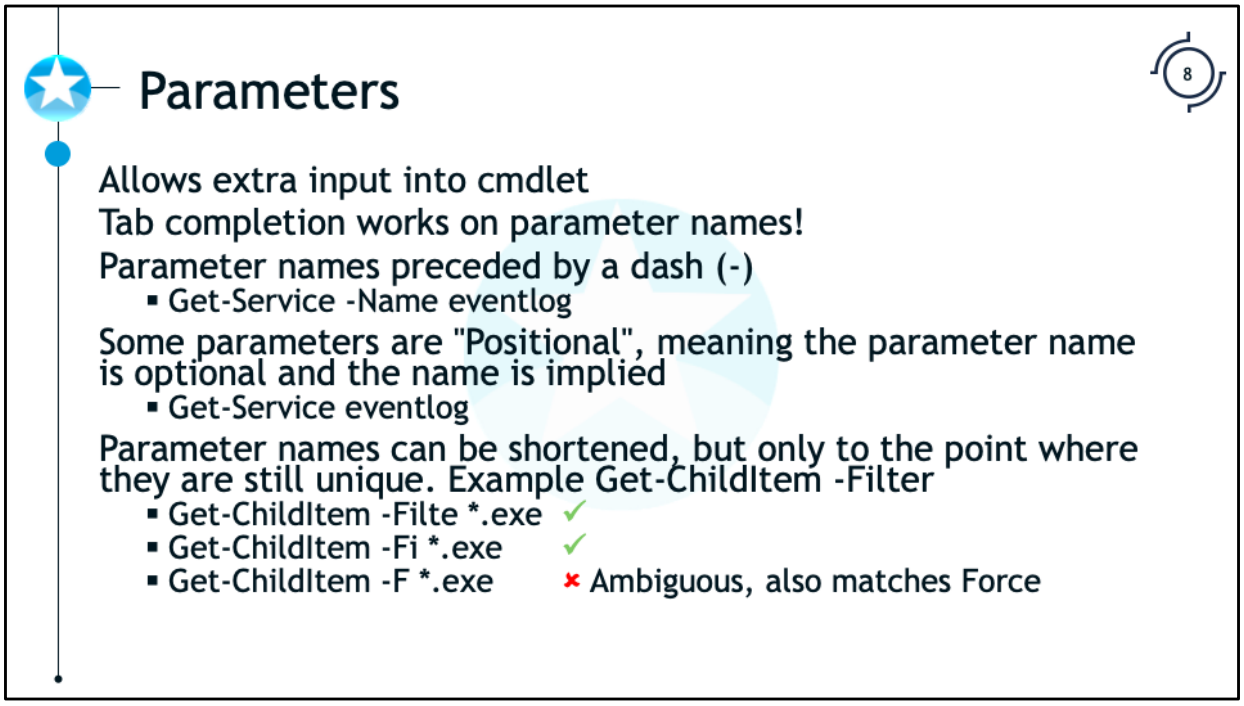
Get-NetworkConfiguration

Cmdlets are named Verb-Noun and the noun is the full name without underscores between words

