Microsoft

WorkshopPLUS - Windows PowerShell: Foundation Skills

Microsoft Services

Microsoft

Flow Control and Collection Types

Microsoft Services

Learning Units covered in this Module

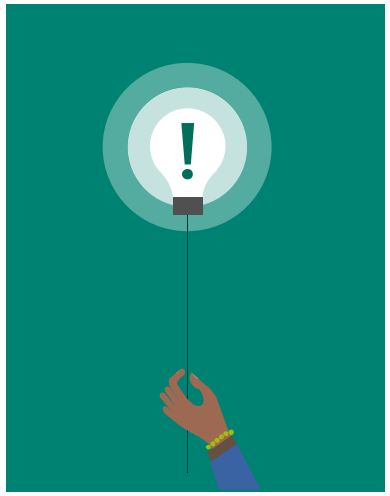
- Flow Control
- Arrays
- Hash Tables

Flow Control

Objectives

After completing Flow Control, you will be able to:

- Work with PowerShell loops
- Controle the flow of PowerShell loops



The Five Loops

PowerShell's Five Loops

- Loops control the flow of code
- Loops can work based on iteration or on objects from a collection

PowerShell's Five Loops

The loops in PowerShell repeat code

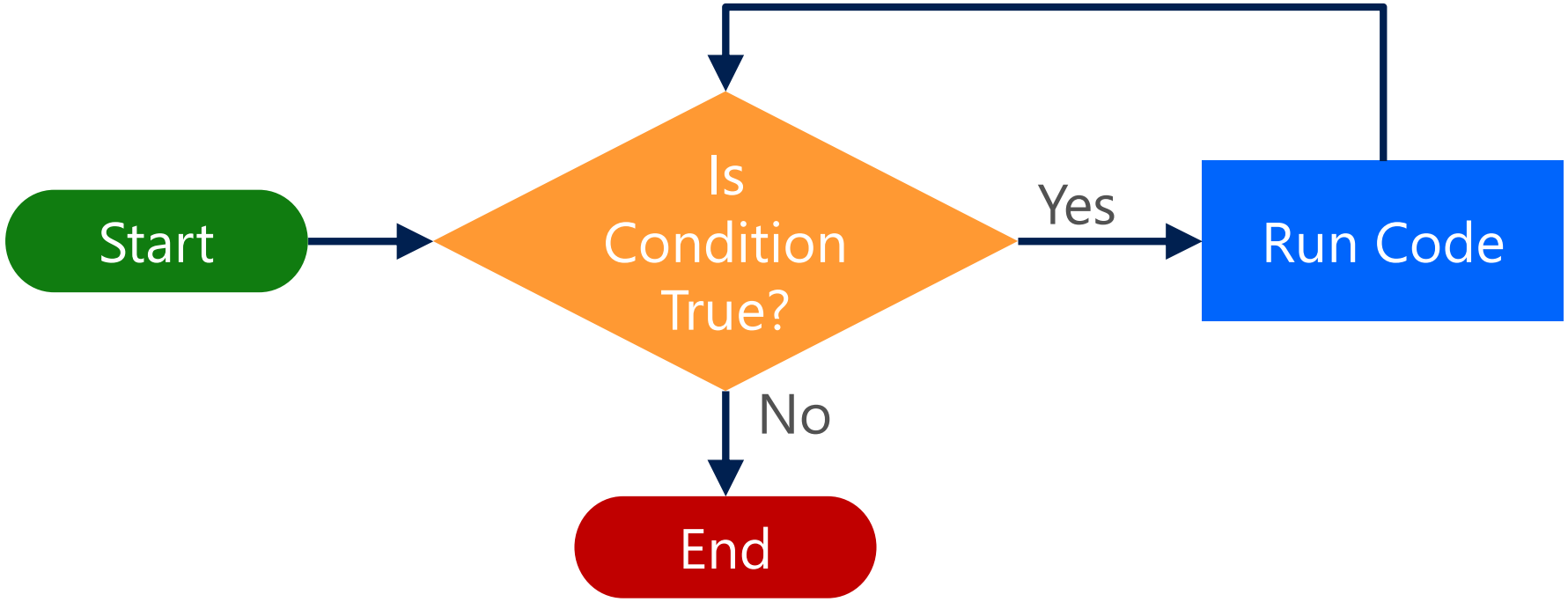
Five different logical variations:

Name	Loop Type	Features
While	Conditional	Tests for $True condition
Do While	Conditional	Tests for $True condition Code block runs at least once
Do Until	Conditional	Tests for $False condition Code block runs at least once
For	Conditional	Tests for $True condition Includes Initialization and Repeat blocks
Foreach	Enumeration	Runs code once for each item in collection/array Choose automatic variable name

While Loop

While Loop

Name	Description	Example
While	Runs script block while conditional test = true	`$a = 0` `while ($a -lt 10) {$a; $a++}`



While

```
$ComputerName = 'DC'

Restart-Computer -ComputerName $ComputerName
Start-Sleep -Seconds 30

While (-not (Test-Connection -ComputerName $ComputerName -Quiet))
{
    "Waiting on restart: $ComputerName"
    Start-Sleep -Seconds 30
}
```

```
Waiting on restart: DC
Waiting on restart: DC
Waiting on restart: DC
PS C:\
```

PowerShell v3.0 Restart-Computer introduced the –Wait parameter

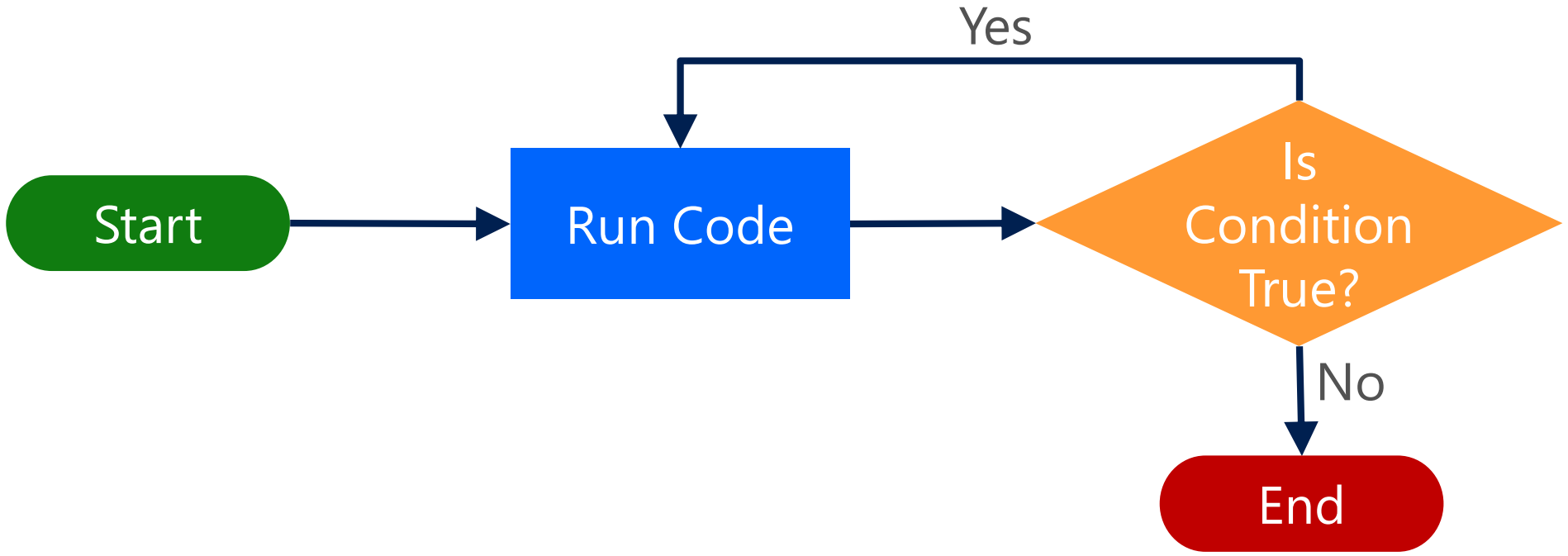
Demonstration

While Loops

Do While loop & Do Until loop

Do While Loop

Name	Description	Example
Do While	Condition evaluated AFTER script block **runs at least once** Runs script block if conditional test = true	`$a = 0` `Do {$a; $a++} while ($a -lt 10)`



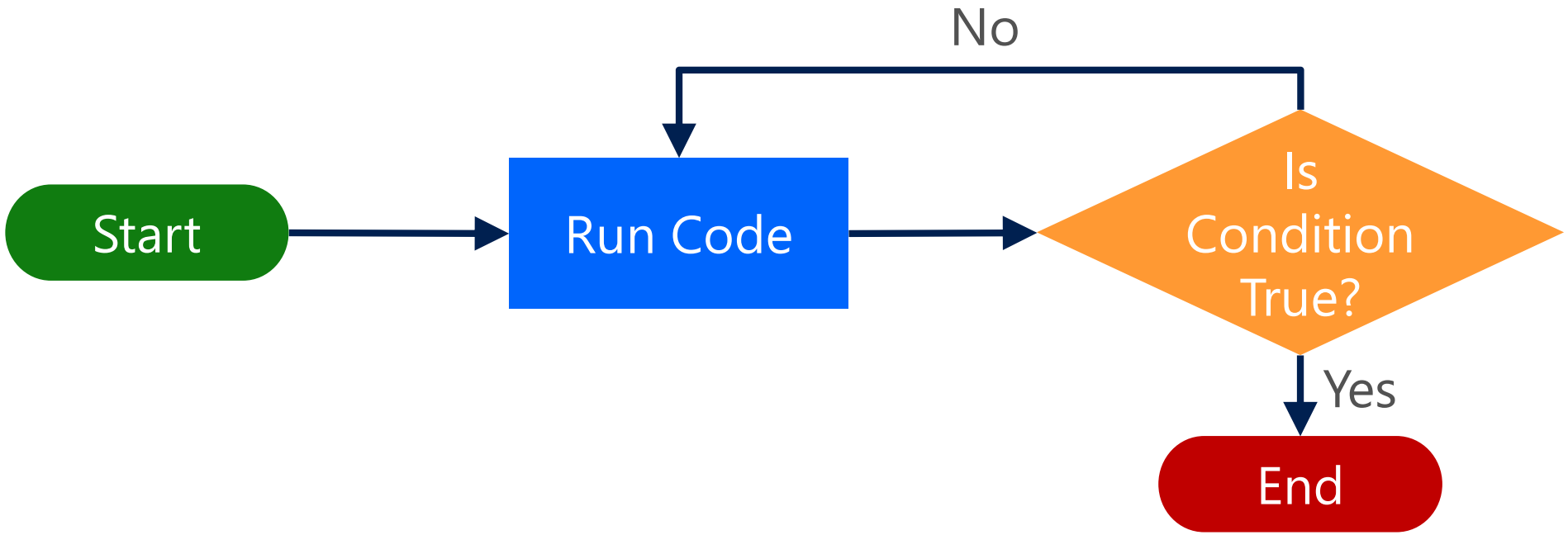
Do While

```
$ComputerName = ‘DC'
Restart-Computer -ComputerName $ComputerName

Do
{
    "Waiting on restart: $ComputerName"
    Start-Sleep -Seconds 30
}
While (-not (Test-Connection -ComputerName $ComputerName -Quiet))
```

```
Waiting on restart: DC
Waiting on restart: DC
Waiting on restart: DC

PS C:\
```

Do Until Loop

Name	Description	Example
Do Until	Condition evaluated AFTER script block **runs at least once** Runs script block if conditional test = false	`$a = 0` `Do {$a; $a++} Until ($a -ge 10)`



