Conditional logic

if, else and else if

If Statements

```
$test = "test"


if ($test -eq "test"){
    Write-Host "if condition met"
}
```

Using else

```
$test = "test"

if ($test -eq "test2"){
    Write-Host "if condition met"
}
else{
    Write-Host "if condition not met"
}
```

Using elseif

```
$test = "test"


if ($test -eq "test2"){
    Write-Host "if condition met"
}
elseif ($test -eq "test"){
    Write-Host "ifelse condition met"
}
```

Negation

```
$test = "test"

if (-Not $test -eq "test2"){
    Write-Host "if condition not met"
}
```

- Or use !

```
if (!($test -eq "test2")){
    Write-Host "if condition not met"
}
```

- Implement the same using the -ne (not equal) operator

Loops

- A loop is a sequence of instruction(s) that is continually repeated until a certain condition is reached. Being able to have your program repeatedly execute a block of code is one of the most basic but useful tasks in programming.
- A loop lets you write a very simple statement to produce a significantly greater result simply by repetition.
- If the condition has been reached, the next instruction "falls through" to the next sequential instruction or branches outside the loop.
- ForEach has two different meanings in PowerShell. One is a keyword and the other is an alias for the ForEach-Object cmdlet.

Foreach

```
$Names = @('Amy', 'Bob', 'Celine', 'David')
ForEach ($Name in $Names){
    Write-Host "Hi, my name is $Name!" }
```

- Capturing the output of a ForEach loop:

```
$Numbers = ForEach ($Number in 1..20) {
$Number # Alternatively, Write-Output $Number }
```

- Creating an array prior to storing the loop:

```
$Numbers = @()
ForEach ($Number in 1..20){
$Numbers += $Number }
```

For

```
for($i = 0; $i -le 5; $i++){
"$i"
}
```

ForEach() Method

- Instead of the ForEach-Object cmdlet, the here is also the possibility to use a ForEach method directly on object arrays like so

```
(1..10).ForEach({$_ * $_})
```

- or - if desired - the parentheses around the script block can be omitted

```
(1..10).ForEach{$_ * $_}
```

ForEach-Object

- The ForEach-Object cmdlet works similarly to the foreach statement, but takes its input from the pipeline.
- Basic Syntax:

```
$object | ForEach-Object {
code_block }
```

- Example:

```
$names = @("Any","Bob","Celine","David")
$names | ForEach-Object {
"Hi, my name is $_!" }
```

Avoiding confusion

- Foreach-Object has two default aliases, foreach and % (shorthand syntax). Most common is % because foreach can be confused with the foreach statement.
- Examples:

```
$names | % { "Hi, my name is $_!" }

$names | foreach { "Hi, my name is $_!" }
```

Continue

- The Continue operator works in For, ForEach, While and Do loops. It skips the current iteration of the loop, jumping to the top of the innermost loop.

```
$i =0
while ($i -lt 20) {
$i++
if ($i -eq 7) { continue }
Write-Host $i }
```

- The above will output 1 to 20 to the console but miss out the number 7.
- Note: When using a pipeline loop you should use return instead of Continue.

Break

- The break operator will exit a program loop immediately. It can be used in For, ForEach, While and Do loops or in a Switch Statement.

```
$i = 0
while ($i -lt 15) {
$i++
if ($i -eq 7) {break}
Write-Host $i }
```

- The above will count to 15 but stop as soon as 7 is reached.

Break Labels

- Break can also call a label that was placed in front of the instantiation of a loop:

```
$i = 0
:mainLoop While ($i -lt 15) {
    Write-Host $i -ForegroundColor 'Cyan'
    $j = 0
    While ($j -lt 15) {
    Write-Host $j -ForegroundColor 'Magenta'
    $k = $i*$j
    Write-Host $k -ForegroundColor 'Green'
    if ($k -gt 100) {
    break mainLoop }
    $j++ }
    $i++ }
```

- This code will increment $i to 8 and $j to 13 which will cause $k to equal 104. Since $k exceed 100, the code will then break out of both loops.

While

- A while loop will evaluate a condition and if true will perform an action. As long as the condition evaluates to true the action will continue to be performed.

```
while(condition){
code_block }
```

- The following example creates a loop that will count down from 10 to 0

```
$i = 10
while($i -ge 0){
$i
$i--
}
```

Do

- Loop while the condition is true:

```
Do {
code_block
} while (condition)
```

- Loop until the condition is true, in other words, loop while the condition is false:

```
Do {
code_block
} until (condition)
```

Basic Functions

- A function can be defined with parameters using the param block:
- function Write-Greeting {

  ```
  param( [Parameter(Mandatory,Position=0)]
  [String]$name,
  [Parameter(Mandatory,Position=1)]
  [Int]$age )
  "Hello $name, you are $age years old." }
  ```
- Or using the simple function syntax:

  ```
  function Write-Greeting ($name, $age) {
  "Hello $name, you are $age years old." }
  ```
- Calling:
- `$greeting = Write-Greeting "Jim" 82`
- Alternatively, this function can be invoked with named parameters
- `$greeting = Write-Greeting -name "Bob" -age 82`

Be Creative :)

- ICMP enumeration

  -
    ```
    1..255 | % {echo "192.168.63.$_"; ping -n 1 -w 100 192.168.63.$_ | Select-String ttl}
    ```
- TCP-connect port scanner

  -
    ```
    1..1024 | % {echo ((New-Object Net.Sockets.TcpClient).Connect("192.168.63.147", $_)) "Open port - $_"} 2>$null
    ```

Offensive Powershell

- We will dive deeper on Offensive PowerShell
- Like:
  - Delivering a backdoor to your target via PowerShell
  - Empire framework
  - Lateral movement
  - AD attacks and More

EOF

John Ombagi
jnyabuti@strathmore.edu