Windows PowerShell Skriptsprache

1 Theorie

PowerShell bietet sämtliche Strukturen (Variablen, Kontrollstrukturen, Schleifen) zur Programmierung komplexer Probleme, wie eine andere höhere Programmiersprache.

1.1 Kommentare

```
# Das ist ein einzeiliger Kommentar

<#
Das ist
Ein mehrzeiliger
Kommentar
#>
```

1.2 Variablen

1.2.1 Grundlagen

Variablen beginnen immer mit dem $-Zeichen!

```
$a = 5
Write-Host "Das Ergebnis lautet: $a"

$prozesse = Get-Process
Write-Host "Anzahl Prozesse: $($prozesse.count)"
```

1.2.2 Variablen-Typen

Bei Bedarf können Variablen auch typisiert werden.

```
[int]$a = 5
$betrag = [double]35.70
```

Die wichtigsten Typen sind:

[bool[]]	Boolean **$True** oder **$False** bzw. ein Array von booleschen Werten. Informationen zu Array folgen später.
[float]	32 Bit lange Fliesskommazahl
[double]	64 Bit lange Fliesskommazahl
[decimal]	86 Bit lange Fliesskommazahl
[string]	Zeichenkette
[array]	Array mit beliebigen Objekten
[hashtable]	Feld beliebig benannter Wertepaare
[xml]	XML-Struktur
[regex]	Regular Expression
[scriptblock]	Windows PowerShell Skriptblock
[psobject]	Windows PowerShell Objekt (.Net-Objekt)
[wmi]	WMI-Object
[adsi]	Active Directroy Object

1.2.3 Zugriff auf nicht definierte Variablen verhindern

```
Set-PSDebug -strict
```

1.2.4 Vordefinierte Variablen

Windows PowerShell kennt viele vordefinierte Variablen. Sie Drives **env:** und **variable:**

1.3 Rechnen mit Zahlen (Operatoren)

PowerShell kennt die arithmetischen Operatoren **+, -, *, /** und **%** (Modulo).

Weitere Operatoren sind:

++	Um eins erhöhen
--	Um eins herunterzählen
..	Bereichsoperator

Logische Operatoren sind:

-not	Nicht
-and	Logisches Und
-or	Logisches Oder
-xor	Exklusiv Oder

Weiterführende mathematische Berechnungen stellt die statische Klasse **[math]** zur Verfügung:

```
[math].GetMethods() | Select-Object -Property Name -Unique
```

Beispiel:

```
[math]::sqrt( 64 )
```

1.4 Manipulation von Zeichenketten

Strings sind Objekte. Die Manipulation von Zeichenketten erfolgt über die entsprechenden Methoden:

```
"Test" | Get-Member
```

```
"Test".ToUpper()
```

Länge einer Zeichenkette eruieren:

```
"Test".Length
```

Die Verkettung von Strings erfolgt mit dem +-Zeichen:

```
"Hans"+" "+"Muster"
```

1.5 Arrays und Hashtables

1.5.1 Arrays

Definition eines Arrays, Wertzuweisung und weitere Funktionen:

```
$Farben = @()                    # leerer Array definieren
$Farben = "rot", "blau", "grün"  # Werte einfügen
$Farben += "gelb"                # Wert hinzufügen
$Farben.Length                   # Länge des Arrays ausgeben
$Farben[0]                       # Wert an Position (Index) 0 ausgeben
$Farben[1..3]                    # Werte der Positionen 1 bis 3 ausgeben
```

1.5.2 Hashtables

Definition einer Hashtable, Wertzuweisung und weitere Funktionen:

```
# Leere Hashtable
$pwd = @{}

# Wertepaare (Key, Value) einfügen
$pwd= @{Muster="12345678"; Adam="master01"; Zimmermann="Bolero1701"}

# Wertepaar hinzufügen
$pwd.Meier = "welcome"+'$'+"2018"

#Wert eines bestimmten Keys (Schlüssel) ausgeben
$pwd["Adam"]

#Anzahl Elementer der Hashtable
$pwd.Count

#Alle Werte in der Hashtable ausgeben
$pwd.Values

#Alle Schlüssel der Hashtable ausgeben
$pwd.Keys

#Element löschen
$pwd.Remove("Adam")
```

1.6 Kontrollstrukturen und Schleifen

1.6.1 Die For-Schleife

```
for ($i=1; $i -le 10; $i++) {
     Write-Host "Durchlauf: $i"
}
```

1.6.2 Do-, While- und Until-Schleifen

```
# while-Schleife
$i = 1
while ($i -le 5) {
      Write-Host "Durchlauf: $i"
      $i++
}

# do-Schleife
$i = 1
do {
      Write-Host "Durchlauf: $i"
      $i++
} while ($i -le 5)

# do-until-Schleife
$i = 1
do {
      Write-Host "Durchlauf: $i"
      $i++
} until ($i -gt 5)
```