Do Until

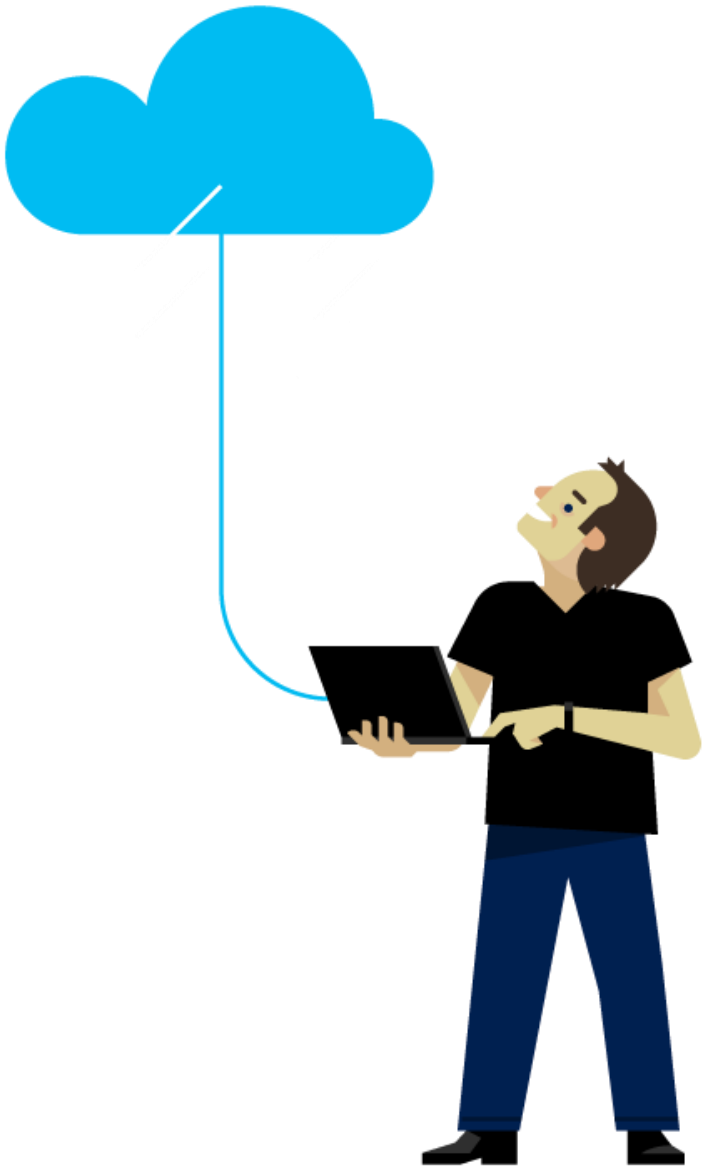
```
$ComputerName = 'DC'
Restart-Computer -ComputerName $ComputerName

Do
{
    "Waiting on restart: $ComputerName"
    Start-Sleep -Seconds 30
}
Until (Test-Connection -ComputerName $ComputerName -Quiet)
```

```
Waiting on restart: DC
Waiting on restart: DC
Waiting on restart: DC

PS C:\
```

Demonstration

Do While Loop & Do Until Loop

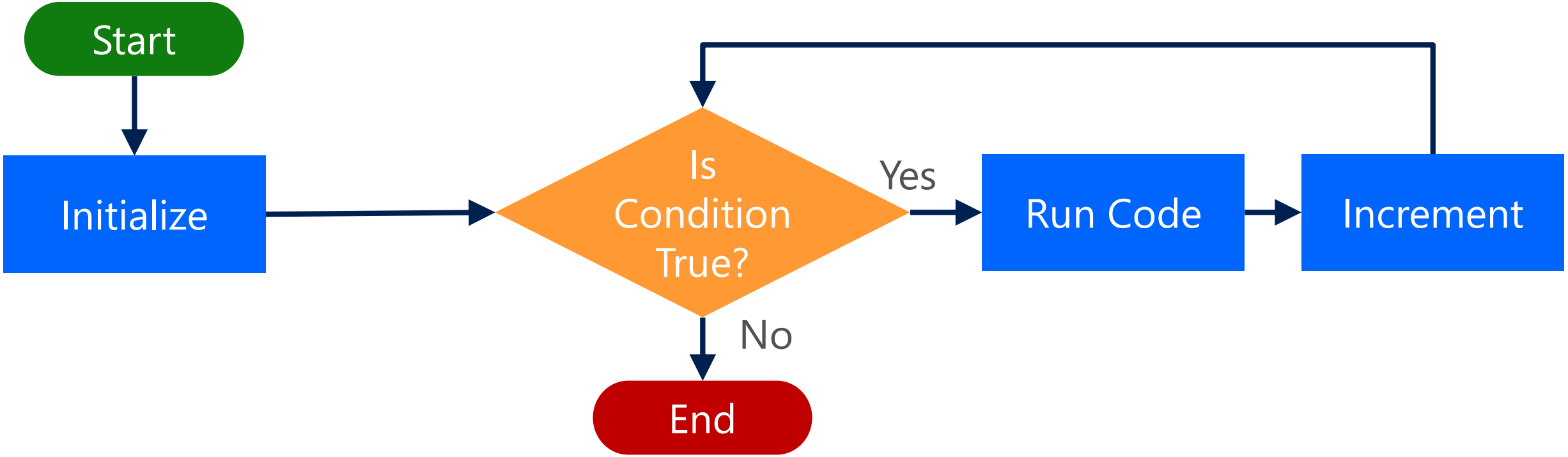
Questions?



For and Foreach Loop

For Loop

For	Runs script block while conditional test = true Useful when targeting a subset of array values
Syntax	`For (<init>; <condition>; <increment>) {<statement list>}`
Example	`For ($a=1; $a -lt 10; $a++) {$a}`



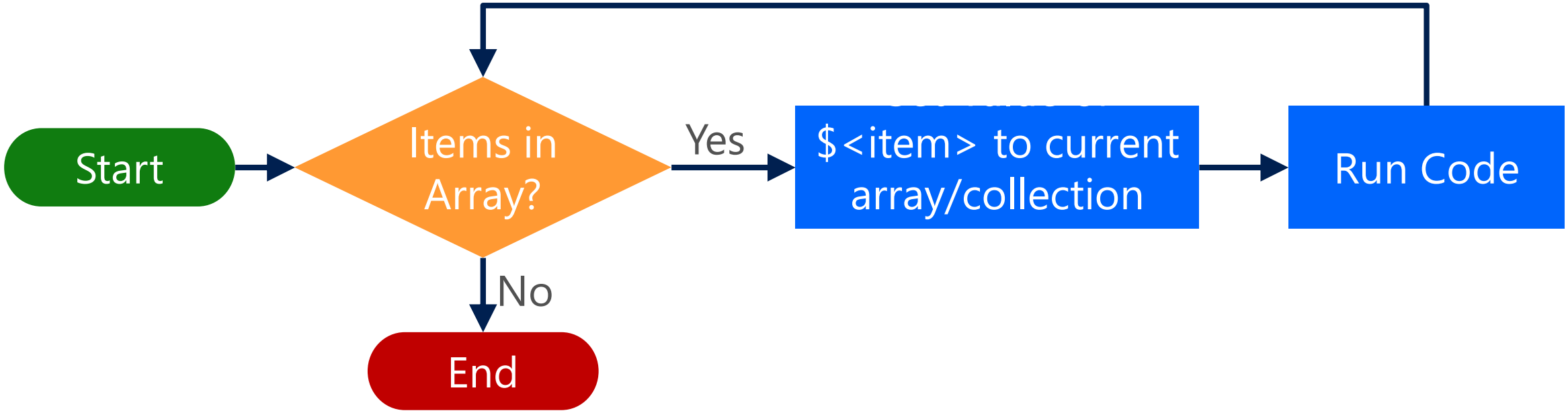
For

```
$Computers = @(Get-ADComputer -Filter {OperatingSystem -
like "*server*"}).Name

For ($i=0 ; $i -lt $Computers.Length ; $i++)
{
    "Computer $($i+1): $($Computers[$i])"
}
```

```
Computer 1: DC
Computer 2: MS
```

ForEach Loop

ForEach	Good when targeting all array values
Syntax	`ForEach ($<item> in $<collection>){<statement list>}`
Example	`ForEach ($file in Get-Childitem c:\windows -File) {$file.name}`



ForEach Loop

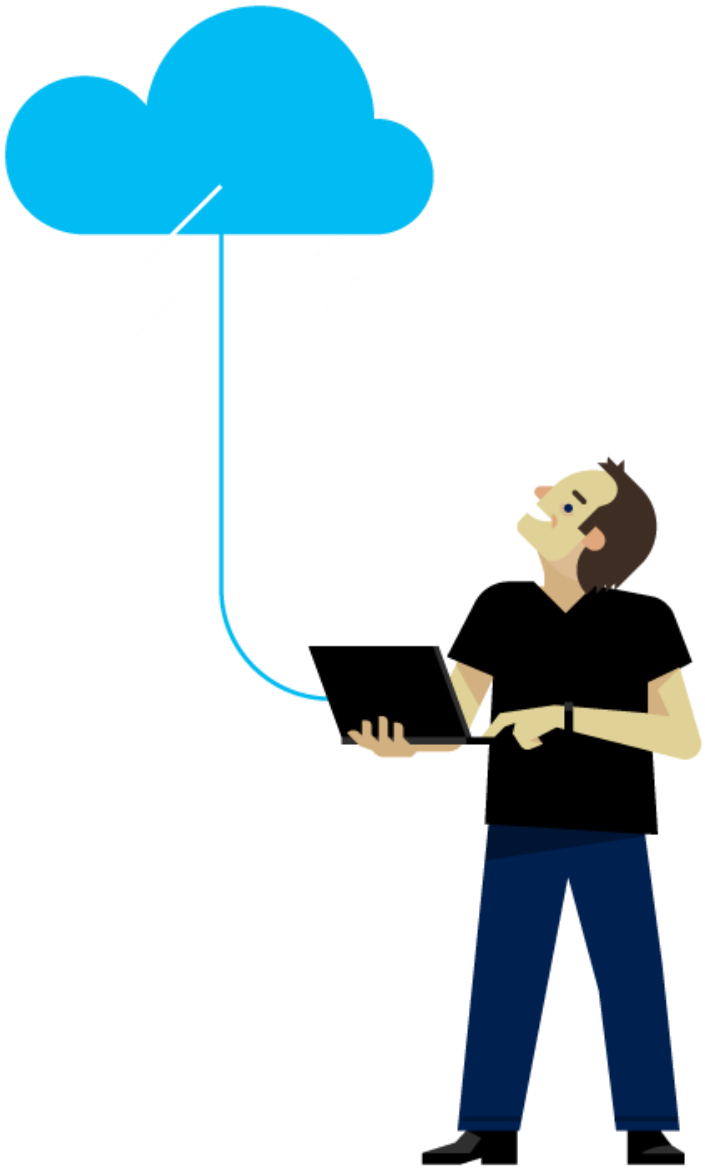
```
$Services = Get-Service
'There are a total of ' + $Services.Count + ' services.'
ForEach ($Service in $Services)
{
    $Service.Name + ' is ' + $Service.Status
}
```

```
There are a total of 167 services.
AeLookupSvc is Stopped
ALG is Stopped
AppIDSvc is Stopped
Appinfo is Stopped
AppMgmt is Stopped
AppReadiness is Stopped
AppXSvc is Stopped
...
```

Demonstration

For and Foreach Loop

Questions?



