# PowerShell – Schleifen kompakt erklärt

### Grundidee

Eine Schleife wiederholt einen Codeblock, solange eine Bedingung erfüllt ist oder bis eine Abbruchbedingung erreicht wird.

---

### Wichtige Schleifentypen

| Schleife | Bedeutung | Prüft wann? | Kann 0-mal laufen? |
|---|---|---|---|
| while | solange eine Bedingung wahr ist | vorher | ja |
| for | Zählschleife mit Start, Bedingung und Änderung | vorher | ja |
| foreach | läuft durch alle Elemente einer Liste | vorher | ja, wenn Liste leer ist |
| do while | läuft mindestens einmal, dann solange Bedingung wahr ist | danach | nein |
| do until | läuft mindestens einmal, dann bis Bedingung wahr wird | danach | nein |

---

### Abweisende Schleifen

Abweisende Schleifen prüfen die Bedingung vor dem ersten Durchlauf.

Sie können deshalb 0-mal laufen.

Beispiele:

```
while ($x -lt 5) {
}

for ($i = 0; $i -lt 3; $i++) {
}

foreach ($item in $liste) {
}
```

---

### Nicht-abweisende / fußgesteuerte Schleifen

Diese Schleifen prüfen die Bedingung erst nach dem ersten Durchlauf.

Sie laufen deshalb mindestens 1-mal.

Beispiele:

```
do {
}
while ($x -lt 5)

do {
}
until ($x -ge 5)
```

---

### Kopfgesteuert und fußgesteuert

| Begriff | Bedeutung | Beispiele |
|---|---|---|
| kopfgesteuert | Bedingung wird vor dem Schleifenkörper geprüft | while, for, foreach |
| fußgesteuert | Bedingung wird nach dem Schleifenkörper geprüft | do while, do until |

---

### Schleifenkopf

Im Schleifenkopf steht die Bedingung oder Steuerung der Schleife.

Beispiel:

```
while ($i -lt 5)
```

---

### Schleifenkörper

Der Schleifenkörper ist der Code, der wiederholt ausgeführt wird.

Beispiel:

```
{
    Write-Host $i
```

```
    $i++
}
```

---

### Iteration

Eine Iteration ist ein einzelner Schleifendurchlauf.

Wenn eine Schleife 5-mal läuft, hat sie 5 Iterationen.

---

### Abbruchbedingung

Die Abbruchbedingung beschreibt, wann die Schleife beendet werden soll.

Beispiel:

```
Abbrechen, sobald $i mindestens 5 ist.
```

Bei do until steht diese Bedingung direkt in der Schleife:

```
do {
    $i++
}
until ($i -ge 5)
```

---

### Weiterlaufbedingung

Die Weiterlaufbedingung beschreibt, solange die Schleife weiterlaufen soll.

Beispiel:

```
while ($i -lt 5)
```

Bedeutung:

```
Wiederholen, solange $i kleiner als 5 ist.
```

---

### while

while prüft vor jedem Durchlauf.

```
$i = 0

while ($i -lt 3) {
    Write-Host $i
    $i++
}
```

Ausgabe:

```
0
1
2
```

Warum keine 3?

```
3 -lt 3 = falsch
```

---

**do while**

do while läuft mindestens einmal und prüft danach.

```
$x = 10

do {
    Write-Host $x
}
while ($x -lt 5)
```

Ausgabe:

```
10
```

Die Schleife läuft nur 1-mal, weil danach geprüft wird:

```
10 -lt 5 = falsch
```

**do until**

do until läuft mindestens einmal und stoppt, sobald die Bedingung wahr ist.

```
$punkte = 0

do {
   $punkte += 10
}
until ($punkte -ge 100)
```

Bedeutung:

```
Wiederholen, bis $punkte mindestens 100 ist.
```

---

**for**

for ist eine typische Zählschleife.

Aufbau:

```
for (Startwert; Bedingung; Veränderung) {
}
```

Beispiel:

```
for ($i = 0; $i -lt 3; $i++) {
   Write-Host $i
}
```

Ausgabe:

```
0
1
2
```

Ablauf:

| Durchlauf | $i | Prüfung |
|---:|---:|---|
| 1 | 0 | 0 -lt 3 = wahr |
| 2 | 1 | 1 -lt 3 = wahr |
| 3 | 2 | 2 -lt 3 = wahr |
| Ende | 3 | 3 -lt 3 = falsch |

---

### Leerer Schleifenkörper

Auch wenn der Schleifenkörper leer ist, läuft die Schleife trotzdem.

```
for ($i = 0; $i -lt 3; $i++) {
}
```

Diese Schleife läuft 3-mal.

