Powershell

Eric Claus

History of Powershell

What is Powershell?

- Powershell uses the .NET framework (same as C#), and is based on VBScript and batch (command prompt).
- Powershell is simpler to read and write than VBScript, and has much greater functionality than batch.
- Microsoft began development of Powershell, then called Monad, in 2002.
- Purpose was to unify the multiple Windows scripting languages and solutions into one, easy to use, all-purpose language.

Uses of Powershell

Why Use Powershell?

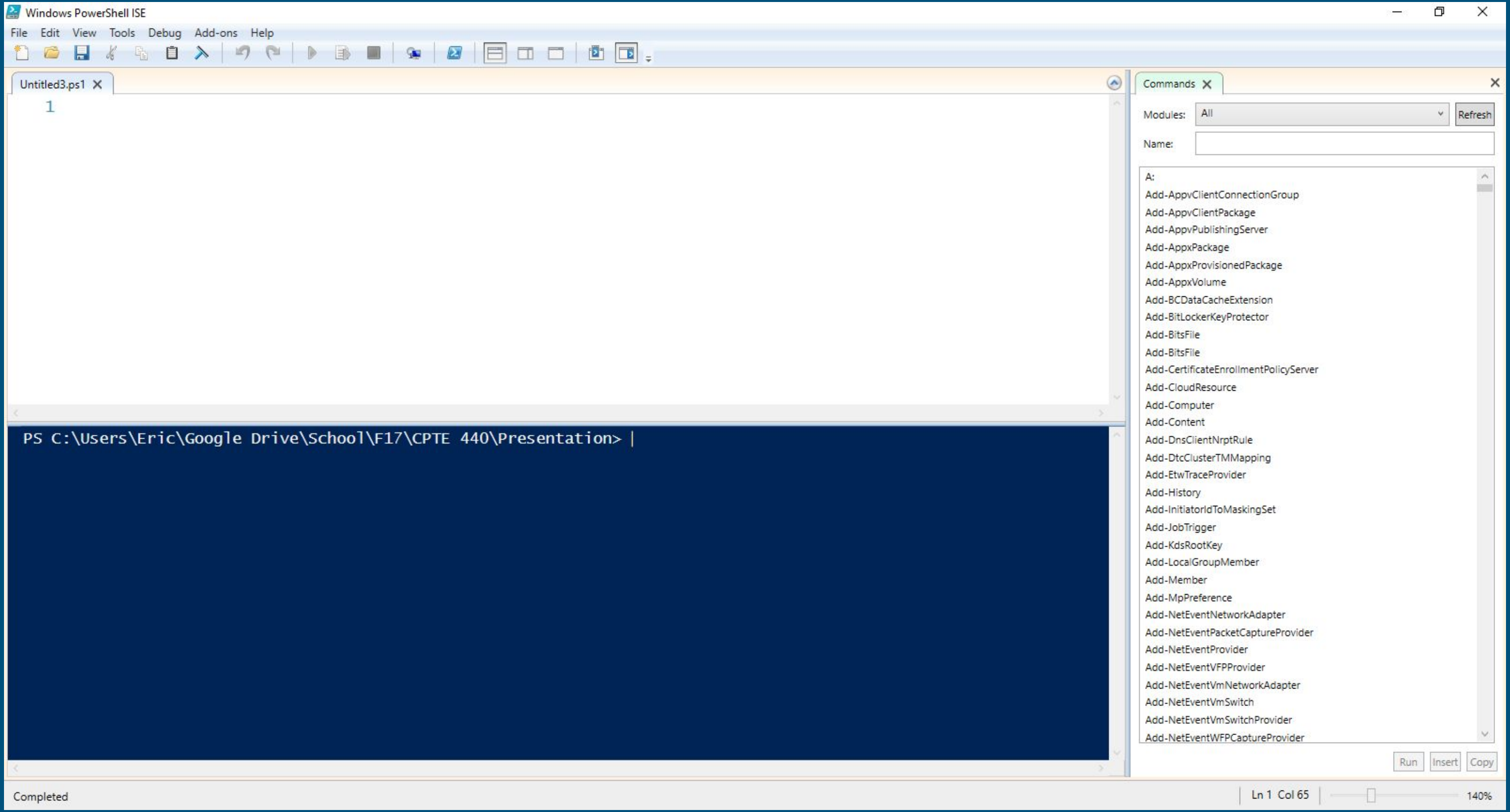
- Automate and consolidate processes
    - Windows admin processes
    - Just about anything else
- High level scripting language (easy to read and write)
- Anything that needs to be done on Windows, Powershell can do
- If you can do it in the GUI, you can do it in Powershell (plus some)
- Replaces and greatly expands upon CMD

Example Uses of Powershell

- Windows Server performance monitoring
- Backup
  - Files and folders
  - Hyper-V guest VMs
  - Applications and roles (Exchange, web servers, etc.)
  - Domain Controller functions
    - AD (system state)
    - DNS
    - DHCP
    - Group Policy
  - Config files from switches, firewalls, etc. via SSH
  - Anything Windows Server Backup and backup
- Management of network appliances via SSH
- Sending emails
  - Alerts, notifications, reminders
- Manage services
- Windows Server configuration
  - Security
  - Network interfaces
  - Roles and Features
  - Anything else
- Data processing
- Manage remote Windows systems
- Automate most Windows SysAdmin tasks

Basic Usage

Powershell ISE



```
PS C:\Users\Eric\Google Drive\School\F17\CPTE 440\Presentation> |
```

```
A:
Add-AppvClientConnectionGroup
Add-AppvClientPackage
Add-AppvPublishingServer
Add-AppxPackage
Add-AppxProvisionedPackage
Add-AppxVolume
Add-BCDataCacheExtension
Add-BitLockerKeyProtector
Add-BitsFile
Add-BitsFile
Add-CertificateEnrollmentPolicyServer
Add-CloudResource
Add-Computer
Add-Content
Add-DnsClientNrptRule
Add-DtcClusterTMMapping
Add-EtwTraceProvider
Add-History
Add-InitiatorIdToMaskingSet
Add-JobTrigger
Add-KdsRootKey
Add-LocalGroupMember
Add-Member
Add-MpPreference
Add-NetEventNetworkAdapter
Add-NetEventPacketCaptureProvider
Add-NetEventProvider
Add-NetEventVFPProvider
Add-NetEventVmNetworkAdapter
Add-NetEventVmSwitch
Add-NetEventVmSwitchProvider
Add-NetEventWFPCaptureProvider
```