IF Statement

IF Statement

Branching structure chooses which code to run

Optional – Elseif(s) used for additional test(s)

Optional – Else code runs if test(s) fail

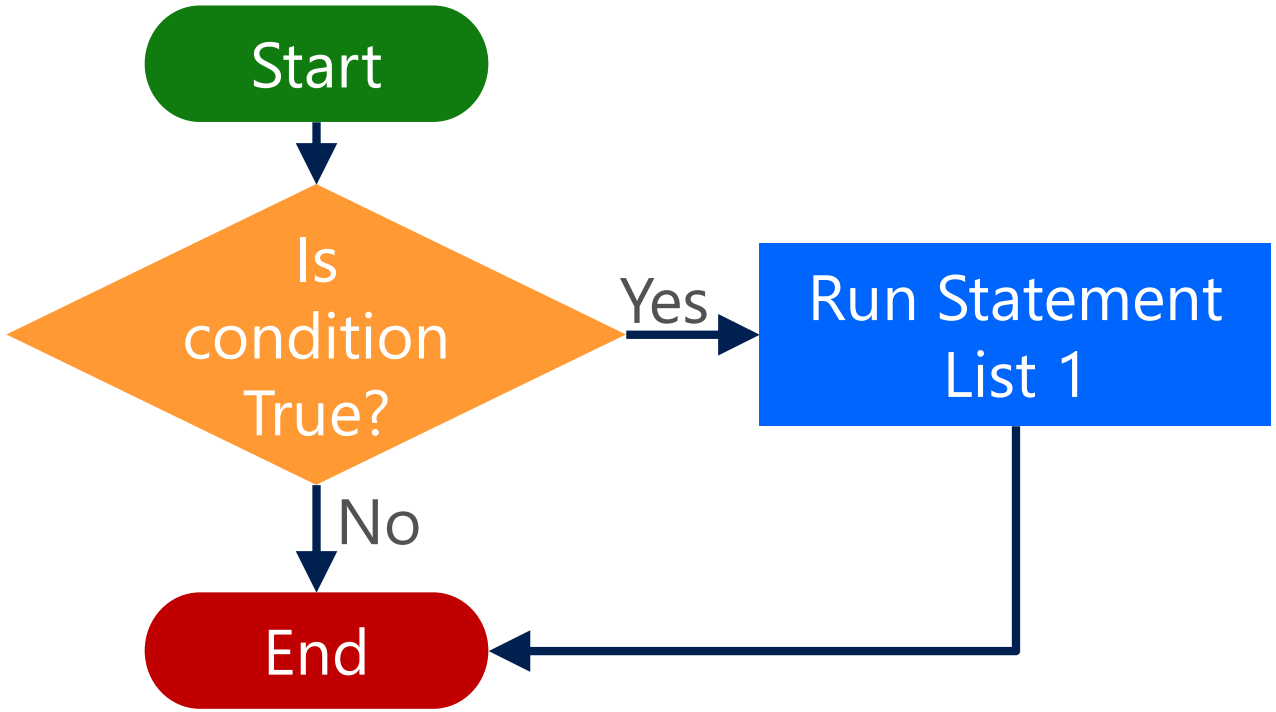
Only one code block will run

```
If      (<test1>) {<statement list 1>}

[ElseIf (<test2>) {<statement list 2>}]

[ElseIf (<test3>) {<statement list 3>}]

[Else             {<statement list 4>}]
```

Optional

If

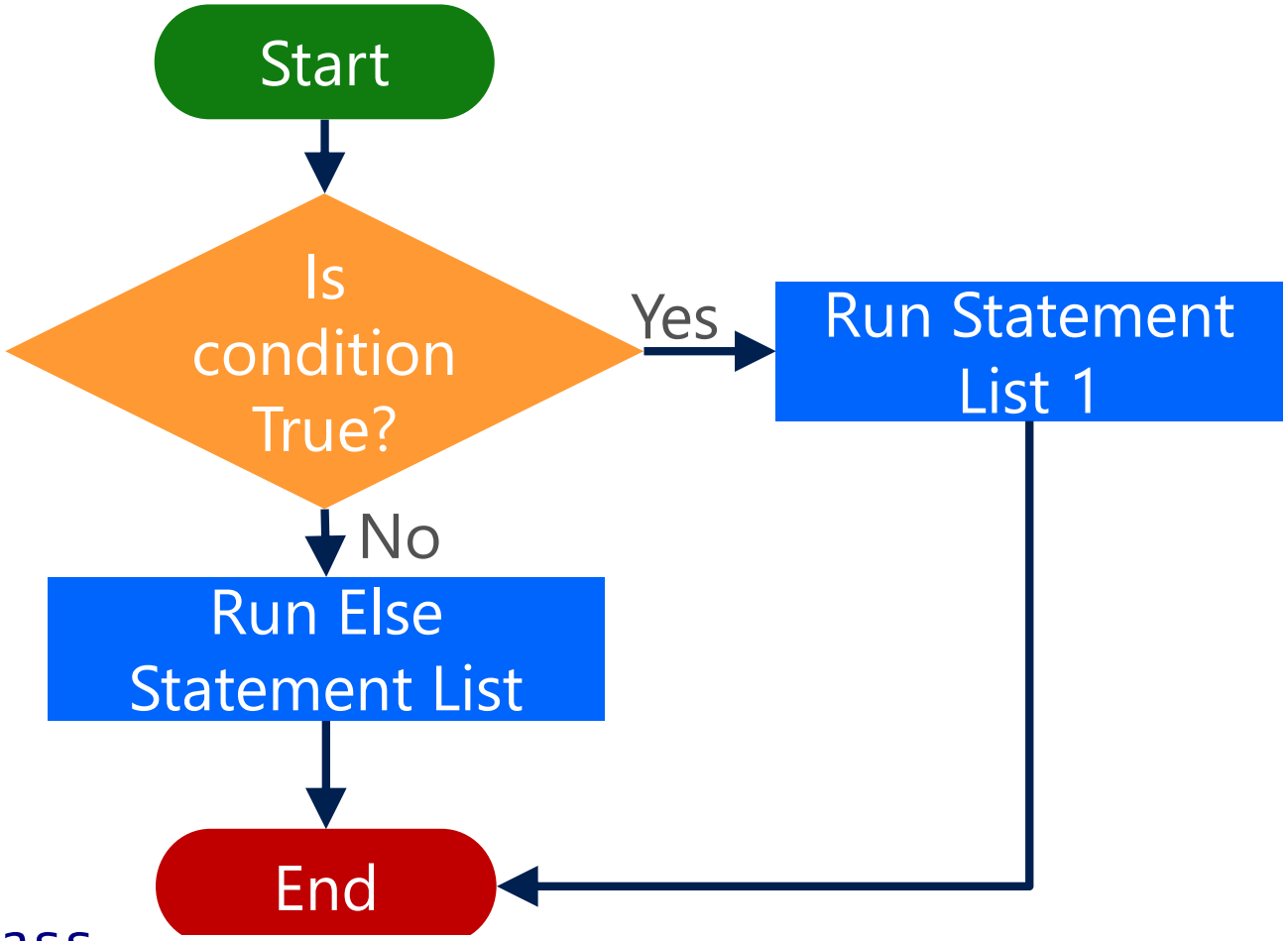
```
If (<test1>)
{
   <statement list 1>
}
```




```
$Language = (Get-CimInstance -class
win32_operatingsystem).OSLanguage

if ($Language -eq "1033")
{
   write-Host "Language = English US" -ForegroundColor Magenta
}
```

If..Else

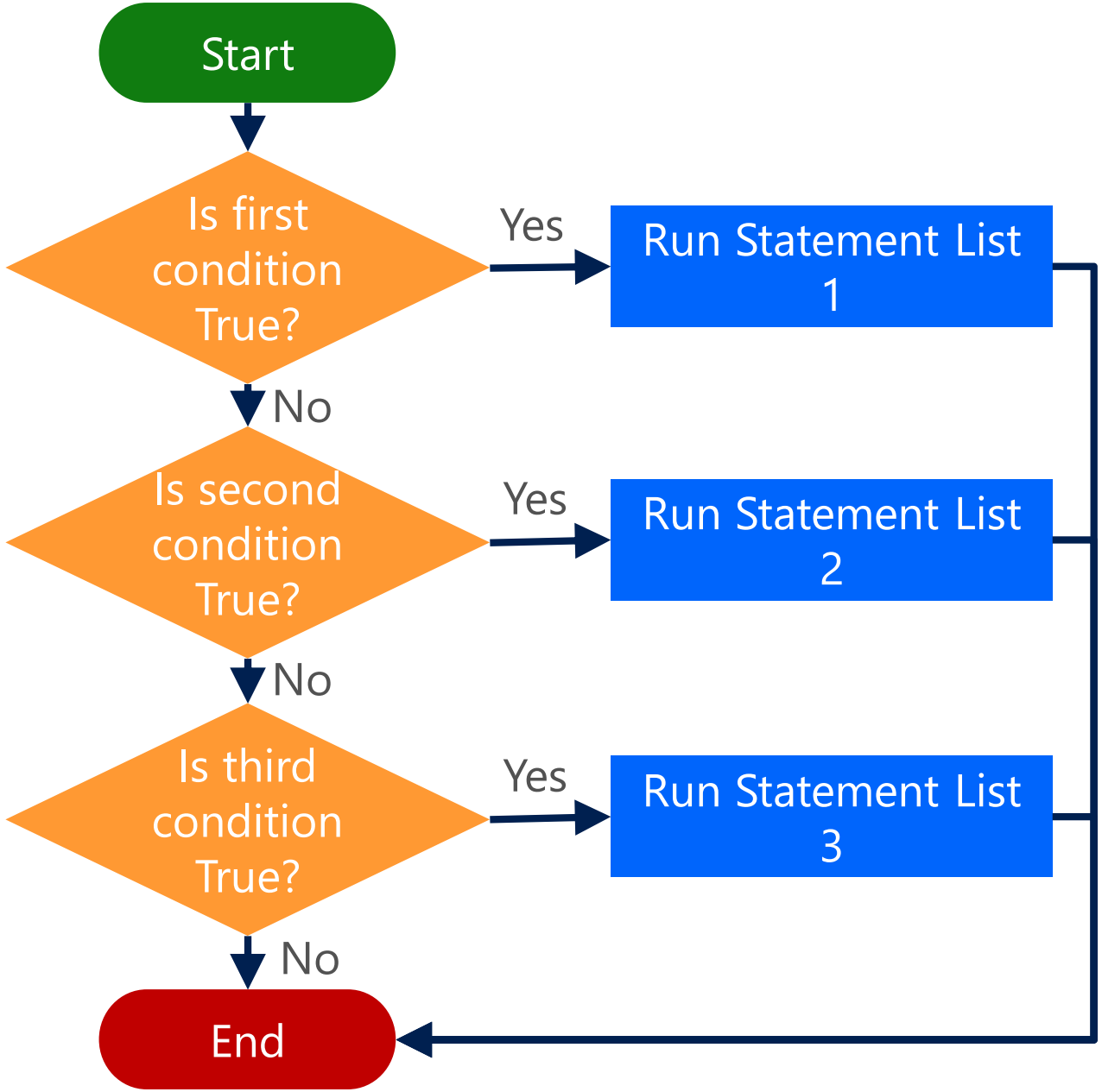
```
If (<test1>)
{
    <statement list 1>
}
Else
{
    <else statement list>
}
```




```
$Language = (Get-CimInstance -class win32_operatingsystem).OSLanguage

if ($Language -eq "1033")
{write-Host "Language = English US" -ForegroundColor Magenta}
else
{Write-Host "Another Language" -ForegroundColor Cyan}
```

If..Elseif(s)

```
If (<test1>)
{
    <statement list 1>
}
ElseIf (<test2>)
{
    <statement list 2>
}
ElseIf (<test3>)
{
    <statement list 3>
}
```

