Powershell

Grundlagen

Grundlagen

Thema 0 - Powershell Wissen

Powershell

ist eine lose typisierte Programmiersprache und arbeitet mit implizierten Konvertierungen

> Skriptsprache: eine Programmiersprache die für Skripte verwendet wird, arbeitet mit meist nach dem Stapelverarbeitungsmodus und ist meist in interpritiert. (Interpreter)

Grundlagen - Operatoren

PowerShell – Grundlagen: Operatoren

Überblick

Operatoren in PowerShell werden verwendet für:

- Vergleiche
- String-Suche
- Listenprüfungen
- Berechnungen
- Zuweisungen

Vergleichsoperatoren

```
-eq   # gleich (equal)
-ne   # ungleich (not equal)
-gt   # größer als (greater than)
-lt   # kleiner als (less than)
-ge   # größer gleich (greater equal)
-le   # kleiner gleich (less equal)

Clear-Host

"Apfel" -eq "Birne"
"Apfel" -ceq "apfel"   # case sensitive Vergleich

12 -eq 12   # true
12 -ne 15   # true
12 -gt 9    # true
12 -lt 9    # false
12 -ge 11   # true
```

```
12 -le 12   # true


"Apfel" -lt "Birne"   # alphabetischer Vergleich
```

Grundlagen Zusammenfassung

Variablen & Strings

- Variablen starten mit `$` und werden mit `=` gesetzt → `$x = 5` (Typ automatisch)
- Zugriff immer mit `$`
- `"..."` → Variable wird **ersetzt** (ausgewertet) → `"Wert: $x"` → Wert: 5
- `'...'` → Variable bleibt **Text** → `'Wert: $x'` → Wert: $x
- `=` → Zuweisung | Vergleich z. B. mit `-eq`

Merksatz:
Doppelte Anführungszeichen = Wert wird eingesetzt
Einfache Anführungszeichen = bleibt nur Text

Datentypen

- PowerShell ist **dynamisch typisiert** → Typ wird automatisch erkannt (`$x = 5`, `$y = "Text"`)
- Häufige Typen: String (`"Text"`), Int (`5`), Double (`3.14`), Boolean (`$true/$false`), Array (`1,2,3`)
- Typ kann **explizit gesetzt** werden → `[int]$x = 5`, `[string]$name = "Derick"`
- Typ prüfen → `$x.GetType()`

Wichtig:

- Rechnen nur mit Zahlen sinnvoll (`"5" + 5` → kann unerwartet sein)
- Arrays werden mit Komma erstellt → `$arr = 1,2,3`

Merksatz:
Typ kommt automatisch – kann aber festgelegt werden

- Ausgabe →
  - `Write-Host "Text"` (nur Anzeige, nicht weiter nutzbar)
  - `Write-Output "Text"` oder `"Text"` (**empfohlen, weiterverarbeitbar**)
- Eingabe → `$x = Read-Host "Frage"`
- Datum → `Get-Date`
- Konsole löschen → `Clear-Host` / `cls` (beides leert die Konsole)

Beispiele anzeigen

```
# Ausgabe 'Write-Output'

# Variante 1 (offiziell)
Write-Output "Hallo"   # → Ausgabe: Hallo

# Variante 2 (Kurzform)
"Hallo"                # → Ausgabe: Hallo

# Konsole löschen 'cls'

# Variante 1 (voller Befehl)
Clear-Host             # → Konsole wird geleert

# Variante 2 (Kurzform / Alias)
cls                    # → Konsole wird geleert
```

Merksatz: **Write-Output besser als Write-Host | Read-Host Input einlesen | Cls entspricht Clear-Host**

Rechnen

- Grundrechenarten → `+` `-` `*` `/`
- Beispiele → `$x = 5 + 3` | `$y = 10 - 2` | `$z = 4 * 2` | `$a = 8 / 2`
- Punkt vor Strich gilt → `$r = 2 + 3 * 4` → Ergebnis: 14
- Klammern steuern Reihenfolge → `$r = (2 + 3) * 4` → Ergebnis: 20
- Zuweisung kombiniert → `$x += 5` (gleich `$x = $x + 5`)