An Example of Some Basic Cmdlets

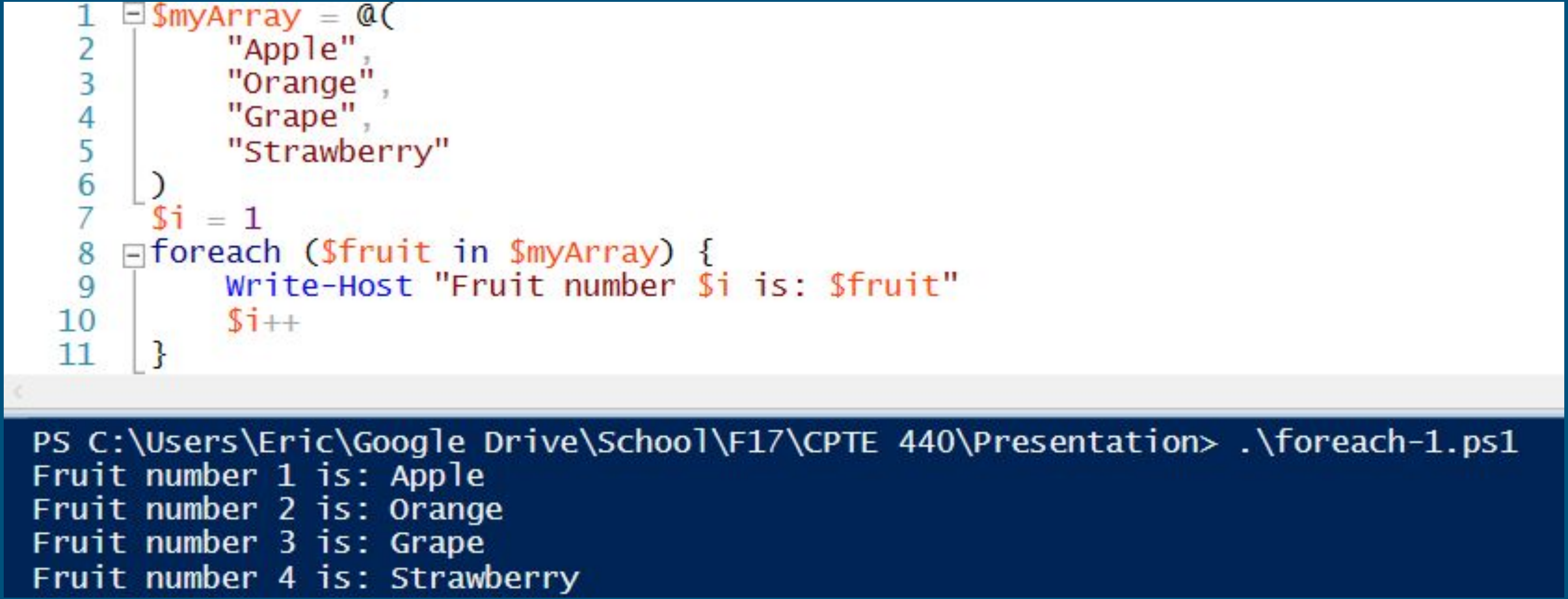
- New-Item
- Move-Item
- Write-Host
- Test-Path
- New-ADUser
- Import- Module
- ConvertTo-SecureString
- Read-Host
- Out-File
- Out-GridView
- Get-Date
- Invoke-Item
- New-Object
- Set-Acl
- Send-MailMessage
- Get-NetIPAddress
- Get-Content
- ForEach-Object
- Get-ChildItem
- Get-Help
- Remove-Item
- Clear-Host
- Compare-Object
- Invoke-WebRequest

See https://blogs.technet.microsoft.com/heyscriptingguy/2015/06/11/table-of-basic-powershell-commands/ for more basic cmdlets.

Loops and Conditionals

- *Foreach () {} # Loads all objects into memory and then processes them one by one*
- *ForEach-Object {} # One object at a time pipes into it, then pipes out (less RAM!)*
- *For () {}*
- *While () {}*
- *If () {}*
- *Elseif () {}*
- *Else {}*
- *Switch () {} # Think case statements*

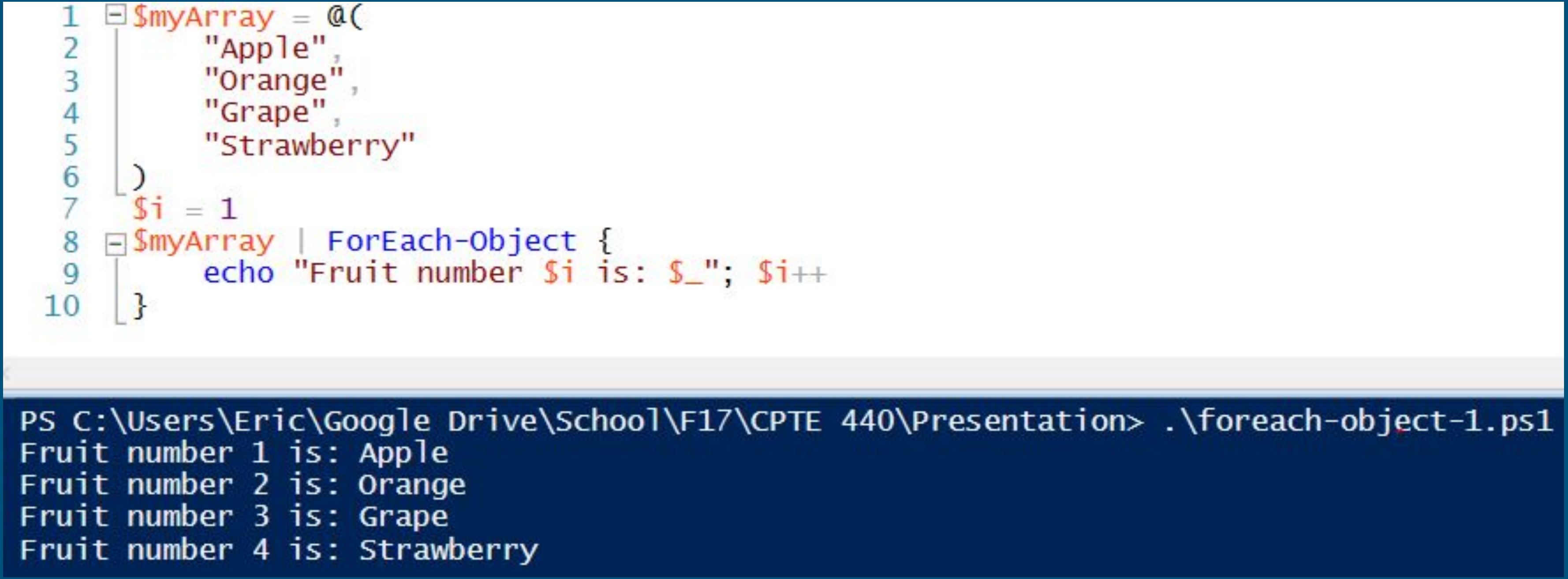
Loops and Conditionals - foreach



```powershell
$myArray = @(
        "Apple",
        "Orange",
        "Grape",
        "Strawberry"
)
$i = 1
foreach ($fruit in $myArray) {
        Write-Host "Fruit number $i is: $fruit"
        $i++
}
```

```
PS C:\Users\Eric\Google Drive\School\F17\CPTE 440\Presentation> .\foreach-1.ps1
Fruit number 1 is: Apple
Fruit number 2 is: Orange
Fruit number 3 is: Grape
Fruit number 4 is: Strawberry
```

