# Lab Exercises

This section of the PowerShell training is supposed to be executed in a PowerShell console or PowerShell ISE. Results may vary depending if the commands are executed in the console, the ISE, 32&64-bit environment or running with administrative privileges. Feel free to experiment with the alternatives to discover the differences.

If there are any questions or if there is anything you would like to have explained in greater detail, feel free to contact me directly. As always there are a hundred different ways of achieving the same result, so trying out different approaches for the same problem can be a learning experience.

## Exercise 1 – Create a variable in PowerShell

Use the New-Variable cmdlet to create a variable named $Text with a value of ‘Hello World!’

New-Variable -Name Text -Value 'Hello World!'

Write-Output -InputObject $Text

Write-Output $Text

Echo $Text

$Text

Using Get-Member it is possible to establish what kind of an object we are dealing with here, execute the following code to view the object type for the $Text variable:

$Text | Get-Member

Using the output of the previous command we discover a number of properties, using the length property we can retrieve the length of the string in our variable $Text:

$Text.Length

There is also a SubString method available on the object:

$Text.Substring(4)

$Text.Substring

$Text.Substring(6,5)

## Exercise 2 – Working with Booleans

In PowerShell in order to display boolean values the following two default variables can be used:

$true

$false

Booleans are important in if-elseif-else statements and therefore it is important to know how PowerShell defines Booleans. For example:

[boolean]0

[boolean]16

Now this also works in a similar way with strings:

[boolean]''

[boolean]'true'

It can get confusing when the following notation is used:

[boolean]'false'

Therefore it is recommended that when creating if-elseif-else statements to use the default values $false & $true

[boolean]$false

Here is an example of what can happen if this is not used correctly:

$value = 16

if (($value -eq $value) -eq 'false') {

"The value '$value' is not equal to '$value'"

}

Attempt to fix this if-statement using the steps described in this exercise.

## Exercise 3 – Create an array in PowerShell

Using the .. notation we can create a list of numbers which we can use to define an array. Enter the following code:

$Array = 1..10

Using the brackets [] it is possible to select a single object in an array:

$Array[5]

The object can also be manipulated as such:

$Array[5] = 'Six'

To add something to an array the following notations can be used:

$Array = $Array + 11

$Array += 12

Display the new contents of the array:

$Array

Now that the array contains two different object types, a string and 11 integers:

$Array | Get-Member | Select-Object -Property TypeName -Unique

## Exercise 4 – Create an hash tables and custom objects

First we create a hash table using the @{} notation:

$HashTable = @{

Key1 = 'value'

'N2' = '2nd value'

}

To view the contents of the $HashTable we can enter it in the console:

$HashTable

Adding a value to a hash table can be done directly in the following manner:

$HashTable.Third = 3

$HashTable

Similarly, a PowerShell custom object can be created using a hash table:

$PSObject = New-Object -TypeName PSCustomObject -Property @{

'Property1' = 1

'123' = 'onetwothree'

'Date' = Get-Date

}

$PSObject

A PowerShell object cannot be extended with new properties in the same manner as a hash table can, the following example will fail:

$PSObject.NewProperty = 1

Another way of creating PowerShell custom object is by using the [pscustomobject] type-accelerator. Note that this only works on PowerShell 3.0 and higher, so if compatibility is a consideration this notation should be avoided:

[pscustomobject]@{

1 = 'one'

2 = 'two'

}

Alternatively, a PowerShell custom object can also be defined directly from a hash table, the following two notations will have the same result:

New-Object -TypeName PSCustomObject -Property $HashTable

[pscustomobject]$HashTable

## Exercise 4 – Manipulating strings

To create a string object the following notation can be used:

$String = 'The quick brown fox jumps over the lazy dog'

Note the difference when using the += notation with a string as compared to an array:

$String += 123

$String

Using the –replace operator it is possible to manipulate text in strings:

$String -replace 'brown fox','polar bear'

The type of quotation marks that are used also change the behavior of string expansion in PowerShell:

"The string is: $String"

'The string is: $String'

It is still possible to add values to a single-quoted string using the –f operator:

'The string is: {0}' -f $String

This can also be used for numeral manipulation:

'This is hexadecimal : {0:X2}' -f 250

'This is a percentage : {0:P2}' -f 0.66666

'Rounded to 2 decimals: {0:N2}' -f 3.99999