01\_PS\_Fundamentals

* **PEs are a reflection of what the test looks like**

**Tips n’ tricks**

* **Will have to make an alias of an alias (like an alias of the man alias (newMan -> man -> help))**
* **CTRL+Spacebar (shows you a dropdown of your possible entries based on what you already have typed)**
* **Get-help <command> -ShowWindow: brings out a searchable window to find the info you want easier**

**REGEX**

* **(Get-Content .\$file | Select-String -CaseSensitive -Pattern $regex -AllMatches).Matches.Value.Count**
  + **Regex in count**
* **(Get-Content $file | Select-String -CaseSensitive -Pattern $regex -AllMatches | ForEach-Object {$\_.Matches.Value})**
  + **Regex match**
* **((Get-Content .\$file | Select-String -CaseSensitive -Pattern $regex -AllMatches).Matches.Value | Sort-Object -Unique).Count**
  + **Regex unique**
* **What is Powershell?**
  + PowerShell is a cross-platform automation solution made up of a command-line shell, a scripting language, and a configuration management framework. PowerShell runs on Windows, Linux, and macOS.
  + As a scripting language, PowerShell is commonly used for automating the management of systems. It is also used to build, test, and deploy solutions, often in CI/CD environments.
  + In software engineering, CI/CD or CICD is the combined practice of continuous integration and either continuous delivery or continuous deployment. CI/CD bridges the gaps between development and operation activities and teams by enforcing automation in building, testing and deployment of applications.
  + **PowerShell is built on the .NET Common Language Runtime (CLR).** 
    - **All inputs and outputs are .NET objects.**
  + **An object in Powershell is simply an enriched data container**.
    - This container represents data in an organized structure with properties and functions / methods.
    - The methods are actions that can be taken by an object.
  + Core cmdlets in PS are built in .NET Core
* **Powershell Features**
  + PowerShell shares some features with traditional shells:
    - **Built-in help system:**
      * The help system in PowerShell provides information about commands and also integrates with online help articles.
    - **Pipeline**:
      * MA pipeline is used to run many commands sequentially.
    - **Aliases**:
      * Aliases are alternate names that can be used to run commands.
* **How is PS Different?**
  + It operates on objects over text. In a command-line shell, you have to run scripts whose output and input might differ. So you end up spending time formatting the output and extracting the data you need. By contrast, in PowerShell you use objects as input and output. That means you spend less time formatting and extracting. Objects retain their properties even as a user manipulates the output they see on their screen or use in pipelines.
  + **Commands in PowerShell are called cmdlets** (pronounced commandlets). Unlike many other shell environments,
    - **In PowerShell, cmdlets are built on a common runtime rather than separate executables.**
  + Cmdlets typically take object input and return objects.
    - The core cmdlets in PowerShell are built in .NET Core and are open source.
    - You can build your own cmdlets in .NET Core or PowerShell.
  + It has many types of commands. Commands in PowerShell can be native executables, cmdlets, functions, scripts, or aliases.
    - Every command you run belongs to one of these types.
    - The words command and cmdlet are often used interchangeably.

**COL**

What are some features PowerShell shares with traditional shells?

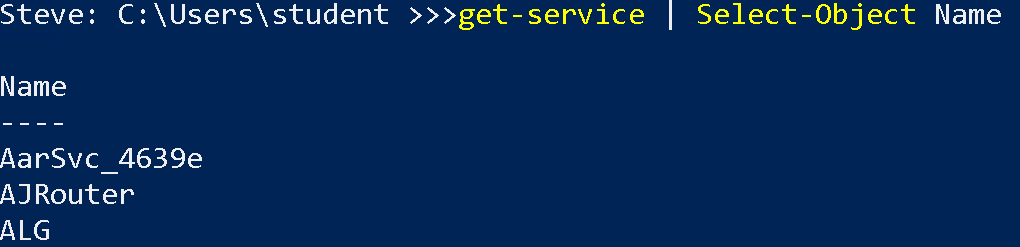
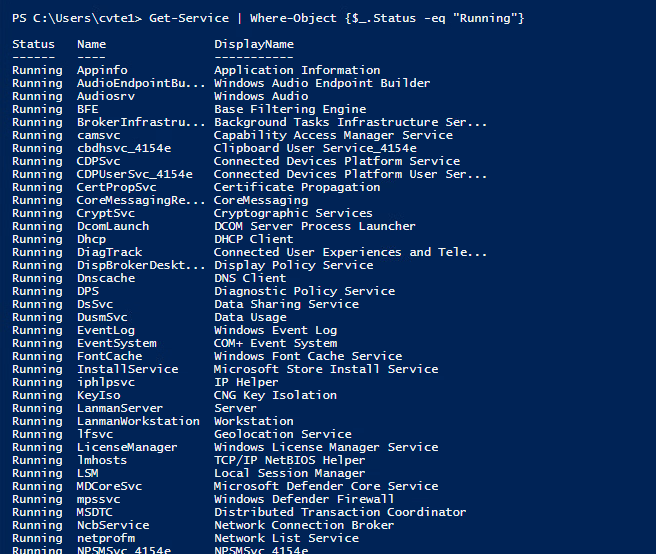
**Built in help system**

**Pipeline**

**aliases**

What are commands in PowerShell called?

**Cmdlets (commandlets)**

* **Commandlets**
  + A cmdlet is a lightweight command that is used in the PowerShell environment.
    - The PowerShell runtime invokes these cmdlets within the context of automation scripts that are provided at the command line.
  + Cmdlets differ from commands in other command-shell environments in the following ways:
    - **Cmdlets are instances of .NET classes; they are not stand-alone executables.**
    - **Cmdlets process input objects from the pipeline rather than from streams of text, and cmdlets typically deliver objects as output to the pipeline.**
  + **Cmdlets employ a verb/noun naming convention that is designed to make each cmdlet easier to remember and read.**
    - As an example, a typical Get-ChildItem command uses the verb Get followed by the noun ChildItem.
  + When executed through the PowerShell runtime environment, the Get-ChildItem command lists or returns the items in one or more specified locations.
    - If items are in a container, the command gets the items inside the container — child items.
* **PS Pipeline**
  + Simply put, a pipe | takes the output of the first part of the pipeline, and then uses it as the input of the next part.
  + 
  + Using the Select-Object cmdlet, how would we display only the DisplayName Property?
* **Parameters and Arguments**
  + Most cmdlets support the use of parameters as part of the input mechanism. Parameters can be added to the cmdlet at the command line or passed to cmdlets through the pipeline as the output from a previous cmdlet.
  + The arguments or values of each parameter detail the actual input that the cmdlet will accept, how the cmdlet should work and what, if any data the cmdlet outputs.
  + On the command line, when you run the Get-Service cmdlet, you get a list of services on your machine.
  + You can further filter these just to show the services that are running:
    - **Get-Service | Where-Object {$\_.Status –eq “Running”}**
  + 

**COL**

What type of naming convention do cmdlets use?