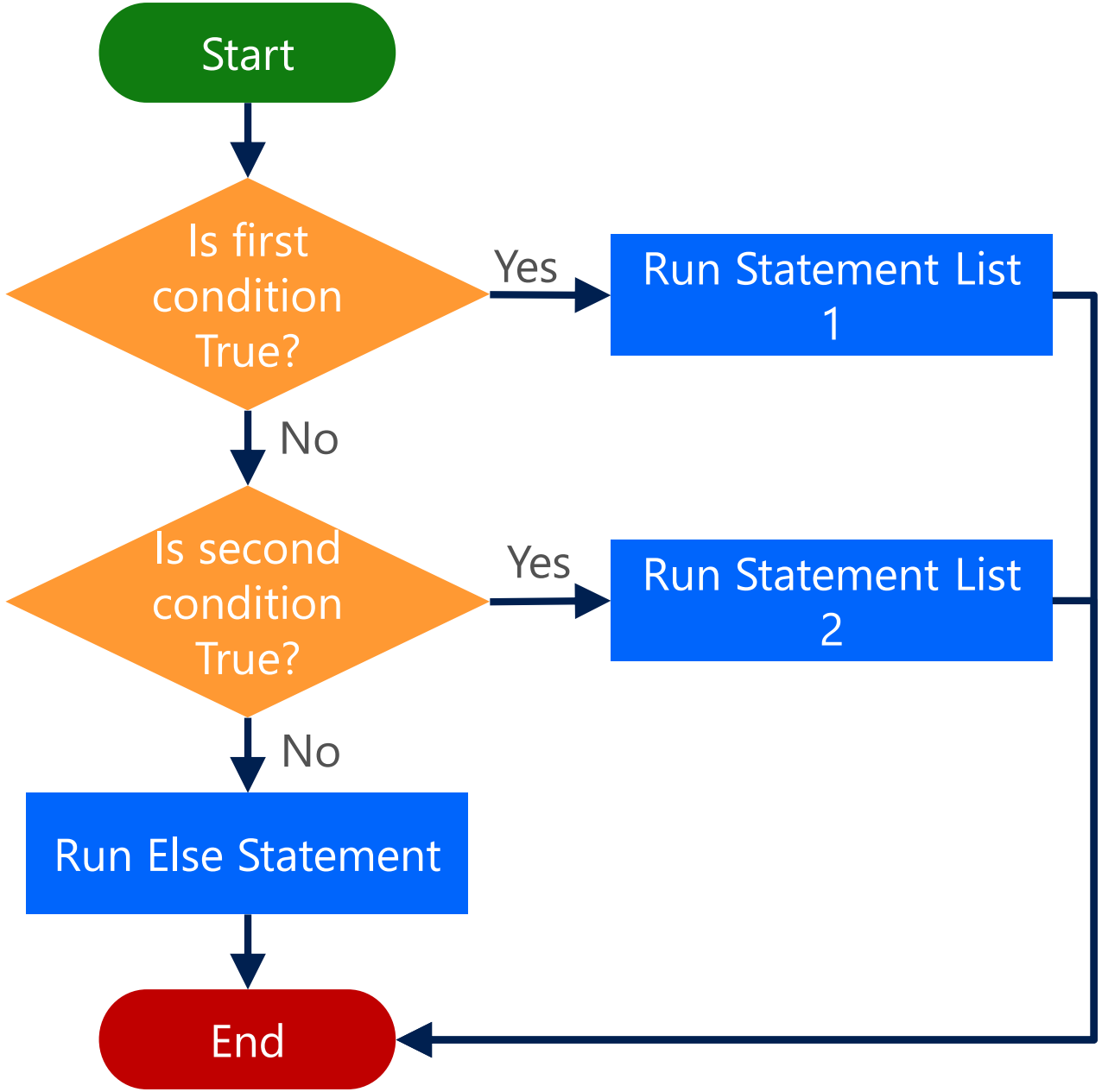


If..Elseif(s)

```
$Language = (Get-CimInstance -ClassName
Win32_OperatingSystem).OSLanguage

if ($Language -eq "1033")
{
  Write-Host "Language = English US" -ForegroundColor Magenta
}
elseif ($Language -eq "1078")
{
  Write-Host "Language = Afrikaans" -Foregroundcolor Green
}
```

If..ElseIf(s)..Else

```
If (<test1>)
{
    <statement list 1>
}
ElseIf (<test2>)
{
    <statement list 2>
}
Else
{
    <else statement list>
}
```



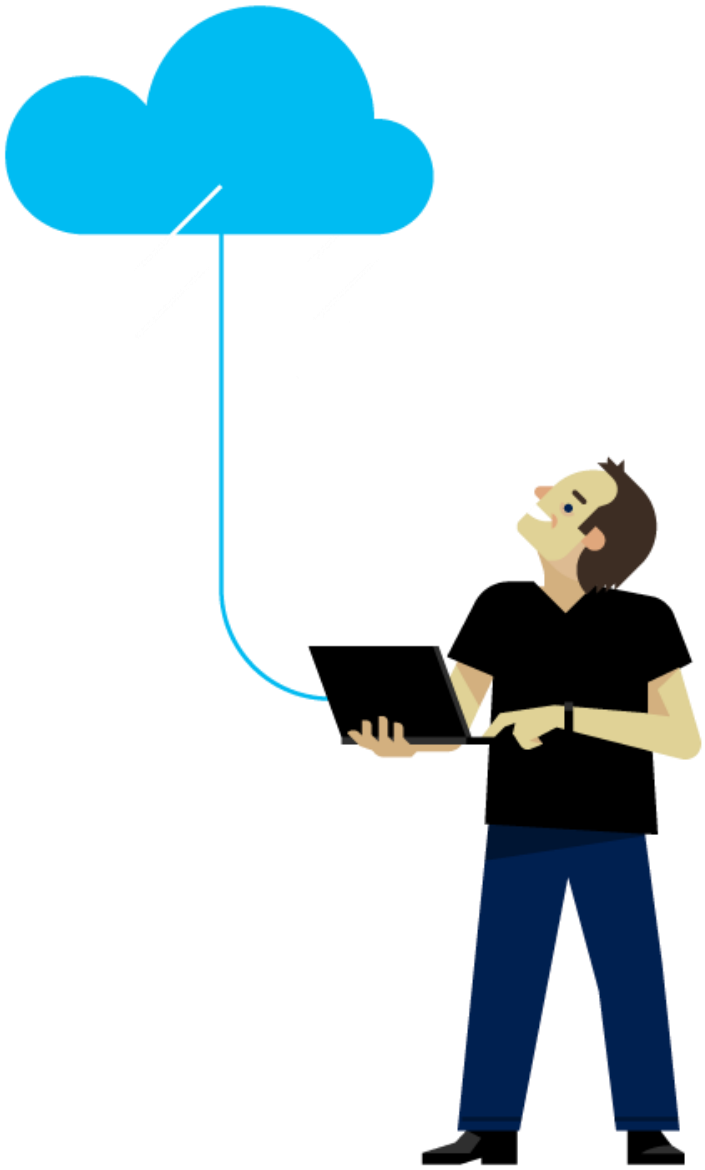
If..Elseif..Else Statement

```
$Language = (Get-CimInstance -ClassName
Win32_OperatingSystem).OSLanguage

if ($Language -eq "1033")
{
    Write-Host "Language = English US" -ForegroundColor Magenta
}
elseif ($Language -eq "1078")
{
    Write-Host "Language = Afrikaans" -Foregroundcolor Green
}
else
{
    Write-Host "Another Language" -ForegroundColor Cyan
}
```

Demonstration

IF Statements

Questions?



SWITCH Statement - Basics

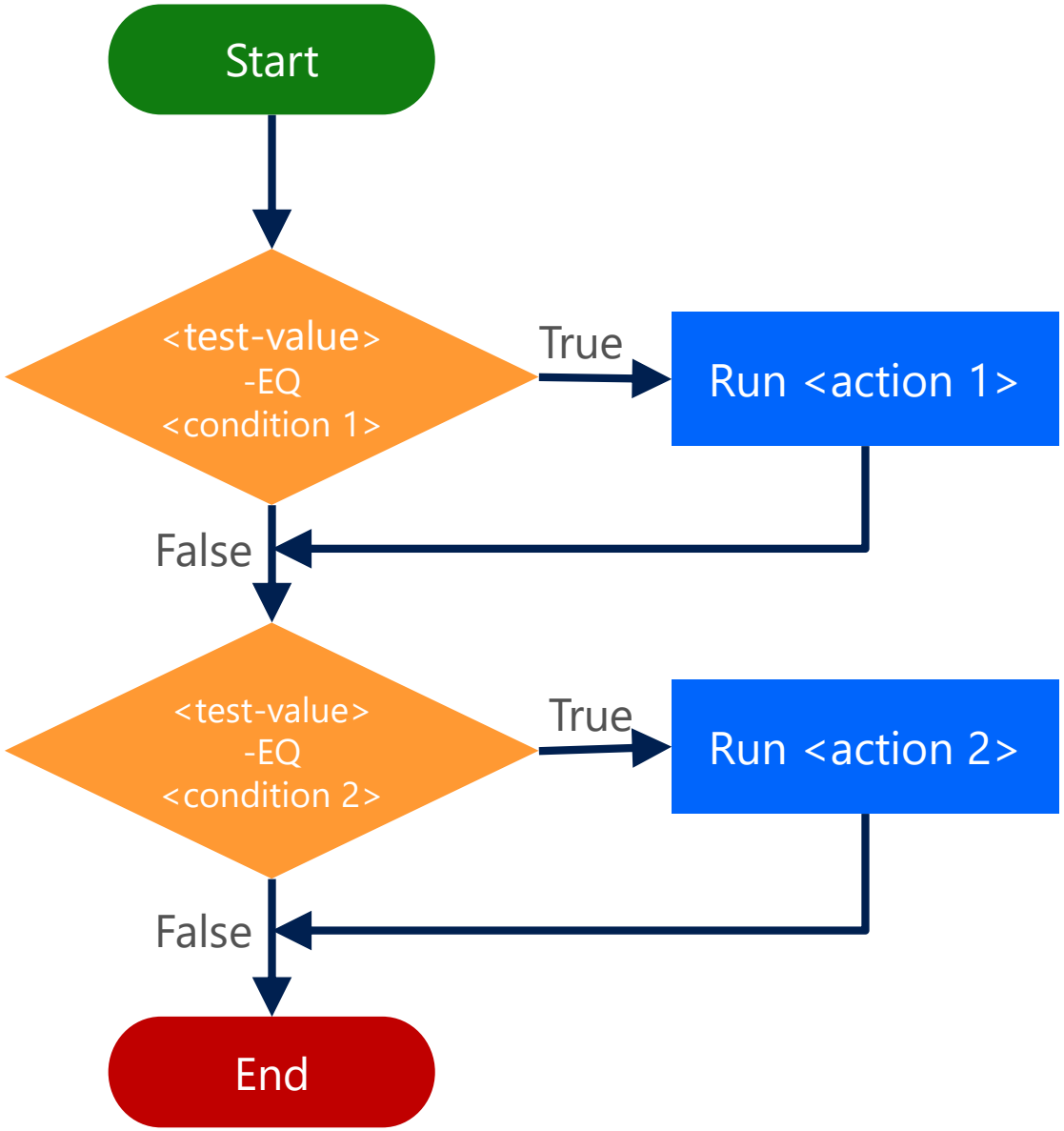
Switch

- Like a simplified version of an If with ElseIfs
- Called Select..Case in some other languages
- Can process multiple test values, operates like a pipeline
- Can accept file paths for process contents

Simple Switch

```
Switch (<test-value>)
   {
      <condition 1> {<action 1>}
      <condition 2> {<action 2>}
   }
```