Beispiel anzeigen

```
# Variante 1 (lang)
$x = $x + 5   # → erhöht $x um 5

# Variante 2 (kurz)
$x += 5       # → erhöht $x um 5
```

Merksatz: **Operatoren wie Mathe | Reihenfolge beachten | Klammern nutzen**

Operatoren

- Vergleich → `-eq` `-ne` `-gt` `-lt` → vergleichen Werte (True/False)
- Rechnen → `+` `-` `*` `/`
- Zuweisung → `=` `+=`
- String → `-like` (Wildcard) | `-match` (Regex)
- Listen → `-contains` (Liste enthält Wert) | `-in` (Wert in Liste)
- Logisch → `-and` `-or` `-not`
- Zusatz → `-not` (verneint) | `-c` (case-sensitive, z. B. `-ceq`)

Beispiele anzeigen

```
# Vergleich
5 -eq 5        # → True
5 -gt 3        # → True

# Rechnen
2 + 3          # → 5

# Zuweisung
$x = 5
$x += 2        # → 7

# String
"Test" -like "T*"    # → True
"Test" -match "es"   # → True

# Listen
1,2,3 -contains 2    # → True
2 -in 1,2,3          # → True

# Logisch
($true -and $false)  # → False

# Zusatz
5 -ne 3              # → True
```

```
"Test" -ceq "test"   # → False (Groß/Klein beachten)
```

PowerShell Verzweigungen (kurz & knapp)

- `if / else` → führt Code je nach Bedingung aus
- `switch` → prüft eine Variable gegen mehrere Fälle (übersichtlicher als viele ifs)

Beispiele anzeigen

```
# if / else
$x = 10

if ($x -gt 5) {
   "größer als 5"
} else {
   "kleiner oder gleich 5"
}

# switch
$farbe = "rot"

switch ($farbe) {
   "rot"   { "Stop" }
   "grün"  { "Go" }
   default { "Unbekannt" }
}
```

Merksatz: **if = einfache Entscheidung | switch = viele Fälle übersichtlich**

Powershell Tests

1. Test Klausur Vorbereitung

1. Wofür wird der Operator `-eq` verwendet?

Antwort anzeigen

Der Operator `-eq` wird verwendet, um zwei Werte auf Gleichheit zu vergleichen.

---

2. Was ist der Unterschied zwischen `-eq` und `-ceq`?

Antwort anzeigen

`-eq` ist nicht case sensitive,
`-ceq` ist case sensitive (Groß-/Kleinschreibung wird berücksichtigt).

---

3. Was prüft der Operator `-ne`?

Antwort anzeigen

Ob zwei Werte ungleich sind.

---

4. Was bedeutet `-gt`?

Antwort anzeigen

„greater than“ → größer als.

---

5. Was bedeutet `-le`?

Antwort anzeigen

„less equal“ → kleiner oder gleich.

---

6. Was passiert bei folgendem Ausdruck?

`"Apfel" -lt "Birne"`

Antwort anzeigen

Es wird ein alphabetischer Vergleich durchgeführt.
Der Ausdruck ergibt true, da „Apfel“ vor „Birne“ kommt.

---

7. Wofür wird der Operator `-like` verwendet?

Antwort anzeigen

Für einfache String-Vergleiche mit Wildcards (`*` und `?`).

---

8. Was bedeutet das `*` bei `-like`?

Antwort anzeigen

`*` steht für beliebig viele Zeichen (auch kein Zeichen).

---

9. Was bedeutet das `?` bei `-like`?

Antwort anzeigen

`?` steht für genau ein Zeichen.

---

10. Was prüft folgender Ausdruck?

`"Apfel" -like "A*"`

Antwort anzeigen

Ob der String mit „A“ beginnt.

---

11. Wofür wird der Operator `-match` verwendet?

Antwort anzeigen

Für die Suche nach Teilstrings mittels regulärer Ausdrücke (Regex).

---

12. Unterschied zwischen `-match` und `-cmatch`?

Antwort anzeigen

`-match` ist nicht case sensitive,
`-cmatch` ist case sensitive.

---

13. Was bedeutet `-notmatch`?

Antwort anzeigen

Der Ausdruck ist wahr, wenn das Suchmuster NICHT enthalten ist.

---

14. Was macht der Operator `-contains`?

Antwort anzeigen

Er prüft, ob eine Liste (links) einen bestimmten Wert enthält.