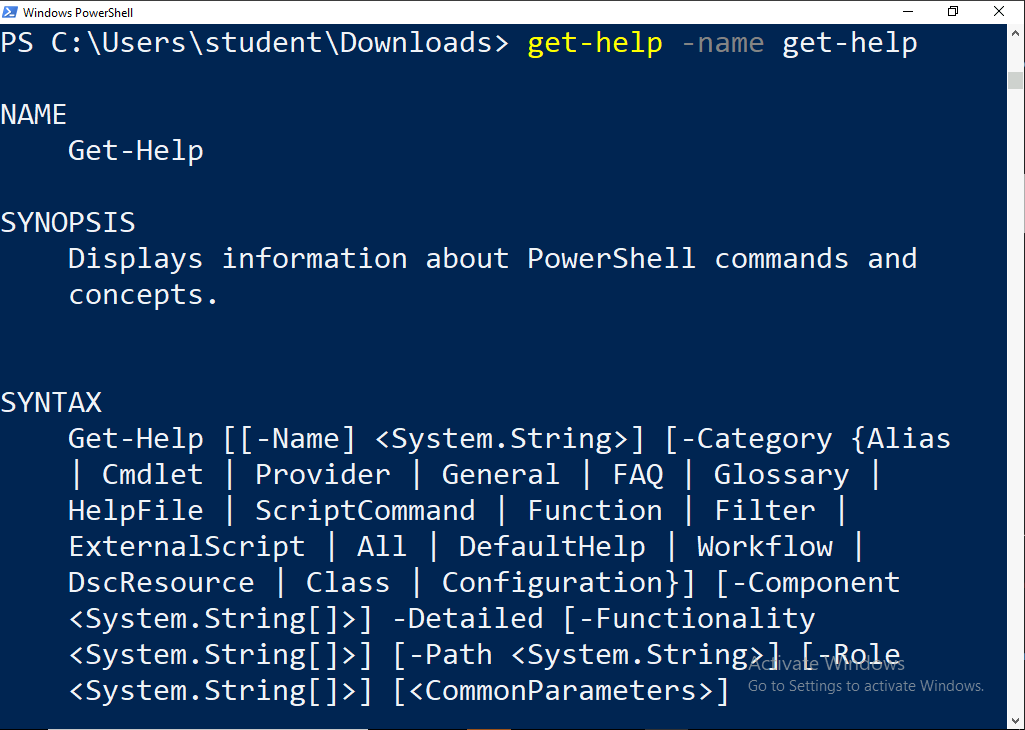
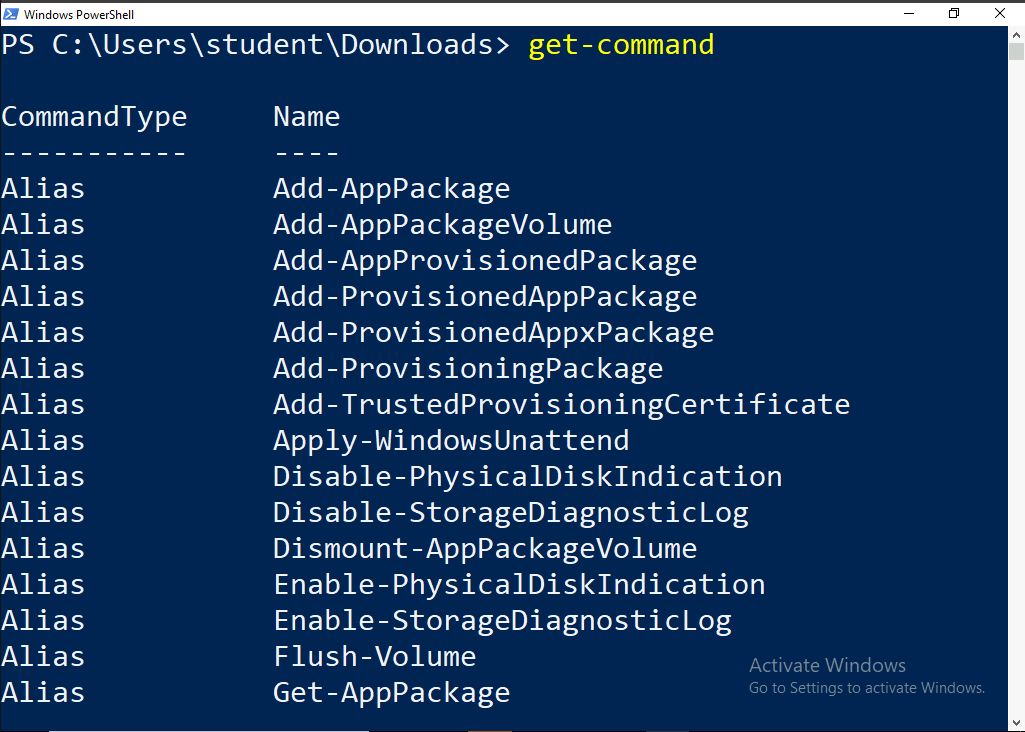
**verb-noun**

Which statement is true about how cmdlets differ from commands?

a. Cmdlets are stand-alone executables.

**b. Cmdlets are instances of .NET classes**

c. both a & b

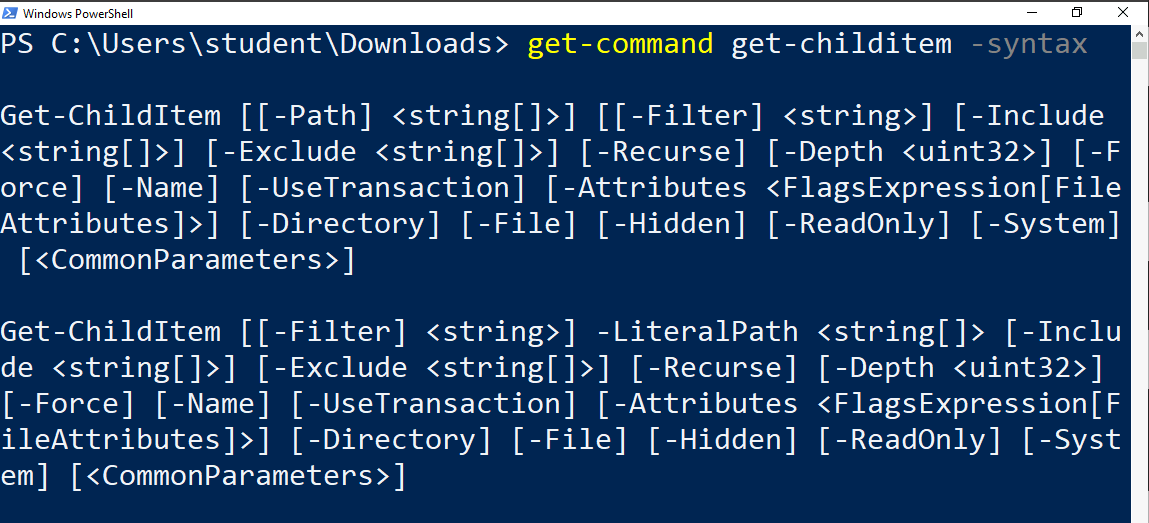
* **PS Help System**
  + Most shells have some kind of help system in which you can learn more about a command. For example, you can learn what the command does and what parameters it supports.
  + **Powershell Help Cmdlets:**
    - **Get-Help**
      * Get-Help is a multipurpose command. Get-Help helps you learn how to use commands once you find them.
      * When Get-Help is used to locate commands, it first searches for wildcard matches of command names based on the provided input. If it doesn’t find a match, it searches through the help topics themselves, and if no match is found an error is returned.
      * **Get-Help -Name Get-Help**
        + We are calling "Get-Help" on the cmdlet named "Get-Help"
      * 
    - **Get-Command**
      * The Get-Command cmdlet gets all commands that are installed on the computer, including cmdlets, aliases, functions, filters, scripts, and applications.
      * This is helpful when we want to get a list of all the commands that are available to us in the current powershell session.
      * 
        + **Get-childterm**

This command grabs the parameter sets of the Get-ChildItem command

**Syntax**: **Get-Command -Name Get-Childitem -Syntax**

We can replace Get-ChildItem with any command we want to get the syntax for.

We are calling "Get-command" on the cmdlet named "Get-Childitem" to see what the accepted syntax is.



* + - **Get-Member**
      * The Get-Member cmdlet gets the members, the properties and methods, of objects.
      * **Syntax: get-process | get-member**
      * Takes Get-Process and pipes it into Get-Member so that we can see all the methods and properties of that object. Think of the methods as “things we can do with these objects,” and properties as “things these objects are.”
* **PS Objects**
  + Made up of three types of data:
    - Object type
    - Properties
    - Methods

.

* **Parameters:**
  + Parameters:
    - NAME
    - SYNOPSIS
    - SYNTAX
    - DESCRIPTION
    - RELATED LINKS
    - REMARKS
  + As you can see, help topics can contain an enormous amount of information and this isn’t even the entire help topic.
  + Take a moment to run that example on your computer, review the output, and take note of how the information is grouped:
  + While not specific to PowerShell, a parameter is a way to provide input to a command. Get-Help has many parameters that can be specified in order to return the entire help topic or a subset of it.
  + The syntax section of the help topic shown in the previous set of results lists all of the parameters for Get-Help.
  + At first glance, it appears the same parameters are listed six different times. Each of those different blocks in the syntax section is a parameter set. This means the Get-Help cmdlet has six different parameter sets. If you take a closer look, you’ll notice that at least one parameter is different in each of the parameter sets.
* **Parameter Sets**
  + Parameter sets are mutually exclusive. Once a unique parameter that only exists in one of the parameter sets is used, only parameters contained within that parameter set can be used. For example, both the Full and Detailed parameters couldn’t be specified at the same time because they are in different parameter sets.
  + Each of the following parameters are in different parameter sets:
    - Full
    - Detailed
    - Examples
    - Online
    - Parameter
    - ShowWindow
* **Wildcard**
  + A wildcard character “\*”, simply matches any and all string filters.
  + **Syntax: Get-Command Get\***
  + We are calling "Get-command" and filtering on command that starts with Get

**COL**

What does the get-member cmdlet do?

**Gets members (properties, methods, etc) of an object**

True or False: Parameter sets are not mutually exclusive.

**False**

* **PS Aliases**
  + An alias is an alternate name for a cmdlet, function, executable file, including scripts. PowerShell includes a set of built-in aliases. You can add your own aliases to the current session and to your PowerShell profile.
  + PowerShell supports the use of common aliases such as cls (clear the screen) and ls (list the files). PowerShell includes built-in aliases that are available in each PowerShell session. So new users can use their knowledge of other frameworks and don’t necessarily have to remember the PowerShell name for familiar commands.
  + We can add new aliases for our own session of Powershell.
  + To create an alias, use the cmdlets Set-Alias or New-Alias.
  + Syntax: **Set-Alias -Name list -Value Get-Childitem**
  + We can verify our alias was created using the Get-Alias cmdlet.
  + **Syntax: Get-Alias -Name list**
  + We have now created an alias called list. This will function EXACTLY the same as Get-CSet-hildItem.
  + Using the Definition parameter, you can find all the aliases of a command. Let’s find all the current aliases of the Get-ChildItem cmdlet.

