Collections powershell scripts

Monday, 18 November 2024 21:02

```
# Step 3.1.1: Install Active Directory Module
# Open a new PowerShell process as Administrator.
Start-Process PowerShell -Verb RunAs
# Install RSAT if the ActiveDirectory module is not available.
if (-not (Get-Module -ListAvailable -Name ActiveDirectory)) {
    Install-WindowsFeature -Name RSAT-AD-PowerShell
}
# Check the availability of Get-ADUser CmdLet
Get-Command -Name Get-ADUser
# Step 3.1.2: User Accounts
# 1. Get all user accounts in the current domain
Get-ADUser -Filter * -Server 'voornaam.local'
# 2. Create a new user account "scripter"
New-ADUser -Name "scripter" -Company "Ehb" -City "Brussel"
# 3. Create a hashtable with additional attributes
$Attributen = @{ Title = 'junior'; Mail = 'juniorscripter@ehb.be' }
# 4. Create a new user account "juniorscripter" with additional attributes
New-ADUser -Name "juniorscripter" -OtherAttributes $Attributen
# 5. Retrieve user accounts with name "scripter" and select Name and
DistinguishedName
Get-ADUser -Filter "Name -like 'scripter*'" | Select-Object Name,
DistinguishedName
# 6. Retrieve user accounts and select Name, Title, and City
Get-ADUser -Filter "Name -like 'scripter*'" | Select-Object Name, Title, City
# 7. Update attributes for user accounts
Set-ADUser -Identity "scripter" -Title "senior"
Set-ADUser -Identity "juniorscripter" -City "Brussel"
# 8. Retrieve both accounts and select SamAccountName, Title, and City
Get-ADUser -Filter "Name -like 'scripter*'" | Select-Object SamAccountName,
Title, City
# 9. Remove the accounts "scripter" and "juniorscripter"
Remove-ADUser -Identity "scripter" -Confirm:$false
Remove-ADUser -Identity "juniorscripter" -Confirm:$false
# Step 3.1.3: Computers
# 1. Get all computers in the current domain
Get-ADComputer -Filter * -Server 'voornaam.local'
# 2. Get all domain controllers and select specific properties
Get-ADComputer -Filter {PrimaryGroupID -eq 516} | Select-Object Name,
SamAccountName, CanonicalName, WhenCreated
# 3. Create an array of domain controllers and loop through them
$dcs = Get-ADComputer -Filter {PrimaryGroupID -eq 516} | Select-Object Name,
Enabled
foreach ($dc in $dcs) {
    Write-Output "$($dc.Name) is $($dc.Enabled)"
}
# 4. Update the Description attribute for the client machine
Set-ADComputer -Identity "CLIENT" -Description "Windows 10 client"
Get-ADComputer -Identity "CLIENT" | Select-Object Name, Description
# Step 3.1.4: Organizational Units
# 1. Create an array of OUs
$ous = @('Brussel', 'Leuven', 'Mechelen')
# 2. Iterate through the array and create an OU for each value
foreach ($ou in $ous) {
    New-ADOrganizationalUnit -Name $ou -Path "DC=voornaam,DC=local"
}
```

```powershell
# 3. Select the OU "Leuven" and set the ManagedBy attribute to Administrator
Set-ADOrganizationalUnit -Identity "OU=Leuven,DC=voornaam,DC=local" -ManagedBy
"CN=Administrator,CN=Users,DC=voornaam,DC=local"
Get-ADOrganizationalUnit -Identity "OU=Leuven,DC=voornaam,DC=local" | Select-
Object Name, ManagedBy
# Step 3.1.5: Active Directory Groups
# 1. Create an AD group "Studenten" in OU Leuven
New-ADGroup -Name "Studenten" -GroupCategory Security -GroupScope Universal -
Path "OU=Leuven,DC=voornaam,DC=local"
# 2. Create an AD group "Docenten" in OU Brussel
New-ADGroup -Name "Docenten" -GroupCategory Distribution -GroupScope Global -
Path "OU=Brussel,DC=voornaam,DC=local"
# 3. Remove both AD groups
Remove-ADGroup -Identity "Studenten" -Confirm:$false
Remove-ADGroup -Identity "Docenten" -Confirm:$false
# Step 3.1.6: Exercise AD 1
# 1. Create a hashtable of OUs
$ous = @{ "Jette" = "Campus Jette"; "Bloemenhof" = "Campus Bloemenhof"; "Kaai"
= "Campus Kaai"; "Canal" = "Campus Canal" }
# 2. Iterate through the hashtable and create OUs with description
foreach ($key in $ous.Keys) {
    New-ADOrganizationalUnit -Name $key -Description $ous[$key] -Path
"OU=Brussel,DC=voornaam,DC=local"
}
# 3. Create 3 AD groups for Campus Kaai
foreach ($group in @('Studenten', 'Docenten', 'Switch2IT')) {
    New-ADGroup -Name $group -GroupCategory Security -GroupScope Universal -
Path "OU=Kaai,OU=Brussel,DC=voornaam,DC=local"
}
# 4. Create a user account "Snover Jef" with password and add to group
"Docenten"
$securePassword = Read-Host -Prompt "Enter Password" -AsSecureString
New-ADUser -Name "Snover Jef" -GivenName "Jef" -Surname "Snover" -Path
"OU=Kaai,OU=Brussel,DC=voornaam,DC=local" -AccountPassword $securePassword -
Enabled $true
Add-ADGroupMember -Identity "Docenten" -Members "Snover Jef"
# Step 3.1.7: Exercise AD 2
# This step would require creating a script to automatically create the AD
structure using CSV input.



