LUZERN

KANTON LUZERN

07_PS_Kontrollstrukturen_I

Modul 122

Abläufe mit einer Scriptsprache automatisieren

Berufsbildungszentrum
Wirtschaft, Informatik und Technik
bbzw.lu.ch

Mitteilungen

- Heute keine Mitteilungen

Handlungsnotwendige Kenntnisse

HAN OK	Kenntnisse
1.3	Kennt grundlegende Kontrollstrukturen und deren Einsatz bei der Ablaufautomatisierung.
1.6	Kennt das Vorgehen zur Realisierung von Scripts in der Systemadministration.
2.1	Kennt grundlegende Funktionalitäten der eingesetzten Scriptsprache.
4.1	Kennt Testverfahren für Scripts.
5.1	Kennt Elemente einer Dokumentation für die involvierten Rollen (z.B. System, Administrator, Entwickler)

Themen

- Kontrollstrukturen
  - Sequenz
  - Alternation
  - Iteration

Ausgangslage

Ein Script soll eine Problemstellung lösen. Z.B. soll ein **Script überprüfen, ob ein Server erreichbar ist**.

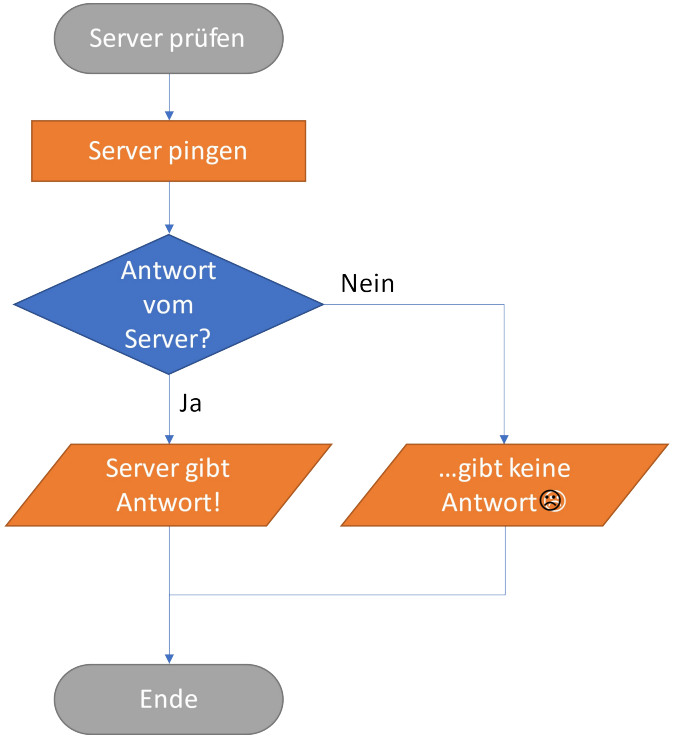
Das Script muss also einen Ping absetzen und dann prüfen, ob der Server eine Antwort gibt. Je nach Antwort fällt die Entscheidung auf **Ja oder Nein**.

Für diese Umsetzung benötigen Sie Kontrollstrukturen.

Ausgangslage

Zusammenfassung

- «Wenn der Server auf den Ping antwortet, dann ist der Server erreichbar, sonst ist er nicht erreichbar.»



Ausgangslage

Wir unterteilen die Kontrollstrukturen in:

