Skriptblöcke, Funktionen, Module

Skriptblöcke sind Einheiten von Kommandos, die als Block ausgeführt werden. Sie stellen einen gemeinsamen Gültigkeitsbereich für Variablen bereit. Alle Variablen, die innerhalb eines Skriptblocks definiert werden, leben nur so lange, wie der Skriptblock ausgeführt wird. Variablen sind auch in Unter-Skriptblöcken sichtbar. Sie können also von allen Skripten und Funktionen aus die Standardvariablen der Konsole sowie aller Skriptblöcke oberhalb Ihrer Ausführungsebene sehen. Eine Variable kann aber durch eine gleichlautende Variable eines Skriptblocks überdeckt werden. Eine globale Variable *$Hostname* ist also nur so lange sichtbar, wie innerhalb eines Skriptblocks keine neue Variable *$Hostname* erstellt wird. Um direkt auf eine globale Variable (=Variable des Powershell-Prozesses) zugreifen zu können, stellen Sie dem Variablennamen das Schlüsselwort *global* voran, für Skriptvariablen verwenden Sie Script: `$global:Hostname`.

Funktionen können als **Skriptmodule** automatisch beim Starten geladen werden. Legen Sie dafür Ihre Skriptdatei in einen Modulordner (*$env:PSModulePath*) mit der Endung .psm1 ab und benennen Sie den Ordner wie Ihre Skriptdatei, jedoch ohne Endung. Metainformationen können Sie in einem Modulmanifest (psd1) hinterlegen.

```
Function Verb-Ding
{
  [CmdletBinding()]
  Param(
    [Validierungsattribut()]
    [ParameterAttribut()]
    [Datentyp]$Param1,
    [Datentyp]$param2
  )
  <Code>
}
```

Validierungsattribute können Parameter und Variablen direkt beim Erstellen prüfen

Attribut	
AllowNull()	Der Parameter kann ohne ein Argument aufgerufen werden
AllowEmptyString()	Das Argument darf ein leerer String ("") sein
AllowEmptyCollection()	Das Argument darf ein leeres Array sein
ValidateNotNull()	Der Parameter darf nicht ohne ein Argument aufgerufen werden
ValidateNotNullOrEmpty()	Der Parameter darf weder NULL (=Nichts) noch ein leerer String sein
ValidateCount(x,y)	Die Anzahl der Elemente eines Arrays muss sich im Wertebereich von x bis y befinden (x,y=Ganzzahlen)
ValidateLength(x,y)	Die Anzahl der Buchstaben eines Strings muss sich im Wertebereich von x bis y befinden (x,y=Ganzzahlen)
ValidatePattern(Regex)	Das Argument muss dem regulären Ausdruck Regex entsprechen
ValidateRange(x,y)	Das Argument (Zahl) muss sich im Wertebereich von x bis y befinden
ValidateScript({ $_ })	Das Validierungsskript darf nicht $false oder leer zurückgeben
ValidateSet('Val1','Val2')	Das Argument muss einem der Werte des Validierungssatzes entsprechen

Ausgabekanäle

Ausgaben von Cmdlets werden in die Standardausgabe geschrieben, von wo aus sie in der Pipeline an andere Cmdlets übernommen oder als Text in die Kommandozeile ausgegeben werden. Damit Fehlerobjekte und sonstige Meldungen nicht die Verarbeitung stören, werden sie in eigenen Kanälen verarbeitet. Mit den Write-Cmdlets kann man gezielt in bestimmte Kanäle schreiben.

Cmdlet	Kanal	Funktion
Write-Host	-/6	Schreibt direkt in die Konsole. Ab PSH5 wird Kanal 6 verwendet
Write-Output	1	Schreibt in die Standardausgabe
Write-Error	2	Schreibt in den Fehlerdatenstrom
Write-Warning	3	Schreibt in den Warnungs-Datenstrom
Write-Verbose	4	Schreibt in den ausführlichen Ausgabestrom
Write-Debug	5	Schreibt in den Debug-Datenstrom
Write-Information	6	Schreibt in den Information-Datenstrom. Erst ab PSH 5 verfügbar.

