

# Informatik I: Einführung in die Programmierung

## 5. Bedingungen, bedingte Ausführung und Schleifen

Albert-Ludwigs-Universität Freiburg



Bernhard Nebel

27. Oktober 2017

# 1 Bedingungen und der Typ bool



- Typ bool
- Vergleichsoperationen
- Logische Operatoren

Bedingungen  
Typ bool  
Vergleichsoperationen  
Logische Operatoren  
Bedingte Anweisungen  
while-Schleifen

27. Oktober 2017

B. Nebel – Info I

3 / 21

# Der Typ bool

- Neben *arithmetischen Ausdrücken* gibt es noch **Boolesche Ausdrücke** mit True oder False als Werte.
- Die einfachsten Booleschen Ausdrücke sind Vergleiche mit dem Gleichheitsoperator ==.
- Die Werte True und False gehören zum Typ bool und werden automatisch nach int konvertiert:

## Python-Interpreter

```
>>> 42 == 42
True
>>> 'egg' == 'spam'
False
>>> type('egg' == 'spam')
<class 'bool'>
>>> True + True
2
```



Bedingungen  
Typ bool  
Vergleichsoperationen  
Logische Operatoren  
Bedingte Anweisungen  
while-Schleifen

27. Oktober 2017

B. Nebel – Info I

4 / 21

# Vergleichsoperatoren



- Es gibt die folgenden Vergleichsoperatoren:

symbolisch	Bedeutung
x == y	Ist x gleich y?
x != y	Ist x ungleich y?
x > y	Ist x echt größer als y?
x < y	Ist x echt kleiner als y?
x >= y	Ist x größer oder gleich y?
x <= y	Ist x kleiner oder gleich y?

- Strings werden anhand der **lexikographischen Ordnung** verglichen, wobei für Einzelzeichen der Unicode-Wert (Ergebnis der ord-Funktion) benutzt wird.
- Werte unvergleichbarer Typen sind ungleich. Bei den Anordnungsrelationen gibt es einen Fehler!

Bedingungen  
Typ bool  
Vergleichsoperationen  
Logische Operatoren  
Bedingte Anweisungen  
while-Schleifen

27. Oktober 2017

B. Nebel – Info I

5 / 21

## Vergleichsoperatoren in Aktion



### Python-Interpreter

```
>>> 'spamer' < 'spam'
False
>>> 'Spam' < 'spam'
True
>>> 2.1 - 2.0 == 0.1
False
>>> False < True
True
>>> 42 == 'zweiundvierzig'
False
>>> 41 < '42'
Traceback (most recent call last): ...
TypeError: unorderable types: int() < str()
```

Bedingungen  
Typ bool  
Vergleichsoperato-  
ren  
Logische  
Operatoren  
Bedingte An-  
weisungen  
while-  
Schleifen

## Logische Operatoren



- Es gibt die folgenden **logischen Operatoren**: or, and, not – mit aufsteigender Operatorpräzedenz.
- Bedeutung wie in **Aussagenlogik**, d.h.
  - $x < 10$  or  $y > 100$  hat den Wert True, wenn  $x$  kleiner als 10 ist, oder falls das nicht der Fall ist, wenn  $y$  größer als 100 ist.
  - $1 \leq x$  and  $x \leq 10$  hat den Wert True, wenn  $x$  zwischen 1 und 10 (inklusive) liegt.
  - Dies kann in Python auch so geschrieben werden (wie in mathematischer Notation):  $1 \leq x \leq 10$ .
  - $\text{not}(x < y)$  ist True wenn  $x \geq y$  ist.
- Alle **Nullwerte**, d.h. None, 0, 0.0, (0 + 0j) und '', werden wie False behandelt, alle anderen Werte wie True!
- Die **Auswertung wird beendet**, wenn das Ergebnis klar ist (wichtig bei Seiteneffekten und nicht-booleschen Werten).

Bedingungen  
Typ bool  
Vergleichsoperato-  
ren  
Logische  
Operatoren  
Bedingte An-  
weisungen  
while-  
Schleifen

## Logische Operatoren in Aktion



### Python-Interpreter

```
>>> 1 < 5 < 10
True
>>> 5 < 1 or 'spam' < 'egg'
False
>>> 'spam' or True
'spam'
>>> '' or 'default'
'default'
>>> 'good night' and 'ding ding ding'
'ding ding ding'
>>> 0 and 10 < 100
0
>>> not 'spam' and (None or 0.0 or 10 < 100)
False
```

Bedingungen  
Typ bool  
Vergleichsoperato-  
ren  
Logische  
Operatoren  
Bedingte An-  
weisungen  
while-  
Schleifen

## 2 Bedingte Anweisungen



- if-Anweisung
- if-else-Anweisung
- elif-Anweisung

Bedingungen  
Bedingte An-  
weisungen  
if-Anweisung  
if-else-  
Anweisung  
elif-Anweisung  
while-  
Schleifen

## Bedingte Ausführung



- Bisher wurde jede eingegebene Anweisung ausgeführt.
- Manchmal möchte man aber eine Anweisung oder einen Anweisungsblock nur unter bestimmten Bedingungen ausführen: **if-Anweisung**.