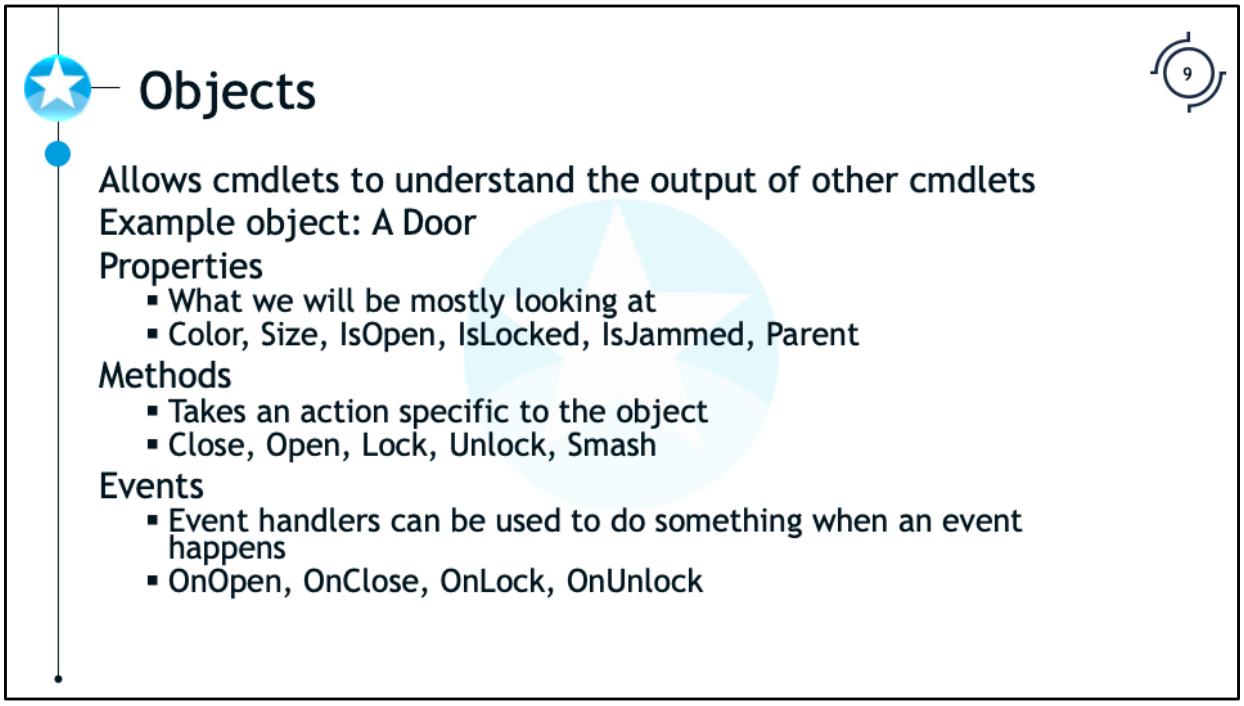
2) Which of these is a standard PowerShell Verb?

Clear

Only "clear" is in the list of standard verbs



Most cmdlets take additional parameters (or arguments). Parameter names are preceded by a dash (-). One such parameter, used by the cmdlet "Get-Service", is "Name". The command "Get-Process -Name svchost" will "get" the objects representing each "process" with the "name" "svchost". Some cmdlets accept positional parameters, meaning a parameter name is not required since it is assumed from its position on the command line. "Get-Service's" "Name" is such a parameter, and the above command can be shortened to "Get-Process svchost". It is pretty convenient that cmdlets can be shortened and some parameter names can be dropped.



In Bash and cmd.exe scripting, you often spend a great deal of time interfacing between applications. In other words, you capture the output of one command, parse out the pieces of data that you need (such as an IP address), and then pass that information on to the next command. Wouldn't it be nice if each command automatically understood the output of the other? Besides being easier to read, it is much easier to write.