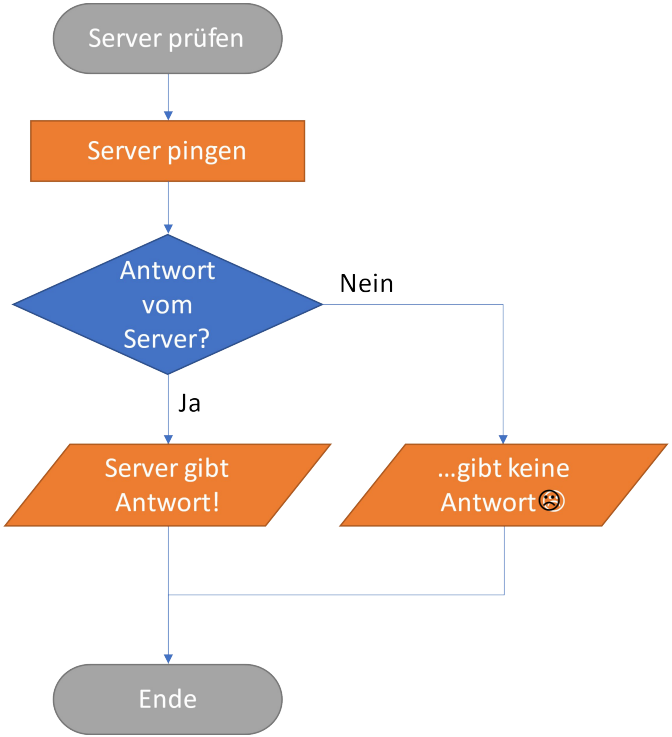
- Sequenz
- Alternation
- Iteration

Ausgangslage

Kontrollstrukturen:
Eine **normale Anweisung** ist eine **Sequenz**, z.B. **den Server pingen**.

Die **Alternation** ist die **Entscheidung**, ob der Server Antwort gibt – er hat also eine Alternative.

Die orangenen Parallelogramme sind Sequenzen und die blaue Raute ist eine Alternation.



Alternation

> Pseudocode

```
if (<test1>)
    {<statement list 1>}

elseif (<test2>)
    {<statement list 2>}

else
    {<statement list 3>}
```

Alternation PS

```
$val=5

if ($val -gt 5) {
    echo "Greater than 5"
}
elseif ($val -eq 5){
    echo "Equal to 5"
}
else {
    echo "Less than 5"
}

10
```

Alternation Bash

```
val=5

if (( $val > 5 )); then
    echo "Greater than 5"
elif (( $val == 5 )); then
    echo "Equal to 5"
else
    echo "Less than 5"
fi
```

Vergleichsoperatoren PS

Operator	Bedeutung	Beispiele
-eq	Ist gleich (*equal to*)	2 -eq 3 (Ergebnis: FALSE) "Harry" -eq "harry" (Ergebnis: TRUE)
-ne	Ist ungleich (*not equal to*)	2 -ne 3 (Ergebnis: TRUE) "Harry" -ne "harry" (Ergebnis: FALSE)
-gt	Größer als (*greater than*)	2 -gt 3 (Ergebnis: FALSE) "Harry" -gt "harry" (Ergebnis: FALSE)
-ge	Größer oder gleich (*greater than or equal to*)	2 -ge 3 (Ergebnis: FALSE) "Harry" -ge "harry" (Ergebnis: TRUE)
-lt	Kleiner als (*less than*)	2 -lt 3 (Ergebnis: TRUE) "Harry" -lt "harry" (Ergebnis: FALSE)
-le	Kleiner oder gleich (*less than or equal to*)	2 -le 3 (Ergebnis: TRUE) "Harry" -le "harry" (Ergebnis: TRUE)
-Contains	Prüft eine gegebene Werteliste, ob ein Wert enthalten ist	"abc ", "def " -Contains "abc" (Ergebnis: TRUE) "abc ", "def " -Contains "bc" (Ergebnis: FALSE)
-NotContains	Umkehrung des Operators -Contains	"abc ", "def " -NotContains "abc" (Ergebnis: FALSE) "abc ", "def " -NotContains "bc" (Ergebnis: TRUE)
-In	Wie -Contains, nur in umgekehrter Reihenfolge	"abc " -In "abc", "def" (Ergebnis: TRUE) "bc " -In "abc","def " (Ergebnis: FALSE)
-NotIn	Umkehrung des Operators -In	"abc " -NotIn "abc", "def" (Ergebnis: FALSE) "bc " -NotIn "abc","def " (Ergebnis: TRUE)
-Is	Prüfung auf Typgleichheit	[string]$string = "Herdt-Verlag" $string -Is [string] (Ergebnis: TRUE)
-IsNot	Umkehrung des Operators -Is	[string]$string = "Herdt-Verlag" $string -IsNot [string] (Ergebnis: FALSE)