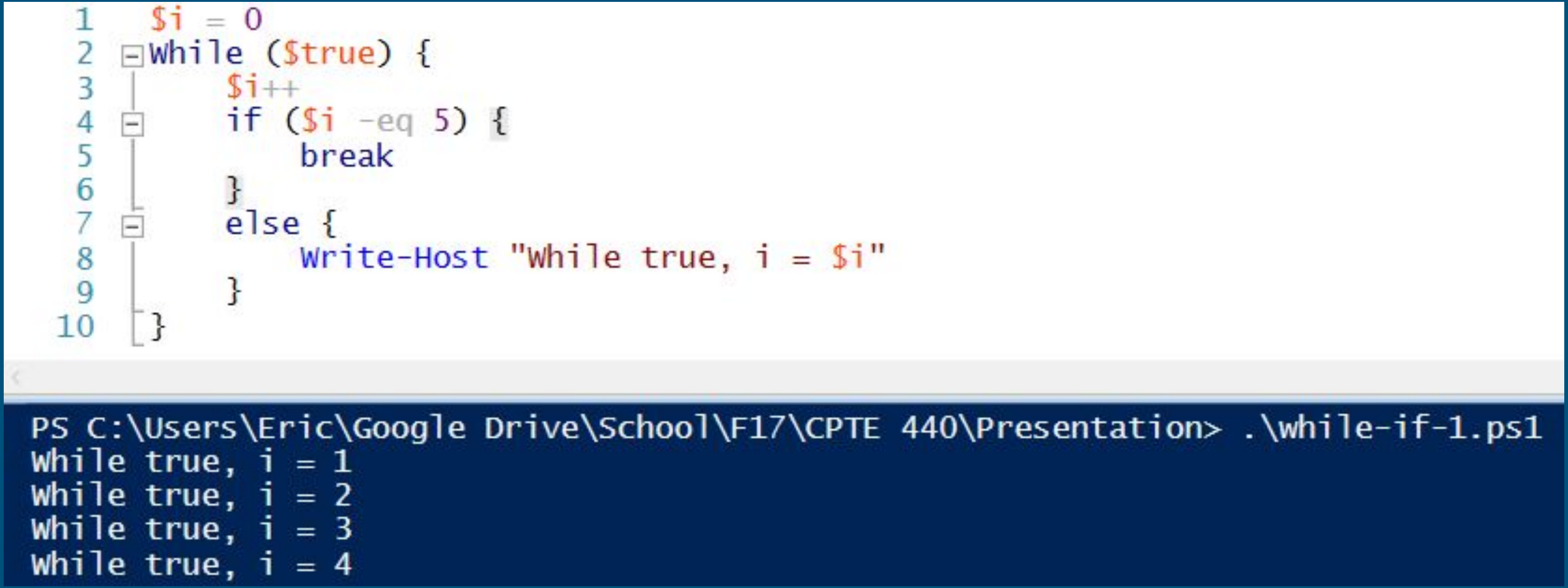
Loops and Conditionals - ForEach-Object



```
$myArray = @(
        "Apple",
        "Orange",
        "Grape",
        "Strawberry"
)
$i = 1
$myArray | ForEach-Object {
        echo "Fruit number $i is: $_"; $i++
}
```

```
PS C:\Users\Eric\Google Drive\School\F17\CPTE 440\Presentation> .\foreach-object-1.ps1
Fruit number 1 is: Apple
Fruit number 2 is: Orange
Fruit number 3 is: Grape
Fruit number 4 is: Strawberry
```

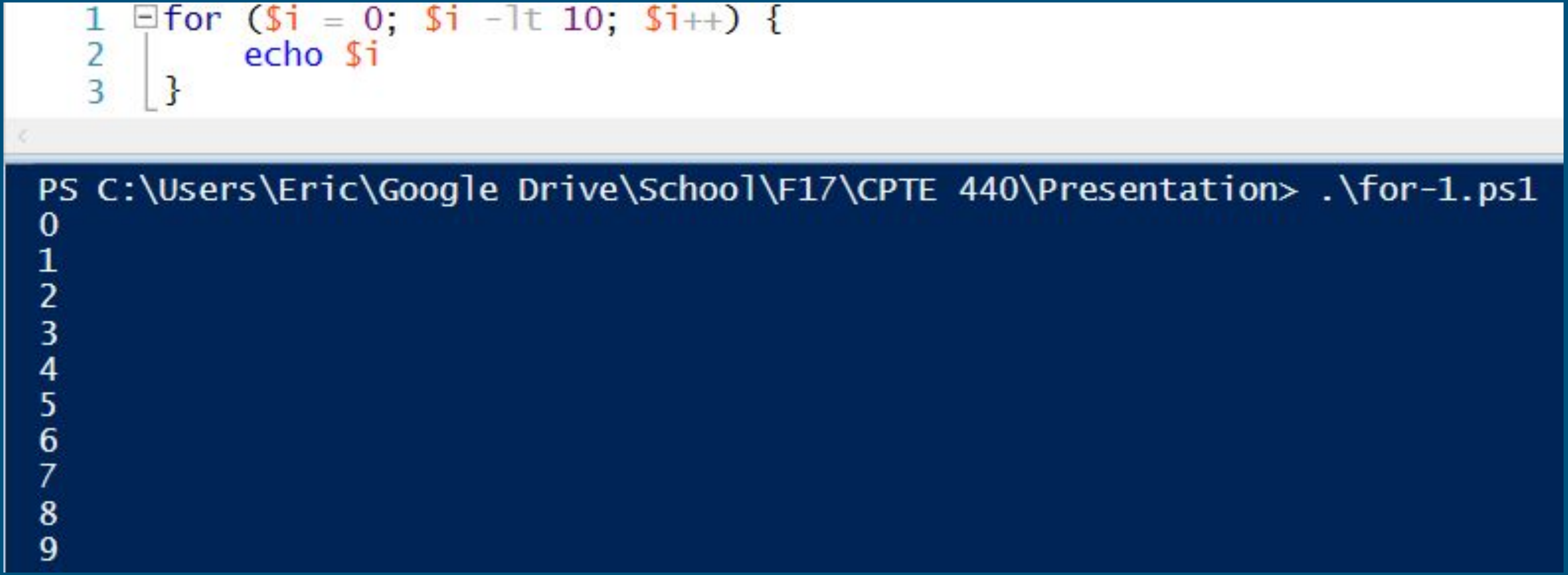
Loops and Conditionals - While and If



```
$i = 0
While ($true) {
    $i++
    if ($i -eq 5) {
        break
    }
    else {
        Write-Host "While true, i = $i"
    }
}
```

```
PS C:\Users\Eric\Google Drive\School\F17\CPTE 440\Presentation> .\while-if-1.ps1
While true, i = 1
While true, i = 2
While true, i = 3
While true, i = 4
```

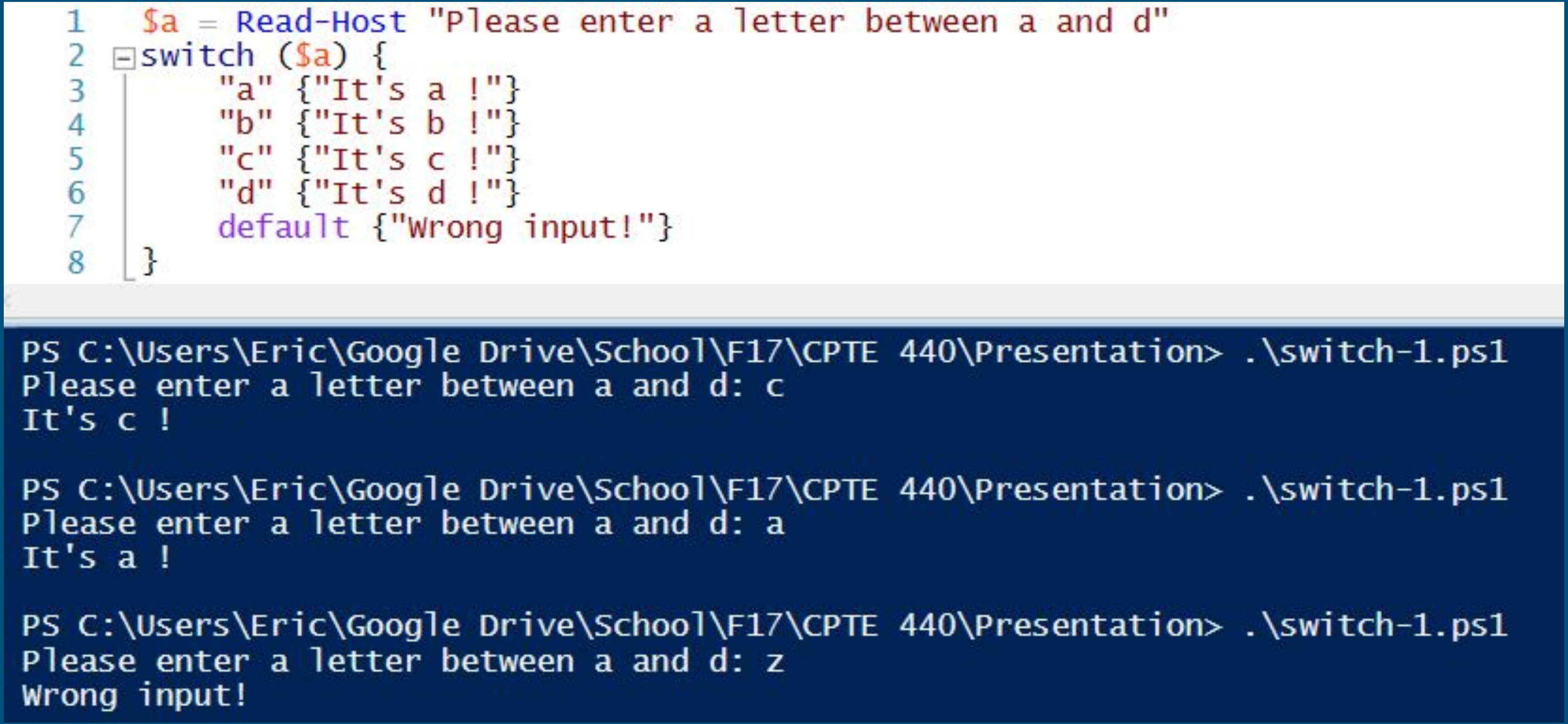
Loops and Conditionals - For



```
for ($i = 0; $i -lt 10; $i++) {
    echo $i
}
```

```
PS C:\Users\Eric\Google Drive\School\F17\CPTE 440\Presentation> .\for-1.ps1
0
1
2
3
4
5
6
7
8
9
```