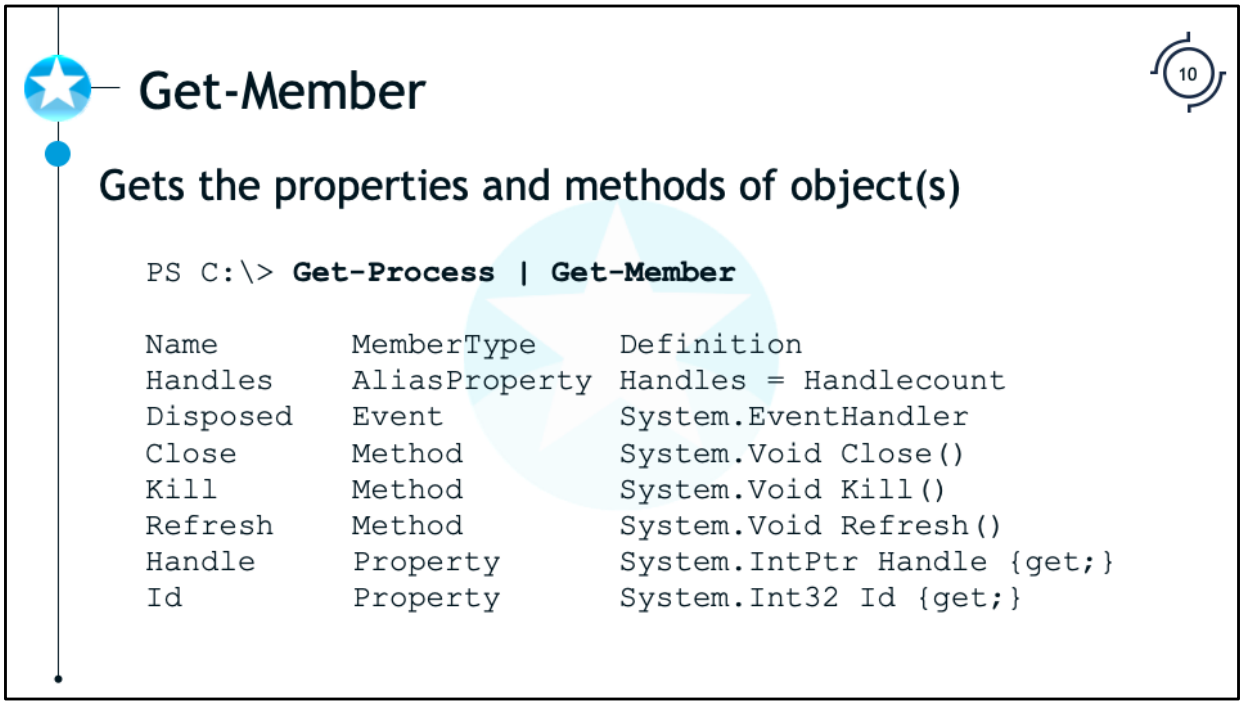
Well, as you probably guessed, that is one of the benefits of the object-oriented nature of PowerShell. In PowerShell, every "cmdlet" has an understanding of the output from other cmdlets, and they can be tied together with powerful results. The objects returned from each cmdlet are understood by other cmdlets. A simple glance at the command reveals which property is being used, and it doesn't require extra effort or intimate knowledge of the output in "field 2."

For example, it isn't immediately clear what is being done in the command below (it gets a list of the process ID's for each running process).

```
$ ps aux | cut -d' ' -f2
```

While the equivalent PowerShell command is much more readable.

```
PS C:\> Get-Process | Select ID
```