$booleanValue = $true
if ($booleanValue) {
    Write-Output 'Dit is waar'
} else {
    Write-Output 'Dit is niet waar'
}
$fruit = 'appel'
if ($fruit -eq 'banaan') {
    Write-Output 'Het is een banaan'
} elseif ($fruit -eq 'appel') {
    Write-Output 'Het is een appel'
} elseif ($fruit -eq 'kiwi') {
    Write-Output 'Het is een kiwi'
} elseif ($fruit -eq 'peer') {
    Write-Output 'Het is een peer'
} else {
    Write-Output 'Onbekend fruit'
}

if($bonus -lt 50)
```

```
{
    $bonus %= 2
}
if($bonus -gt 50)
{
    $bonus %= 5
}
$waarde = Get-Process | Sort-Object -Property NPM -Descending |
Select-Object -Property Name,
@{ Label='Geheugenpaginering';Expression={ $_.NPM} } -First 1
if($waarde.Geheugenpaginering -lt 256000000) { Write-Host 'laag' }
elseif($waarde.Geheugenpaginering -lt 512000000) { Write-Host 'gemiddeld' }
elseif($waarde.Geheugenpaginering -le 1024000000) { Write-Host 'hoog' }
else { Write-Host 'te hoog' }


$leeftijd = 18
if ($leeftijd -lt 16) {
    Write-Output 'Toegang verboden'
} elseif ($leeftijd -eq 16) {
    Write-Output 'Toegang toegestaan, alcohol verboden'
} elseif ($leeftijd -eq 17 -or $leeftijd -eq 18) {
    Write-Output 'Toegang toegestaan, sterke drank verboden'
} else {
    Write-Output 'Toegang toegestaan, sterke drank toegestaan'
}

$fruit = 'appel'
switch ($fruit) {
    'banaan' { Write-Output 'Het is een banaan'; break }
    'appel' { Write-Output 'Het is een appel'; break }
    'kiwi' { Write-Output 'Het is een kiwi'; break }
    'peer' { Write-Output 'Het is een peer'; break }
    default { Write-Output 'Onbekend fruit' }
}
$dag = 'maandag'
switch ($dag) {
    'maandag' { Write-Output 'Het is maandag'; break }
    'dinsdag' { Write-Output 'Het is dinsdag'; break }
    'woensdag' { Write-Output 'Het is woensdag'; break }
    'donderdag' { Write-Output 'Het is donderdag'; break }
    'vrijdag' { Write-Output 'Het is vrijdag'; break }
    'zaterdag' { Write-Output 'Het is zaterdag'; break }
    'zondag' { Write-Output 'Het is zondag'; break }
    default { Write-Output 'Onbekende dag' }
}
$lampAan = $true
if ($lampAan) {
    Write-Output 'De lamp is aan'
} else {
    Write-Output 'De lamp is uit'
}
$maand = (Get-Date).Month
switch ($maand) {
    {$_ -eq 1} { Write-Output 'Het is januari'; break }
    {$_ -eq 2} { Write-Output 'Het is februari'; break }
    {$_ -eq 3} { Write-Output 'Het is maart'; break }
    {$_ -eq 4} { Write-Output 'Het is april'; break }
    {$_ -eq 5} { Write-Output 'Het is mei'; break }
    {$_ -eq 6} { Write-Output 'Het is juni'; break }
    {$_ -eq 7} { Write-Output 'Het is juli'; break }
    {$_ -eq 8} { Write-Output 'Het is augustus'; break }
```

```
    {$_ -eq 9} { Write-Output 'Het is september'; break }
    {$_ -eq 10} { Write-Output 'Het is oktober'; break }
    {$_ -eq 11} { Write-Output 'Het is november'; break }
    {$_ -eq 12} { Write-Output 'Het is december'; break }
    default { Write-Output 'Onbekende maand' }
}
$bankrekening = 600
switch ($bankrekening) {
    {$_ -le 250} { Write-Output 'Bankrekening lager of gelijk aan 250 euro';
break }
    {$_ -le 500} { Write-Output 'Bankrekening lager of gelijk aan 500 euro';
break }
    {$_ -gt 500 -and $_ -lt 750} { Write-Output 'Bankrekening hoger dan 500
euro of lager dan 750 euro'; break }
    {$_ -ge 750 -and $_ -lt 1000} { Write-Output 'Bankrekening hoger of gelijk
aan 750 euro of lager dan 1000 euro'; break }
    {$_ -ge 1000} { Write-Output 'Bankrekening hoger of gelijk aan 1000 euro';
break }
    default { Write-Output 'Onbekende waarde'; break }
}



# 1. Array
# Task 1.1: Create an array named weekdagen with the following String values
$weekdagen = @('Ma', 'Di', 'Wo', 'Do', 'Vr')
# Task 1.2: Try adding 'Za' and 'Zo' using .Add(), observe the error
# Uncomment the following line to see the error
# $weekdagen.Add('Za')
# Task 1.3: Add 'Za' and 'Zo' using +=
$weekdagen += 'Za', 'Zo'
# Task 1.4: Create a new array named vakantieDagen, copy only the values of
weekdagen, ensure it accepts only String values
$vakantieDagen = [string[]]$weekdagen.Clone()
# Task 1.5: Replace 'Wo' in weekdagen with 0.5
$weekdagen[2] = 0.5
# Task 1.6: Check the contents of both variables
Write-Output "\$weekdagen"
$weekdagen
Write-Output "\$vakantieDagen"
$vakantieDagen
# 2. ArrayList
# Task 2.1: Create an ArrayList named weekdagenLijst with the following String
values
$weekdagenLijst = [System.Collections.ArrayList]@('Ma', 'Di', 'Wo', 'Do',
'Vr')
# Task 2.2: Add 'Za' and 'Zo' to the ArrayList
$weekdagenLijst.Add('Za')
$weekdagenLijst.Add('Zo')
# Task 2.3: Add an array with the values 5, 10, 15, 20 to the ArrayList
$weekdagenLijst.AddRange(@(5, 10, 15, 20))
# Task 2.4: Check the contents of the ArrayList using Foreach-Object
$weekdagenLijst | ForEach-Object { Write-Output $_ }
# 3. List
# Task 3.1: Create a generic list of type DayOfWeek named generiekeWeekdagen
$generiekeWeekdagen =
[System.Collections.Generic.List[System.DayOfWeek]]::new()
# Task 3.2: Create a DateTime variable named datumTijd with the current date
and time
$datumTijd = [DateTime]::Now
# Task 3.3: Use a For loop with 3 iterations, add a value of type DayOfWeek to
generiekeWeekdagen
```

```powershell
For ($i = 0; $i -lt 3; $i++) {
    $generiekeWeekdagen.Add($datumTijd.AddDays($i).DayOfWeek)
}
# Task 3.4: Present the contents of the generic list in a GridView (Windows
only)
$generiekeWeekdagen | Out-GridView
# 4. HashTable
# Task 4.1: Create a HashTable named company
$company = @{
    'HR' = @('Els', 'Piet', 'Bjorn', 'Sandra', 'Kim')
    'LOGISTICS' = @('Kurt', 'Tania', 'Tom')
    'MAINTENANCE' = @('Isabel', 'Eric', 'Sven')
}
# Task 4.2: Iterate through the HashTable and display key and value on the
host
foreach ($key in $company.Keys) {
    Write-Output $key
    Write-Output ($company[$key] -join ' ')
}
# 5. Ordered Dictionary
# Task 5.1: Create an Ordered Dictionary with the same data as the HashTable
$companyOrdered = [System.Collections.Specialized.OrderedDictionary]::new()