Loops and Conditionals - Switch



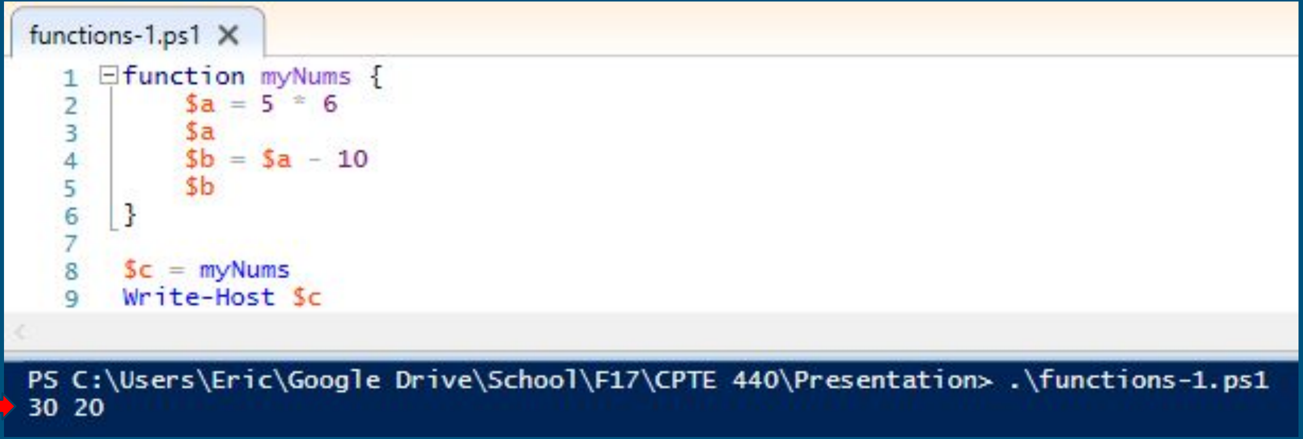
```
 $a = Read-Host "Please enter a letter between a and d"
switch ($a) {
        "a" {"It's a !"}
        "b" {"It's b !"}
        "c" {"It's c !"}
        "d" {"It's d !"}
        default {"Wrong input!"}
}
```

```
PS C:\Users\Eric\Google Drive\School\F17\CPTE 440\Presentation> .\switch-1.ps1
Please enter a letter between a and d: c
It's c !

PS C:\Users\Eric\Google Drive\School\F17\CPTE 440\Presentation> .\switch-1.ps1
Please enter a letter between a and d: a
It's a !

PS C:\Users\Eric\Google Drive\School\F17\CPTE 440\Presentation> .\switch-1.ps1
Please enter a letter between a and d: z
Wrong input!
```

Functions

- Declaration: *function <function name> {}*
- Do not have have to declare a type (anything can be returned)
- Do not require a "return" statement
    - Anything outputted is returned
    - 

functions-1.ps1

```
function myNums {
    $a = 5 * 6
    $a
    $b = $a - 10
    $b
}

$c = myNums
Write-Host $c
```

```
PS C:\Users\Eric\Google Drive\School\F17\CPTE 440\Presentation> .\functions-1.ps1
30 20
```

Functions - Parameters

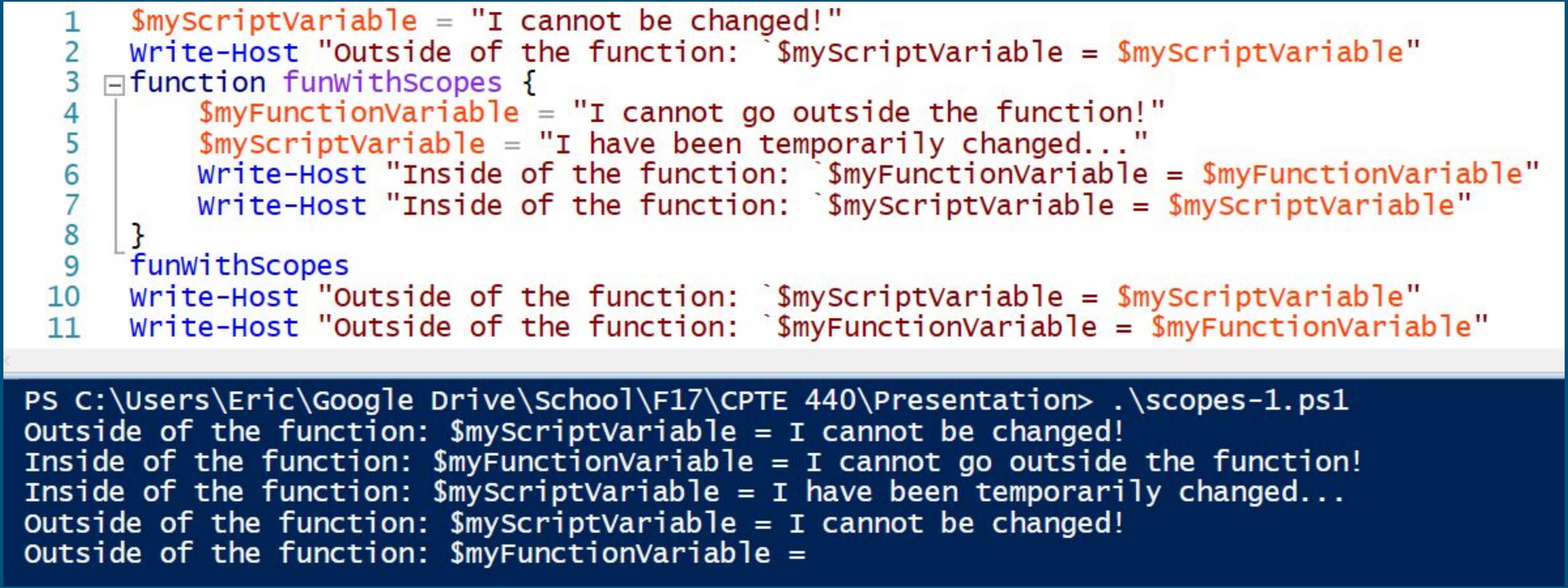
Parameters specified in a *Param* block

```
Param(
    # A required string parameter
    [Parameter(Mandatory=$True)][string]$myString,
    # If not specified when calling the function, it will have a default value of 10
    $myInt = 10,
    # An optional PSCredential object parameter
    [System.Management.Automation.PSCredential]$credentials,
    # An optional boolean switch parameter
    [switch]$enableLogging
)
```

