Module 2: Powershell’s Scripting Language

* Use F8 to execute a specific line in the Powershell ISE
* Use Write-Output to output a variable
* Specify the type of data that will be in a variable **[int]$Variable=5**
* Use [validateset()] to set specific values that are valid for a variable
* Double quotes resolve variables, single quotes do not
* If statement syntax If ($this -eq $that) {

#commands

}

* Switch can be easier to maintain than If statements
* Do loop executes code while a certain condition is met
* ForEach executes code for each object in a group of objects

Module 3: Simple Scripts and Functions

* Param sets certain variables to be parameters
* [CmdletBinding()] turns the script into a cmdlet
* Use the function keyword to make the code in the { } a function

Module 4: Advanced Functions

* The purpose of advanced functions is to create functions that are similar to cmdlets
* Use ‘get-help \*function\*’ to find help files for functions
* Advanced functions have a Begin, Process, and End
* Begin and End run once and Process runs for each value in the pipeline
* Use ctrl + H to search and replace in powershell ISE
* ; is a statement terminator
* ; is necessary if you have multiple statements on the same line

Module 5: More on Parameters

* Use the param attribute [Parameter(Mandatory=$True)] to make parameters mandatory
* Pipe a command to clip to copy the output to the clipboard
* The [] in [string[]]$ComputerName will allow you to enter multiple values for the variable ComputerName
* Use the parameter attribute ValueFromPipeline=$true to make your cmdlet support pipeline input
* Use the parameter attribute HelpMessage to set a help message for the script
* Display the help message by typing !?
* Parameter attribute [ValidateCount()] allows you to set the amount of arguments the user can set to a parameter
* Use the [ValidatePattern()] parameter attribute to specify a pattern using regular expressions that input for a parameter must fit
* Use [ValidateLength()] parameter attribute to set how many characters the value of a variable can be

Module 6: Writing Help

* Put comment based help above the function
* Include multiple examples of how your program is intended to be used in your help page
* Use [Alias()] to create aliases for parameters

Module 7: Error Handling

* -ErrorAction can be abbreviated as -EA
* -ErrorVariable can be abbreviated as -EV
* -EA Stop will stop the command if it encounters an error
* -EA Inquire will open a window when the command encounters an error and you can select what to do
* In a Try Catch block, the Try code will be executed and if it encounters an error it will execute the Catch code

Module 8: Tools that Make Changes

* Values for Impact Level are Low, Medium, and High
* If the Impact Level of a script is set to High, you will be prompted to confirm the changes when running the script
* .ShouldProcess returns a boolean value, and if it returns True it will execute the code block
* Tools that make changes should include SupportsShouldProcess=$true

Module 9: Script and Manifest Modules

* Modules have the extension .psm1
* Use the Import-Module command to use the functions stored in a module
* When you edit a module you have to remove it and re-import it to test the changes
* Store Modules in \Documents\WindowsPowershell\Modules
* \WindowsPowershell\Modules doesn’t exist by default
* When you put modules in \Documents\WindowsPowershell\Modules you don’t have to manually import the module
* A module manifest has the extension .psd1
* The filename of the manifest needs to match the module folder name
* The cmdlet for making a module manifest is New-ModuleManifest