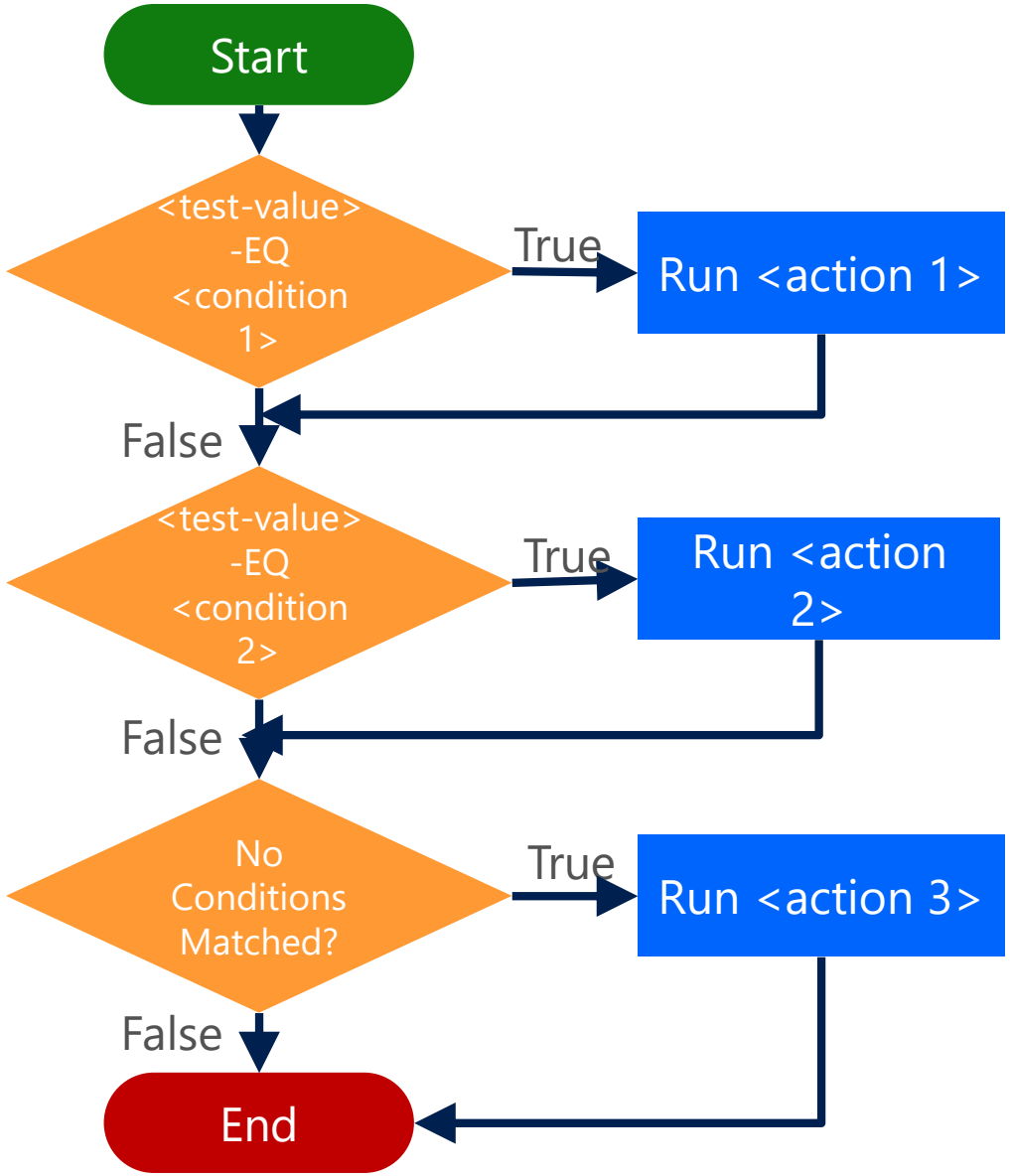
```
$number = 0
switch ($number)
{
   0      {write-Host “Its 0"}
   0      {write-Host “Its Zero"}
   2      {write-Host “Its 2"}
}
```



Switch – With Default Case

```
Switch (<test-value>)
   {
      <condition 1> {<action 1>}
      <condition 2> {<action 2>}
      Default       {<action 3>}
   }
```

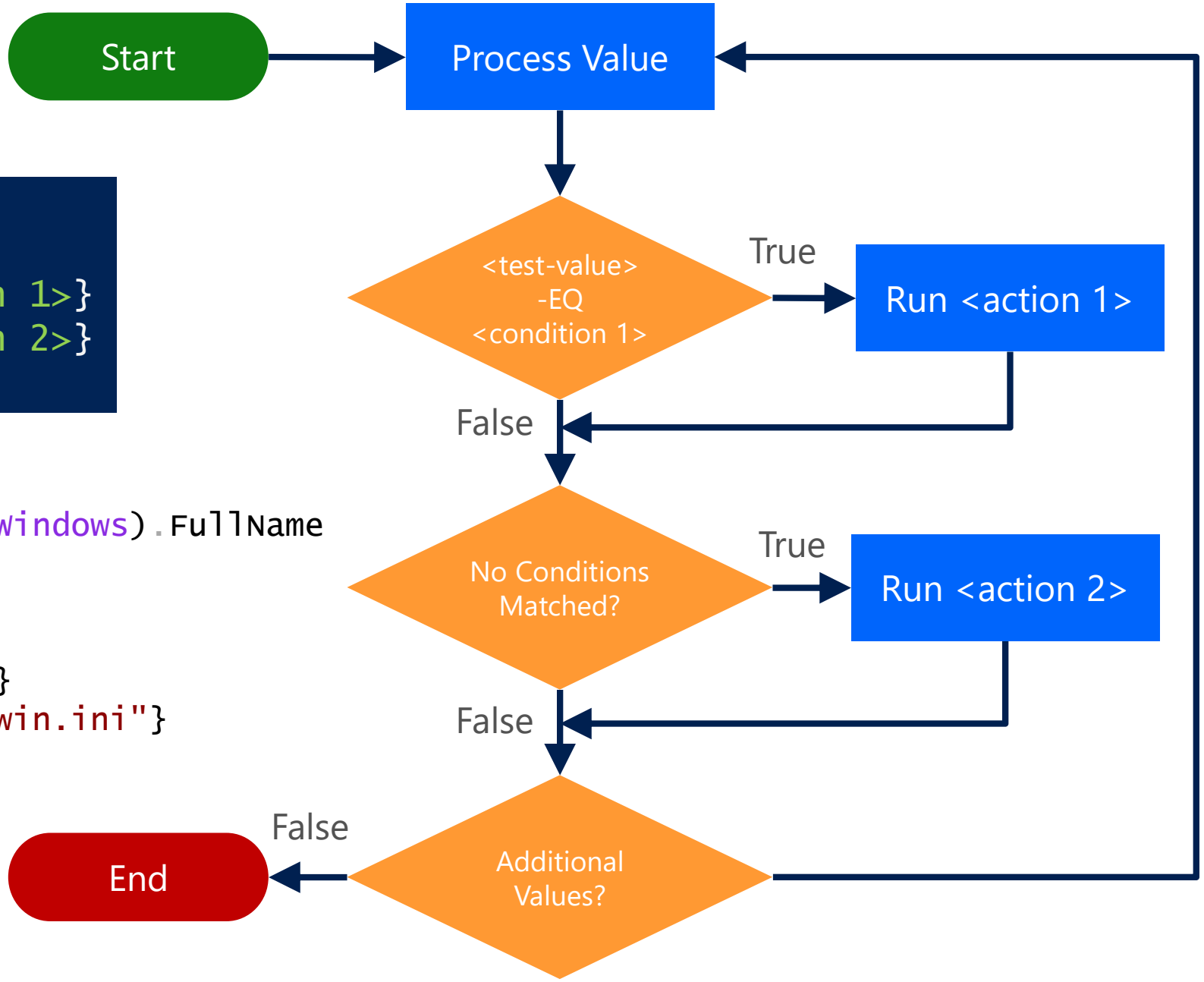
```
$number = 0
switch ($number)
{
   0      {write-Host “Its 0"}
   0      {write-Host “Its Zero"}
   2      {write-Host “Its 2"}
   Default {write-Host “Its Not 2 or 0"}
}
```



Switch
Multiple Values

```
Switch (<test-value-array>)
   {
      <condition 1> {<action 1>}
      Default       {<action 2>}
   }
```

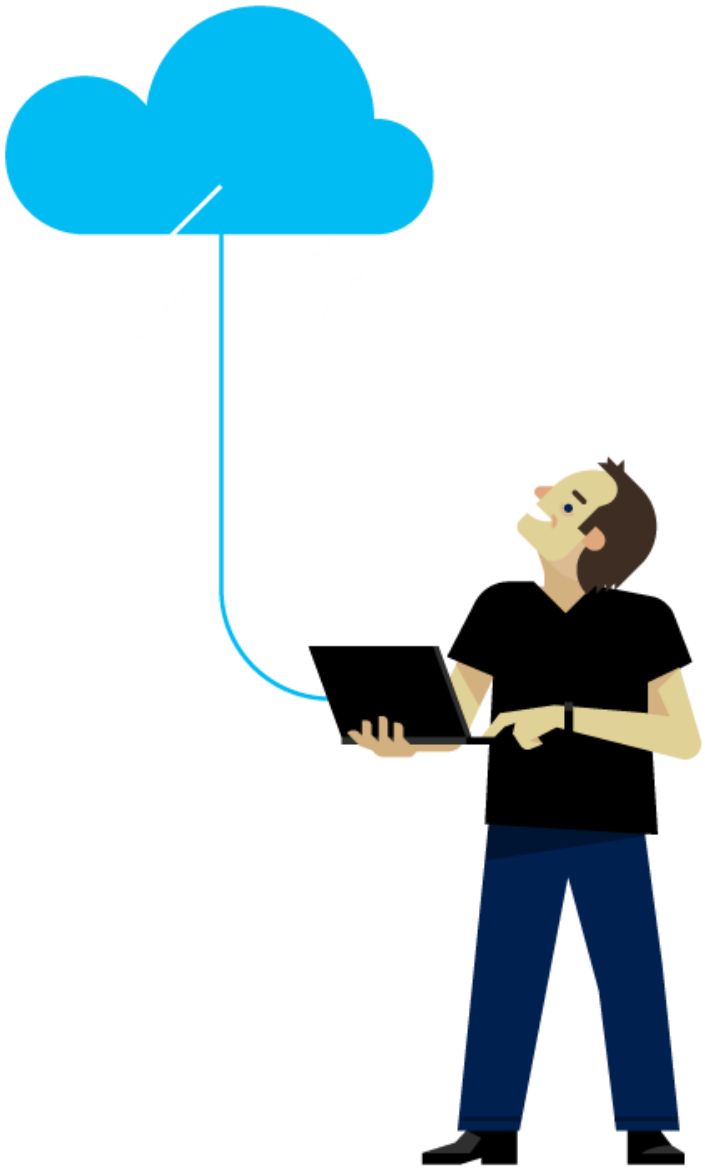
```
$FileNames = (Get-ChildItem C:\Windows).FullName
Switch -Wildcard ($FileNames)
{
    “hh.exe"    {"Found hh.exe"}
    “win.ini"   {"Found win.ini"}
    Default     {"Not hh.exe or win.ini"}
}
```



Demonstration

Basic Switch Statement

Questions?



