Flow control

In Programmen hat man oft “wenn … dann …” Situationen. Im “wenn”-Teil steht eine Bedingung, die entweder wahr (True) oder falsch (False) ist. Diese Bedingungen werden in Python mit boolschen Ausdrücken dargestellt.

Häufig ergeben sich boolsche Ausdrücke mit Vergleichsoperatoren.

Symbol	Bedeutung
`==`	Gleichheit
`!=`	Ungleichheit
`<`	kleiner
`>`	größer
`<=`	kleiner oder gleich
`>=`	größer oder gleich

```
print(f"5 == 5 ist {5 == 5}")
print(f"5 == 3 ist {5 == 3}")

print(f"5 != 3 ist {5 != 3}")
print(f"5 != 5 ist {5 != 5}")

print(f"3 < 5 ist {3 < 5}")
print(f"5 < 5 ist {5 < 5}")
```

```
5 == 5 ist True
5 == 3 ist False
5 != 3 ist True
5 != 5 ist False
3 < 5 ist True
5 < 5 ist False
```

Vergleiche funktionieren auch mit Variablen:

```
a = 10
b = 20
print(f"Für a = {a} und b = {b} ist a < b {a < b}.")
```

```
Für a = 10 und b = 20 ist a < b True.
```

Man kann boolsche Werte in Variablen abspeichern:

```
vergleich1 = 5 > 3
vergleich2 = 3 < 1
print(f"vergleich1 ist {vergleich1}, vergleich2 ist {vergleich2}")
```

```
vergleich1 ist True, vergleich2 ist False
```

Boolsche Ausdrücke können mit logischen Operatoren kombiniert werden.

Python	Bedeutung
`and`	und-Verknüpfung (beide Bedingungen müssen wahr sein)
`or`	oder-Verknüpfung (mindestens eine Bedingung muss wahr sein)
`not`	Negation (kehrt den Wahrheitswert um)

```
print(f"(5 > 3) and (3 < 1): {(5 > 3) and (3 < 1)}")
print(f"(5 > 3) and (3 > 1): {(5 > 3) and (3 > 1)}")
print(f"(5 > 3) or (3 < 1): {(5 > 3) or (3 < 1)}")
print(f"(5 < 3) or (3 < 1): {(5 < 3) or (3 < 1)}")
print(f"not (5 > 3): {not (5 > 3)}")
print(f"not (5 < 3): {not (5 < 3)}")
```

```
(5 > 3) and (3 < 1): False
(5 > 3) and (3 > 1): True
(5 > 3) or (3 < 1): True
(5 < 3) or (3 < 1): False
not (5 > 3): False
not (5 < 3): True
```

Besonderheiten: `int`- und `float`-Typen können untereinander verglichen werden, `String`-Ausdrücke können nicht mit Zahlentypen verglichen werden. Jedoch können Strings untereinander verglichen werden.

```
print(f"5 == 5.0: {5 == 5.0}")
print(f"5 == '5': {5 == '5'}")
print(f"'Hallo' == 'Hallo': {'Hallo' == 'Hallo'}")
```

```
5 == 5.0: True
5 == '5': False
'Hallo' == 'Hallo': True
```

Bedingungen und Blöcke

Bei der Ablaufkontrolle (flow control) gibt es immer eine **Bedingung** und einen **Codeabschnitt**, der in Abhängigkeit von der Bedingung einmal, mehrmals oder gar nicht ausgeführt wird. Der Codeabschnitt, auf den sich die Bedingung bezieht, wird als **eingerückter Codeblock** eingegeben.

Wichtig: - Die Einrückung muss im ganzen Codeblock die gleiche Tiefe haben. - Die Einrückung sollte mit der Tabulator-Taste und nicht mit Leerzeichen erfolgen.

richtig:

```
weiterer code
weiterer code
flow control-Anweisung u. Bedingung:
    code
    code
    code
    code
weiterer code
weiterer code
```

falsch:

```
weiterer code
weiterer code
flow control-Anweisung u. Bedingung:
      code
      code
```

```
     code
      code
weiterer code
weiterer code
```

if-Anweisung

Die `if`-Anweisung wird verwendet, um Code nur dann auszuführen, wenn eine bestimmte Bedingung erfüllt ist. Ist die Bedingung wahr (`True`), wird der eingerückte Codeblock ausgeführt. Ist sie falsch (`False`), wird der Codeblock übersprungen.

Die Syntax lautet:

```
if [boolscher Ausdruck]:
    ...
    ...
```

Beispiel:

```
if a < b:
    print("a ist kleiner als b")
```

In diesem Beispiel wird die Ausgabe nur angezeigt, wenn `a` tatsächlich kleiner als `b` ist.

```
a = 5
b = 10
if a < b:
    print("a ist kleiner als b")

if b == a:
    print("b ist gleich a")
```

```
a ist kleiner als b
```

Die `else`-Anweisung wird verwendet, um einen Codeblock auszuführen, wenn die Bedingung der vorherigen `if`-Anweisung **nicht** erfüllt ist. Sie ergänzt die `if`-Anweisung und sorgt dafür, dass genau einer der beiden Blöcke ausgeführt wird.