Man kann die Ausgaben eines Kanals mit dem Umleitungs-Operator aus dem Kanal ausleiten.

```
Get-ChildItem -Path C:\Windows -Recurse 2> 'c:\Error.log' # Kanal 2 in Datei
```

Mit & kann man einen Kanal in einen anderen umleiten, z.B. Fehler in die Standardausgabe.

```
$AllOutput = Get-Childitem -Path C:\Windows -Recurse 2>&1
```

Parameter-Argumente

Argument	
Mandatory	Pflichtparameter - Bei Auslassung wird er erfragt
HelpMessage='Hilfe'	Hilfemeldung für Pflichtparameter. Kann durch !? angezeigt werden
Position = 0	das Argument kann ohne den Parameternamen übergeben werden. Das erste Parameterlose Argument wird in den Parameter mit Position 0 übergeben usw.
ValueFromPipeLine	Der Parameter kann ein Objekt direkt aus der Pipeline übernehmen
ValueFromPipelineByPropertyName	Der Parameter kann aus der Pipeline übernommen werden. Die Zuordnung erfolgt über den Eigenschaftsnamen des Pipelineobjekts
ValueFromRemainingArguments	Alle nicht zugeordneten Parameter werden diesem Parameter zugewiesen. Der Parameter sollte ein String-Array sein.
ParameterSetName='Satzname'	Der Parameter gehört dem Parametersatz 'Satzname' an. Sobald ein Satz eindeutig erkannt ist, werden nur noch Satz-Parameter angezeigt.

Vergleichs-Operatoren

Operator	Bedeutung
-eq, -ne	= (gleich), <> (ungleich)
-lt, -le	< (kleiner als), <= (kleiner oder gleich)
-gt, -ge	> (größer als), => (größer oder gleich)
-like [1]	Wie –eq, aber es können Platzhalter [2] verwendet werden
-match [1]	Vergleich mit regulären Ausdrücke (s. umseitig)
-contains [1]	$array –contains $singleValue: Der Wert muss Teil des Arrays sein
-in [1]	$singleValue –in $array: Wie contains mit umgekehrten Vorzeichen

[1] werden mit vorangestelltem not negiert: -notlike, -notmatch, -notcontains, -notin

[2] [ABC] für genau ein Zeichen aus der Liste in [], ? für **ein** beliebiges Zeichen und * für beliebige Zeichen

Seminare: https://www.netz-weise-it.training/seminare.html
Der Netz-Weise Blog: https://www.netz-weise-it.training/weisheiten/tipps.html
ebooks und Dokus: https://www.netz-weise-it.training/weisheiten/doku.html

Netz-Weise
Lernen von den Besten.

Reguläre Ausdrücke

Reguläre Ausdrücke sind komplexe Suchmuster, die es erlauben, in einem String Teilstrings zu finden und zu extrahieren. Reguläre Ausdrücke sind normalerweise Case-sensitiv, bei Powershell aber nur in Verbindung mit dem Operator -cmatch. Achten Sie auch darauf, dass die gleichen Zeichen innerhalb und außerhalb eckiger Klammern unterschiedliche Bedeutung haben können.
[^ABC] = Nicht ABC
^[ABC] = ABC am Zeilenanfang

Platzhalter	Beschreibt
[ABC]	[] = 1 Zeichen, das A, B oder C sein darf
[^ABC]	^ = Nicht -> Alle Zeichen außer ABC
[A-Z]	Ein Wertbereich -> hier alle Zeichen von A bis Z
.	Ein beliebiges Zeichen außer einem Zeilenumbruch
****	Escape-Sequenz: das folgende Zeichen hat eine besondere Bedeutung
\w	Wort (Word): Alle Buchstaben + Zahlen + Unterstrich (_), entspricht [A-Za-z0-9_]
\W	Alle Zeichen, die in \w nicht enthalten sind
\d	Zahl (digit), entspricht [0-9]
\D	Alle Zeichen, die in \d nicht enthalten sind: [^0-9]
\s	Alle Leerzeichen: Space, Tabulator, Zeilenumbruch
\S	Alle Zeichen außer Leerzeichen. Entspricht \w plus Sonderzeichen
\t	Tabulator
\n	Zeilenumbruch – Steuerzeichen, bedeutet: in die nächste Zeile springen
\r	Carriage Return – Steuerzeichen, bedeutet: Zum Anfange der Zeile zurück

Um einen String auf Suchmuster zu untersuchen, verwenden Sie den *-match*-Operator:

```
'Bezeichner {9dea862c-5cdd-4e70-acc1-f32b344d4795}' -match '(^\w+)\s+(.+)'
```