Bedingungen  
Bedingte Anweisungen  
if-Anweisung  
if-else-Anweisung  
elif-Anweisung  
while-Schleifen

### Python-Interpreter

```
>>> x = 3
>>> if x > 0:
...     print('x ist strikt positiv')
...
x ist strikt positiv
>>> x = 0
>>> if x > 0:
...     print('x ist strikt positiv')
...
>>>
```

## If-else



- Möchte man im positiven und im negativen Fall etwas machen: **if-else-Anweisung**.

Bedingungen  
Bedingte Anweisungen  
if-Anweisung  
if-else-Anweisung  
elif-Anweisung  
while-Schleifen

### Python-Interpreter

```
>>> x = 3
>>> if x%2 == 0:
...     print('x ist gerade')
... else:
...     print('x ist ungerade')
...
x ist ungerade
```

- Soll ein Anweisungsblock leer bleiben, kann man dafür pass einsetzen.

## Verkettete bedingten Anweisungen



- Will man mehrere Fälle behandeln, gibt es die **verketteten bedingten Anweisungen**

Bedingungen  
Bedingte Anweisungen  
if-Anweisung  
if-else-Anweisung  
elif-Anweisung  
while-Schleifen

### Python-Interpreter

```
>>> x = 3
>>> y = 0
>>> if x < y:
...     print('x ist kleiner als y')
... elif x > y:
...     print('x ist größer als y')
... else:
...     print('x und y sind gleich')
...
x ist größer als y
```

- Es wird immer der Block ausgeführt, bei dem die Bedingung das erste Mal wahr wird.

## Geschachtelte Konditionale



- Man kann auch bedingte Anweisungen als Block in bedingten Anweisungen unterbringen.

Bedingungen  
Bedingte Anweisungen  
if-Anweisung  
if-else-Anweisung  
elif-Anweisung  
while-Schleifen

### Python-Interpreter

```
>>> x = 5
>>> if x > 0:
...     if x > 10:
...         print('successful encyclopedia salesman')
...     else:
...         print('unsuccessful encyclopedia salesman')
...
>>>
```

- Durch Einrückung ist immer klar, wozu die bedingte Anweisung gehört!

## 3 while-Schleifen



Bedingungen  
Bedingte Anweisungen  
while-Schleifen

## while-Schleifen



Bedingungen  
Bedingte Anweisungen  
while-Schleifen

- Oft muss etwas wiederholt gemacht werden, bis ein bestimmter Wert erreicht wird.
- Hier benutzt man die **while-Schleife**:  
`while Bedingung:`  
`Anweisungen`
- Die *Anweisungen* werden wiederholt, solange die *Bedingung* `True` (oder einen äquivalenten Wert) liefert.
- Damit könnten wir unseren „Multiplikationsalgorithmus“ umsetzen.

## Multiplikations-Algorithmus



Bedingungen  
Bedingte Anweisungen  
while-Schleifen

### Eingabe und Ausgabe

Eingabe: Zwei natürliche Zahlen  $L$  und  $R$

Ausgabe: Das Produkt von  $L$  und  $R$

### Algorithmus

- 1 Setze  $P$  auf 0.
- 2 Falls  $R = 0$ , gebe  $P$  als Ergebnis zurück.
- 3 Addiere  $L$  zu  $P$  hinzu.
- 4 Reduziere  $R$  um 1.
- 5 Mache bei Schritt 2 weiter.

## Multiplikations-Programm in Python



Bedingungen  
Bedingte Anweisungen  
while-Schleifen

### Python-Interpreter

```
>>> def mult(l, r):  
...     p = 0  
...     while r != 0:  
...         p = p + l  
...         r = r - 1  
...     return p  
...  
>>> mult(3, 2)  
6
```

- Was passiert hier genau?
- **Visualisierung** der Ausführung: <http://pythontutor.com>

## Ein weiteres Beispiel: Summe aller Zahlen bis $n$



- Wir wollen alle Zahlen von 1 bis  $n$  aufsummieren:  $\sum_{i=1}^n i$ .

Bedingungen

Bedingte Anweisungen

while-Schleifen

### Python-Interpreter

```
>>> def sumup(n):
...     i = 1
...     result = 0
...     while i <= n:
...         result = result + i
...         i = i + 1
...     return result
...
>>> sumup(10)
55
```

- Ginge auch einfacher:  $\sum_{i=1}^n i = \frac{(n+1) \times n}{2}$

### Visualisierung

## Zusammenfassung



- `bool` ist ein weiterer **Typ**, dessen beide Werte `True` und `False` sind.
- Vergleiche, wie z.B. `==` oder `<`, liefern **Boolesche Werte**.
- Boolesche Werte werden automatisch nach `int` konvertiert, wobei `True` gleich 1 und `False` gleich 0 ist.
- Alle **Nullwerte** werden als `False` interpretiert, alle **Nichtnullwerte** als `True`.
- Mit `if-(elif)-else`-Anweisungen kann man bei der Ausführung verschiedene Anweisungen wählen.
- **while-Schleifen** erlauben die bedingte Wiederholung von Anweisungen im Körper der Schleife.
- Nutzen Sie `pythontutor.com`, um Programmausführungen zu verstehen.

Bedingungen

Bedingte Anweisungen

while-Schleifen