---

15. Was ist der Unterschied zwischen `-contains` und `-in`?

Antwort anzeigen

`-contains` → Liste steht links
`-in` → Liste steht rechts

---

16. Was prüft folgender Ausdruck?

```
"Apfel" -in "Birne","Apfel"
```

Antwort anzeigen

Ob „Apfel“ in der rechten Liste enthalten ist → Ergebnis: true

---

17. Wofür steht `-notin`?

Antwort anzeigen

Der Ausdruck ist wahr, wenn ein Wert NICHT in der Liste enthalten ist.

---

18. Nenne zwei mathematische Operatoren in PowerShell.

Antwort anzeigen

Zum Beispiel:

`+` (Addition)
`*` (Multiplikation)

---

19. Wofür wird der Operator `%` verwendet?

Antwort anzeigen

Für die Modulo-Berechnung (Rest einer Division).

---

20. Was macht der Operator `+=`?

Antwort anzeigen

Er addiert einen Wert zu einer Variable und speichert das Ergebnis in der gleichen Variable.

---

21. Was macht der Operator `-=`?

Antwort anzeigen

Er subtrahiert einen Wert von einer Variable.

---

22. Wofür wird `*=` verwendet?

Antwort anzeigen

Für Multiplikation mit gleichzeitiger Zuweisung.

---

23. Was bedeutet der Operator `++`?

Antwort anzeigen

Er erhöht den Wert einer Variable um 1.

---

24. Was bedeutet der Operator `--`?

Antwort anzeigen

Er verringert den Wert einer Variable um 1.

---

25. Warum wird in PowerShell `-eq` statt `=` verwendet?

Antwort anzeigen

`=` ist ein Zuweisungsoperator,
`-eq` ist ein Vergleichsoperator.

---

26. Was bedeutet `-ge`?

Antwort anzeigen

„greater equal“ → größer oder gleich.

---

27. Was bedeutet `-lt`?

Antwort anzeigen

„less than“ → kleiner als.

---

28. Was prüft folgender Ausdruck?

`12 -gt 9`

Antwort anzeigen

Ob 12 größer als 9 ist → Ergebnis: true

---

29. Was prüft folgender Ausdruck?

`12 -le 12`

Antwort anzeigen

Ob 12 kleiner oder gleich 12 ist → Ergebnis: true

---

30. Was prüft folgender Ausdruck?

`"Apfel","Birne" -contains "Birne"`

Antwort anzeigen

Ob „Birne“ in der Liste enthalten ist → Ergebnis: true

Selbstbewertung

0–10 richtig → □ wiederholen
11–20 richtig → ⚠ unsicher
21–27 richtig → □ gut vorbereitet
28–30 richtig → □□ prüfungsbereit

PowerShell – Klausurvorbereitung (Übersicht + Test)

PowerShell – Klausurvorbereitung (Übersicht + Test)

Grundlagen

PowerShell ist eine lose typisierte Skriptsprache und arbeitet mit automatischer Typkonvertierung.

Das bedeutet:

- Datentypen werden automatisch angepasst
- keine feste Typdefinition notwendig

Begriffe:

- Deklarierung → Variable erstellen
- Initialisierung → Wert zuweisen
- Konvertierung → Datentyp ändern

Variablen

- beginnen immer mit `$`
- erlaubte Zeichen: Buchstaben, Zahlen, `_`
- Leerzeichen nur mit `${}` möglich

Beispiele:

`$name`
`$_abc`
`$123`
`${mein Wert}`

Ungültig:

`$abc-12`
`$mein Name`

Geschützte Variablen:

`$true`, `$false`, `$_`

Datentypen

- String → `"Text"`
- Int → `12`
- Double → `4.5`
- Boolean → `$true / $false`
- DateTime → `Get-Date`

Wichtig für IHK:

- `Read-Host` liefert IMMER einen String → oft Konvertierung nötig

Typ anzeigen:

`$variable.GetType().Name`

Beispiel:

`[int]$zahl = Read-Host`

Cmdlets

- Ausgabe → `Write-Host`
- Eingabe → `Read-Host`
- Datum → `Get-Date`
- Konsole löschen → `Clear-Host` (Alias: `cls`)