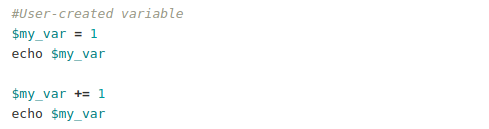
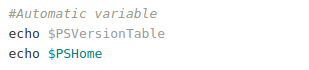
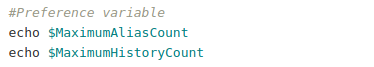
**COL**

What is the full command to create an alias?

**Set-alias -name <aliasNAME> -value <cmdNAME>**

True or False: The get-alias cmdlet can verify an alias was created

**True**

* **Variables**
  + A variable is a unit of memory in which values are stored. In PowerShell, variables are represented by text strings that begin with a dollar sign ($), such as $a, $process, or $my\_var.
  + Variable names are not case-sensitive in Powershell, and can include spaces and special characters.
  + **3 types of Variables**
    - **User Created Variables**
      * ***User-created variables are created and maintained by the user.*** By default, the variables that you create at the PowerShell command line exist only while the PowerShell window is open.
      * When the PowerShell window is closed, the variables are deleted.
      * To save a variable, add it to your PowerShell profile.
      * You can also create variables in scripts with global, script, or local scope.
      * 
    - **Automatic Variables**
      * ***Automatic variables store the state of PowerShell.*** These variables are created by PowerShell, and PowerShell changes their values as required to maintain their accuracy.
      * **Users can’t change the value of these variables.** 
        + For example, the $PSHOME variable stores the path to the PowerShell installation directory.
      * 
    - **Preference Variables**
      * ***Preference variables store user preferences for PowerShell***. These variables are created by PowerShell and are populated with default values.
      * **Users can change the values of these variables.** 
        + For example, the $MaximumHistoryCount variable determines the maximum number of entries in the session history.
      * 

**COL**

List some of the different types of variables discussed in Powershell

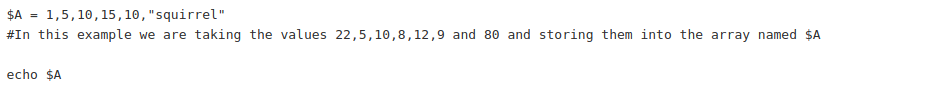
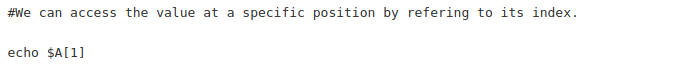
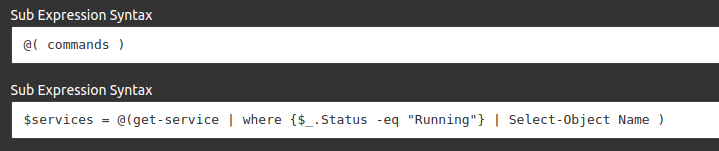
**User created**

**Automatic**

**Preference**

True or False: Users can change preference variables

**true**

* **Arrays**
  + An array is a data structure that is designed to store a collection of items. The items can be the same type or different types. An array is organized by a zero-based index.
  + To create and initialize an array, assign multiple values to a variable.
  + The values stored in the array are delimited with a comma and separated from the variable name by the assignment operator (=).
    - 
  + This data is now associated in memory with the array named $A. Anytime we want to access this information we can interact with the array it is stored in.
    - 
  + The value 5 is stored at index 1.
  + $A[-2] would be equal to 10 because it basically starts at the back of the array and goes from there for negative indices
* **Array Sub-Expressions**
  + The array sub-expression operator creates an array from the statements inside it. Whatever the statement inside the operator produces, the operator will place it in an array. Even if there is zero or one object.
    - 
  + Now we have an array named $services that is a collection of all the service names with a status of "Running.”
  + We can now interact with our service names as if they were just a regular array.
    - 
  + We’ve taken output from a commandlet and have organized it into an array for further data processing or analyzing.
  + One of the properties of an array is the length. This is simply how many objects are in an array. We can access this value using the .length method of an array.

.

* **Hash Tables**
  + A hash table, also known as a dictionary or associative array, is a compact data structure that stores one or more key/value pairs.
  + For example, a hash table might contain a series of IP addresses and computer names, where the IP addresses are the keys and the computer names are the values, or vice versa.
  + To create a hash table, follow this syntax:
    - 
  + We now have an array where keys are associated with their corresponding values.
  + Unlike a regular array hash tables use key based indexing. We cannot refer to values in their numerical position. We have to reference the specific key in order to access the value.
    - 

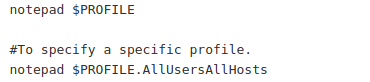
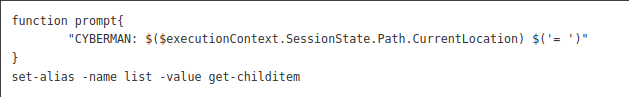
**COL**

What is an array?

**data structure that is designed to store a collection of items.**

What are the values stored in an array delimited with?

**A comma**

* **Powershell Features**
  + You can create a PowerShell profile to customize your environment and to add session-specific elements to every PowerShell session that you start.
  + A PowerShell profile is a script that runs when PowerShell starts. You can use the profile as a logon script to customize the environment.
  + You can add commands, aliases, functions, variables, snap-ins, modules, and PowerShell drives. You can also add other session-specific elements to your profile so they are available in every session without having to import or re-create them.
* **Powershell Profile Files**
  + *PowerShell supports several profiles for users and host programs. However, it does not create the profiles for you.*
  + 
  + **Editing a Profile**
    - You can open any PowerShell profile in a text editor, such as Notepad.
    - Here we are taking advantage of the automatic variable $PROFILE
      * 
    - Test the path of the $PROFILE to see if it exists
      * 
    - It may show false because the file doesn’t exist, so let’s create it.
      * 
    - Now if we test it again it should show us true. Type $PROFILE to see what the default path to the file we just created is. Navigate to it and open it up. In here, the profile file, we can make any changes that we want to happen everytime we open the console.
    - We are going to add the following to our profile:
      * 
    - To apply the changes, save the profile file, and then restart Powershell

**COL**

True or False: A powershell profile is a script that runs when PowerShell starts?