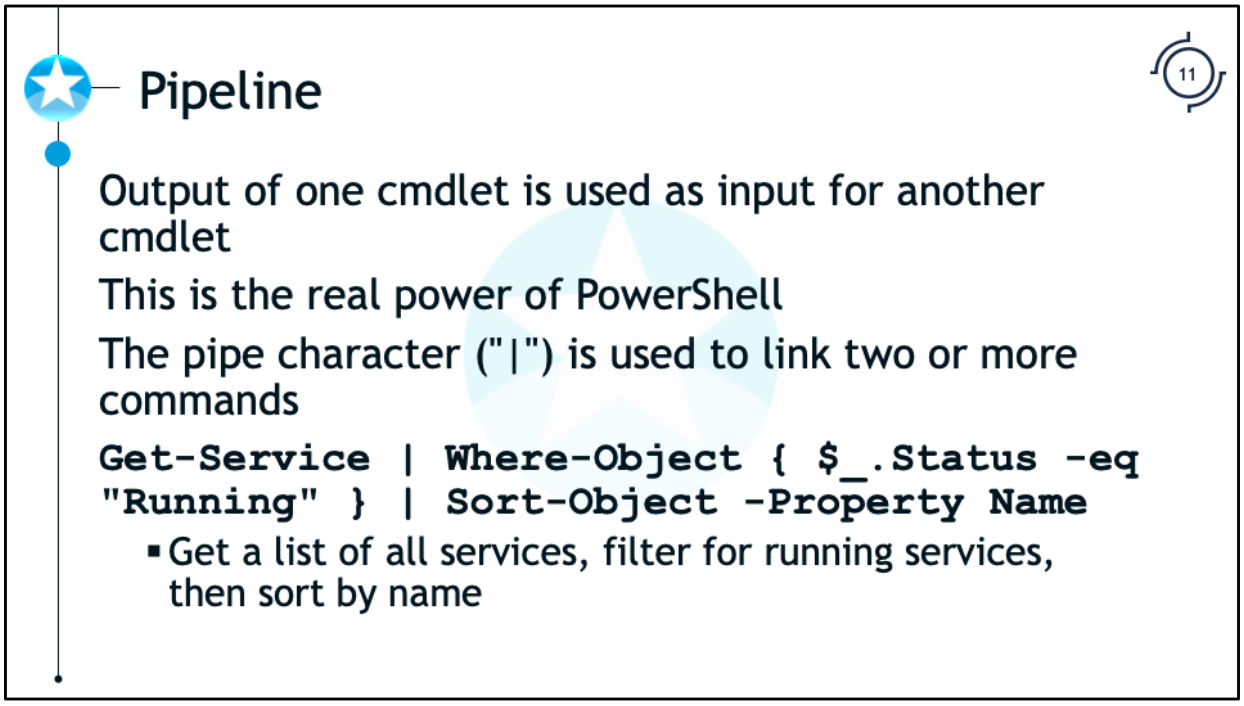
A new object type (or set of objects) may be encountered for the first time and you may not know what properties and methods are available to interact with the object. How do we know which properties and methods are available? The cmdlet "Get-Member" can be used to show available properties, methods, and events as shown above (the output has been modified for brevity).

We can kill a process by calling the Kill method.

```
PS C:\> $a = Get-Process spoolsv
PS C:\> $a.Kill()
```

Or more tersely:

```
PS C:\> (Get-Process spoolsv).Kill()
```



The real "power" in PowerShell is using the objects with the pipeline. This pipeline takes the output objects from one command and sends it as input to the next command. Simply use the pipe character ("|") to link our two commands. Here is a real-world example of the use of the pipeline:

