# 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 – Working with PowerShell Modules

To get a list of module that are currently loaded execute the following cmdlet:

Get-Module

To get a full list of installed modules, the –ListAvailable switch can be used:

Get-Module -ListAvailable

If the Bitlocker module is available on your system, use the following example, otherwise select another module from the list generated by the previous command:

Import-Module -Name BitLocker

To see the commands that have become available by loading the BitLocker module, we can use Get-Command in combination with the –Module parameter:

Get-Command -Module BitLocker

Get-BitLockerVolume

The Find-Module cmdlet can be used to find modules in the PowerShell Gallery or in other repositories:

Find-Module -Name CustomizeWindows10

To find more information about this module, you can view the metadata property of any module:

Find-Module -Name CustomizeWindows10 | Select-Object -ExpandProperty AdditionalMetadata

(Find-Module -Name CustomizeWindows10).AdditionalMetadata

To install the module, the output of Find-Module can be transferred using the pipeline to Install-Module:

Find-Module CustomizeWindows10 | Install-Module -Verbose -Scope AllUsers

To explore the available cmdlets in this module the following code can be executed:

Get-Command -Module CustomizeWindows10

## Exercise 2 – Creating basic functions and modules

Create a basic PowerShell function, when called this function will output hello to the console:

function Write-HelloWorld {

'Hello'

}

To execute this function, execute the following code:

Write-Hello

Now we will create a module named HelloWorld, in this module we will place a similar function to the previous function we created in the previous example. This function is called Write-HelloWorld. Execute the following code to create the module:

'function Write-HelloWorld{"Hello World!"}' | Set-Content -Path HelloWorld.psm1

Import-Module .\HelloWorld.psm1 -Force

To check the commands that the module provides we can execute the following code:

Get-Command -Name Write-HelloWorld

To use the Write-HelloWorld function, as provided by the HelloWorld module we can now simply run:

Write-HelloWorld

For more information about the module Get-Module can be used:

Get-Module -Name HelloWorld | Format-List \*

## Exercise 3 – Create an advanced function

To create a more advanced function that combines information from several sources into a single PowerShell custom object try the following example, feel free to experiment and add to the code.

function Get-SystemInformation {

[CmdletBinding()]

param(

$ComputerName = $env:COMPUTERNAME

)

$Bios = Get-CimInstance -ComputerName $ComputerName -ClassName CIM\_BIOSElement

$OSInfo = Get-CimInstance -ComputerName $ComputerName -ClassName CIM\_OperatingSystem

$Disk = Get-CimInstance -ComputerName $ComputerName -ClassName CIM\_StorageVolume -Filter "Name LIKE 'C:%'"

return [pscustomobject]@{

BiosName = $Bios.Caption

BiosManufacturer = $Bios.Manufacturer

WindowsVersion = $OSInfo.Caption

WindowsBuild = $OSInfo.BuildNumber

WindowsFolder = $OSInfo.SystemDirectory

CDriveSizeGB = $Disk.Capacity / 1GB -as [int]

CDriveFreeGB = $Disk.FreeSpace / 1GB -as [int]

}

}

Get-SystemInformation