SWITCH Statement - Advanced

Switch With $_

- SWITCH works like pipeline $_ or $PSItem can be used

```
$number = “1”,”2”,”3”

switch ($number)
{
   1 {Write-Host "$_ : The number 1"}
   2 {Write-Host "$_ : The number 2"}
}
```

```
PS C:\> 1 : The number 1
PS C:\> 2 : The number 2
```

SWITCH Case Insensitive

Case-insensitive by default

```
switch ("HELLO")
{
"hello" {"lowercase"}
"HELLO" {"uppercase"}
}
```

```
lowercase
uppercase
```

SWITCH Case Sensitive

Case sensitive switch

```
switch -CaseSensitive ("HELLO")
{
"hello" {"lowercase"}
"HELLO" {"uppercase"}
}
```

```
uppercase
```

SWITCH Without -Wildcard

Name	Date modified	Type
PerfLogs	8/22/2013 11:22 AM	File folder
Program Files	11/24/2013 10:52 ...	File folder
Program Files (x86)	8/22/2013 11:36 AM	File folder
PShell	4/22/2014 10:24 PM	File folder
Transcripts	9/8/2014 12:57 PM	File folder
Users	4/6/2014 9:14 PM	File folder
Windows	4/13/2014 9:35 PM	File folder

```
Switch (Get-ChildItem -Path c:\)
{
"program*" {Write-Host $_ -ForegroundColor Green}
"windows"  Write-Host $_ -ForegroundColor Cyan}
}
```

Wildcard characters not matched

```
Windows
```

SWITCH With -Wildcard

Name	Date modified	Type
PerfLogs	8/22/2013 11:22 AM	File folder
Program Files	11/24/2013 10:52 ...	File folder
Program Files (x86)	8/22/2013 11:36 AM	File folder
PShell	4/22/2014 10:24 PM	File folder
Transcripts	9/8/2014 12:57 PM	File folder
Users	4/6/2014 9:14 PM	File folder
Windows	4/13/2014 9:35 PM	File folder

```
switch -wildcard (Get-ChildItem -Path c:\)
{
"program*" {Write-Host $_ -ForegroundColor Green}
"windows"  {Write-Host $_ -ForegroundColor Cyan}
}
```

```
Program Files
Program Files (x86)
Windows
```

SWITCH With -Regex

```
switch -Regex (Get-ChildItem -Path c:\)
{
 "^program" {Write-Host $_ -ForegroundColor Green}
 "s$" {Write-Host $_ -ForegroundColor Cyan}
}
```

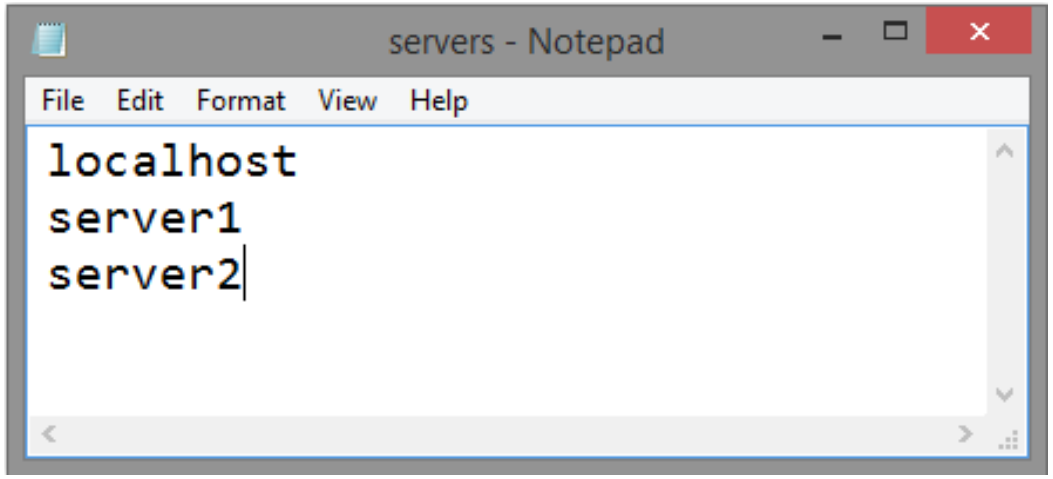
```
Program Files
Program Files
Program Files (x86)
Scripts
Users
Windows
```

Note	^	Matches beginning character(s)
	$	Matches end character(s)
	Get-Help about_Regular_Expressions	

SWITCH Expression Matches

```
PS C:\> switch (123) {

{$_ -lt 124} {Write-Host $_ -ForegroundColor Green}
{$_ -gt 200} {Write-Host $_ -ForegroundColor Cyan}

}

123
```

SWITCH -File




```
switch -File .\servers.txt
{
"server1" {Write-Host "$_ is in file" -F Green}
"server10" {Write-Host "$_ is in file" -F Cyan}
}
```

```
server1 is in file
```

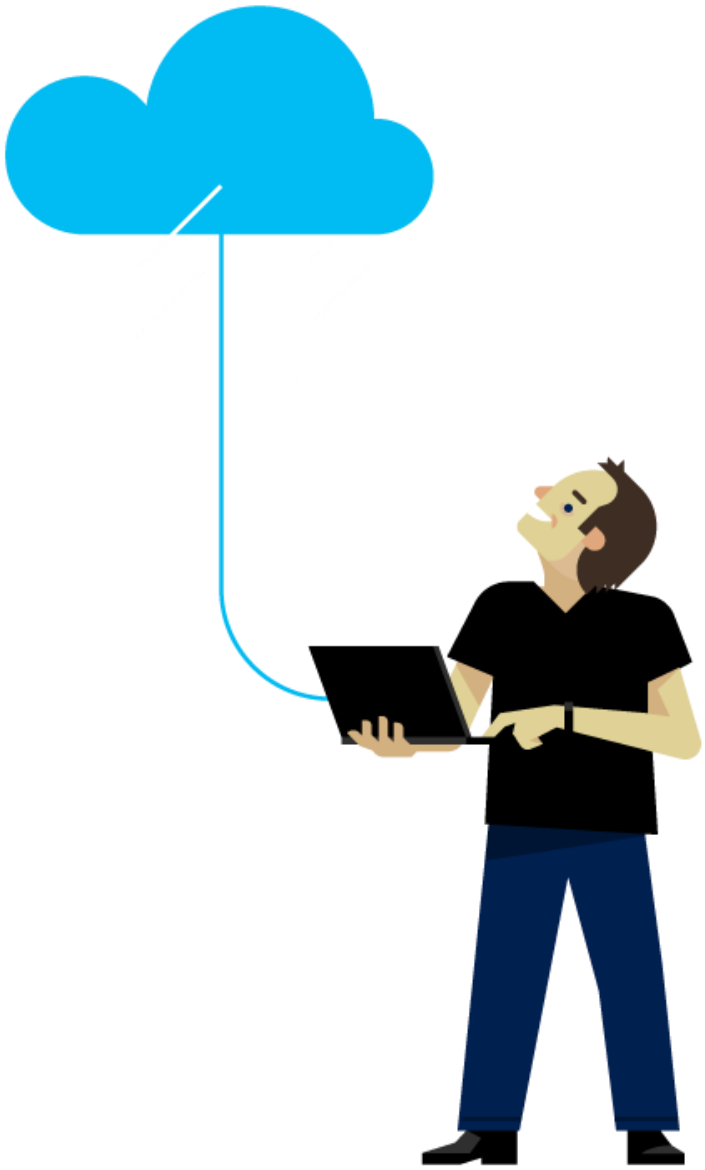
IF and SWITCH Difference

- IF statement terminates when a match is found
- SWITCH statement does not terminate when a match is found
- SWITCH is more suitable when multiple conditions are evaluated

Demonstration

Switch Statement - Advanced

Questions?



Flow Control Keywords

Flow Control Keywords

Keyword	Description	Example(s)

Break	Immediately exit Loop
	
	Example breaks after 1 match to avoid multiple matches
		`Switch -Wildcard ("WMF 5.0")`
		`{`
		`  "WMF 5.0" {"Matched First"; Break}`
		`  "W*"  {"Matched Second"}`
		`}`
		
		`Matched First`

Continue	Immediately returns to top of loop
	
	Example skips over 2
		`$c = 0`
		`while ($c -lt 3)`
		`{`
		`  $c++`
		`  if ($c -eq 2) {Continue}`
		`  Write-Host $c`
		`}`
		
		`1`
		`3`

Flow Control Keywords

Keyword	Description	Example(s)

Return	Exits current 'scope', which can be a function, script, or script block
	
	Note: Return can appear alone or followed by a value or expression
		`function Test-Return ($val)`
		`{`
		`  if ($val -ge 5) {return $val}`
		`  Write-Host "Reached end of function"`
		`}`
		
		`PS C:\> Test-Return 1`
		`Reached end of function`
		`PS C:\> Test-Return 6`
		`6`

Exit	Exit current script or session – Optional ErrorLevel Numeric Code	`PS C:\> Exit 10`

Note: 'Scopes' are covered in a different course

Exit and ErrorLevel From Cmd Prompt

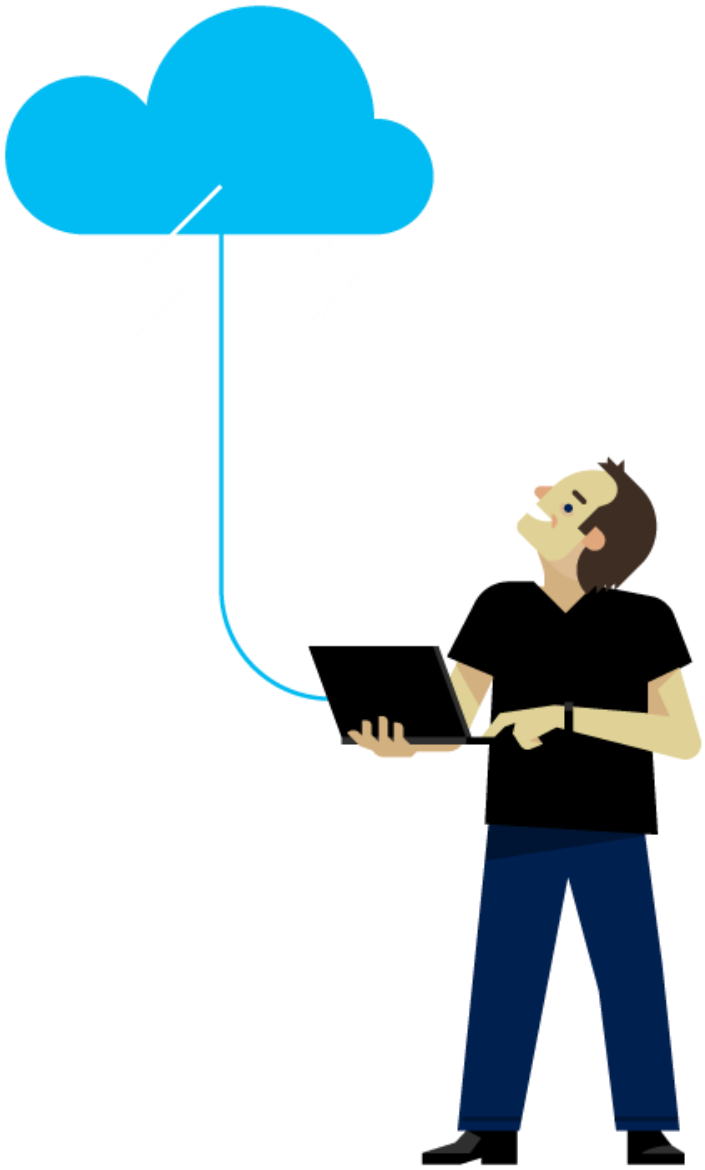
Administrator: Command Prompt

```
C:\>powershell.exe -noprofile
Windows PowerShell
Copyright (C) 2013 Microsoft Corporation. All rights reserved.

PS C:\> "Something went wrong"
Something went wrong
PS C:\> Exit 10

C:\>echo %ErrorLevel%
10

C:\>_
```

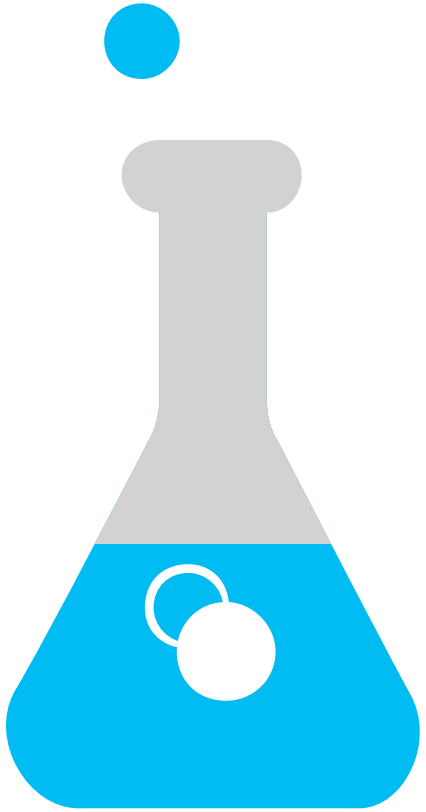
Demonstration

Flow Control

Questions?