Rechnen

`(4 + 4) * 12`
`12 - 3`
`16 * 4`

Wichtig:

- Punktrechnung vor Strichrechnung

Im String (Unterausdruck):

`"Ergebnis: $(4*12)"`

Datum:

`Get-Date("2026-04-14")`

Operatoren

Vergleich

`-eq -ne -gt -lt -ge -le`
`-ceq` → case sensitive

Beispiel:

`12 -gt 9` → true

Rechenoperatoren

`+ - * / %`

`%` = Modulo (Rest)

Zuweisung

`=` → Wert setzen
`+=` → addieren und speichern

Beispiel:

`$x = 5`
`$x += 3` → Ergebnis: 8

Stringoperatoren

`-like` → Wildcards
`*` → beliebig viele Zeichen
`?` → genau ein Zeichen

Beispiel:

`"Apfel" -like "A*"`

`-match` → Teilstring / Regex

Beispiel:

`"Apfel" -match "pf"` → true

Listenoperatoren

`-contains` → Liste links

`"A","B","C" -contains "A"`

`-in` → Liste rechts

`"A" -in "A","B","C"`

Wichtig:

- nur ganze Werte (keine Teilstrings!)

Logische Operatoren

`-and` → UND
`-or` → ODER
`-not` → NICHT

Unäre Operatoren

`++` → erhöhen
`--` → verringern

Verzweigungen

if / else

`if($zahl -lt 12){...} elseif(...) {...} else {...}`

Wichtig:

- Bedingungen müssen TRUE oder FALSE ergeben

---

switch

`switch($wert){ {$_ -lt 12}{"kleiner 12"; break} default{"größer"} }`

Wichtig:

- `$_` steht für aktuellen Wert
- `break` beendet den Block

TEST

---

1. Was bedeutet lose Typisierung?

Antwort anzeigen

Variablen haben keinen festen Datentyp und werden automatisch konvertiert.

---

2. Unterschied zwischen Deklarierung und Initialisierung?

Antwort anzeigen

Deklarierung = Variable erstellen
Initialisierung = Wert zuweisen

---

3. Woran erkennt man Variablen?

Antwort anzeigen

Am `$` Zeichen.

---

4. Was ist bei `Read-Host` wichtig?

Antwort anzeigen

Es liefert immer einen String.

---

5. Wofür wird `Get-Date` verwendet?

Antwort anzeigen

Für Datum und Uhrzeit.

---

6. Was macht `Clear-Host`?

Antwort anzeigen

Löscht die Konsole.

---

7. Ergebnis von `(4+4)*12`?

Antwort anzeigen

96

---

8. Wofür steht `-eq`?

Antwort anzeigen

Vergleich auf Gleichheit.

---

9. Unterschied `-eq` und `-ceq`?

Antwort anzeigen

`-ceq` beachtet Groß-/Kleinschreibung.

---

10. Was macht `-like`?

Antwort anzeigen

Vergleich mit Wildcards.

---

11. Bedeutung von `*` bei `-like`?

Antwort anzeigen

Beliebig viele Zeichen.

---

12. Bedeutung von `?` bei `-like`?

Antwort anzeigen

Genau ein Zeichen.

---

13. Was macht `-match`?

Antwort anzeigen

Teilstring-Suche.

---

14. Unterschied `-contains` und `-in`?

Antwort anzeigen

`-contains` → Liste links
`-in` → Liste rechts

---

15. Was macht `+=`?

Antwort anzeigen

Addiert und speichert den Wert.

---

16. Was machen `++` und `--`?

Antwort anzeigen

Erhöhen bzw. verringern um 1.

---

17. Wofür wird `if` verwendet?

Antwort anzeigen

Für Bedingungen.

---

18. Wann wird `else` ausgeführt?

Antwort anzeigen

Wenn keine Bedingung zutrifft.

---

19. Wofür wird `switch` verwendet?

Antwort anzeigen

Für mehrere Bedingungen.

---

20. Was macht `break` im switch?

Antwort anzeigen

Beendet den Block.

Selbstbewertung

0–7 richtig → □ wiederholen
8–14 richtig → ⚠ unsicher
15–18 richtig → □ gut
19–20 richtig → □□ bereit für die Klausur