Variable Scopes

- Variable scopes include:
    - Global - Powershell session
    - Script - Entire script
    - Local - Inside of a function
    - Private - Only in the current scope, not in child scopes
- Can be manually set by the scope modifier:
    - *$<scope>:<variable name>*
- To create a variable in the script scope inside of a function:
    - *$script:myVariable = “I’m a string!”*

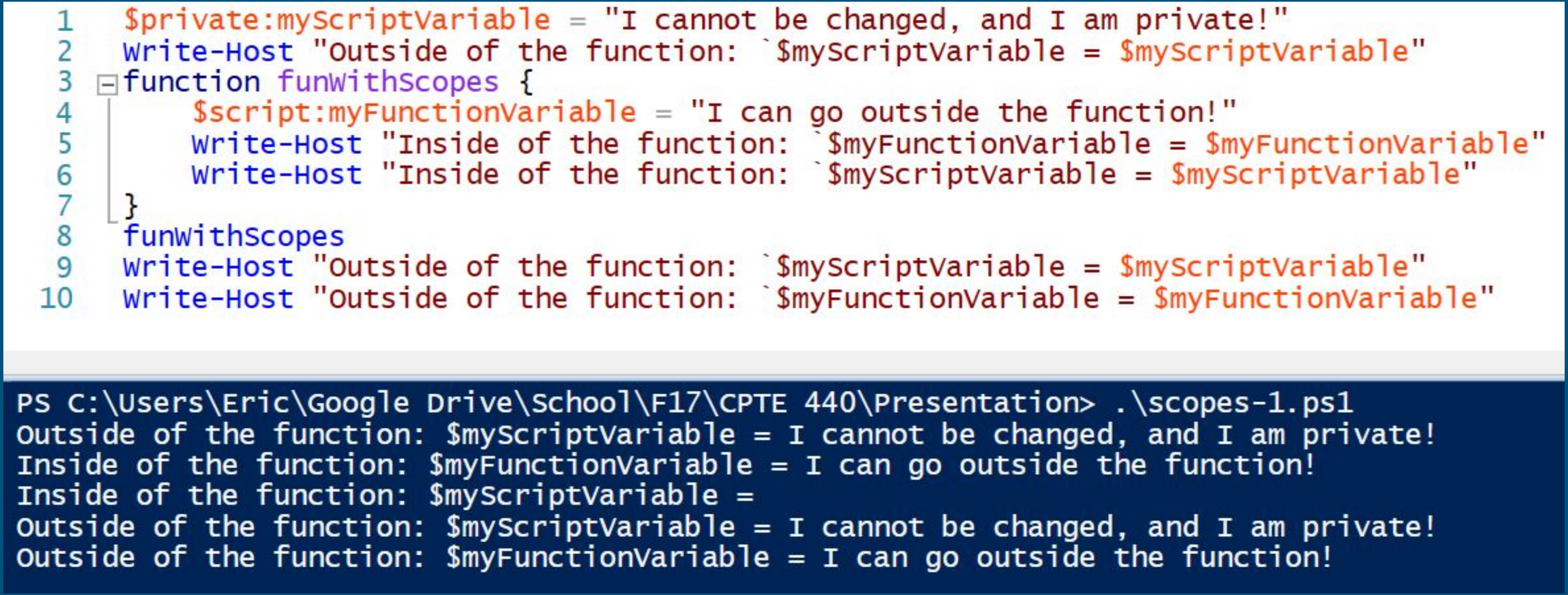
Variable Scopes



```powershell
$myScriptVariable = "I cannot be changed!"
Write-Host "Outside of the function: `$myScriptVariable = $myScriptVariable"
function funWithScopes {
    $myFunctionVariable = "I cannot go outside the function!"
    $myScriptVariable = "I have been temporarily changed..."
    Write-Host "Inside of the function: `$myFunctionVariable = $myFunctionVariable"
    Write-Host "Inside of the function: `$myScriptVariable = $myScriptVariable"
}
funWithScopes
Write-Host "Outside of the function: `$myScriptVariable = $myScriptVariable"
Write-Host "Outside of the function: `$myFunctionVariable = $myFunctionVariable"
```

```
PS C:\Users\Eric\Google Drive\School\F17\CPTE 440\Presentation> .\scopes-1.ps1
Outside of the function: $myScriptVariable = I cannot be changed!
Inside of the function: $myFunctionVariable = I cannot go outside the function!
Inside of the function: $myScriptVariable = I have been temporarily changed...
Outside of the function: $myScriptVariable = I cannot be changed!
Outside of the function: $myFunctionVariable =
```

Variable Scopes

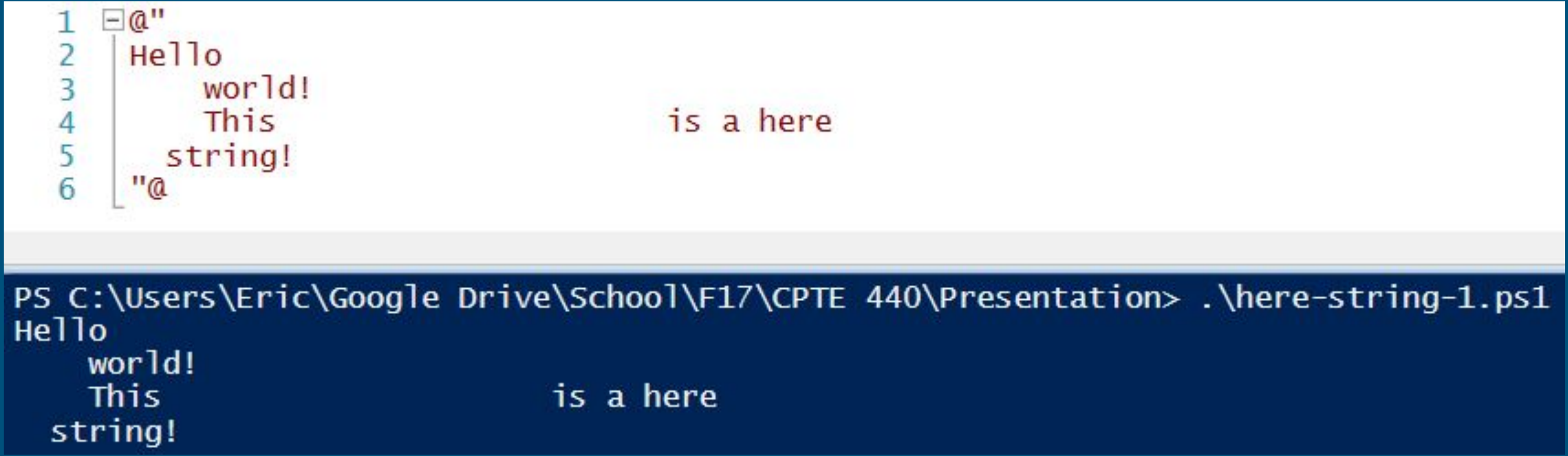


```
$private:myScriptVariable = "I cannot be changed, and I am private!"
Write-Host "Outside of the function: `$myScriptVariable = $myScriptVariable"
function funWithScopes {
    $script:myFunctionVariable = "I can go outside the function!"
    Write-Host "Inside of the function: `$myFunctionVariable = $myFunctionVariable"
    Write-Host "Inside of the function: `$myScriptVariable = $myScriptVariable"
}
funWithScopes
Write-Host "Outside of the function: `$myScriptVariable = $myScriptVariable"
Write-Host "Outside of the function: `$myFunctionVariable = $myFunctionVariable"
```

```
PS C:\Users\Eric\Google Drive\School\F17\CPTE 440\Presentation> .\scopes-1.ps1
Outside of the function: $myScriptVariable = I cannot be changed, and I am private!
Inside of the function: $myFunctionVariable = I can go outside the function!
Inside of the function: $myScriptVariable =
Outside of the function: $myScriptVariable = I cannot be changed, and I am private!
Outside of the function: $myFunctionVariable = I can go outside the function!
```

Here Strings

Multi-line strings formatted exactly as typed. @” and “@ must be on their own lines.



```
@"
Hello
    world!
    This                              is a here
  string!
"@
```

```
PS C:\Users\Eric\Google Drive\School\F17\CPTE 440\Presentation> .\here-string-1.ps1
Hello
    world!
    This                              is a here
  string!
```

Comparison Operators

- -eq
- -ne
- -gt
- -ge
- -lt
- -le
- -Like
- -NotLike
- -Match
- -NotMatch
- -Contains
- -NotContains
- -In
- -NotIn
- -Replace

Dot Sourcing

- Process of including files (functions) in your script
- Loads file into memory
- Dot (‘.’) then space (‘ ‘) then path to file
- Example:

  ```
  . C:\Path\to\file.ps1
  ```
- Make sure there is no executable code in the file being dot sourced, otherwise it will execute as soon as it is dot sourced.
- Instead, make sure everything is wrapped in a function.
- After dot sourcing, call the function as you would a built in cmdlet.