**true**

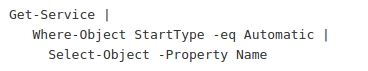
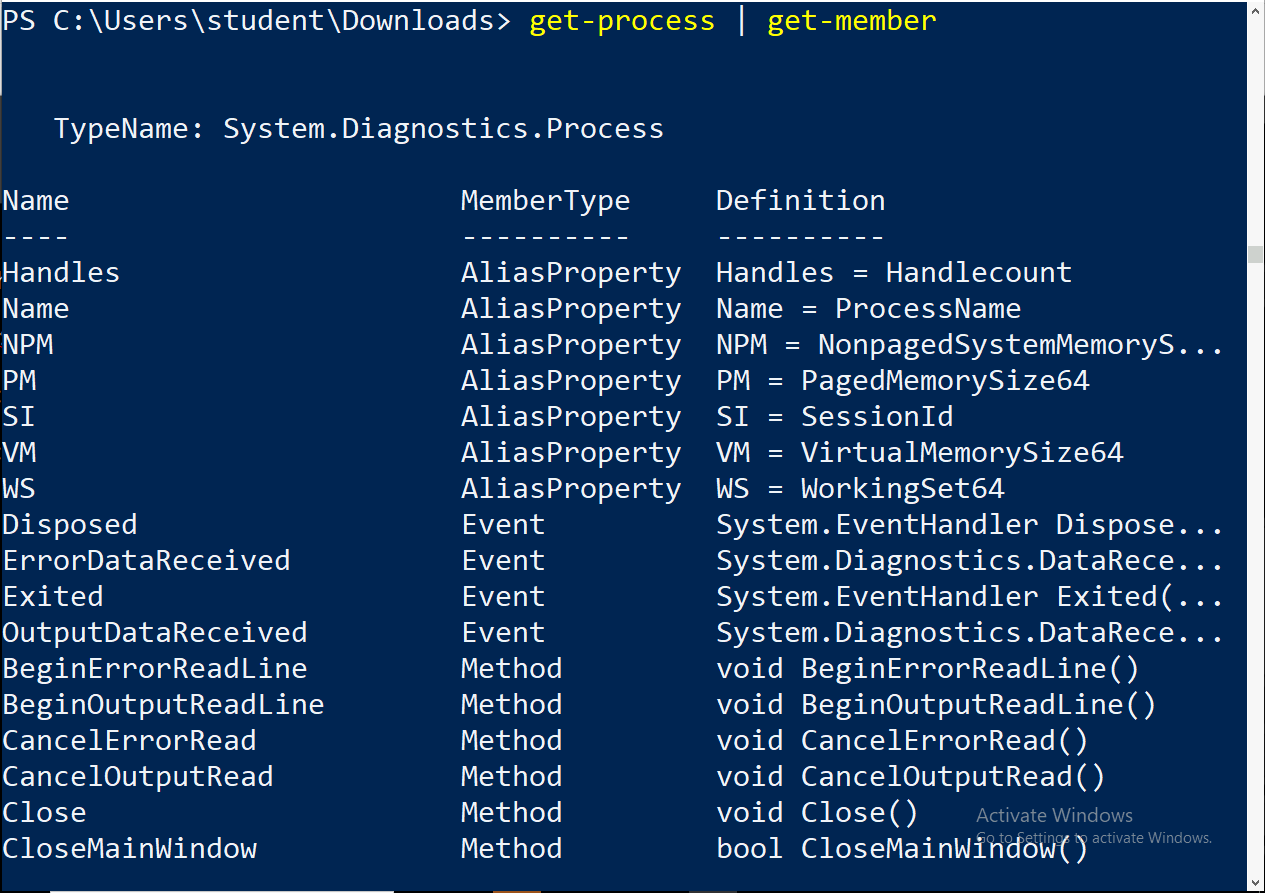
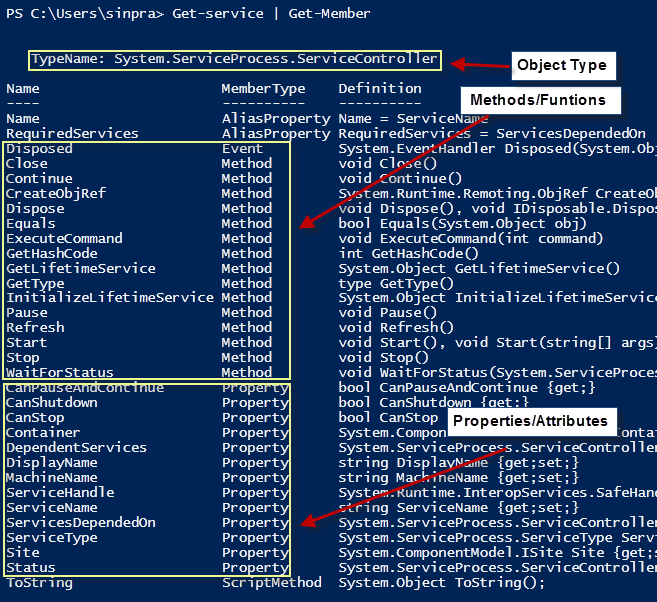
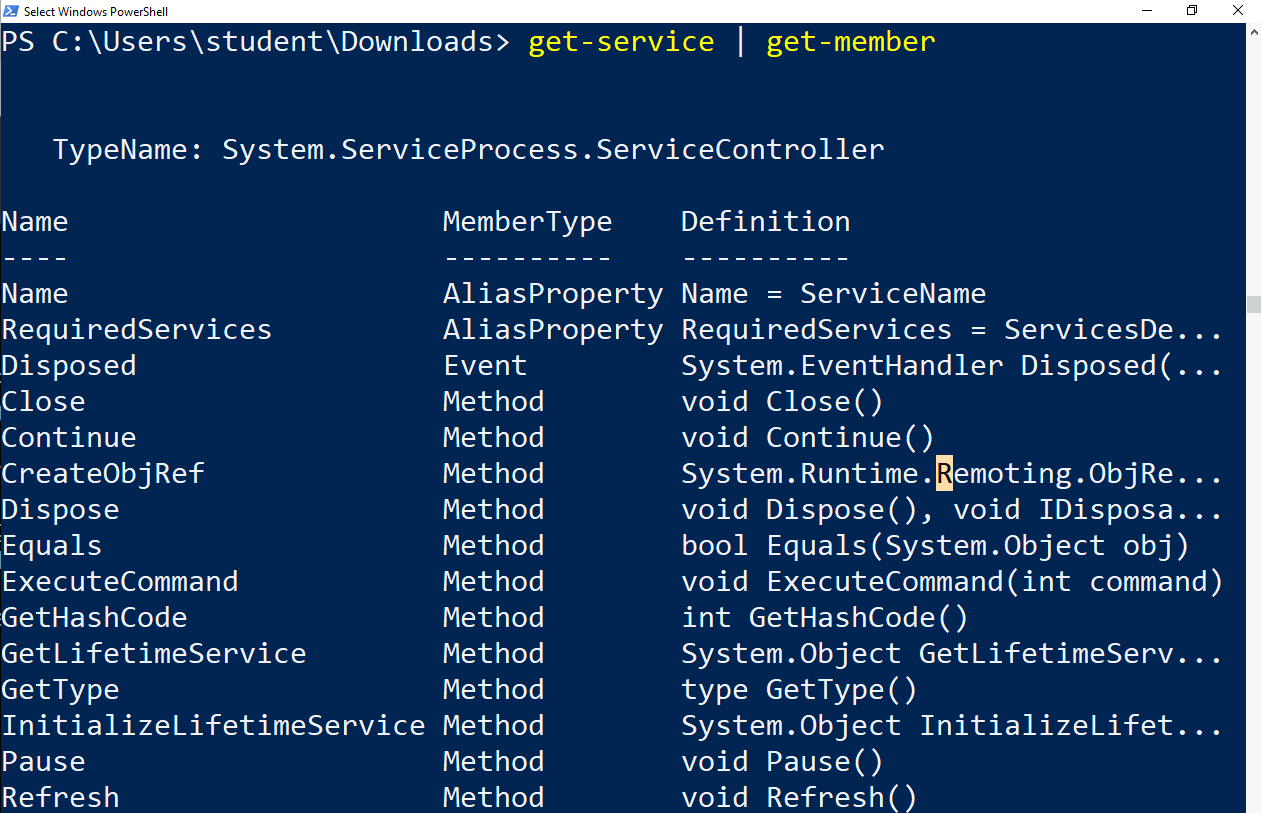
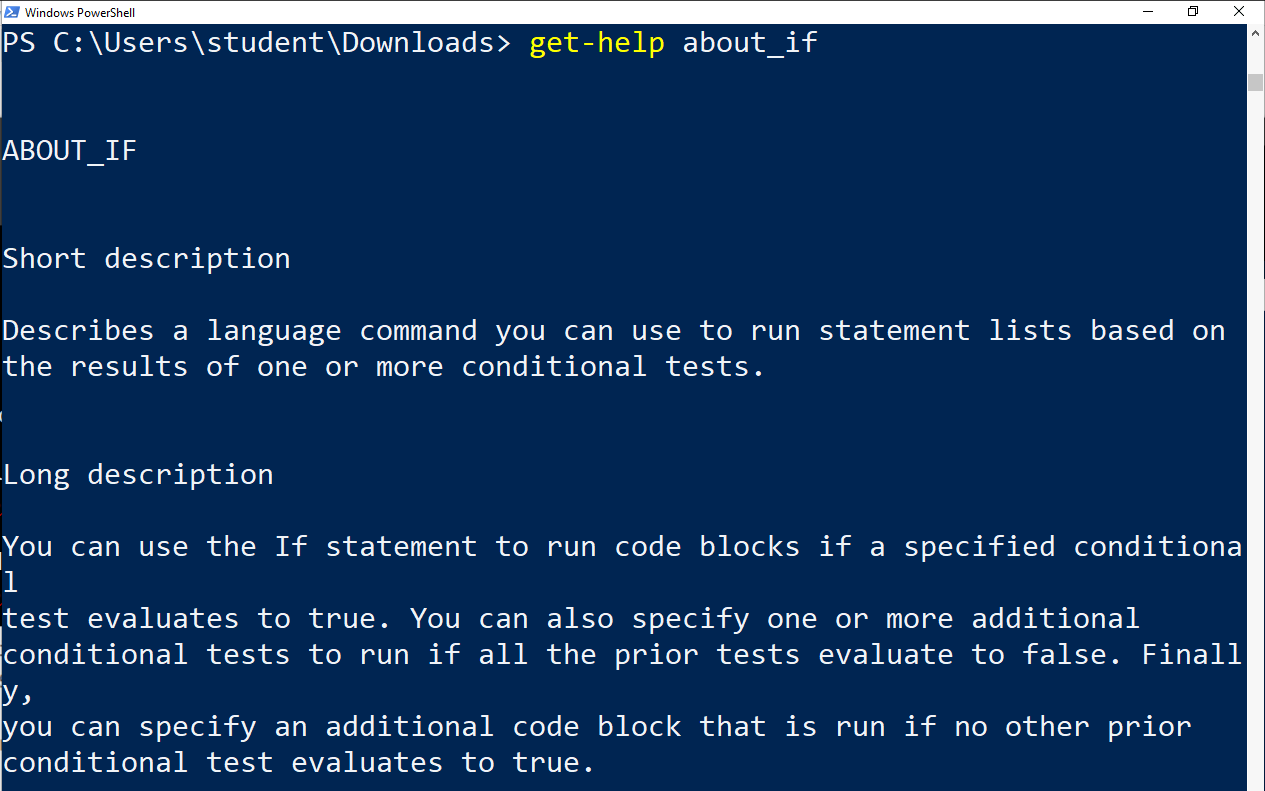
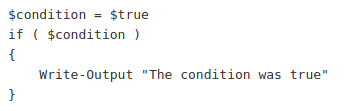
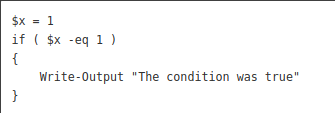
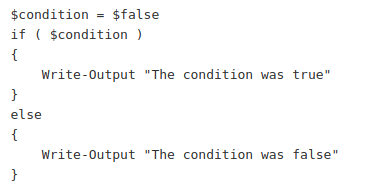
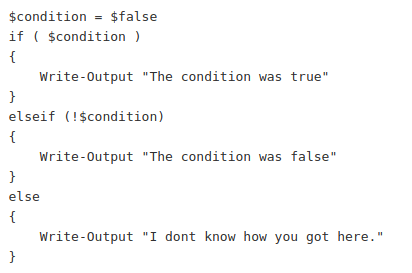