Operator	Bedeutung	Beispiele
-Like	Einfacher Text-abgleich; prüft, ob sich eine Zeichen-kette in einer ande-ren befindet	"abcdef " -Like "abc" (Ergebnis: FALSE) "abcdef " -Like "abc*" (Ergebnis: TRUE) "abcdef " -Like "*c*" (Ergebnis: TRUE)
-NotLike	Umkehrung des Operators -Like	"abcdef " -NotLike "abc" (Ergebnis: TRUE) "abcdef " -NotLike "abc*" (Ergebnis: FALSE) "abcdef " -NotLike "*c*" (Ergebnis: FALSE)
-Match	Suche nach Substring ohne Wildcards (auch Muster-vergleich mit regu-lären Ausdrücken)	"abcdef " -Match "c" (Ergebnis: TRUE)
-NotMatch	Umkehrung des Operators -Match	"abcdef " -NotMatch "c" (Ergebnis: FALSE)

Auszug aus dem Herdt-Buch Kapitel 7.5, S.101 zu Vergleichsoperatoren

Arithmetische Operatoren: Zu Operatoren wie Addition „+“ usw, siehe die Übersicht im Herdt-Buch, im Kapitel 7.4, S. 98

Vergleichsoperator String

- PS:

```
($val -eq "abc")
($val -ceq "abc")      # Case sensitive
```

- Bash:

```
[ $val == "abc" ]      # Case sensitive
[ ${val,,} == "abc" ]
```

Logische Verknüpfung

- PS:

```
($val -lt 10 -and $val -gt 5)
($val -lt 5 -or $val -gt 10)
($val -gt 5 -xor $val -lt 10)
```

- Bash:

```
(( $val < 10 )) && (( $val > 5 ))
(( $val < 5 )) || (( $val > 10 ))
```

Aufgabe 1a - Alternation 5‘

Eine **Sequenz** ist also eine Zeile Code. Z.B. den ping testen.

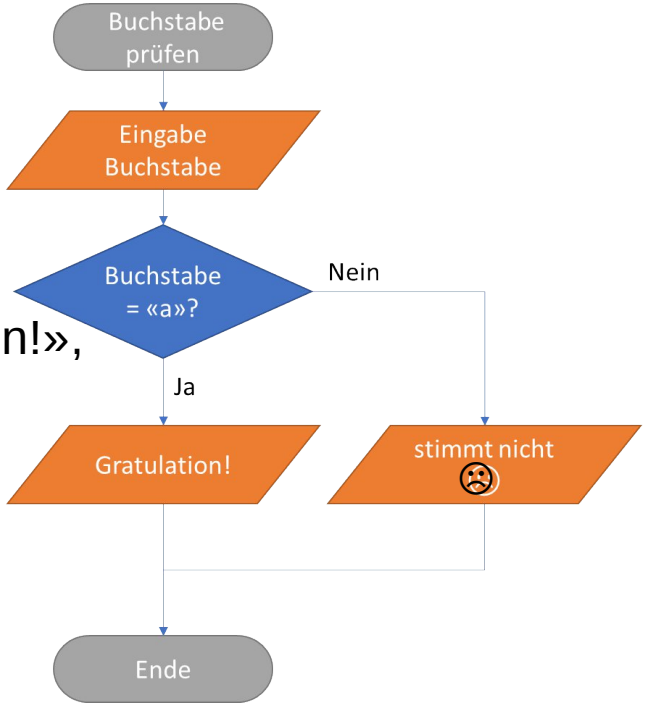
Eine **Alternation** ist eine Entscheidung und das sieht dann schon in PowerShell etwas spektakulärer aus.

```
if(Test-Connection -Count 1 www.bbzw.lu.ch){
  echo "Server gibt Antwort!"
}
else {
  echo "Server gibt keine Antwort ☹"
}
```

Setzen Sie das Beispiel in PowerShell um, indem Sie die Sequenz und Alternation zusammenführen in einer Funktion «ServerCheck». Gibt der Server www.bzw.lu.ch.ch Antwort?