Dot Sourcing

One method to dot source multiple files (with some very basic error handling):

```
# Create an array with all of the needed files
$myFunctions = @(
        “C:\path\to\file1.ps1”,
        “\\filesrv1\path\to\file2.ps1”,
        “C:\path\to\file3.ps1”
)
# Loop through each file and test to see if it exists. If so, dot source it.
Foreach ($function in $myFunctions) {
        If (Test-Path $function) {. $function}
        Else {Write-Host “At least one necessary function was not found”}
}
```

Powershell Profiles

- Powershell profiles are simply scripts that are run whenever a Powershell session is started
- You can put variables, functions, and dot sourced functions into your Powershell profile so that you have them every time you use Powershell
- Powershell has six profiles, each affecting different scopes
- Primary profile is Current User, Current Host[1]
- See the path to the profiles via the $profile variable
- If in Powershell ISE, you can edit them with the command: *psEdit $profile*

1. https://blogs.technet.microsoft.com/heyscriptingguy/2012/05/21/understanding-the-six-powershell-profiles/

Powershell Profiles

Description	Path	Variable
Current User, Current Host - console	$Home\[My]Documents\WindowsPowerShell\Profile.ps1	$PROFILE.CurrentUserCurrentHost
Current User, All Hosts	$Home\[My]Documents\Profile.ps1	$PROFILE.CurrentUserAllHosts
All Users, Current Host - console	$PsHome\Microsoft.PowerShell_profile.ps1	$PROFILE.AllUsersCurrentHost
All Users, All Hosts	$PsHome\Profile.ps1	$PROFILE.AllUsersAllHosts
Current User, Current Host - ISE	$Home\[My]Documents\WindowsPowerShell\Microsoft.PowerShellISE_profile.ps1	$profile.CurrentUserCurrentHost
All Users, Current Host - ISE	$PsHome\Microsoft.PowerShellISE_profile.ps1	$profile.AllUsersCurrentHost

https://blogs.technet.microsoft.com/heyscriptingguy/2012/05/21/understanding-the-six-powershell-profiles/

Powershell Profiles

Transcripts

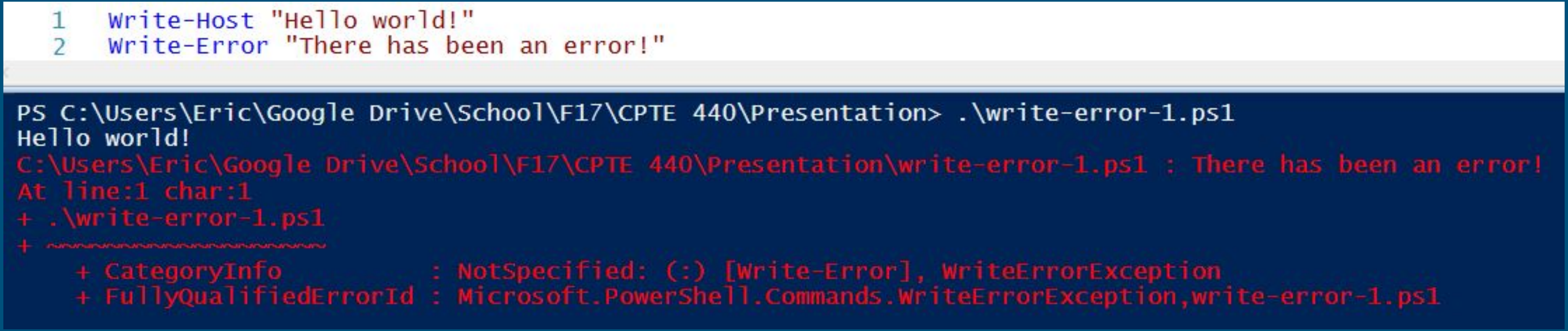
- Transcripts are a log of everything outputted during a Powershell session.
- *Start-Transcript -Path C:\path\to\transcript.log*
- *Stop-Transcript*
- Very useful, especially when automating scripts, as you can go back and see a transcript of what happened while the script was running.

Error Handling

Error Handling

- Methods of error handling:
  - Write-Error
  - Throw statements
  - -ErrorAction
  - $ErrorActionPreference
  - Trap statements
  - Try/Catch blocks

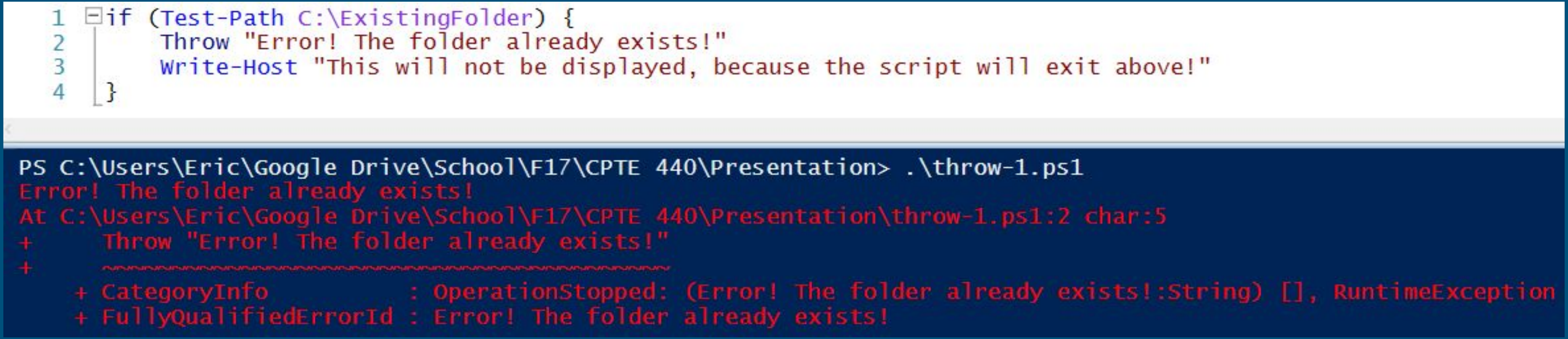
Error Handling - Write-Error



```
Write-Host "Hello world!"
Write-Error "There has been an error!"
```

```
PS C:\Users\Eric\Google Drive\School\F17\CPTE 440\Presentation> .\write-error-1.ps1
Hello world!
C:\Users\Eric\Google Drive\School\F17\CPTE 440\Presentation\write-error-1.ps1 : There has been an error!
At line:1 char:1
+ .\write-error-1.ps1
+ ~~~~~~~~~~~~~~~~~~~
    + CategoryInfo          : NotSpecified: (:) [Write-Error], WriteErrorException
    + FullyQualifiedErrorId : Microsoft.PowerShell.Commands.WriteErrorException,write-error-1.ps1
```

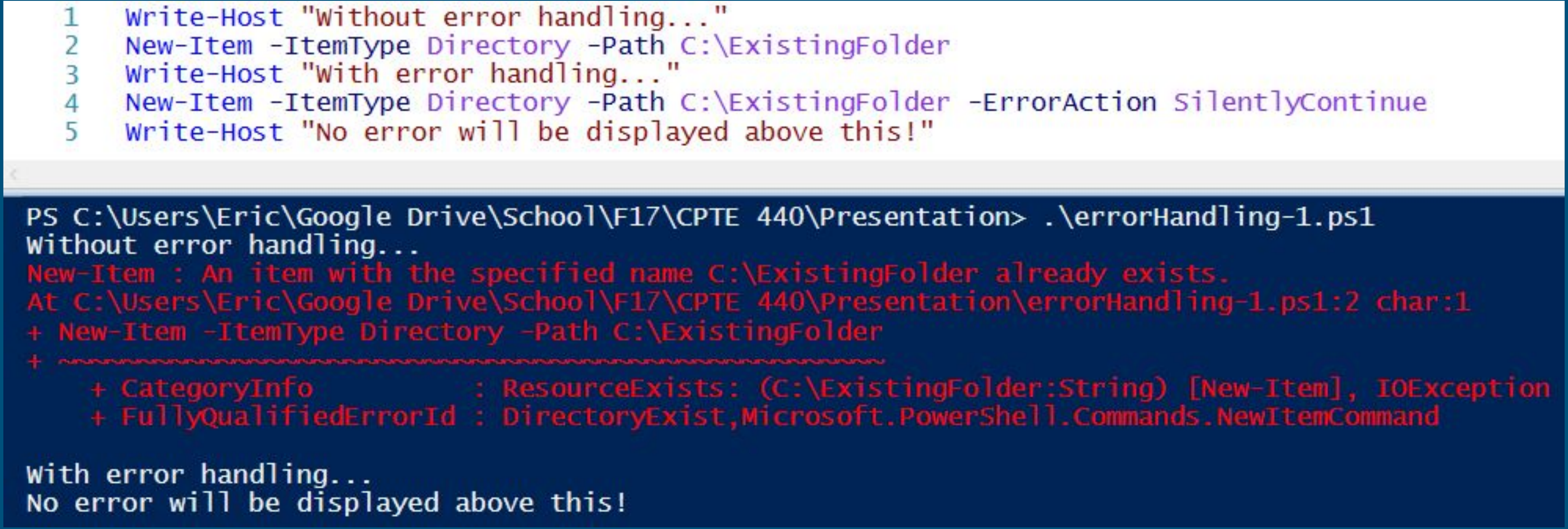
Error Handling - Throw