Flow Control



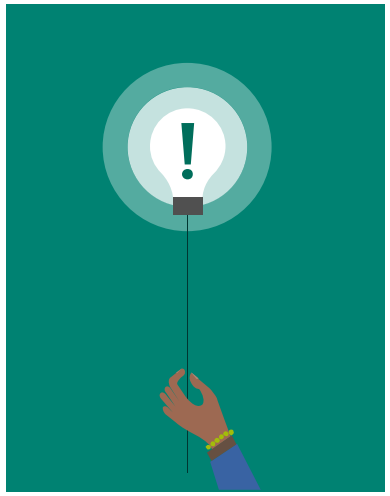
LAB

Arrays

Objectives

After completing Arrays, you will be able to:

- Work and manipulate PowerShell arrays



Creating Arrays

Creating Arrays

Arrays can be created in a number of ways:

Cmdlets that return multiple items

```
PS C:\> $processarray = Get-Process
```

Assigning multiple values to a variable

```
PS C:\> $array = 22,5,10,8,12,9,8
```

Array sub-expression operator

```
PS C:\> $b = @()
```

Demonstration

Creating Array Objects

Accessing Array Items

Accessing Array Items

Display all items in an array

```
PS C:\> $array
22
5
10
8
12
9
8
```

First item in array – using index position

```
PS C:\> $array[0]
22
```

Last item in array – using index position

```
PS C:\> $array[-1]
8
```

Display first 3 items in array

```
PS C:\> $array[0..2]
22
5
10
```

Display first item and last item in array

```
PS C:\> $array[0,-1]
22
8
```

Determine Number of Items in Array

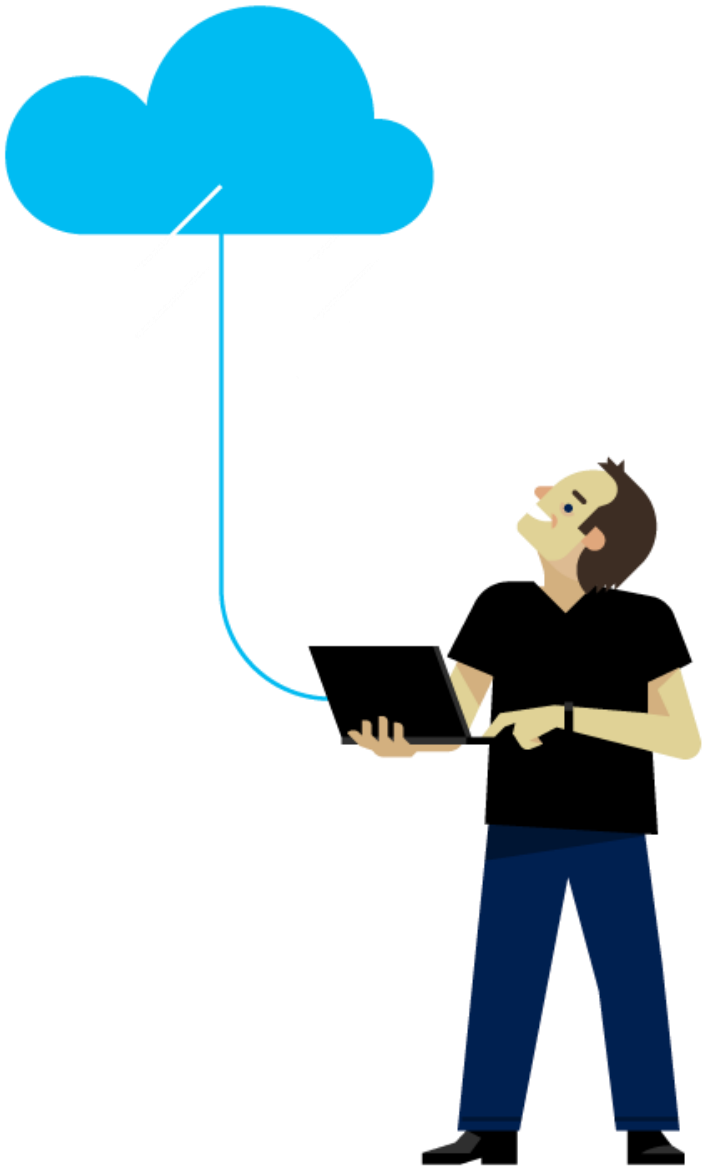
Determine the number of items in an array

```
PS C:\> $array.Count
7

PS C:\> $array.Length
7
```

Demonstration

Accessing Array Items

Questions?



Adding and Modifying Array Items

Adding Items to an Array

Adding items to an array

```
PS C:\> $array += 999

PS C:\> $array
22
5
10
8
12
9
8
999
```

Manipulating Items in an Array

Manipulating items in an array – Using assignment operator

```
PS C:\> $array[0] = 100

PS C:\> $array
100
8
8
9
10
12
22
999
```

Manipulating items in an array – Using array "Set" method

```
PS C:\> $array.Set(0,200)

PS C:\> $array
200
8
8
9
10
12
22
999
```

Demonstration

Adding and Modifying Array Objects

Sorting Arrays

Sorting Array Display

Sort-Object only sorts the console output – Array order is not changed

```
PS C:\> $array | Sort-Object -Descending
999
22
12
10
9
8
8
5
```

Sorting Array

Array Type Static Method Sort changes item order

```
PS C:\> [array]::Sort($myarray)
PS C:\> $myarray
5
8
8
9
10
12
22
999
```

Determine Array Object Members

Determine Array Object Members

Piping to Get-Member discovers item members – Not array members

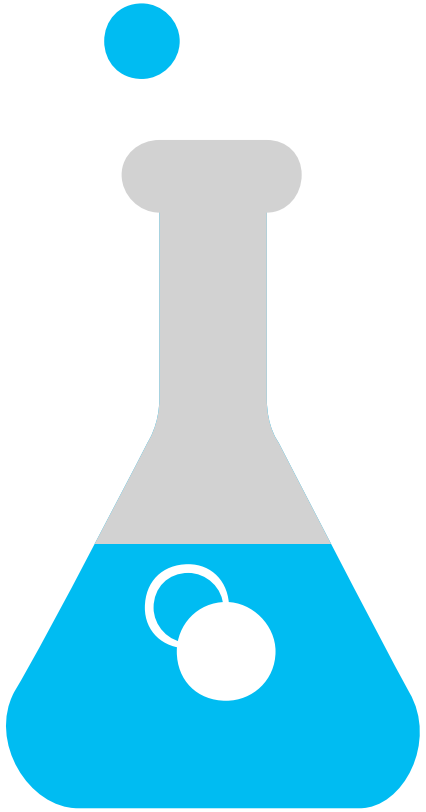
```
PS C:\> $array | Get-Member
TypeName: System.Int32

Name          MemberType      Definition
----          -----------     -----------
CompareTo     Method          int Compare
…
```

Determine Array Object Members

Use Get-Member –Inputobject parameter to get array members

```
PS C:\> Get-Member -InputObject $array
TypeName: System.Object[]

Name          MemberType          Definition
-------       ----------------    ------------
Count         AliasProperty       Count = Length
Add           Method              int
…
```

Arrays



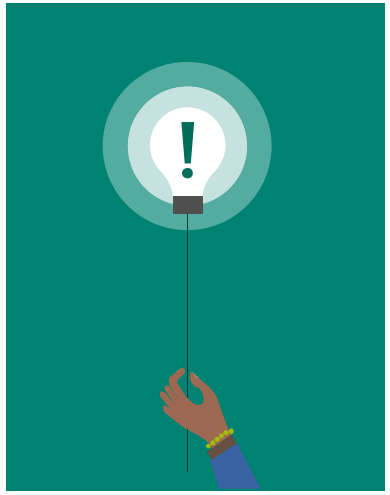
LAB

Hash Tables

Objectives

After completing Hash Tables, you will be able to:

- Work and manipulate PowerShell Hash tables



Creating Hash Tables

Creating Hash Tables

Empty hash table

```
PS C:\> $hash = @{}
```

Create and populate hash table

```
PS C:\> $Server = `
@{ 'HV-SRV-1'='192.168.1.1' ; Memory=64GB ; Serial='THX1138'}

PS C:\> $Server

Name                                   Value
----                                   -----
HV-SRV-1                               192.168.1.1
Serial                                 THX1138
Memory                                 68719476736
```

Creating Hash Tables

Create a hash table from here string data

```
PS C:\> $string = @"
Msg1 = Hello
Msg2 = Enter an email alias
Msg3 = Enter an username
Msg4 = Enter a domain name
"@

PS C:\> ConvertFrom-StringData -StringData $string

Name                                   Value
----                                   -----
Msg4                                   Enter a domain name
Msg3                                   Enter an username
Msg2                                   Enter an email alias
Msg1                                   Hello
```

Creating Hash Tables

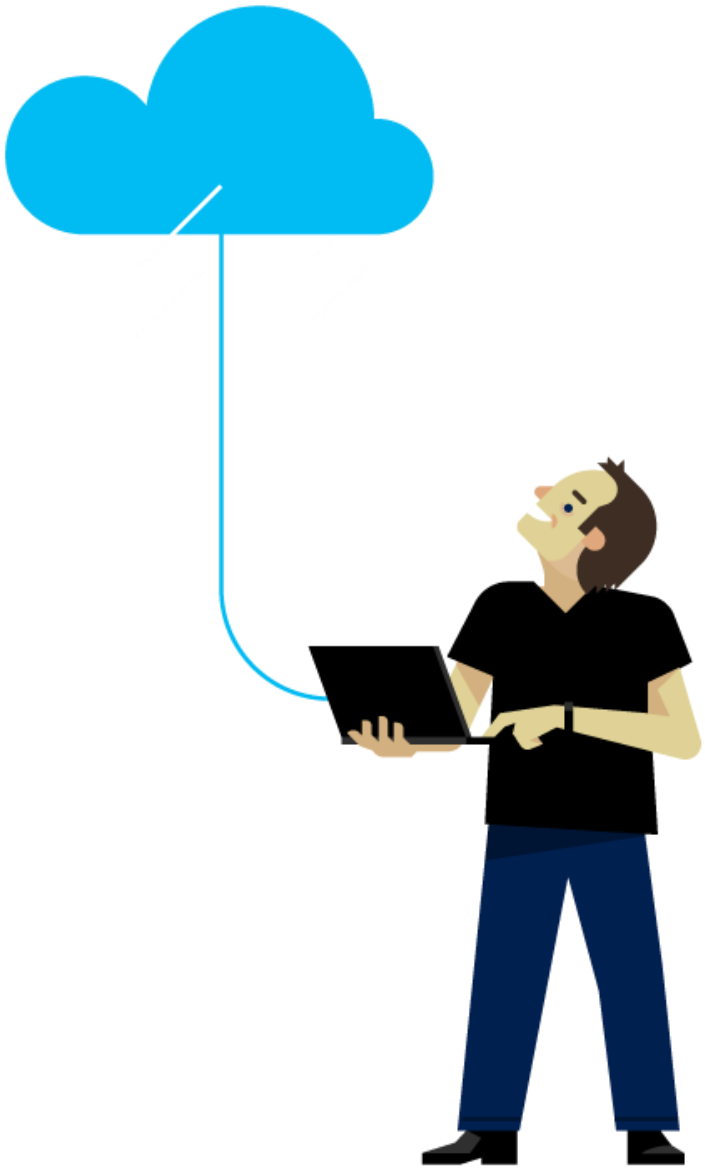
Create a hash table of services using Group-Object -AsHashTable

```
PS C:\> $svcshash = Get-Service |
Group-Object Status -AsHashTable -AsString

PS C:\> $svcshash

Name        Value
----        -----
Stopped     {AeLookupSvc, ALG, AppMgmt, AppReadiness...}
Running     {AppIDSvc, Appinfo, AudioEndpointBuilder,...}

PS C:\> $svcshash.Stopped

Status    Name           DisplayName
------    ----           -----------
Stopped   AeLookupSvc    Application Experience
...
```

Demonstration

Creating Hash Tables

Questions?