$companyOrdered['HR'] = @('Els', 'Piet', 'Bjorn', 'Sandra', 'Kim')
$companyOrdered['LOGISTICS'] = @('Kurt', 'Tania', 'Tom')
$companyOrdered['MAINTENANCE'] = @('Isabel', 'Eric', 'Sven')
# Display the Ordered Dictionary
foreach ($key in $companyOrdered.Keys) {
    Write-Output $key
    Write-Output ($companyOrdered[$key] -join ' ')
}
# 6. PSCustomObject
# Task 6.1: Create a PSCustomObject named $AdministratorInfo
$AdministratorInfo = [PSCustomObject]@{
    Username       = 'Administrator'
    Groupname      = ''
    Description    = ''
    Enabled        = ''
    Membership     = @()
}
# Task 6.2: Set Description for the local user Administrator
$AdministratorInfo.Description = (Get-LocalUser -Name
$AdministratorInfo.Username).Description
# Task 6.3: Set Enabled status for the local user Administrator
$AdministratorInfo.Enabled = (Get-LocalUser -Name
$AdministratorInfo.Username).Enabled
# Task 6.4: Create a variable localGroups with the array from Get-LocalGroup
$localGroups = Get-LocalGroup
# Task 6.5: Iterate through localGroups and check if Administrator is a
member, add to Groupmembership
foreach ($group in $localGroups) {
    if (Get-LocalGroupMember -Group $group.Name -Member 'Administrator' -
ErrorAction SilentlyContinue) {
        $AdministratorInfo.Membership += $group.Name
    }
}
# Task 6.6: Display the PSCustomObject
$AdministratorInfo


# Step 2.1: Query DNS Zones
# 1. Query all DNS zones in the current domain
Get-DnsServerZone
```

```
# 2. Query only the reverse lookup zones
Get-DnsServerZone | Where-Object { $_.IsReverseLookupZone -eq $true }
# 3. Query forward lookup zones and filter properties
Get-DnsServerZone | Where-Object { $_.IsReverseLookupZone -eq $false } |
Select-Object ZoneName, IsReverseLookupZone
# Step 2.2: Query DNS Records
# 1. Query all A records where "dc" is in the HostName
Get-DnsServerResourceRecord -ZoneName "vincent.local" | Where-Object
{ $_.HostName -like '*dc*' -and $_.RecordType -eq 'A' }
# 2. Query A record for IP address "192.168.1.3"
Get-DnsServerResourceRecord -ZoneName "vincent.local" | Where-Object
{ $_.RecordData.IPv4Address -eq "192.168.1.3" }
# 3. Query all pointer (PTR) records
Get-DnsServerResourceRecord -ZoneName "vincent.local" | Where-Object
{ $_.RecordType -eq 'PTR' }
# 4. Query all Alias (CNAME) records
Get-DnsServerResourceRecord -ZoneName "vincent.local" | Where-Object
{ $_.RecordType -eq 'CNAME' }
# Step 2.3: Create DNS Records
# 1. Create a new static A record with a PTR record
Add-DnsServerResourceRecordA -ZoneName "voornaam.local" -Name "dc5" -
IPv4Address "192.168.1.8" -CreatePtr
# 2. Create an Alias (CNAME) for dc5
Add-DnsServerResourceRecordCName -ZoneName "voornaam.local" -Name
"Domeincontroller" -HostNameAlias "dc5.voornaam.local"
# 3. Verify that the PTR record exists for dc5
Get-DnsServerResourceRecord -ZoneName "voornaam.local" | Where-Object
{ $_.RecordType -eq 'PTR' -and $_.HostName -eq 'dc5' }
# 4. Query all CNAME records
Get-DnsServerResourceRecord -ZoneName "voornaam.local" | Where-Object
{ $_.RecordType -eq 'CNAME' }
# Step 2.4: Delete DNS Records
# 1. Delete the alias record "Domeincontroller"
Remove-DnsServerResourceRecord -ZoneName "voornaam.local" -Name
"Domeincontroller" -RRType "CNAME" -Force
# 2. Delete the static record "dc5" and its associated PTR record
Remove-DnsServerResourceRecord -ZoneName "voornaam.local" -Name "dc5" -RRType
"A" -Force
# Step 3.1: Access Control List (ACL) for Folder
# 1. Get the ACL of the new folder and assign it to a variable
$acl = Get-Acl -Path "C:\temp\testfolder"
# 2. Create a new access rule for a user account
$accessRule = New-Object
System.Security.AccessControl.FileSystemAccessRule("VINCENT\student1",
"ReadAndExecute, Synchronize", "ContainerInherit, ObjectInherit", "None",
"Allow")
# 3. Add the access rule to the ACL and apply the changes
$acl.AddAccessRule($accessRule)
Set-Acl -Path "C:\temp\testfolder" -AclObject $acl
# 4. Verify the ACL of the folder
Get-Acl -Path "C:\temp\testfolder" | Select-Object Owner, Access | Format-List



$aProc = Get-Process | Select-Object -First 1 -Property Id, ProcessName
Write-Host $aProc
$bProc = $aProc
Write-Host $aProc
# Filter the object to show only Id and ProcessName properties
$aProc | Select-Object Id, ProcessName
if($aProc -eq $bProc) {
    Write-Host "aProc and bProc are equal"
```

```
} else {
    Write-Host "aProc and bProc are not equal"
}
if($aProc.Id -gt $bProc.Id) {
    Write-Host "aProc.Id is bigger than bProc.Id"
}elseif ($aProc.Id -eq $bProc.Id) {
    Write-Host "aProc.Id is equal to bProc.Id"
}
 else {
    Write-Host "aProc.Id is not bigger than bProc.Id"
}
$bProc = Get-Process | Select-Object -Last 1 -Property Id, ProcessName
Write-Host $bProc
if($aProc -eq $bProc) {
    Write-Host "aProc and bProc are equal"
} else {
    Write-Host "aProc and bProc are not equal"
}
if($aProc.Id -gt $bProc.Id) {
    Write-Host "aProc.Id is bigger than bProc.Id"
}elseif ($aProc.Id -eq $bProc.Id) {
    Write-Host "aProc.Id is equal to bProc.Id"
}
 else {
    Write-Host "aProc.Id is not bigger than bProc.Id"
}
$files = Get-ChildItem -File | Select-Object Name, Length
$files
Write-Host "Number of files:"
$sortedFiles = $files | Sort-Object Name -Descending
$sortedFiles | Format-Table -AutoSize



# Step 3.1.1: Try Catch Example with Array
# Declare an array and initialize it
$weekend = @('zaterdag', 'zondag')
# Try block to add an element
try {
    $weekend.Add('maandag')
} catch {
    Write-Output "Deze collectie is een fixed type, de add methode is hier
niet toegelaten."
}
# Step 3.1.2: Try Catch Example for File Path
# Ask the user for a file path
$filePath = Read-Host -Prompt "Voer een bestandspad in"
# Try to open the file
try {
    Get-Content -Path $filePath -ErrorAction Stop
} catch {
    Write-Output "Het bestand kan niet worden geopend. Controleer het pad en
probeer het opnieuw."