```
if (Test-Path C:\ExistingFolder) {
    Throw "Error! The folder already exists!"
    Write-Host "This will not be displayed, because the script will exit above!"
}
```

```
PS C:\Users\Eric\Google Drive\School\F17\CPTE 440\Presentation> .\throw-1.ps1
Error! The folder already exists!
At C:\Users\Eric\Google Drive\School\F17\CPTE 440\Presentation\throw-1.ps1:2 char:5
+     Throw "Error! The folder already exists!"
+     ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
    + CategoryInfo          : OperationStopped: (Error! The folder already exists!:String) [], RuntimeException
    + FullyQualifiedErrorId : Error! The folder already exists!
```

Error Handling -ErrorAction Parameter

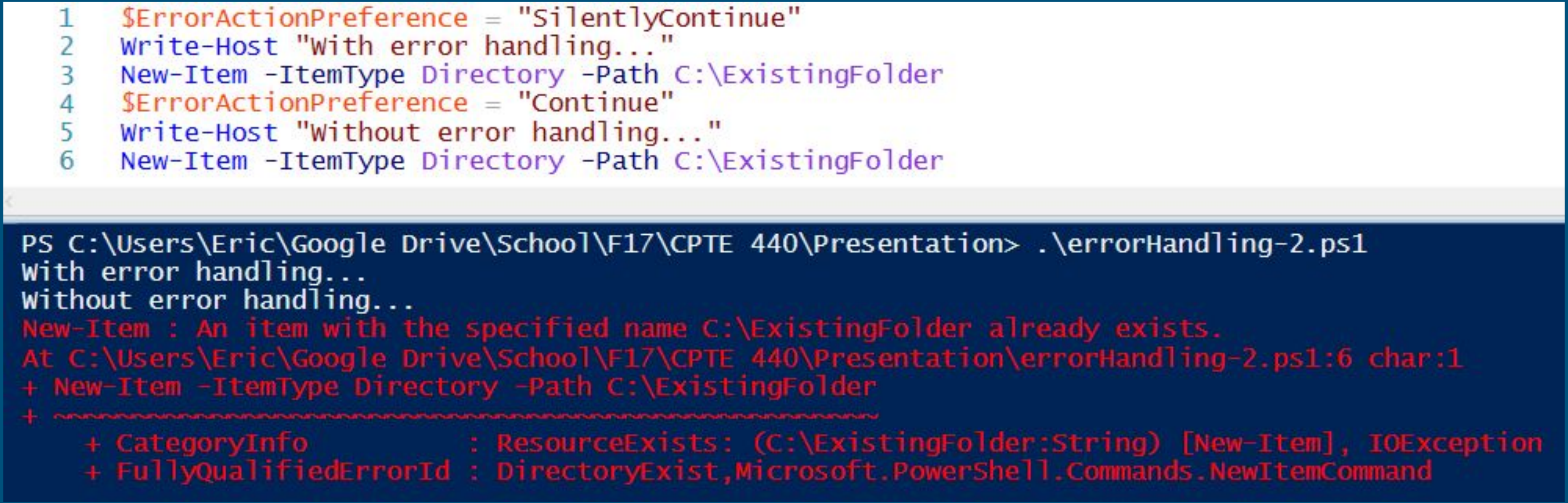


```
Write-Host "Without error handling..."
New-Item -ItemType Directory -Path C:\ExistingFolder
Write-Host "With error handling..."
New-Item -ItemType Directory -Path C:\ExistingFolder -ErrorAction SilentlyContinue
Write-Host "No error will be displayed above this!"
```

```
PS C:\Users\Eric\Google Drive\School\F17\CPTE 440\Presentation> .\errorHandling-1.ps1
Without error handling...
New-Item : An item with the specified name C:\ExistingFolder already exists.
At C:\Users\Eric\Google Drive\School\F17\CPTE 440\Presentation\errorHandling-1.ps1:2 char:1
+ New-Item -ItemType Directory -Path C:\ExistingFolder
+ ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
    + CategoryInfo          : ResourceExists: (C:\ExistingFolder:String) [New-Item], IOException
    + FullyQualifiedErrorId : DirectoryExist,Microsoft.PowerShell.Commands.NewItemCommand