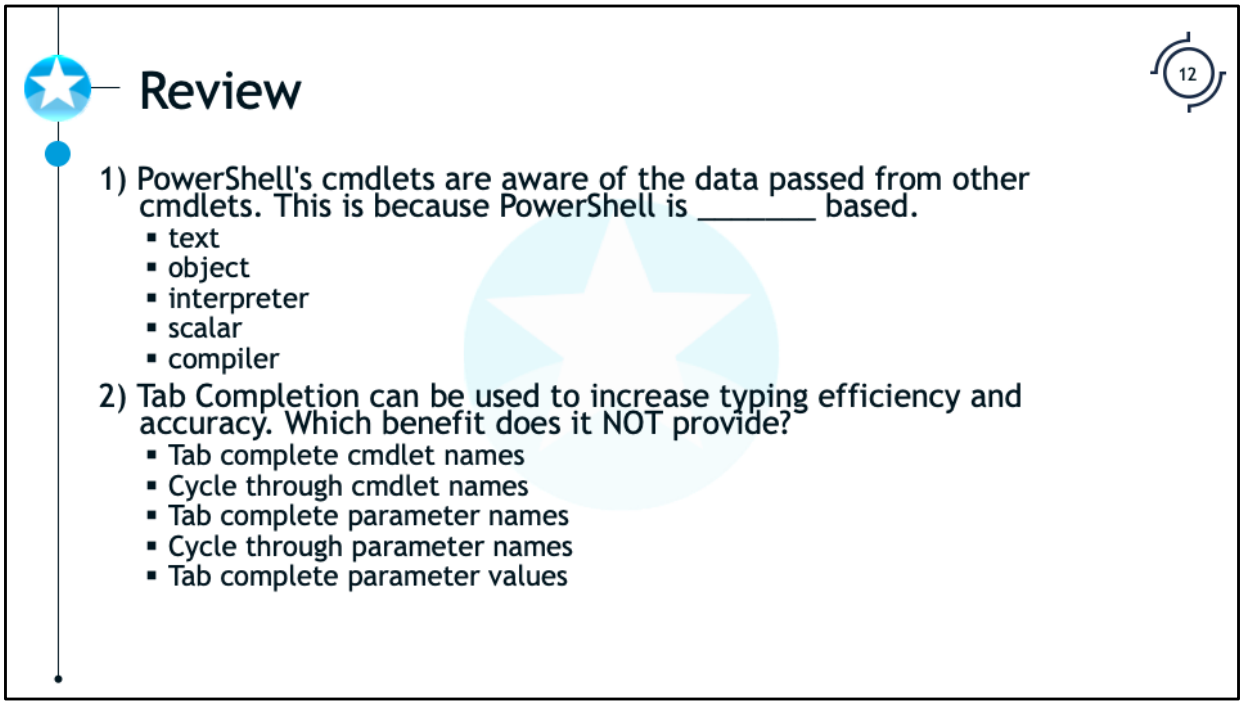
```
PS C:\> Get-Service | Where-Object { $_.Status -eq
        "Running" } | Sort-Object -Property Name
```

This command will return all the services, filter for the running ones, and sort them by name. Don't worry about the syntax of "Where-Object" for now, that will be covered in a bit.

While this is highly dangerous, we could even use this same syntax to stop all running services (don't try this at home!):

```
PS C:\> Get-Service | Where-Object { $_.Status -eq
        "Running" } | Stop-Service
```

All sorts of commands can be chained together to create some really powerful and flexible commands.



1) PowerShell's cmdlets are aware of the data passed from other cmdlets. This is because PowerShell is _____ based.

   text

   object

   interpreter

   scalar

   compiler

2) Tab Completion can be used to increase typing efficiency and accuracy. Which benefit does it NOT provide?