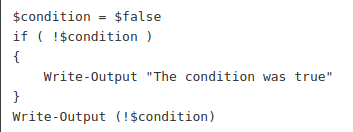
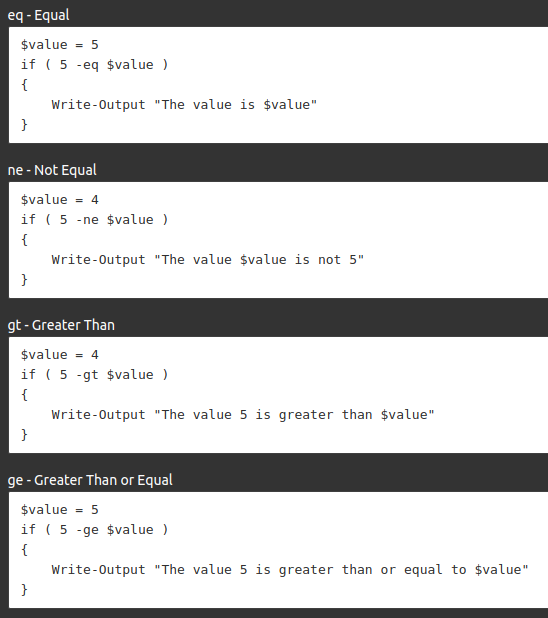
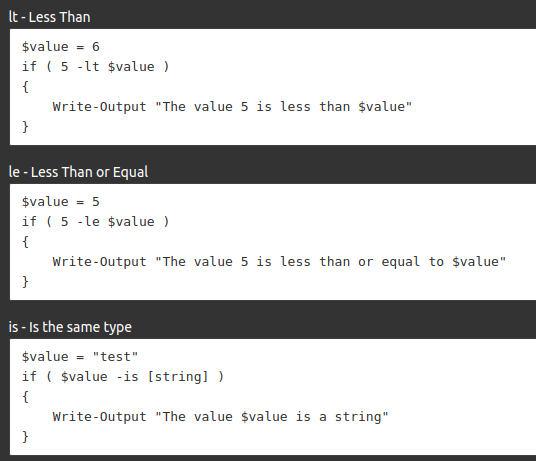
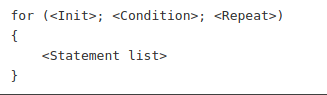
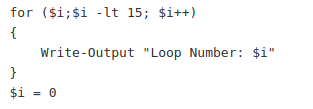
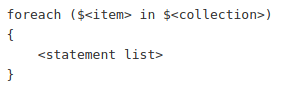
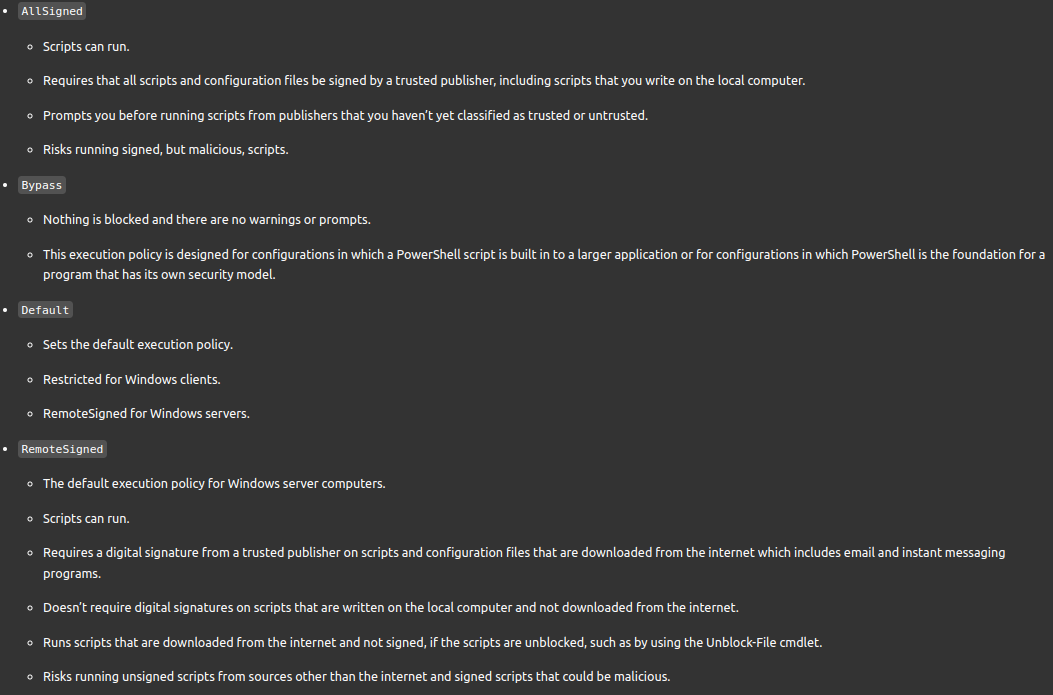
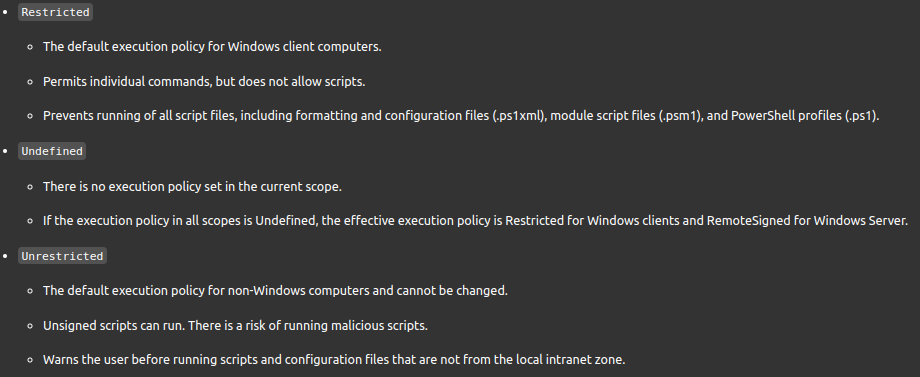
List two of the PowerShell profiles