}
# Step 3.1.3: Try Catch Example for Division by Zero
# Ask the user for a number
$number = Read-Host -Prompt "Voer een getal in"
# Try to divide the number by zero
try {
    $result = $number / 0
} catch {
    Write-Output "Er is een fout opgetreden: Delen door nul is niet mogelijk."
```

```
}


# 3.1.1 Eenvoudige functies
# Task 1: Write a simple function
function New-ShowText {
    param (
        [string]$tekst = 'Oproep van deze functie'
    )
    Write-Output $tekst
}
# Task 2: Test the function
# PS C:> New-ShowText
# Oproep van deze functie
# PS C:> New-ShowText 'Testbericht'
# Testbericht
# Task 3: Write a function named 'Add-Numbers'
function Add-Numbers {
    param (
        [int]$num1,
        [int]$num2
    )
    return $num1 + $num2
}
# Test the function
# $result = Add-Numbers 5 3
# Write-Host "De som is: $result"
# Task 4: Write a function named 'Get-MaxNumber'
function Get-MaxNumber {
    param (
        [int]$num1,
        [int]$num2,
        [int]$num3
    )
    return ($num1, $num2, $num3 | Measure-Object -Maximum).Maximum
}
# Test the function
# $maxNumber = Get-MaxNumber 10 25 15
# Write-Host "Het grootste nummer is: $maxNumber"
# 3.1.2 Uitgebreide functies
# Task 1: Create a function to manage new enrollments
function New-Inschrijving {
    param (
        [Parameter(Mandatory=$true)]
        [string]$Naam,
        [bool]$Ingeschreven = $false,
        [ValidateScript({ $_.Count -le 3 })]
        [hashtable]$Inschrijvingen
    )
    # Task 2: Check if someone is enrolled
    if ($Ingeschreven) {
        Write-Output "$Naam is ingeschreven"
    } else {
        Write-Output "$Naam is niet ingeschreven"
    }
    # Task 7: Check if Name is a key in the hashtable and update value if
different
    if ($Inschrijvingen.ContainsKey($Naam)) {
        if ($Inschrijvingen[$Naam] -ne $Ingeschreven) {
            $Inschrijvingen[$Naam] = $Ingeschreven
        }
    }
```

```powershell
    # Display the hashtable
    Write-Output "Inhoud van de hashtable:"
    $Inschrijvingen
}
# Task 5: Create a hashtable $inschrijvingen with 4 items
$inschrijvingen = @{
    'Vincent' = $true
    'Jack' = $false
    'Anna' = $true
    'Liam' = $false
}
# Task 6: Remove a random item from the hashtable
$inschrijvingen.Remove(($inschrijvingen.Keys | Get-Random))
# Test the function
# PS C:> New-Inschrijving -Naam Jack -Ingeschreven $true -Inschrijvingen
$inschrijvingen
# Jack is ingeschreven


# Werkcollege 08 - Herhalingen
# Oefening 1 For
# 1.Zoek de gegevens op om firewall regels op te vragen.
#   gebruik hiervoor de ingebouwde hulpfunctionaliteit van Powershell.
Get-Command -Verb Get -Noun *Firewall*
Get-Help Get-NetFirewallRule
Get-Help Get-NetFirewallRule -Examples
Get-NetFirewallRule
# 2.Filter de resultaten zodat je enkel de inbound firewall regels te zien
krijgt.
Get-NetFirewallRule -Direction Inbound
# 3.Wijs deze resultaten vervolgens toe aan een variabele met de naam
$firewallRegels
$firewallRegels = Get-NetFireWallRule -Direction Inbound
# 4.Voer nu een For herhaling uit, zorg dat bij elke iteratie de eigenschappen
DisplayName, Direction en Action zichtbaar zijn
# mogelijkheid 1
For($i = 0; $i -le $firewallRegels.Count; $i++ )
{
    $firewallRegels[$i].DisplayName
    $firewallRegels[$i].Direction
    $firewallRegels[$i].Action
}
# mogelijkheid 2
For($i = 0; $i -le $firewallRegels.Count; $i++ )
{
    $firewallRegels[$i] | Select-Object -Property DisplayName, Direction,
Action |
    Format-Table
}
# 5.Genereer een For loop die start met het geheel getal 100 en eindigt op
200.
#   Zorg ervoor dat enkel de even waarden zichtbaar zijn op het scherm.
For($i = 100; $i -le 200; $i+=2)
{
    $i
}
# Oefening 2 ForEach
# 1.Zoek het CmdLet op dat alle variabelen weergeeft van de huidige Powershell
sessie.
Get-Command -Verb Get -Noun *Variable*
Get-Variable
# 2.Gebruik de pipeline om het resultaatset van de variabelen door te geven
```

```
naar een ForEach herhaling.
#   Geef telkens de naam van de variabele weer bij elke iteratie.
Get-Variable | ForEach-Object { $_.Name }
# 3.Gebruik dezelfde iteratie en zorg voor een correcte beslissingstructuur
waarbij een controle uitgevoerd
# wordt bij de variabelen die bepalen of je in een Windows, Linux of MacOS
werkzaam bent.
# Geef een melding terug naar de host met de boolean waarde van deze
variabelen.
Get-Variable | ForEach-Object {
        if ($_.Name -eq 'IsWindows') {
            Write-Host "Windows omgeving = $IsWindows"
        }
        elseif ($_.Name -eq 'IsLinux') {
            Write-Host "Linux omgeving = $IsLinux"
        }
        elseif ($_.Name -eq 'IsMacOS') {
            Write-Host "MacOS omgeving = $IsMacOS"
        }
    }
# Oefening 3 ForEach-Object
# 1.Creeër een array met de waarden 1 tot 100, vervolgens sturen we het
resultaat naar een Foreach-Object CmdLet, geef op het scherm bij elke iteratie
de waarde van de array vermenigvuldigd met de tafel van 2.
@(1..100) | ForEach-Object { $_ * 2 }
# 2.Maak een nieuw tekstbestand aan met de naam sites.txt, gebruik hiervoor
een editor naar keuze notepad, nano.
# C:\Temp\sites.txt
# 3.Zoek het CmdLet op om de content op te vragen van een bestand.
Get-Command Get-Content
Get-Help Get-Content -Examples
# 4.Geef de gegevens door naar het Foreach-Object CmdLet en controleer of er
een verbinding tot stand kan gebracht worden met elke website.
# Zoek de correcte parameter om slechts 1 reply te ontvangen per website.
Get-Content -Path 'C:\Temp\sites.txt' | ForEach-Object { Test-Connection $_ -
Count 1 }
# 5.Zorg bij bovenstaande iteratie dat bij elke reply de eigenschappen bron,
bestemming, ip adres en de status zichtbaar zijn.