Sie gibt nur nichts aus.

---

### foreach

foreach wird verwendet, um jedes Element einer Liste zu verarbeiten.

```
foreach ($datei in $dateien) {
    Write-Host $datei
}
```

Bedeutung:

```
Für jede Datei in $dateien wird der Code einmal ausgeführt.
```

Wenn die Liste leer ist, läuft foreach 0-mal.

---

### Vergleichsoperatoren in PowerShell

| Operator | Bedeutung |
|---|---|
| -lt | kleiner als |
| -le | kleiner oder gleich |
| -gt | größer als |

| Operator | Bedeutung |
|---|---|
| -ge | größer oder gleich |
| -eq | gleich |
| -ne | ungleich |

Beispiele:

```
$i -lt 5    # kleiner als 5
$i -le 5    # kleiner oder gleich 5
$i -gt 5    # größer als 5
$i -ge 5    # größer oder gleich 5
$i -eq 5    # genau 5
$i -ne 5    # nicht 5
```

---

### Wichtig: -lt und -le

```
-lt 5 = kleiner als 5
```

Läuft bei:

```
0, 1, 2, 3, 4
```

Nicht bei:

```
5
```

```
-le 5 = kleiner oder gleich 5
```

Läuft bei:

```
0, 1, 2, 3, 4, 5
```

---

### while vs until

| Schleife | Bedeutung |
|---|---|
| while | läuft solange die Bedingung wahr ist |

| Schleife | Bedeutung |
| --- | --- |
| until | läuft bis die Bedingung wahr wird |

Beispiel mit while:

```
do {
   $punkte += 10
}
while ($punkte -lt 100)
```

Bedeutung:

```
Wiederholen, solange $punkte kleiner als 100 ist.
```

Beispiel mit until:

```
do {
   $punkte += 10
}
until ($punkte -ge 100)
```

Bedeutung:

```
Wiederholen, bis $punkte mindestens 100 ist.
```

---

**Bei 100 abbrechen**

Wenn bei 100 Schluss sein soll, ist bei do until logisch:

```
until ($punkte -ge 100)
```

Oder bei do while:

```
while ($punkte -lt 100)
```

Nicht passend wäre:

```
while ($punkte -le 100)
```

Denn bei 100 ist diese Bedingung noch wahr und die Schleife läuft nochmal.

---

**Zähler erhöhen**

```
$i++
```

ist dasselbe wie:

```
$i += 1
```

Beide erhöhen den Wert um 1.

---

**Zähler verringern**

```
$i--
```

ist dasselbe wie:

```
$i -= 1
```

Beide verringern den Wert um 1.

---

**Größere Schritte**

```
$i += 5
```

erhöht um 5.

```
$i -= 5
```

verringert um 5.

---

**Initialisierung**

Initialisierung bedeutet: Startwert setzen.

Beispiel:

```
$i = 0
```

---

**Inkrement**

Inkrement bedeutet: Wert erhöhen.

Beispiel:

```
$i++
```

oder:

```
$i += 1
```

---

**Dekrement**

Dekrement bedeutet: Wert verringern.

Beispiel:

```
$i--
```

oder:

```
$i -= 1
```

---

**Endlosschleife**

Eine Endlosschleife endet nicht von selbst.

Beispiel:

```
while ($true) {
    Write-Host "läuft"
```

```
}
```

Oder:

```
$x = 1

do {
    Write-Host $x
}
while ($x -lt 5)
```

Wenn $x nie verändert wird, bleibt die Bedingung immer wahr.

---

**break**

break beendet eine Schleife sofort.

```
while ($true) {
    break
}
```

---

**continue**

continue überspringt den Rest des aktuellen Durchlaufs und springt direkt zum nächsten Durchlauf.

```
for ($i = 1; $i -le 5; $i++) {
    if ($i -eq 3) {
        continue
    }

    Write-Host $i
}
```

Ausgabe:

```
1
2
4
```

```
5
```

---

### Off-by-one-Fehler

Ein Off-by-one-Fehler passiert, wenn eine Schleife einmal zu viel oder einmal zu wenig läuft.

Typischer Unterschied:

```
$i -lt 5
```

läuft bis 4.

```
$i -le 5
```

läuft bis 5.

---

### Merksätze

```
while = solange

until = bis

for = Zählschleife

foreach = für jedes Element

do while = mindestens einmal, dann solange

do until = mindestens einmal, dann bis

break = sofort raus

continue = nächster Durchlauf

-lt = kleiner als

-le = kleiner oder gleich

-gt = größer als
```

```
-ge = größer oder gleich

-eq = gleich

-ne = ungleich
```