**All users/all hosts**

**Current user/current host**

* Observing the minimal help documentation for “get-help get process” which group contains a notice to see more examples?
  + **REMARKS**
* Observing the -full help documentation of the get-command, which group contains information about piping into the cmdlet?
  + Make sure PS is updated to 5.1.19041.4170
  + **INPUTS**

02\_PS\_Objects

* **Powershell Objects**
  + PowerShell is an object-oriented language and shell. This is a departure from the traditional shells like cmd and Bash.
  + These traditional shells focused on text aka strings and while still useful, are limited in their capabilities. **Nearly everything in PowerShell is an object**.
  + We can maximize Powershell’s potential by leveraging its object oriented structure to accomplish our goals.
* **Powershell Pipeline**
  + ***A pipeline is a series of commands connected by pipeline operators: |***
  + Each pipeline operator sends the results of the preceding command to the next command.
  + The output of the first command can be sent for processing as input to the second command. And that output can be sent to yet another command.
  + The result is a complex command chain or pipeline that is composed of a series of simple commands.
  + 
  + In the above example, we started an instance of notepad.exe
  + We then use Get-Process with the name of "notepad" and PIPE that object into the cmdlet Stop-Process.
  + The notepad process then stops. We took the object from the left side of the pipeline and piped it as the input to the right side of the pipeline.
  + Simply put, the pipeline takes the output of an object and pipes it as the input of the next item in the pipe.
* **One Liner**
  + ***A PowerShell one-liner is one continuous pipeline and not necessarily a command that’s on one physical line. Not all commands that are on one physical line are one-liners.***
  + Even though the following command is on multiple physical lines, it’s a PowerShell one-liner:
  + 
  + Let’s break down what is happening in this one liner:
    - Step 1: The cmdlet Get-Service is ran and instead of the output being printed to the screen it is then piped into the next item in the pipe.
    - Step 2: The Where-Object cmdlet takes the previous item in the pipes output (in this case a Collection of services) as input and outputs each object that matches the Where-Object filter clause. The filter is looking for any service from the collection whose property "StartType" equals "Automatic".
    - Step 3: The Select-Object cmdlet takes the collection of objects from the previous pipe as input, it then selects only the objects property called "Name". Because it is the last pipe, it outputs it to the screen.
* **Powershell Objects**
  + Objects have many different types of information associated with them.
  + In PowerShell, this information is sometimes called members.
  + An object member is a generic term that refers to all information associated with an object.
  + An object can have multiple members. But each member belongs to a single Object.
  + 
  + To discover information about an object, you can use the Get-Member cmdlet.
  + The Get-Member cmdlet is a handy command that allows you find available properties, methods, and so on for any object in PowerShell.
  + For example: If we take the cmdlet Get-service and pipe it into Get-Member:
  + **Get-service | Get-Member**
  + We can see that the Get-service cmdlet returns a System.serviceProcess.t. This object has members that belong to that object.
  + 
  + **We can Pause, Start, and Stop services using the Methods: Pause, Stop and Start.**
  + **These are actions taken on an object.**
  + We can get information about each service by looking at the Properties of that object.
  + Properties such as DisplayName, ServiceName and Status etc.
  + 
  + The Method members are Methods that can be called.
    - Methods are actions that can be taken on an object. In the human example, a Method of the Human Object would be Walk, Run, Breath.
    - These are actions that immediately affect that object that calls them.
  + The Property members are attributes that describe an object.
    - An object can have many different properties attached to it representing various attributes.
  + If we want to search for these properties we can use the Where-Object.
  + All procedural programming languages require constructs that allow them to execute instructions that are dependent on certain conditions.
  + Procedural programming is a programming paradigm, derived from imperative programming, based on the concept of the procedure call.
  + Procedures simply contain a series of computational steps to be carried out.
* **If Statement**
  + Because Powershell is procedural in nature, it has built in conditionals that allow certain blocks of code to be executed or ignored based on a specific condition.
  + These conditions in Powershell are determined using “If” statements.
  + The first thing the if statement does is evaluate the expression in parentheses.
  + If it evaluates to $true, then it executes the scriptblock in the braces.
  + If the value was $false, then it would skip over that scriptblock.
  + 
  + 
  + 
* **Else statement**
  + The else statement does not accept any condition.
  + The statement list in this statement contains the code to run if all the prior conditions tested are false.
  + Else will always resolve to $True if all conditions prior to it are False.
  + Think of the else as a last resort condition.
  + 
* **ElseIf Statement**
  + The elseif statement is where you add conditions.
  + You can add multiple ElseIf statements when you multiple conditions.
  + PowerShell will evaluate each of these conditions sequentially.
  + 
* **Comparison Operators**
  + The most common use of the if statement is comparing two items with each other.
  + PowerShell has special operators for different comparison scenarios.
  + When you use a comparison operator, the value on the left-hand side is compared to the value on the right-hand side.
* **Not -! Operator**
  + not - !
  + The not operand simply inverts the boolean value of a condition or boolean object.
  + A false statement becomes true and a true statement becomes false, just like in Computer Organization and Architecture.
  + 
* **Other Operators**
  + 
  + 
* **For Loop**
  + Init - The Init placeholder represents one or more commands that are run before the loop begins.
  + Condition - The Condition placeholder represents the portion of the For statement that resolves to a $true or $false Boolean value. PowerShell evaluates the condition each time the For loop runs. If the statement is $true, the commands in the command block run. The loop is repeated until the condition becomes $false.
  + Repeat - The Repeat placeholder represents one or more commands, separated by commas, that are executed each time the loop repeats.
  + Statement - The Statement list placeholder represents a set of one or more commands that are run each time the loop is entered or repeated. The contents of the Statement list are surrounded by braces.
  + 
  + 
* **For Each Loop**
  + A foreach loop reads a set of objects (iterates) and completes when it’s finished with the last one.
  + The simplest and most typical type of collection to traverse is an array. Within a foreach loop, it is common to run one or more commands against each item in an array.
  + The difference between a For loop and Foreach is that a for loop is typically looped until a condition is true.
  + It will loop forever until that condition is true.
  + In a foreach loop a set amount of loops are determined based on the size of a collection.
  + The $<item> is created locally within the loop and the variable is name by the user.
  + 
* **PS Scripts**
  + A script is a plain text file that contains one or more PowerShell commands. PowerShell scripts have a .ps1 file extension.
  + Running a script is a lot like running a cmdlet. You type the path and file name of the script and use parameters to submit data and set options. You can run scripts on your computer or in a remote session on a different computer.
  + Writing a script saves a command for later use and makes it easy to share with others. Most importantly, it lets you run the commands simply by typing the script path and the filename. Scripts can be as simple as a single command in a file or as extensive as a complex program.
  + Scripts have additional features, such as the #Requires special comment, the use of parameters, support for data sections, and digital signing for security. You can also write Help topics for scripts and for any functions in the script.
* **Execution Policy**
  + Before you can run a script on Windows, you need to change the default PowerShell execution policy. Execution policy does not apply to PowerShell running on non-Windows platforms.
  + In order to change the execution policy we will use the built-in powershell cmdlet Set-ExecutionPolicy.
  + 
  + 
  + You can find all the descriptions and differences by checking out the help page for about\_Execution\_policies

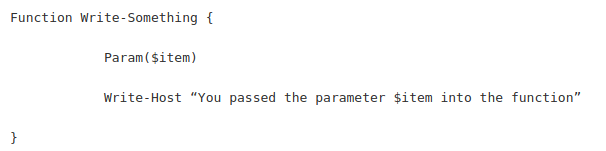
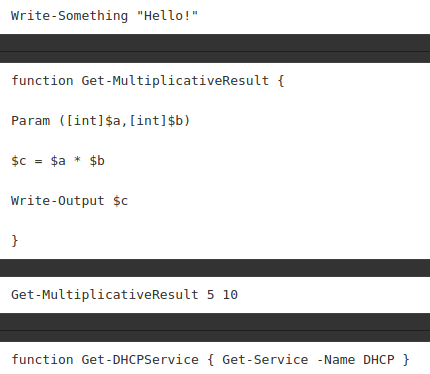
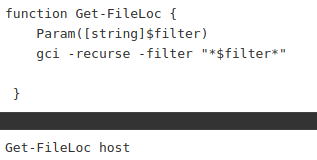
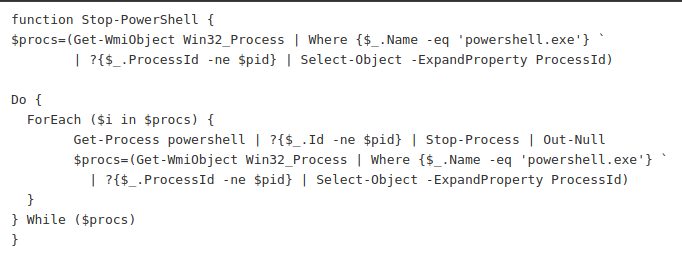
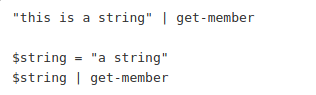
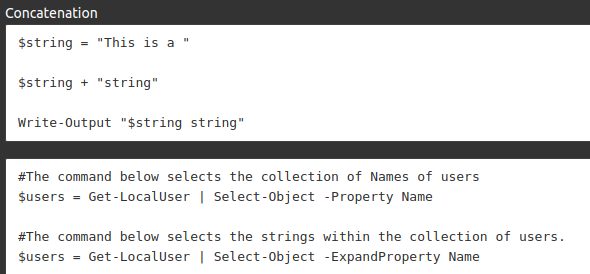
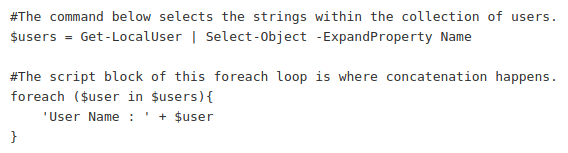
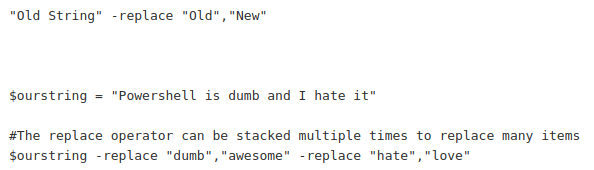
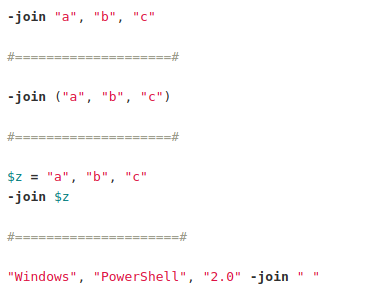
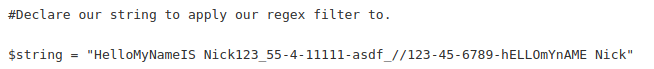
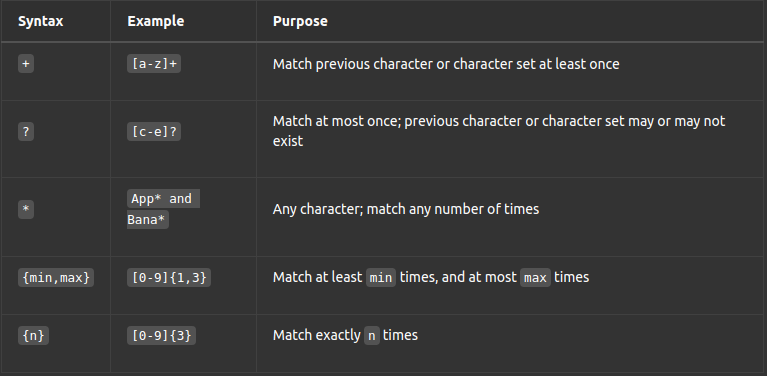
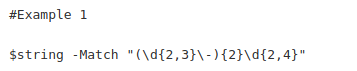
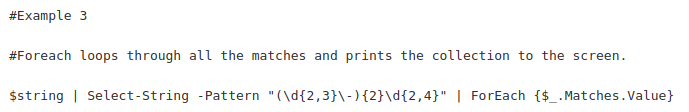
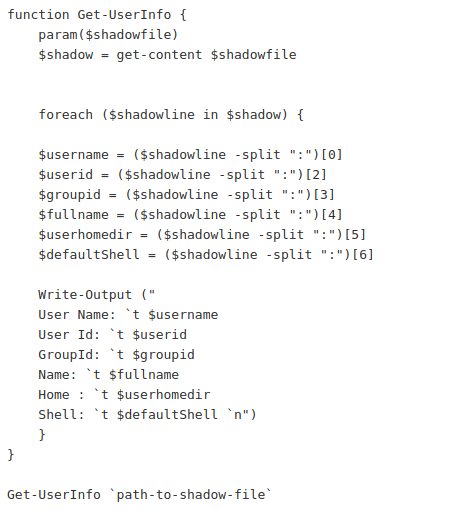
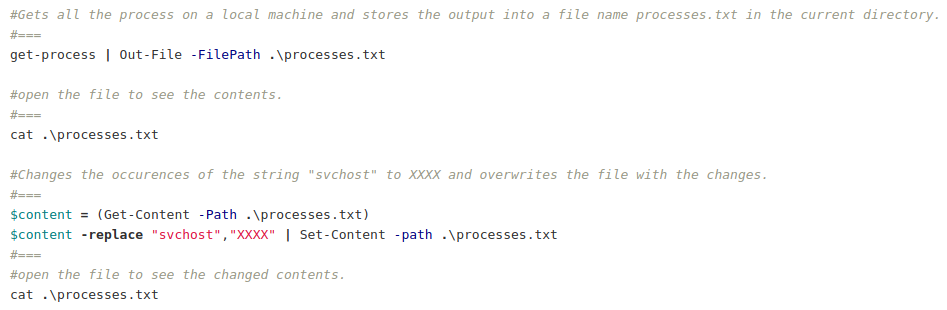
**PE Help**