Accessing Hash Table Items

Access Hash Tables Items

Display all items in hash table

```
PS C:\> $Server

Name           Value
----           -----
HV-SRV-1     192.168.1.1
Serial       THX1138
Memory       68719476736
```

Return value using dot notation

```
PS C:\> $Server.'HV-SRV-1'
192.168.1.1

PS C:\> $Server.Serial
THX1138
```

Return value using index notation

```
PS C:\> $Server["Serial"]
THX1138
```

Display All Hash Tables Keys

Display all keys in hash table

```
PS C:\> $Server.Keys
HV-SRV-1
Serial
Memory
```

Display All Hash Tables Values

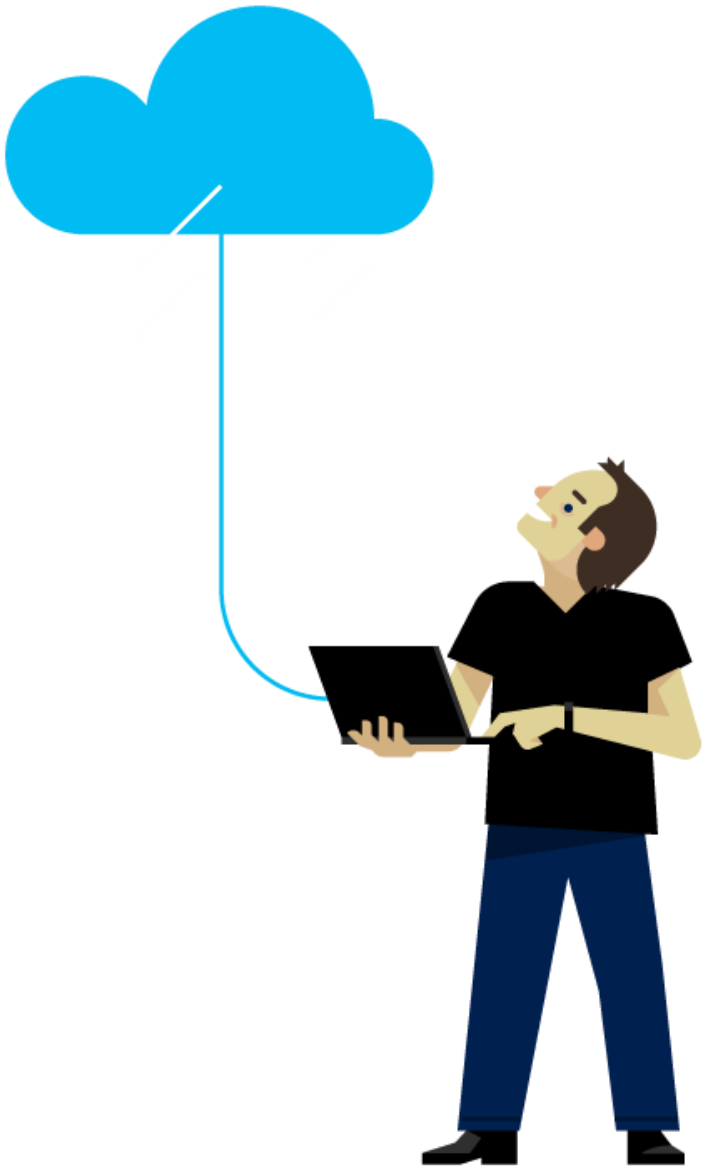
Display all values in hash table

```
PS C:\> $Server.Values
192.168.1.1
THX1138
68719476736
```

Demonstration

Accessing Hash Table Keys

Questions?



Modifying Hash Table Items

Adding Items To a Hash Table

Add or set key and value using index notation

```
PS C:\> $Server["CPUCores"] = 4
```

Add or set key and value using dot notation

```
PS C:\> $Server.Drives="C", "D", "E"
```

Add key and value using hash table ADD method

```
PS C:\> $Server.Add("HotFixCount", `
(Get-HotFix -Computer $Server[“HV-SRV-1”]).count)
```

Removing Items From a Hash Table

Remove key

```
PS C:\> $Server.Remove("HotFixCount")
```

Demonstration

Modifying Hash Table Items

Sorting Hash Tables

Sorting Hash Tables

- Hash tables are intrinsically unordered
- It is not possible to sort a hash table
- GetEnumerator() method used with Sort-Object Cmdlet

Sort hash table display by key

```
PS C:\> $Server.GetEnumerator() | Sort-Object -
Property key

Name          Value
----          -----
CPUCores      4
Drives        {C, D, E}
HV-SRV-1      192.168.1.1
Memory        68719476736
```

Ordered Dictionary

- Alternative to regular hash tables
- Works similarly to a hash table but order is preserved

```
PS C:\> @{firstname = "John" ; lastname = "Smith"}

Name                           Value
----                           -----
lastname                       Smith
firstname                      John

PS C:\> [ordered]@{firstname = "John" ; lastname = "Smith"}

Name                           Value
----                           -----
firstname                      John
lastname                       Smith
```

Order not preserved

Insertion order preserved

Demonstration

Sorting Hash Tables

Searching Hash Tables

Searching Hash Tables

Searching a hash table

```
PS C:\>  $hash =
@{"John"=23342;"Linda"=54345;"James"=65467}

PS C:\>  $hash.ContainsKey("Linda") #Fast hashed key search
True

PS C:\>  $hash.ContainsValue(19)    # Slow non-hashed search
False

PS C:\>  $hash.ContainsValue(65467)
True
```

Hash Table Example

Calculated Property

- Customizing property value on pipeline with Select-Object and a hash table
- Length property is in kilobytes and limited to 2 decimal points before displayed

```
PS C:\> Get-ChildItem C:\Windows | Select-Object Name,
@{Name="Size (KB)";Expression={"{0:N2}" -f ($_.Length/1Kb)}}

Name          Size (KB)
----          ---------
HelpPane.exe  950.50
un_dext.exe   94.91
```

Key Value Key Value

```
@{Name="Size (KB)";Expression={"{0:N2}" -f ($_.Length/1Kb)}}
```

Hash Table

Splatting

Passing a hash table as parameters to a cmdlet, function or script
Referred to as 'Splatting'

```
$params = @{
  LogName       = "application"
  Newest        = 10
  EntryType     = "Warning"
  ComputerName  = "localhost"
}

Get-EventLog @Params
```

Custom PSObject

Create a customized object (PS v2.0+) – Ordering Not Preserved

```
$props = @{
  Computer = (Get-WmiObject -Class Win32_computersystem).Name

  Name = (Get-NetAdapter -Physical |
    Where-Object {$_.status -eq “up”}).Name

  Speed = (Get-NetAdapter -Physical |
    Where-Object {$_.Status -eq “up”}).Linkspeed
}

$notpreserved = New-Object PSObject -Property $props
```

Custom PSObject

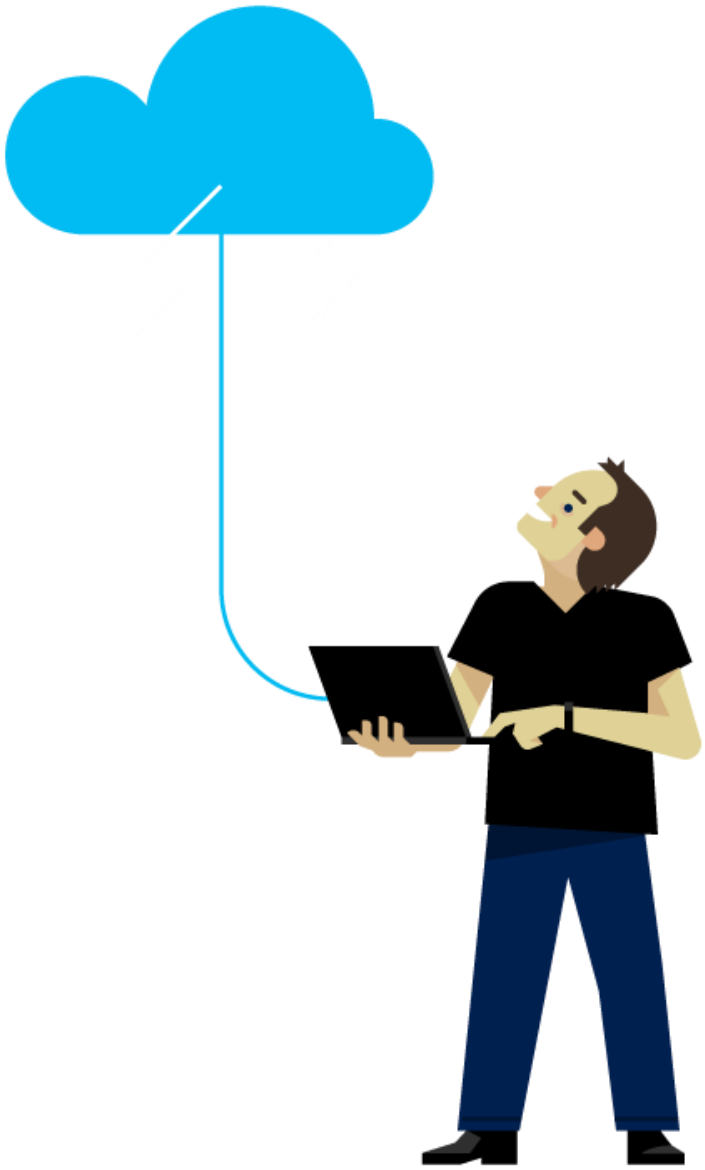
Create a customized object (PS v3.0+) – Ordering Preserved

```
$preserved = [PSCustomObject]@{
  Computer = (Get-WmiObject -Class Win32_computersystem).Name

  Name = (Get-NetAdapter -Physical |
    Where-Object {$_.status -eq “up”}).Name

  Speed = (Get-NetAdapter -Physical |
    Where-Object {$_.Status -eq “up”}).Linkspeed
}
```

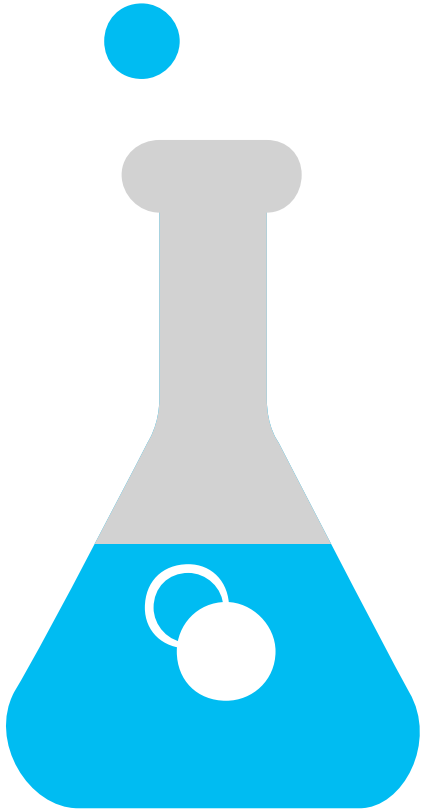
Demonstration

Hash Table

Questions?



Hash Tables



LAB

Microsoft