Test-Connection -TargetName 'ehb.be' -Count 1 | Get-Member -MemberType
Property
 |
    ForEach-Object { Test-Connection $_ -Count 1 |
    Select-Object -Property Source, Destination, Address, Status }
# Oefening 4 Do While
# 1.Genereer een Do While herhaling waarbij een variabele met de naam getal
# van het type Integer de beginwaarde 0 bevat, gebruik bij elke iteratie
# de increment operator waarbij getal met 1 wordt verhoogd, herhaal deze
iteratie
# tot de variabele het getal 10 bevat,
# controleer de waarde van getal met een weergave naar de host toe.
[Int32]$getal = 0
Do{
    $getal++
    Write-Host $getal
}While($getal -lt 10)
<#
2.Zoek het CmdLet op om de huidige datum en tijd te verkrijgen,
wijs dit CmdLet toe aan de variabele met de naam datumTijd,
maak een 2de variabele aan met de naam huidigeDatumTijd.
Maak nu een Do While herhaling gebruik het Cmdlet bij elke iteratie en wijs
deze toe
aan de variabele huidigeDatumTijd.
```

```
zolang de minuutwaarde van de variabele Voer de herhaling uit zolang de
minuutwaarde van datumTijd gelijk is aan de minuutwaarde van huidigeDatumTijd.
Zorg bij elke iteratie voor een weergave naar de host van de datumTijd en de
huidigeDatumTijd.
Gebruik bij elke iteratie het CmdLet Start-Sleep -Seconds 1 om de activiteit
bij elke iteratie 1 seconde te pauzeren.
#>
Get-Command -Verb Get -Noun *Date*
Get-Help Get-Date
Get-Date | Get-Member -Name *Minute* -MemberType Property
[datetime]$datumTijd = Get-Date
[datetime]$huidigeDatumTijd
Do{
$huidigeDatumTijd = Get-Date
Write-Host "Huidige DatumTijd : $huidigeDatumTijd"
Write-Host "Initiële DatumTijd : $datumTijd"
Start-Sleep -Seconds 1
}While($datumTijd.Minute -eq $huidigeDatumTijd.Minute)
# Oefening 5 Do Until
# 1.Maak gebruik van 3.1.4 oefening 2 met een Do Until herhaling, let op de
evalutie op het einde van de herhaling.
#   Maak gebruik van de toetsencombinatie CTRL + C om een uitvoering te
onderbreken.
[datetime]$datumTijd = Get-Date
[datetime]$huidigeDatumTijd
Do{
$huidigeDatumTijd = Get-Date
Write-Host "Huidige DatumTijd : $huidigeDatumTijd"
Write-Host "Initiële DatumTijd : $datumTijd"
Start-Sleep -Seconds 1
}Until($datumTijd.Minute -ne $huidigeDatumTijd.Minute)

# Step 3.1: Export
# 1. Export current folder contents to CSV
Get-ChildItem | Export-Csv -Path "$HOME\folder_contents.csv"
# 2. Export current folder contents to CSV without type information
Get-ChildItem | Export-Csv -Path "$HOME\folder_contents_no_type.csv" -
NoTypeInformation
# 3. Export current folder contents to CSV using ";" as delimiter
Get-ChildItem | Export-Csv -Path "$HOME\folder_contents_semicolon.csv" -
Delimiter ";" -NoTypeInformation
# 4. Export current folder contents to XML
Get-ChildItem | Export-Clixml -Path "$HOME\folder_contents.xml"
# 5. Export list of services on the computer to CSV
Get-Service | Export-Csv -Path "$HOME\services_list.csv" -NoTypeInformation
# 6. Export list of services on the computer to XML
Get-Service | Export-Clixml -Path "$HOME\services_list.xml"
# 7. Export the $exportObject to CSV
$exportObject = [PSCustomObject]@{
    "hashtable1" = @{
        "hashtable1.1" = @{
            "hashtable1.1.1" = @{
                "key1.1.1.1" = "value1.1.1.1"
                "key1.1.1.2" = "value1.1.1.2"
            }
        }
    }
}
$exportObject | Export-Csv -Path "$HOME\export_object.csv" -NoTypeInformation
# 8. Export the $exportObject to XML with depth 5
$exportObject | Export-Clixml -Path "$HOME\export_object.xml" -Depth 5
# Step 3.2: Import
```

```powershell
# 1. Import CSV from 3.1.5 and show only running services
$importedServices = Import-Csv -Path "$HOME\services_list.csv"
$importedServices | Where-Object { $_.Status -eq "Running" }
# 2. Import XML from 3.1.6 and show only running services
$importedServicesXml = Import-Clixml -Path "$HOME\services_list.xml"
$importedServicesXml | Where-Object { $_.Status -eq "Running" }
# 3. Import CSV from 3.1.7 and observe lost data
$importedObjectCsv = Import-Csv -Path "$HOME\export_object.csv"
# 4. Import XML from 3.1.8 and observe retained data
$importedObjectXml = Import-Clixml -Path "$HOME\export_object.xml"
# Step 3.3: ConvertTo
# 1. Convert top 10 processes by RAM usage to HTML, overwrite every minute
while ($true) {
    Get-Process | Sort-Object -Property WorkingSet -Descending | Select-
Object -First 10 | ConvertTo-Html | Out-File -FilePath "$HOME\processes.html"
    Start-Sleep -Seconds 60
}
# 2. Convert $exportObject to JSON without specifying depth, save to file
$exportObject | ConvertTo-Json | Out-File -FilePath "$HOME\export_object.json"
# 3. Convert $exportObject to JSON with depth 10, save to file
$exportObject | ConvertTo-Json -Depth 10 | Out-File -FilePath "$HOME
\export_object_depth10.json"
# Step 3.4: ConvertFrom
# Import JSON from 3.3.3 and navigate through the object
$importedJsonObject = Get-Content -Path "$HOME\export_object_depth10.json" |
ConvertFrom-Json
$importedJsonObject.hashtable1.'hashtable1.1'.'hashtable1.1.1'


# Step 1: Define the Computer class
class Computer {
    [string]$Naam
    [string]$Model
    [string]$OS
    [int]$Schijfruimte
    [int]$AantalProcessors
    [int]$AantalBestellingen
    Computer([string]$naam, [string]$model, [string]$os, [int]$schijfruimte,
[int]$aantalProcessors, [int]$aantalBestellingen) {
        $this.Naam = $naam
        $this.Model = $model
        $this.OS = $os
        $this.Schijfruimte = $schijfruimte
        $this.AantalProcessors = $aantalProcessors
        $this.AantalBestellingen = $aantalBestellingen
    }
    [void]BestellingToevoegen([int]$aantal) {
        $this.AantalBestellingen += $aantal
    }
    [void]BestellingVerwijderen([int]$aantal) {
        if ($aantal -le $this.AantalBestellingen) {
            $this.AantalBestellingen -= $aantal
        } else {
            Write-Output "Cannot remove more orders than available."