Aufgabe 1b - Alternation 5‘

Für das nächste Beispiel überprüfen wir eine Eingabe. Sie geben einen Buchstaben ein und prüfen, ob dieser Buchstabe mit einem vorgegebenen Buchstaben übereinstimmt.
Wenn der Buchstabe «a» ist, dann «Gratulation!», sonst «Schade, stimmt nicht ☹.»
Wichtig ist zu wissen, wie die Vergleichsoperatoren definiert sind (Vorherige Folien checken☺).



- **Setzen Sie die Alternation in PowerShell um**, inkl. Eingabe des Buchstaben.

Aufgabe 1c - Alternation 10‘

Mehr Informationen zu Alternationen finden Sie hier: IF, ELSE, SWITCH: Bedingte Anweisungen in PowerShell | WindowsPro

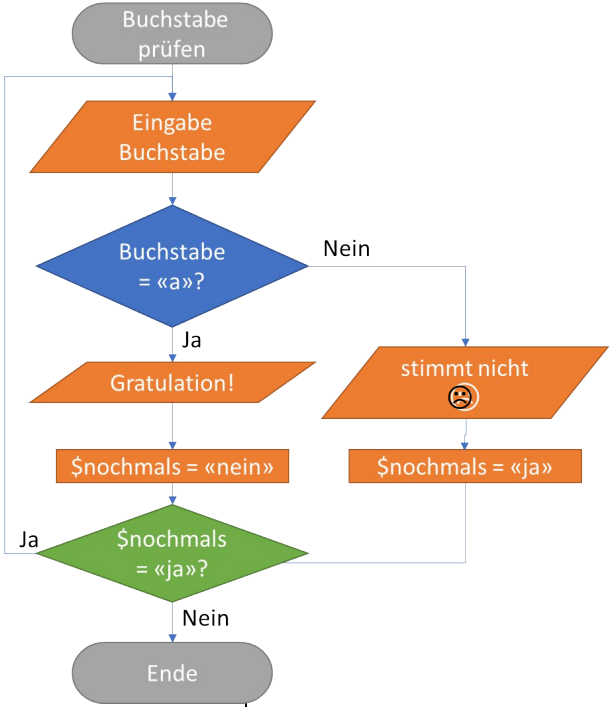
- Lesen Sie mindestens die Abschnitte:
  - Serie von Abfragen mit elseif
  - switch als Alternative zu langen elseif-Listen
  - PowerShell wertet bei switch alle Vergleiche aus
- Testen Sie das gelernte in der Powershell ISE

Iteration

- Ein Script, das nur Sequenzen und Alternationen hat, startet oben und arbeitet sich Schritt für Schritt nach unten durch.
- Wenn das Script wieder gestartet werden soll, dann müssen Sie es nochmals ausführen. **Wiederholungen** nennen wir **Iterationen**. Das sind **Schleifen**, z.B. um eine Abfrage solange auszuführen, bis sie einem bestimmten Wert entspricht.
- Nehmen wir das Beispiel mit der Eingabe des Buchstabens. **Solange der Buchstabe nicht korrekt eingegeben ist, muss er wieder eingegeben werden.**

Iteration

- Wenn Sie den PAP nebenan betrachten, dann wird am Ende geprüft (grün), ob die Eingabe stimmt, wenn nicht, dann muss nochmals ein Buchstabe eingegeben werden. Das so lange, bis die Eingabe stimmt.