* If the following text file “test.txt” exists in $home with the words “hello world” written inside of it, what will be the outcome if I run the command:
  + Get-content $home\test.txt | Remove-Item

03\_PS\_Functions

* **` (back quote)**
  + Sends signal PS to go to another line
  + Like:
    - Asdfsadf `

| asdfdfa

* + - That first line pipes into the second line
* **Powershell Functions**
  + A function in PowerShell is a grouping of code that has an optional input and output.
  + **Developers, when making a script or function, use parameters to provide input at runtime**
  + It’s a way of collecting up a lot of code in order to perform one or more times by just pointing to it instead of duplicating that code repeatedly.
  + Think of it as a method in objects. Anything within your script can call it.
  + A function is a list of PowerShell statements that has a name that you assign. When you run a function, you type the function name. The statements in the list run as if you had typed them at the command prompt.
  + Functions need to be re-declared if the terminal exits and reopens
  + **Functions take priority over built in commands**
    - Function Get-ChildItem would run over Get-Command Get-ChildItem
      * You can get around this by having (Get-Command Get-ChildItem).invoke() in your function to also call the other name
      * ***This is bad practice, unless used exploitatively*** 
        + *For instance you can have Get-ChildItem invoke the real command but also create a file full of ipconfig info*
  + Functions can be as simple as:
  + 
  + 
  + This is a function that we can call like a cmdlet.
  + 
  + We’ve just created an easier way to get the DHCP service.
  + Functions are useful if you are reusing code alot.
  + You can put all of your reused code into a single function statement and call the code when you reference the name of the function.
  + The example here is the same idea but a little more complicated.
  + 
  + Open up powershell and let’s build this function:
  + 
* **String Manipulation**
  + It is important to understand that strings in PowerShell are always objects, whether you are dealing with literal strings or variables.
  + Consequently, the methods of string objects provide most of the functions you need. As usual, you can display them with the Get-Member cmdlet.
  + 
  + Like we learned in objects, a variable is just a name that represents an object. You can see that when we can get a string literal and a string variable we get the same output because we are dealing with a string object in both scenarios.
  + We can use the built in string methods to manipulate and interact with string objects.
* **Select-Object**
  + The difference in the command below is that Select-Object -Property creates a collection of the different user object names. We would then have to further convert the results of this command to get our output to a string.
  + 
* **Expand Property**
  + The Select-Object -ExpandProperty, takes the property argument passed, in this case Name and converts it to a string that is ready for string manipulation.
  + 
* **Replace**
  + The replace operator replaces instances of string occurrences (old values) with the specified string (new value).
  + 
* **Join**
  + The join operator concatenates a set of strings into a single string. The strings are appended to the resulting string in the order that they appear in the command.
    - No parentheses, it will create an array
    - With parentheses, it concatenates everything
      * Joining an array of strings will concatenate it all into one string
      * Join at the end with a “ “ will add a space in between each letter
  + 
* **Split**
  + The split() method is on every string object and is capable of splitting a string into an array based on a non-regex character.
    - Splits strings into an array
  + 
* **Regular Expressions**
  + Regular Expressions (REGEX) are used synonymously across many programming language variants, and serve as a set of tools for filtering strings in large data outputs.
  + Their value cannot be overstated, as they have helped operators and analysts accomplished the most mundane of tasks to the most extreme and complex.
  + Think of regex as [CTRL]+F on adrenaline.
  + 
  + 
* **Match and Pattern**
  + In Regex the \d character class is used to designate numbers, just as the [0-9] character class does.
  + The - is escaped as it’s usually used in regex to denote a character range.
  + The {2,3} repetition quantifier denotes that whatever is previously stated, in this case the character class \d, which again is any number 0-9, must contain at least 2 numbers, and may be repeated up to three instances in the string to be matched.
  + The repetition quantifier, {1,} is often used to denote one or more, when the quantity to match of a designated character class is not known.
  + The -Match shell built-in parameter (Example 1) can be used to match a regex pattern of any provided string.
  + 
  + To match a string using the contents of a file(s), the CMDLET Select-String (Example 2) is used with the parameter -Pattern, which unlike the -Match parameter, does not just return a boolean indication of match.
  + Instead returns the entire line of the actual string that was matched.
  + 
  + Passing the complete line of output with matching string to the ForEach loop, discussed in a previous lesson, used in conjunction with methods .Matches(), and .Value().
  + Example 3 extracts only the string that the -Pattern Select-String parameter matches.
  + 
* **Regex**
  + This command goes and grabs a pentesting guide on the internet and stores the content of that page to a file called ptes.txt:
  + Invoke-WebRequest -Uri "http://www.pentest-standard.org/index.php/PTES\_Technical\_Guidelines" -UseBasicParsing | Select-Object -ExpandProperty Content | Out-File ptes.txt
  + We can see that this file is \*HUGE\*. It would take us hours to skim through it without a regex. It’s over a Gigabyte:
  + (Get-Content .\ptes.txt).Length
  + Get-Content .\ptes.txt | Select-String -Pattern "nmap" | % {$\_.Matches.Value} | Select -First 2
  + Get-Content .\ptes.txt | Select-String -Pattern "nmap.+" | % {$\_.Matches.Value} | Select -First 30 | Select -Last 10
  + Get-Content .\ptes.txt | Select-String -Pattern "nmap\s\-.+" | % {$\_.Matches.Value} | Sort -Unique
  + We are able to grab contextual information by filtering on the string "nmap" using regex without having to search this file manually. Reinforce this concept to the students and remind them how powerful regex is.
* **Anchors**
  + Two regex characters we can use separately or together are the ‘^’ and ‘$’. Both are referred to as anchor characters as they search for patterns that are anchored to one end or the other.
  + ^ - searches for pattern at the beginning of a line
  + $ - searches for a pattern at the end of a line
* **Powershell Scripts**
  + 
* **Search and Replace Strings in a File**
  + 

**PE HELP**

* A sequence of characters that specifies a search pattern
  + Regex
* Regex is short for:
  + Regular expression

PEs

**MOD F01: PowerShell Fundamentals (PE1)**

Note

Before Starting these practical exercises:

Run Powershell as an administrator

[Net.ServicePointManager]::SecurityProtocol = [Net.SecurityProtocolType]::Tls12

Update-Help -Verbose -Force -ErrorAction SilentlyContinue

2 The PowerShell format for cmdlets is

**Verb/Noun**

3 (T or F) Powershell is an open-source proprietary programming language developed by the Linux foundation.

**False**

4 All output and inputs from PowerShell are **.NET Objects**.

5 PowerShell methods are **Actions** that can be taken by an object.

6 Which of the following cmdlet will provide PowerShell help functionality?

**Get-Help**

7 I want to see examples of the cmdlet get-process. What would I type in?

**Get-Help Get-Process -Examples**

8 **Pipelines** are used to run many commands sequentially.

9 **Aliases** are alternate names that can be used to run commands.

10 A PowerShell object is made up of three types of data: **the object type, its properties, and its methods**.

11 The core cmdlets in PowerShell are built in **.NET Core**.

12 Cmdlet is often interchangeably called a **Command**.

13 Which of the following is a wildcard character for any string in Powershell?

**"\*"**

14 Observing the minimal help documentation for `get-help get-process` which group contains a notice to see more examples?

**Remarks**

15 Observing the -full help documentation of get-command, which group contains information about piping into the cmdlet?

**Inputs**

16 You've been entering gci in PowerShell forever since you copied and pasted it from online. What is it really running as?