        }
    }
}
# Create instances of Computer class
$computer1 = [Computer]::new('Intel', 'I7', 'Windows', 1240, 12, 8)
$computer2 = [Computer]::new('AMD', 'Ryzen 9', 'Linux', 1000, 16, 5)
$computer3 = [Computer]::new('Apple', 'M1', 'macOS', 512, 8, 2)
# Display properties of computer1
```

```powershell
$computer1 | Format-List
# Step 2: Define the Student class
class Student {
    [string]$Naam = 'Doe'
    [string]$Voornaam = 'John'
    [datetime]$GeboorteDatum
    [string]$Afdeling
    Student() {
        $this.GeboorteDatum = Get-Date
    }
    [datetime]GetGeboorteDatum() {
        return $this.GeboorteDatum
    }
    [void]SetGeboorteDatum([datetime]$geboorteDatum) {
        $this.GeboorteDatum = $geboorteDatum
    }
    [void]SetAfdeling([string]$afdeling) {
        $this.Afdeling = $afdeling
    }
}
# Create instances of Student class
$student1 = [Student]::new()
$student2 = [Student]::new()
$student3 = [Student]::new()
$student4 = [Student]::new()
$student5 = [Student]::new()
# Set unique properties for each student
$student2.Naam = 'Smith'; $student2.Voornaam = 'Jane';
$student2.SetAfdeling('Informatica')
$student3.Naam = 'Brown'; $student3.Voornaam = 'Chris';
$student3.SetAfdeling('Wiskunde')
$student4.Naam = 'Johnson'; $student4.Voornaam = 'Emily';
$student4.SetAfdeling('Geschiedenis')
$student5.Naam = 'White'; $student5.Voornaam = 'Robert';
$student5.SetAfdeling('Fysica')
# Add students to array
$studenten = @($student1, $student2, $student3, $student4, $student5)
# Iterate over students and display their properties
foreach ($student in $studenten) {
    Write-Output "***Student***"
    Write-Output $student.Naam
    Write-Output $student.Voornaam
    Write-Output $student.Afdeling
}
# Create a new instance of Student with specific values
$student1 = [Student]::new()
$student1.Naam = 'Vlasmeer'
$student1.Voornaam = 'Dave'
# Set birth date for $student1
[datetime]$date = Get-Date -Date "14-10-1976"
$student1.SetGeboorteDatum($date)
# Display details of $student1
Write-Output "***Student***"
Write-Output $student1.Naam
Write-Output $student1.Voornaam
Write-Output $student1.GetGeboorteDatum()


# Step 3.1: Logging to Files with Out-File and Add-Content
# Write an initial message to a new log file
$logFilePath = "$HOME\logfile.txt"
"Dit is het begin van de logfile" | Out-File -FilePath $logFilePath
```

```powershell
# Loop 5 times and log each iteration to the file
for ($teller = 1; $teller -le 5; $teller++) {
    "Run: $teller" | Add-Content -Path $logFilePath
}
# Step 3.2: Logging to Event Log
# Define the event source
$eventSource = "MyScript"
# Check if the event source exists, if not, create it
if (-not [System.Diagnostics.EventLog]::SourceExists($eventSource)) {
    New-EventLog -LogName "Application" -Source $eventSource
}
# Log messages to the Event Log
Write-EventLog -LogName "Application" -Source $eventSource -EventId 1000 -
EntryType Information -Message "Dit is het begin van de Event Log."
for ($teller = 1; $teller -le 5; $teller++) {
    Write-EventLog -LogName "Application" -Source $eventSource -EventId 1001 -
EntryType Information -Message "Run: $teller"
}
# Step 3.3: Using Transcript Logging with Try Catch
# Start Transcript Logging
$transcriptPath = "$HOME\transcript_log.txt"
Start-Transcript -Path $transcriptPath
# Declare an array and initialize it
$weekend = @('zaterdag', 'zondag')
# Try block to add an element
try {
    $weekend.Add('maandag')
} catch {
    Write-Output "Deze collectie is een fixed type, de add methode is hier
niet toegelaten."
}
# Ask the user for a file path
$filePath = Read-Host -Prompt "Voer een bestandspad in"
# Try to open the file
try {
    Get-Content -Path $filePath -ErrorAction Stop
} catch {
    Write-Output "Het bestand kan niet worden geopend. Controleer het pad en
probeer het opnieuw."
}
# Ask the user for a number
$number = Read-Host -Prompt "Voer een getal in"
# Try to divide the number by zero
try {
    $result = $number / 0
} catch {
    Write-Output "Er is een fout opgetreden: Delen door nul is niet mogelijk."
}
# Stop Transcript Logging
Stop-Transcript

# According to PowerShell conventions, the module name and directory should
match.
$ModulePath = "$HOME\Documents\System Automation & Scripting\Modules
\Berichten"
New-Item -ItemType Directory -Path $ModulePath -Force
# Create a .psm1 file inside the module directory
$ModuleFile = "$ModulePath\Berichten.psm1"
New-Item -ItemType File -Path $ModuleFile -Force
# Step 2: Write the function "Get-Message" inside the module
$FunctionContent = @'
function Get-Message {
```

```
    [CmdletBinding()]
    param (
        [Parameter(Mandatory = $true)]
        [ValidateLength(3, 30)]
        [string]$Message
    )
    Write-Output $Message
}
<### Help Content ###>
.SYNOPSIS
    Displays a message passed as a parameter.
.DESCRIPTION
    The `Get-Message` function accepts a message string (between 3 and 30
characters) and displays it.
.PARAMETER Message
    The message that will be displayed. Must be between 3 and 30 characters in
length.
.EXAMPLE
    Get-Message -Message "Hello World"
    Displays "Hello World".
'@
# Save the function content to the module file
$FunctionContent | Set-Content -Path $ModuleFile
# Step 3: Verify available modules
Get-Module -ListAvailable
# Step 4: Import the module
Import-Module Berichten
# Step 5: Verify if the module is functional and check the help information
Get-Command -Module Berichten
Get-Help Get-Message


$optellen = 5+12
Write-Host $optellen
$vermen = 3*3
Write-Host $vermen
$restwaarden = 10%3
Write-Host $restwaarden
$deling = (200/50)
$vermenig = $deling * (2 + 2)
Write-Host $vermenig
$toewijzing = 12
$toewijzing += 5
Write-Host $toewijzing
$toewijzing -= 3
Write-Host $toewijzing
$toewijzing /= 2
Write-Host $toewijzing
$restwaarde = $toewijzing % 4
Write-Host $restwaarde
$toewijzing++
Write-Host $toewijzing
$toewijzing--
Write-Host $toewijzing

<#
3.  Output Streams
3.1 Output Streams 1 tot 6
Schrijf een script met daarin output voor elke stream van 1 tot 6, gebruik
makende van de daarbijhorende Cmdlet.
Voeg ook de Write-Host Cmdlet toe om de Host stream te testen.
Probeer dit script zo uit te voeren dat je de output van elke stream te zien
```

```
krijgt.