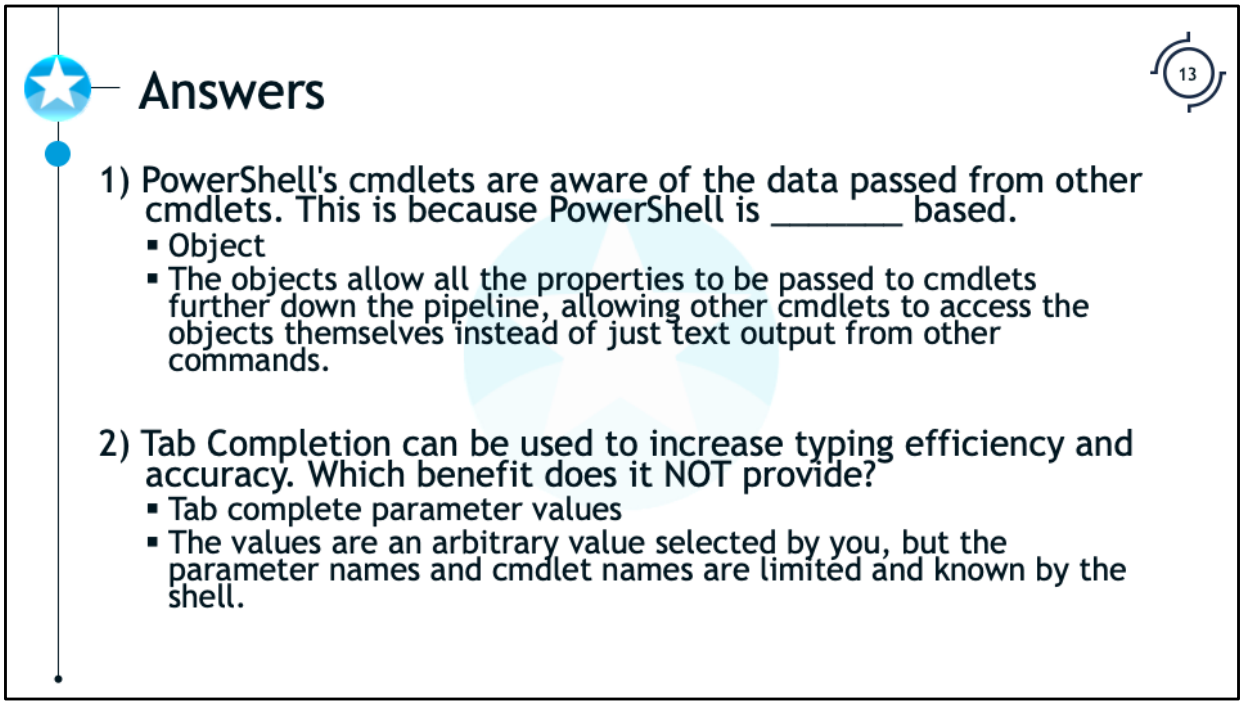
   Tab complete cmdlet names

   Cycle through cmdlet names

   Tab complete parameter names

   Cycle through parameter names

   Tab complete parameter values



Answers

1) PowerShell's cmdlets are aware of the data passed from other cmdlets. This is because PowerShell is _____ based.