1.6.3 Die For-Each-Schleife

Wird zum Durchlaufen von Arrays mit Objekten verwendet.

```
$processlist = Get-Service
foreach($process in $processlist) {
      Write-Host $process.name " " $process.status
}
```

1.6.4 If..Else-Bedingung

Die Else-Anweisung ist optional.

```
[int]$a = Read-Host "Geben Sie eine Zahl ein:"
if ($a -ge 0) {
      Write-Host "$a ist positiv"
} else {
      Write-Host "$a ist negativ" -Foregroundcolor Red
}
```

1.6.5 Switch-Bedingung

```
[int]$a = Read-Host "Geben Sie eine Zahl zwischen von 1 bis 10 ein:"
switch ($a)
{
      1 {Write-Host "Eins"}
      2 {Write-Host "Zwei"}
      3 {Write-Host "Drei"}
      4 {Write-Host "Vier"}
      5 {Write-Host "Fünf"}
      6 {Write-Host "Sechs"}
      7 {Write-Host "Sieben"}
      8 {Write-Host "Acht"}
      9 {Write-Host "Neun"}
      10 {Write-Host "Zehn"}
      default {Write-Host "Ungültige Zahl"}
}
```

2 Übungen

Aufgabe 1

Sie erhalten eine CSV-Datei mit Lernenden (**`lernende.csv`**). Erstellen Sie ein PowerShell-Skript, welches die Daten importiert und pro Lernendem ein Verzeichnis der Form **`vorname.nachname`** erstellt. Stellen Sie sicher, dass spezielle Zeichen, wie Umlaute und andere, ersetzt werden: **`ä`** mit **`ae`**, **`ö`** mit **`oe`**, **`ü`** mit **`ue`**, **`é`** mit **`e`**. Zudem sollen bei zweiteiligen Vornamen der zweite Teil weggelassen werden.

Aufgabe 2

Sie erhalten eine CSV-Datei mit Schweizer Ortschaften. Erstellen Sie eine **Hashtable** mit **Key**: Ortschaft, **Value**: Postleitzahl. Falls eine Ortschaft bereits existiert, wird der Key mit " **(Kanton)**" erweitert. Der Benutzer kann jetzt eine Ortschaft eingeben und die entsprechende Postleitzahl wird angezeigt. Das Programm wird mit "exit" beendet.

Aufgabe 3

Erstellen Sie ein Skript, welches alle EXE-Dateien im Verzeichnis C:\Windows auflistet und deren Name, Produktversion und Dateigrösse (in Bytes) ausgibt.

Aufgabe 4

Erstellen Sie ein Skript, welches alle Prozesse auflistet. Prozesse, die weniger als 300 Sekunden der CPU-Zeit benötigen, sollen in grüner Schrift erscheinen (Cmdlet **`Write-Host`** mit der Option **`– ForegroundColor`**), Prozesse die zwischen 300 und 1000 CPU Sekunden benötigen sollen weisser Schrift erscheinen und darüber liegende Prozesse in Rot. Am Ende soll eine Zusammenfassung der Anzahl grüner, weisser und roter Prozesse ausgegeben werden.

Aufgabe 5

Gegeben sei eine Verzeichnisstruktur mit mehreren Textdateien, die einen bestimmten Begriff ein- oder mehrmals beinhalten. Entwickeln Sie ein Skript, welches alle Begriffe in allen Textdateien durch einen anderen Ersatzbegriff ersetzt. Begriff und den Ersatzbegriff, sowie den Pfad zum Basisverzeichnis können in der Kommandozeile angeben werden.

Aufgabe 6

Sie erhalten die Datei **`schueler.csv`** mit Schülerdaten. Die Testplattform istest.ch erlaubt einen automatischen Import von Schülerdaten. Die Import-Dateien müssen als CSV-Dateien pro Klasse vorliegen. Der Dateiname entspricht dem Namen der Klasse (z.B. **`INF17A.csv`**). Die CSV Datei weist pro Schüler die Daten Name, Vorname, Benutzername, Kennwort auf (z.B. "**`muster;peter;peter.muster;peter,muster`**"). Als Kennwort wird initial der Benutzername gesetzt. Erstellen Sie ein PowerShell-Skript, welches aus der gegebenen Datei **`schueler.csv`**, wie beschrieben, eine CSV-Datei pro Klasse für den Import nach istest.ch erstellt.