Bekijk voor elke variant welke output wel en niet worden uitgevoerd.
Tips:
•   Je zal een param blok moeten toevoegen aan het script met daarin de
[cmdletbinding] term in het begin om
    de common parameters te kunnen aanroepen. Zoek het even op op of gebruik
een vscode snippet.
#>
# Function to test the output streams
function Test-OutputStreams {
    [CmdletBinding()]
    param ()
    Write-Output "Success stream"
    Write-Error "Error stream"
    Write-Warning "Warning stream"
    Write-Verbose "Verbose stream"
    Write-Debug "Debug stream"
    Write-Information "Information stream"
    Write-Host "Host stream"
}
# test the output streams by uncommenting the desired parameters below and
changing
# the value of the actions to "Continue" or "SilentlyContinue"
$TestOutputStreamParams = @{
     Verbose           = $true
     Debug             = $true
     InformationAction = "Continue"
     WarningAction     = "Continue"
     ErrorAction       = "Continue"
}
# Test-OutputStreams @TestOutputStreamParams
<#
3.2 Progress stream
Schrijf een script dat een loop genereert voor waarden van 1 tot 10 waarbij
bij elke iteratie 1 seconde
gewacht wordt. Toon een progressiebalk op het scherm bij het uitvoeren van het
script met de tekst
“script in uitvoering”. Zorg ook voor een titel. Gebruik hiervoor de juiste
CmdLet, en gebruik Get-Help
om de parameters te vinden.
Tips:
•   Voor het script uit in een console/terminal, niet in vscode. De balk zal
in vscode vaak niet correct
    worden weergegeven.
#>
[int]$max = 10
for ($i = 1; $i -le $max; $i++) {
    $progressParams = @{
        Activity        = "Verwerken van items"
        PercentComplete = ($i / $max) * 100
    }
    Write-Progress @progressParams
    Start-Sleep -Seconds 1
}
<#
•   Op zoek naar extra uitdaging?
    o   Probeer de progressie van de loop ook mee te geven in de balk.
Bijvoorbeeld “3 van 10”.
    o   Maak een subloop in je loop van telkens 5 iteraties van 1 seconde, en
dus ook een onderliggende balk
        onder je bestaande balk. Laat beide balken hun respectievelijke
progressie tonen.
```

```powershell
#>
[int]$maxOuterLoop = 10
[int]$maxInnerLoop = 5
for ($i = 1; $i -le $maxOuterLoop; $i++) {
    $progressParamsOuterLoop = @{
        ID              = 0
        Activity        = "Verwerken van items"
        Status          = "Bezig met verwerken van item $i van $maxOuterLoop"
        PercentComplete = ($i / $maxOuterLoop) * 100
    }
    Write-Progress @progressParamsOuterLoop
    for ($j = 1; $j -le $maxInnerLoop; $j++) {
        $progressParamsInnerLoop = @{
            ID              = 1
            ParentID        = 0
            Activity        = "Verwerken van items"
            Status          = "Bezig met verwerken van item $i van
$maxInnerLoop"
            PercentComplete = ($j / $maxInnerLoop) * 100
        }
        Write-Progress @progressParamsInnerLoop
        Start-Sleep -Seconds 1
    }
}

<#
4.  Redirection
4.1 Redirection naar een file
Schrijf een script waarbij de de inhoud van een directory (via de output van
Get-ChildItem) geredirect wordt
naar een file.
#>
# 1.    Doe dit eerst via Out-File
Get-ChildItem | Out-File -FilePath "./Get-ChildItem_Output.txt"
# 2.    Vervolgens via de > operator
Get-ChildItem > "./Get-ChildItem_Output.txt"
# 3.    Tenslotte via de >> operator
Get-ChildItem >> "./Get-ChildItem_Output.txt"
# Vergelijk de resultaten
# de inhoud van 2 zal de inhoud van 1 overschrijven, terwijl 3 de inhoud zal
toevoegen aan de file.
# 4.2   Redirection naar een stream
# 1.    Gebruik de output van de oefening 3.1 in zijn verschillende varianten
om deze te redirecten naar
#       de succes stream.
Get-Content ./get-childitem_output.txt>&1
# 2.    Probeer ook eens alle streams te redirecten naar de succes stream.
*>&1
# 5.    Output onderdrukken
# 1.    Maak een nieuw bestand aan in je test directory via het CmdLet New-
Item. Gebruik Get-Help om de
#       correcte syntax te vinden. Bemerk de output die je krijgt.
New-Item -Path "./test.txt" -ItemType "file"
# 2.    Verwijder het bestand. Gebruik Get-Command en Get-Help om het juiste
commando en bijhorende
#       syntax te vinden.
Remove-Item -Path "./test.txt" -Force
# 3.    Maak terug een nieuw bestand, maar zorg er deze keer voor dat er geen
output van de actie op het
#       scherm verschijnt. Doe dit door gebruik te maken van:
# a.    Out-Null
New-Item -Path "./test.txt" -ItemType "file" | Out-Null
```

```powershell
# b.    > $null
New-Item -Path "./test.txt" -ItemType "file" > $null
# c.    $null =
$null = New-Item -Path "./test.txt" -ItemType "file"
# Observeer gelijkenissen en verschillen.
# De output van de actie wordt onderdrukt in 1,2 en 3, maar de actie wordt wel
uitgevoerd.


# 3.1 Eenvoudige parameters
# Task 1.1: Write a simple script to display text
Write-Output "Dit is mijn scripttekst."