-match liefert *$true* zurück, wenn das Suchmuster gefunden wurde. Das gefunden Muster kann über die Hashtable *$Matches* abgerufen werden. Jede gefundene Suchgruppen ist ein nummeriertes Key-Value-Paar. *–match* liefert aber nur den ersten Treffer zurück. Will man alle Treffer innerhalb eines Strings erhalten, muss man auf die Regex-Methode *Matches()* zurückgreifen.

```
[regex]$regex = '(\d)+GB'
$Pattern = $regex.Matches('100GB sind keine 10GB')
```

In der Eigenschaft Value des zurückgelieferten Match-Objektes finden Sie die gefunden Treffer:

```
$Pattern | select-object -property Value
```

Achten Sie darauf, dass auch der *-replace-Operator* mit regulären Ausdrücken arbeitet. Um einen Backslash zu ersetzen müssen Sie also folgendermaßen vorgehen:

```
"C:\windows\" -replace '\\','/'
```

Quantifizierer geben die Anzahl der gewünschten Platzhalter an, | verknüpft Muster per or

Quant.	Beispiel	Bedeutung
+	[ABC]+	Mindestens 1 oder mehr Zeichen
*****	[^ABC]*	0 oder mehr Zeichen
{1,3}	[A-Z]{1,3}	Bereich, hier 1 – 3 Zeichen
{4,}	\d{4,}	Bereich, hier 4 oder mehr Zeichen
?	\.psm?1	Das vorstehende Zeichen ist optional (.ps1 oder .psm1)
<Q>?	a+?	Ein Quantifiz. ist gierig (greedy), erkennt also alle vorstehenden Zeichen, die dem Suchmuster entsprechen (z.B. 'aaaaa' mit dem Ausdruck a+), als 1 Treffer. Das ? macht den Qantifiz. faul (lazy) – die minimale Anzahl Zeichen ('aaaaa') entspricht 1 Treffer.
|	\.exe|\.cmd	Oder (.exe oder .cmd)

Anker können einen Ausdruck an Wort- und Zeilengrenzen suchen

Anker	Beispiel	
^	^\d{4}	Zeilenanfang. 4 Zahlen am Anfang der Zeile
$	(\w*)$	Zeilenende. Das letzte Wort vor dem Zeilenende
\b	\bAn	Wort-Grenze, das Beispiel finde in AnnaAnna nur das erste An, das es am Beginn des Wortes steht
\B	\BAn	Nicht Wort-Grenze, findet in AnnaAnna nur das zweite An

Suchgruppen kann man verwenden, um wiederkehrende Zeichenfolge mit Optionen wie Quant. zu versehen, oder um die gefundenen Muster zu extrahieren. Verwendet man die Operatoren -match, -cmatch oder -notmatch, enhält die automatische Variable $matches alle gefunden Suchmuster.

Gruppe	Beispiel	Bedeutung
(ABC)	(\w+\).exe\s	Suchgruppe: ein "Untermuster" in einem Suchtext. Extrahiert den Dateinamen. Gruppen können in Powershell aus der Variable $matches ausgelesen werden.
(?:ABC)	(?:\d{1,3}){4}	Erstellt ein Untermuster, wird aber nicht extrahiert. Wenn man Gruppen quantifizieren möchte, ohne Sie zu extrahieren.
\1		Referenzier auf das Ergebnis Suchgruppe. \1 entspricht dem Ergebnis der ersten Suchgruppe, \2 der zweiten usw.

Lookaround sind vergleichbar mit Ankern, denn sie beschreiben Positionen im Text. Ein Lookahead sucht ein Muster hinter dem eigentlichen Suchbegriff, ein Lookbehind davor.

Lookaround	Beispiel	Bedeutung
(?=Muster)	\d+(?=GB)	Lookahead: \d+ ist nur ein Treffer, wenn im direkten Anschluß GB folgt. GB wird nicht in $matches angegeben.
(?!Muster)	\d+(?!KB)	Negativer Lookahead: \d+ ist ein Treffer, wenn KB nicht direkt folgt.
(?<=Muster)	(?=<f\d+)GB	Lookbehind: GB ist ein Treffer, wenn keine Zahl voran steht.
(?<!Muster)	(?<!\d) GB	