**Get-ChildItem**

16 Create an alias every time np is entered will open notepad.exe

**Set-Alias -Name np -Value notepad.exe**

18 An input mechanism for users to select options or provide input is called **parameters**.

19 Variables that you create at the PowerShell command line exist only while the PowerShell window is open are called **User-Created Variables**.

20 Variables that store state information for PowerShell. These variables created and maintained by Windows PowerShell are called **Automatic Variables**.

21 Customized variables that affect the PowerShell operating environment and all commands that run in the environment are called **Preference Variables**.

22 Based on the following, what would be the outcome?

$apple = "pears","watermelon","heap","waterfall",15,10,"squirrel"

$apple[3]

**It will select a waterfall. It is the 4th element because indexes start at 0.**

23 Based on the following, what would be the outcome?

$apple = "pears","watermelon","heap","waterfall",15,10,"squirrel"

$apple[-4]

**It will select a waterfall. It is the 4th element when counting backwards. There is no index [-0]because reverse indexes start at [-1].**

**MOD F02: PowerShell Objects (PE2)**

1 (True or False) PowerShell is an object-oriented language and shell.

**True**

2 (True or False) A pipeline is a series of commands connected by "|".

**True**

3 (True or False) Every cmdlet can be piped to any other command.

**False**

4 I am running powershell with administrative permissions; Based on the following:

If the following text file "test.txt" exists in $home with the words "hello world" written inside of it, what will be the outcome if I run the following command?

Get-content $Home\test.txt | Remove-Item

**An error will occur looking for "C:\Users\student\hello world"**

5 Based on the following, what is Standard-In for Where-Object?

Get-ChildItem C:\Windows\System32 | Where-Object {$\_.Name -like "bad\*"}

**The Standard out of Get-ChildItem**

6 Based on the following, what is Standard-In for Where-Object?

Get-ChildItem C:\Windows\System32 | Where-Object {$\_.Name -like "bad\*"}

**The output of the pipelined series anything in system32 starting with bad**

7 Based on the following "-eq Automatic " is what?

Get-Service | Where-Object StartType -eq Automatic | Select-Object -Property Name

**Argument**

8 From the following what is "fl"

Get-Process -id 1360 | fl \* -Force

**Format-List**

9 Property members are **attributes** that describe an object.

10 Based on the following "Select-Object" is what?

Get-Service | Where-Object StartType -eq Automatic | Select-Object -Property Name

**Statement**

11 Which of the following is not a method available in get-process? (In Powershell version 5)

Get-Process | Get-Member -MemberType Method

**"Strings"**

12 Which of the following is not a method available in get-help?

Get-Help | Get-Member -MemberType Method

**"Length"**

13 Which of the following is not a method available in get-typedata?

Get-TypeData | Get-Member -MemberType Method

**"Members"**

14 What is the GeoID member type in Get-WinHomeLocation?

Get-WinHomeLocation | Get-Member

**"Property"**

15 If given the following code, what would you write in the if condition to evaluate the statement as true and output "You win!"

$teehee="xD"

if ($teehee -eq "xD")

{

write-output "You win!"

}

**MOD F03: PowerShell Functions**

1 In PowerShell grouping code together and giving it a name, so that we can call it by that name later, is called **Function**.

2 How do you run a function?

**Enter the name of function**

3 Based on the following, what is the outcome once you run the function?

$arg="Talking Cat"

function hello { "Hello there $arg, how are you?" }

$arg="Schrodinger"

**Hello there Schrodinger, how are you?**

4 Developers, when making a script or function, use this to enable script users to provide input at runtime.

**Parameters**

5 Based on the following, what is the function name?

$arg="Talking Cat"

function hello { "Hello there $arg, how are you?" }

$arg="Schrodinger"

**"Hello"**

6 Missing Question

7 (True or False) Functions declared will remain if the terminal exits and reopens.

**False**

8 Functions you want to remain persistent should be defined in/at/with

**PowerShell Profile**

9 (True or False) Strings in PowerShell are always objects.

**True**

10 The following are two strings when running the command will \_\_\_\_\_\_\_ the strings

$string = "Wubba Lubba"; $string + " Dub Dub"

**Concatenate**

11 What takes a property argument passed and converts it to a string for string manipulation?

**Expand Property**

12 This operator will take a string occurrence and put a new one in its place.

**Replace**

13 What will be the output of the following?

$YearofCovid = "Sometimes I think if all this is happening because I didn't forward that email to 10 people"

$YearofCovid -replace "think","wonder" -replace "email","meme"

**Sometimes I wonder if all this is happening because i didn't forward that meme to 10 people**

14 What will be the output of the following?

"Be8azgoodhpersonxbutzdont8wasteztimextrying8tohprove it." -split {$\_ -eq "8" -or $\_ -eq "h" -or $\_ -eq 'x' -or $\_ -eq "z"}

**Be a good person but don't waste time trying to prove it.**

15 Based on the following, two words will be on the same line which is one of them

"Be8azgoodhpersonxbutzdont8wasteztimextrying8tohprove it." -split {$\_ -eq "8" -or $\_ -eq "h" -or $\_ -eq 'x' -or $\_ -eq "z"}

**prove**

16 A sequence of characters that specifies a search pattern.

**Regex**

17 REGEX is short for **Regular Expression**.

18 The following will result in True: 'big' -match 'b[iou]g

**True**

19 The following will result in True: 'bog' -match 'b[iou]g'

**True**

20 The following will result in True: 'boig' -match 'b[iou]g'

**False**

21 The result of the following would be? (Terrible question, but answer is true.)

write-host 100

100 -match '[0-9][0-9]'

**True**

22 Which of the following regex character ranges will match any number?

**\d**

23 Which of the following regex character ranges will match any word?

**\w**

24 The period "." will match any character except a **newline**.

25 Which quantifier is for zero or more times?

**"\*"**

26 Which quantifier is at least one or more times?

**"+"**

27 Which quantifier is for zero or one time?

**"?"**

28 In Powershell a variable $message contains text. Apply a regex to validate true if it begins with the word "error".

**$message -match “^error”**

29 Within the HR Employees file, count the total # of times the ": S" tag appears.

548

**$employees | Select-String ": S" -CaseSensitive | Measure-Object | Select-Object -ExpandProperty Count**

Note: I simply declared $employees = Get-Content "HR\_Employee\_list.txt' IOT to avoid typing Get-Content "HR\_Employee\_list.txt” every single time during PEs. You may see me use $employees and Get-Content "HR\_Employee\_list.txt' interchangeably below.

30 Within the HR Employees file, count the total # of times "Antarctica" appears anywhere in the file.

179

**Get-Content "HR\_Employee\_list.txt' | Select-String "Antarctica" | Measure-Object | Select-Object -ExpandProperty Count**

31 Within the HR Employees file, count the total # of times the "Na" appears at the beginning of the line.

1500

**Get-Content "HR\_Employee\_list.txt' | Select-String "^Na" | Measure-Object | Select-Object -ExpandProperty Count**

32 Within the HR employees file, count the total # of times a "." appears in the file.

7874

**($employees -join "" -replace "[^.]", "").Length** OR

**($employees | Select-String -Pattern "\." -AllMatches).Matches.Count**

33 Count the total # of times "84" appears in the middle of a Social Security Number.

25

**Get-Content "HR\_Employee\_list.txt” | Select-String "-84-" | Measure-Object | Select-Object -ExpandProperty Count**

34 Within the HR Employees file, count the total # of times two digits appear at the end of the line, using any combination of the numbers "1,2,or 3".

553

**Get-Content "HR\_Employee\_list.txt' | Select-String -Pattern "[123{2}$]" | Measure-Object**

35 Within the HR Employees file, count the total # of times "72" appears at the end of a line.

60

**Get-Content ".\HR\_Employee\_list.txt" | Select-String -Pattern "(72)$" -AllMatches | Measure-Object**

36 Within the HR Employees file, count the total # of unique area codes anywhere in the file.

716

**$employees | Select-String -Pattern "\((\d{3})\) \d{3}-\d{4}" -AllMatches | ForEach-Object {$\_.Matches.Groups[1].Value } | Sort-Object -Unique | Measure-Object**

37 Within the HR Employees file, count the total # of IPs in the 150.0.0.0/8 network anywhere in the file.

11

**($employees | Select-String -Pattern "\b150\.(\d{1,3})\.(\d{1,3})\.(\d{1,3})\b" -AllMatches).Count** or

**(Get-Content .\HR\_Employee\_list.txt | Select-String -Pattern "\b150\.(\d{1,3})\.(\d{1,3})\.(\d{1,3})\b" -AllMatches).Count**

38 Within the HR Employees file, count the total # of ".mil" emails in the file.

215

**(Get-Content .\HR\_Employee\_list.txt | Select-String -Pattern "\b\.mil\b" -AllMatches).Count**

39 Create a string that matches the following regex pattern[A-Za-z]{1,3}z[0-9a-z].\:"

**Abz8X: or az0.: or xyz3?: (Basically any expression that satisfies the above condition)**

40 Create a string that matches with the following pattern "((.?.?a[0-9]){2}){2}"

**a5a7a2a3**