With error handling...
No error will be displayed above this!
```

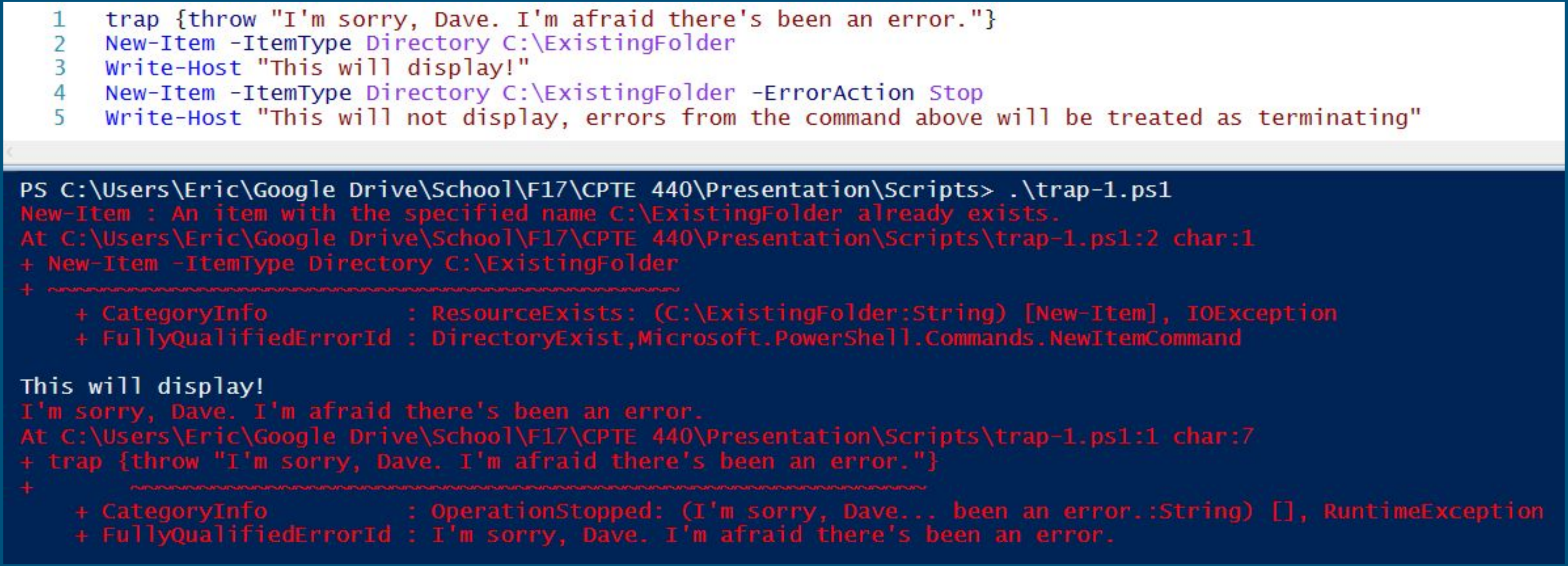
Error Handling - $ErrorActionPreference



```powershell
$ErrorActionPreference = "SilentlyContinue"
Write-Host "With error handling..."
New-Item -ItemType Directory -Path C:\ExistingFolder
$ErrorActionPreference = "Continue"
Write-Host "Without error handling..."
New-Item -ItemType Directory -Path C:\ExistingFolder
```

```
PS C:\Users\Eric\Google Drive\School\F17\CPTE 440\Presentation> .\errorHandling-2.ps1
With error handling...
Without error handling...
New-Item : An item with the specified name C:\ExistingFolder already exists.
At C:\Users\Eric\Google Drive\School\F17\CPTE 440\Presentation\errorHandling-2.ps1:6 char:1
+ New-Item -ItemType Directory -Path C:\ExistingFolder
+ ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
    + CategoryInfo          : ResourceExists: (C:\ExistingFolder:String) [New-Item], IOException
    + FullyQualifiedErrorId : DirectoryExist,Microsoft.PowerShell.Commands.NewItemCommand
```

Error Handling - Trap

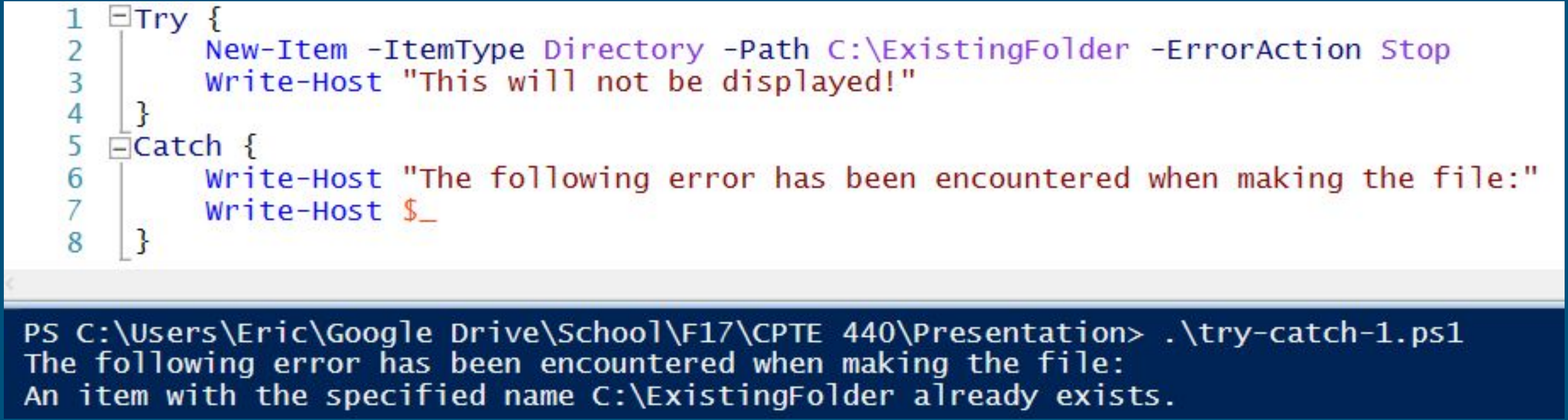


```
trap {throw "I'm sorry, Dave. I'm afraid there's been an error."}
New-Item -ItemType Directory C:\ExistingFolder
Write-Host "This will display!"
New-Item -ItemType Directory C:\ExistingFolder -ErrorAction Stop
Write-Host "This will not display, errors from the command above will be treated as terminating"
```

```
PS C:\Users\Eric\Google Drive\School\F17\CPTE 440\Presentation\Scripts> .\trap-1.ps1
New-Item : An item with the specified name C:\ExistingFolder already exists.
At C:\Users\Eric\Google Drive\School\F17\CPTE 440\Presentation\Scripts\trap-1.ps1:2 char:1
+ New-Item -ItemType Directory C:\ExistingFolder
+ ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
    + CategoryInfo          : ResourceExists: (C:\ExistingFolder:String) [New-Item], IOException
    + FullyQualifiedErrorId : DirectoryExist,Microsoft.PowerShell.Commands.NewItemCommand

This will display!
I'm sorry, Dave. I'm afraid there's been an error.
At C:\Users\Eric\Google Drive\School\F17\CPTE 440\Presentation\Scripts\trap-1.ps1:1 char:7
+ trap {throw "I'm sorry, Dave. I'm afraid there's been an error."}
+       ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
    + CategoryInfo          : OperationStopped: (I'm sorry, Dave... been an error.:String) [], RuntimeException
    + FullyQualifiedErrorId : I'm sorry, Dave. I'm afraid there's been an error.
```

Error Handling - Try/Catch Block



```
Try {
    New-Item -ItemType Directory -Path C:\ExistingFolder -ErrorAction Stop
    Write-Host "This will not be displayed!"
}
Catch {
    Write-Host "The following error has been encountered when making the file:"
    Write-Host $_
}
```

```
PS C:\Users\Eric\Google Drive\School\F17\CPTE 440\Presentation> .\try-catch-1.ps1
The following error has been encountered when making the file:
An item with the specified name C:\ExistingFolder already exists.
```

Documenting Powershell

Commenting in Powershell

Comments begin with the ‘#’ symbol

- # At the beginning of a line (prefered)
- On an existing line # everything after the ‘#’ is treated as a comment

Block comments begin with ‘<#’ and end with ‘#>’

<# This is a comment
Everything within this is also comments
#>

Get-Help

- Get-Help is the Powershell command to see documentation regarding cmdlets, functions, and scripts.
- The Powershell version of Linux's “man” command.
- *Get-Help <cmdlet name> [-Full | -Examples | etc.]*
- *Get-Help about_<term>*
- *Get-Help <search term (part of cmdlet name)>*

Adding Get-Help Content to Your Script

In a block comment (<# … #>) at the top of your script or function. Indent each section under the headers.

```
.SYNOPSIS
.DESCRIPTION
.INPUTS
.OUTPUTS
.NOTES
.LINK
.EXAMPLE
.COMPONENT
```