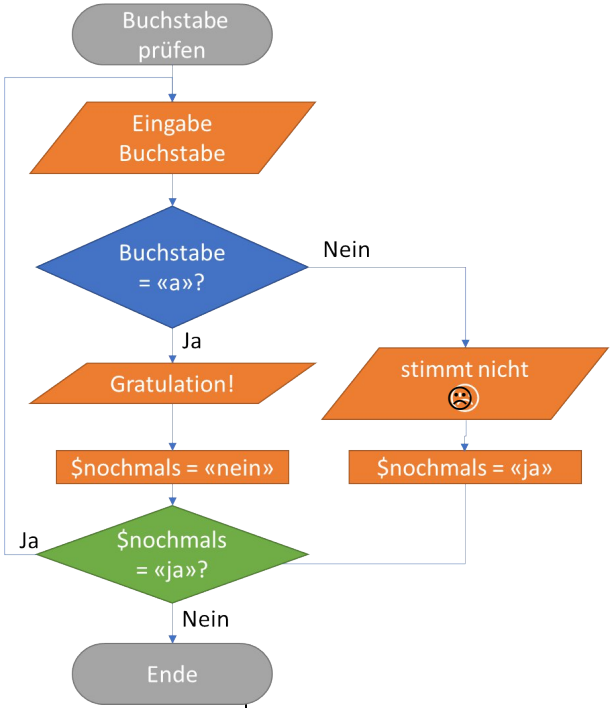


Aufgabe 2 - Iteration 5‘

Wenn Sie den PAP nebenan betrachten, dann wird am Ende geprüft (grün), ob die Eingabe stimmt, wenn nicht, dann muss nochmals ein Buchstabe eingegeben werden. Das so lange, bis die Eingabe stimmt.
Darum ist dies eine Do-While Iteration. Der Syntax dazu in PowerShell ist:

```
do{
  …
}while ($nochmals -eq "ja")
```

> Ergänzen Sie Ihr Script mit der Iteration.



Iteration

20‘

Es gibt **3 Grundtypen von Iterationen**:

- Fussgesteuert (`«do while()», «do until()»`)
- Kopfgesteuert (`«while()»`)
- Zählschleife (`«for()»`)

Schauen Sie sich das Video an:
https://www.youtube.com/watch?v=YcRxyEv78qg

- Versuchen Sie die Aufgaben im Video zu verstehen/ zu lösen

Für PowerShell finden Sie den Syntax auf folgender Seite:
https://www.windowspro.de/script/schleifen-powershell-foreach-while-do-until-continue-break

Zählerschleife «for»

- PS:

```
for ($i=1; $i -le 10; $i++) {
    echo "Number: $i"
}
```

- Bash:

```
for ((i=1; i<=10; i++))
do
    echo "Number: $i"
done
```

Kopfgesteuerte Schleife «while»

> PS:

```
$i=1
while ( $i -le 10 ) {
    echo "Number: $i"
    $i++
}
```

Kopfgesteuerte Schleife «while»

- Bash:

```
i=1
while (( $i <= 10 ))
do
    echo "Number: $i"
    ((i++))
done
```

Fussgesteuerte Schleife «do-while»

> PS:

```
$i=1
do {
    echo "Number: $i"
    $i++
} while ( $i -le 10 )
```

Fussgesteuerte Schleife «do-while»

- Gibt es in Bash nicht direkt

- Bash:

```
i=1
while true; do
    echo "Number: $i"
    ((i++))
    if (( $i > 10 )); then
        break
    fi
done
```

Hausaufgaben

- Fragen Sie Ihre Lehrperson

Quellen

- BBB. (2022). M122-master
- WindowsPro. (2022). https://www.windowspro.de/script/

Vielen Dank für Ihre Aufmerksamkeit!

KANTON LUZERN

Berufsbildungszentrum Wirtschaft, Informatik und Technik BBZW

www.bbzw.lu.ch