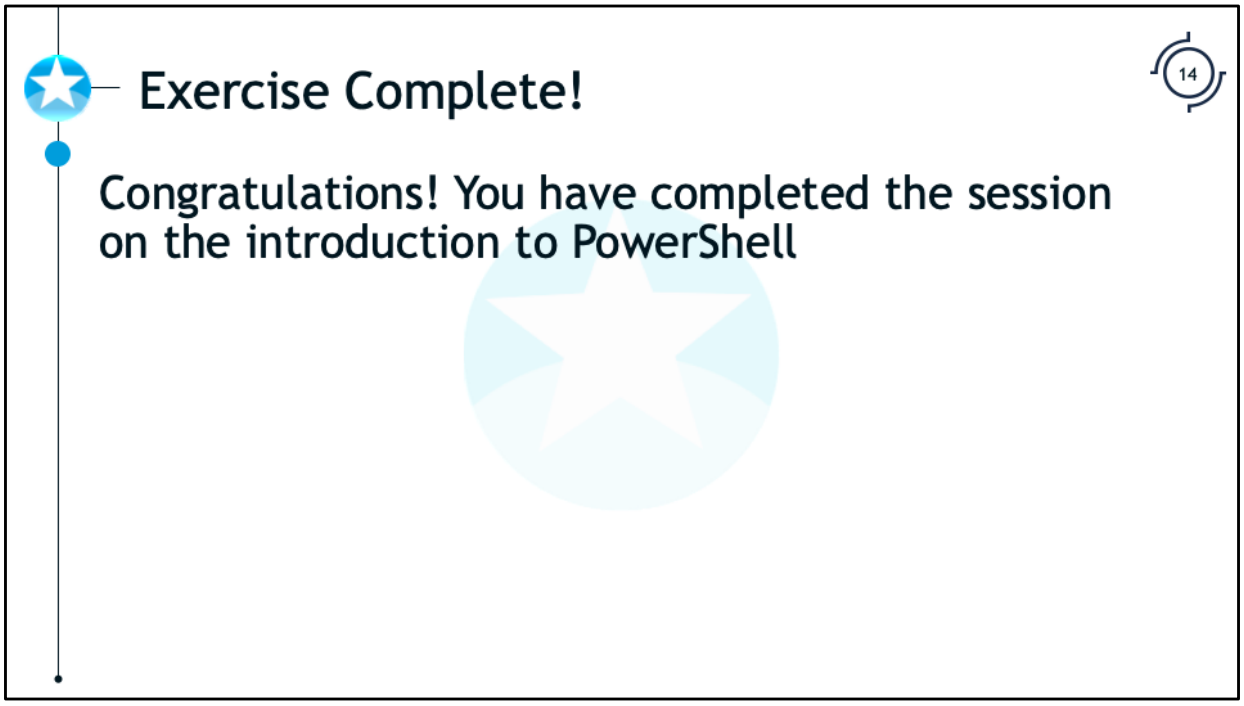
   Object

   The objects allow all the properties to be passed to cmdlets further down the pipeline, allowing other cmdlets to access the objects themselves instead of just text output from other commands.

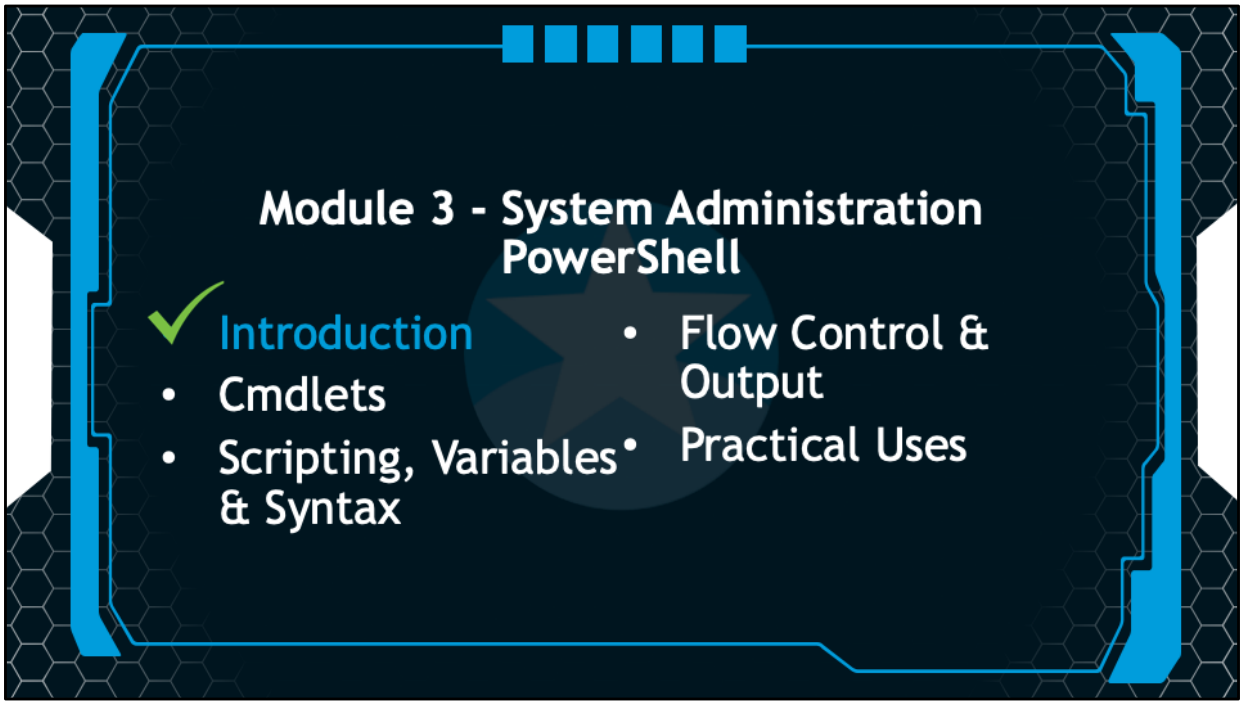
2) Tab Completion can be used to increase typing efficiency and accuracy. Which benefit does it NOT provide?

   Tab complete parameter values

   The values are an arbitrary value selected by you, but the parameter names and cmdlet names are limited and known by the shell.



Exercise Complete



This portion intentionally left blank.