Beispiel:

```
if a < b:
    print("a ist kleiner als b")
else:
    print("a ist nicht kleiner als b")
```

In diesem Beispiel wird die Ausgabe im `else`-Block nur angezeigt, wenn die Bedingung `a < b` **falsch** ist.

```
a = 10
b = 5
if a < b:
    print("a ist kleiner als b")
else:
    print("a ist nicht kleiner als b")
```

```
a ist nicht kleiner als b
```

Die `elif`-Anweisung steht für "else if" und wird verwendet, um mehrere Bedingungen in einer if-else-Struktur zu überprüfen. Sie folgt auf eine `if`-Anweisung und vor einer optionalen `else`-Anweisung. Sobald eine Bedingung wahr ist, wird der zugehörige Codeblock ausgeführt und die restlichen Bedingungen werden übersprungen.

Beispiel:

```
if a < b:
    print("a ist kleiner als b")
elif a == b:
    print("a ist gleich b")
else:
    print("a ist größer als b")
```

Mit `elif` können also mehrere Alternativen übersichtlich hintereinander geprüft werden.

```
a = 10
b = 5
if a < b:
    print("a ist kleiner als b")
elif a == b:
    print("a ist gleich b")
else:
    print("a ist größer als b")
```

```
a ist größer als b
```

Achtung: Sobald eine elif-Bedingung richtig ist, wird der Rest nicht mehr geprüft:

```
a = 1000
b = 5
if a > b:
    print("a ist größer als b")
elif a > b + 100:
    print("a ist viel größer als b")
else:
    print("a ist nicht größer als b")
```

```
a ist größer als b
```

Ändert man die Reihenfolge, funktioniert es:

```
a = 1000
b = 5
if a > b + 100:
    print("a ist viel größer als b")
elif a > b:
    print("a ist größer als b")
else:
    print("a ist nicht größer als b")
```

```
a ist viel größer als b
```

While-Schleifen

Die `while`-Schleife wird verwendet, um einen Codeblock so lange zu wiederholen, wie eine bestimmte Bedingung wahr (`True`) ist. Die Bedingung wird vor jedem Durchlauf überprüft. Sobald sie falsch (`False`) wird, endet die Schleife. Die Syntax ist:

```
while [boolscher Ausdruck]:
    ...
    ...
```

```
eingabe = int(input("Was ist 5 + 3? ")) # Eingabe: 7
while eingabe != 8:
    print("Falsch! Versuch es noch einmal.")
    eingabe = int(input("Was ist 5 + 3? ")) # Einfabe: 8
print("Richtig!")
```

```
Was ist 5 + 3? 7
Falsch! Versuch es noch einmal.
Was ist 5 + 3? 8
Richtig!
```

Wir suchen durch Ausprobieren die kleinste Zahl, die durch 18 und 12 teilbar ist:

```
zahl = 1
while (zahl % 18 != 0) or (zahl % 12 != 0):
    zahl = zahl + 1
print(f"Die kleinste Zahl, die durch 18 und 12 teilbar ist, ist {zahl}.")
```

```
Die kleinste Zahl, die durch 18 und 12 teilbar ist, ist 36.
```

Die `break`-Anweisung wird verwendet, um eine Schleife vorzeitig zu beenden. Sobald `break` im Schleifen-Block ausgeführt wird, wird die Schleife sofort verlassen und das Programm läuft mit dem Code nach der Schleife weiter.

Typischerweise wird `break` eingesetzt, wenn eine bestimmte Bedingung innerhalb der Schleife erfüllt ist und keine weiteren Durchläufe mehr nötig sind.

Beispiel von oben in abgeänderter Form:

```
zahl = 1
while True:
    if (zahl % 18 == 0) and (zahl % 12 == 0):
        print(f"Die kleinste Zahl, die durch 18 und 12 teilbar ist, ist {zahl}.")
        break
    zahl = zahl + 1
```

```
Die kleinste Zahl, die durch 18 und 12 teilbar ist, ist 36.
```

Die `continue`-Anweisung springt wieder zum Beginn der Schleife, der Rest der Anweisung wird für diesen Durchgang ausgelassen.

```
while True:
    name = input("Wer bist du? ") # Eingabe: Master, danach: Doctor
    if name != "Doctor":
        continue
    print("Hallo Doctor! Was ist das Passwort?")
    passwort = input("Passwort: ") # Eingabe: Tardis
    if passwort == "Tardis":
        break
print("Zutritt gewährt.")
```

```
Wer bist du? Master
Wer bist du? Doctor
Hallo Doctor! Was ist das Passwort? Tardis
Zutritt gewährt.
```