# Task 1.2: Modify the script to display args, each on a new line
foreach ($arg in $args) {
    Write-Output $arg
}
# Task 1.3: Modify the script to use a parameter named 'tekst'
param (
    [string]$tekst = 'Dit is mijn scripttekst'
)
Write-Output $tekst
# Task 1.4: Add an integer parameter 'aantal' to repeat the text
param (
    [int]$aantal = 3
)
for ($i = 1; $i -le $aantal; $i++) {
    Write-Output "$tekst $i"
}
# 3.2 Uitgebreide parameters
# Task 2.1: Create a script with 2 parameters, one required and one optional
switch
param (
    [Parameter(Mandatory=$true, HelpMessage="De naam van de student")]
    [ValidateLength(3, 20)]
    [string]$Naam,
    [switch]$Ingeschreven
)
# Task 2.2: If-else to check if someone is enrolled
if ($Ingeschreven) {
    Write-Output "$Naam is ingeschreven"
} else {
    Write-Output "$Naam is niet ingeschreven"
}
# 3.3 Validatie
# Task 3.2: Add an array parameter 'vakken' with validation
param (
    [Parameter(Mandatory=$true, HelpMessage="De naam van de student")]
    [ValidateLength(3, 20)]
    [string]$Naam,
    [switch]$Ingeschreven,
    [ValidateCount(0, 3)]
    [ValidateSet('Wiskunde', 'Nederlands', 'Engels', 'Frans', 'Duits')]
    [string[]]$vakken
)
# Display enrollment status
if ($Ingeschreven) {
    Write-Output "$Naam is ingeschreven"
} else {
    Write-Output "$Naam is niet ingeschreven"
}
# Display the subjects
foreach ($vak in $vakken) {
```

```powershell
    Write-Output $vak
}
# Task 3.4: Add a parameter 'Opleiding' with validation
param (
    [Parameter(Mandatory=$true, HelpMessage="De naam van de student")]
    [ValidateLength(3, 20)]
    [string]$Naam,
    [switch]$Ingeschreven,
    [ValidateCount(0, 3)]
    [ValidateSet('Wiskunde', 'Nederlands', 'Engels', 'Frans', 'Duits')]
    [string[]]$vakken,
    [ValidateScript({ $_.StartsWith('Graduaat') -or
$_.StartsWith('Bachelor') })]
    [string]$Opleiding
)
# Display enrollment status
if ($Ingeschreven) {
    Write-Output "$Naam is ingeschreven"
} else {
    Write-Output "$Naam is niet ingeschreven"
}
# Display the subjects
foreach ($vak in $vakken) {
    Write-Output $vak
}
# Display the course
if ($Opleiding) {
    Write-Output $Opleiding
}

# Step 3.1.1: Check connectivity to dc1 and CentOS machines
# Test connection to dc1
Test-Connection 192.168.1.3 -Count 1
# Test connection to CentOS
Test-Connection 192.168.1.2 -Count 1
# Step 3.1.2: Test WSMan protocol
# Test WSMan on client
Test-WSMan
# Test WSMan on dc1
Test-WSMan -ComputerName 192.168.1.3
# Step 3.1.3: Configure WinRM if needed
# Run WinRM quick configuration if WSMan test failed
winrm quickconfig
# Step 3.1.4: Verify WinRM service
# Check the status of WinRM service on client and dc1
Get-Service -Name winRM
# Step 3.1.5: Create a PowerShell session using WSMan
# Create a PSCredential object
[PSCredential]$credential = Get-Credential
# Create PowerShell session to dc1 using WSMan
New-PSSession -Name 'DC1' -ComputerName 192.168.1.3 -Credential $credential
# Verify active sessions
Get-PSSession
# Enter the session
Enter-PSSession -Name 'DC1'
# Verify the current computer name
$env:COMPUTERNAME
# Exit the session
Exit-PSSession
# Remove the session
Get-PSSession -Name 'DC1' | Remove-PSSession
# Remove the PSCredential object
```

```
Remove-Variable -Name credential
# Step 3.2.1: Configure SSH on dc1
# Install OpenSSH Client and Server on dc1
Add-WindowsCapability -Online -Name OpenSSH.Client~~~~0.0.1.0
Add-WindowsCapability -Online -Name OpenSSH.Server~~~~0.0.1.0
# Start and configure sshd service to start automatically
Start-Service -Name sshd
Set-Service -Name sshd -StartupType Automatic
# Verify SSH firewall rule
Get-NetFirewallRule -Name *ssh* | Select-Object -Property DisplayName,
Direction, Action
# Edit SSHD configuration file to allow password authentication
notepad "$env:ProgramData\ssh\sshd_config"
# Restart SSHD service after modifying configuration
Restart-Service sshd
# Step 3.2.2: Create PowerShell session using SSH
# Create PowerShell session to dc1 using SSH
New-PSSession -Name 'DC1' -HostName 192.168.1.3 -UserName 'gebruikersnaam'
# Enter the session
Enter-PSSession -Name 'DC1'
# Verify active sessions and remove the session
Exit-PSSession
Get-PSSession -Name 'DC1' | Remove-PSSession
# Step 3.3.1: Remoting Exercises
# Create PowerShell session to dc1 using WSMan and get disk information
[PSCredential]$credential = Get-Credential
$sess = New-PSSession -Name 'DC1' -ComputerName 192.168.1.3 -Credential
$credential
Invoke-Command -Session $sess -ScriptBlock { Get-Volume }
# Exit and remove the session
Remove-PSSession -Session $sess
Remove-Variable -Name credential
# Step 3.3.2: Use SSH for remoting
# Create PowerShell session to dc1 using SSH and get the process with the
highest CPU
$sess = New-PSSession -Name 'DC1' -HostName 192.168.1.3 -UserName
'gebruikersnaam'
Invoke-Command -Session $sess -ScriptBlock {
    Get-Process | Sort-Object CPU -Descending | Select-Object -First 1
}
# Exit and remove the session
Remove-PSSession -Session $sess
# Step 3.3.3: Invoke-Command for OS Information
# Get processor information from dc1 using CIM_Processor class
Invoke-Command -ComputerName 192.168.1.3 -ScriptBlock {
    Get-CimInstance -ClassName CIM_Processor | Select-Object Name,
MaxClockSpeed
}
# Step 3.3.4: Invoke-Command with Parameters
# Define parameters
[string]$naam = 's*'
[bool]$inbound = $true
# Use Invoke-Command to get firewall rules based on parameters
$firewallRegels = Invoke-Command -ComputerName 192.168.1.3 -ScriptBlock {
    param ($naam, $inbound)
    if ($inbound) {
        Get-NetFirewallRule -Name $naam | Where-Object { $_.Direction -eq
'Inbound' }
    } else {
        Get-NetFirewallRule -Name $naam | Where-Object { $_.Direction -eq
'Outbound' }
    }
```

```powershell
} -ArgumentList $naam, $inbound
# Step 3.3.5: Create SSH Session to CentOS Machine
# Edit SSHD configuration file on CentOS and restart SSHD
# Subsystem powershell /usr/bin/pwsh -sshs -NoLogo -NoProfile
# Create a new SSH session to CentOS machine
$sess = New-PSSession -Name 'CentOS' -HostName 192.168.1.2 -UserName 'Student'
# Enter the session and get all processes
Enter-PSSession -Session $sess
Get-Process
# Exit and remove the session
Exit-PSSession
Remove-PSSession -Session $sess

[string]$cursus = 'Powershell'
$favorieteCursus = "$cursus is mijn favoriete cursus"
$favorieteCursus = $favorieteCursus -replace 'Powershell', 'Bash'
$favorieteCursus = $favorieteCursus.ToUpper()
$favorieteCursus = "$cursus is mijn favoriete cursus"
$favorieteCursus = "$($cursus) is mijn favoriete cursus"


[int]$bonus = 45.50
Write-Output $bonus
Write-Output "Maximum Integer: [int]::MaxValue"
Write-Output "Minimum Integer: [int]::MinValue"
$bonus = Read-Host 'Geef een numerieke waarde: '
$bonus = [int]$bonus
Write-Output "De waarde van bonus is: $bonus"
```