Python Kurs 2019/2020

2: Programme, Vergleiche, If, Kommentare

Bernhard Mallinger

`b.mallinger [at] gmx.at`

https://totycro.github.io/python-kurs

Programme (1/3)

Under the hood

- Historisch: Befehle abarbeiten
- CPU (Prozessor) verarbeitet Befehle
  - CPU ist "schlau"
    → Hat Schaltkreise zum Ausführen von Befehlen
- Zwischenergebnisse in Arbeitsspeicher (RAM)
- Input von Eingabegeräten, Festplatte, Internet
- Ouput auf Bildschirm, Festplatte, Internet
- Betriebssystem & Python kümmern sich um Details dieser Ressourcen

Programme (2/3)

- Programme sind Textdateien*
  - in menschen- und computerlesbarer Sprache
- Spannungsfeld Programmierung:
  - Befehle für die Maschine formulieren (*imperativ*)
  - Beschreiben, welches Ergebnis du haben möchtest (*deklarativ*)

Programme (3/3)

Gegeben: Liste von Zahlen `l`

Beispiel imperativ:

```
Setze x auf 0
Für alle Indizes i zwischen 0 und der Länge von l:
    Setze x auf x plus dem i-ten Element von l
→ x ist die Summe der Elemente
```

Programme (3/3)

Gegeben: Liste von Zahlen `l`

Beispiel imperativ:

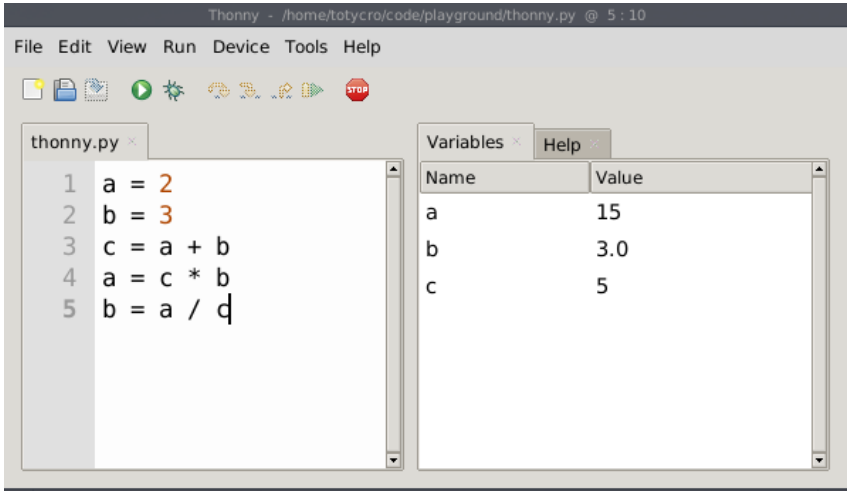
```
Setze x auf 0
Für alle Indizes i zwischen 0 und der Länge von l:
    Setze x auf x plus dem i-ten Element von l
→ x ist die Summe der Elemente
```

Beispiel deklarativ:

```
Verbinde alle Elemente der Liste l mit der Operation "plus"
```

Programme und Variablen in Thonny

- Script als Datei speichern (nicht in Shell)
- "Play"-Button zum Ausführen
- Ausführung lädt neue Werte auch in Shell



`print`

Während des Programmablaufs mit `print` aktuellen Wert ausgeben (in Shell)

```
a = 2
print(a)
a = "apple"
print(a)
```

Output:

```
2
apple
```

Vgl. `print("a")` und `"a"` in Shell und Programm

Vergleichsoperatoren (1/4)

- Wie in Mathematik
  - Funktion mit 2 Parametern, gibt `bool` zurück
- Gleichheit wird als `==` geschrieben
  (`=` bedeutet Variablenzuweisung!)
- ≠ als `!=`
  ≥ als >=
  ≤ als <=

```
>>> 2 > 3
False
>>> 5.5 <= 20.3
True
>>> 4 == 4
True
>>> 3 != 3
False
```

Vergleichsoperatoren (2/4)

Verkettung auch möglich:

```
>>> weight = 3.2
>>> 2.0 <= weight < 4.0
True
>>> 2.0 <= weight < 2.333
False
```

Bei Strings nur `==` und `!=` wirklich sinnvoll:

```
>>> a = "apple"
>>> a == "apple"
True
>>> a != "pear"
True
```

Vergleichsoperatoren (3/4)

Gewisse Typkombinationen möglich:

```
>>> "apple" < 4
Traceback (most recent call last):
  File "<pyshell>", line 1, in <module>
TypeError: '<' not supported between instances of 'str' and 'int'
>>> 2 < 2.2
True
```

Auch möglich, aber sinnvoll?

```
>>> "2" == 2
False
>>> "2" != 2
True
```

Vergleichsoperatoren (4/4)

Achtung bei `float`:

```
>>> 2 == 2.0
True
>>> 0.1 + 0.1 == 0.2
True
>>> 0.1 + 0.1 + 0.1 == 0.3
False
```

`if-then-else` (1/2)

```
weight = 3.2

if weight > 2:
    use_production_process = "A"
else:
    use_production_process = "B"

print(use_production_process)
```

Output:

```
A
```

`if`-`then`-`else` (2/2)

- Einrückung entscheidend
  - 4 Leerzeichen laut Style Guide
  - `then`- und `else`-Teil können beliebig lang sein
    → jede Zeile muss aber eingerückt sein
- `else` ist optional
- Nach `if` muss ein `bool` stehen, oder ein zu `bool` konvertierbarer Wert

```
if False:
    print("never executed")
```

Logische Operatoren

- `and`, `or` und `not`
- Oder ist nicht ausschließend

```
>>> True or True
True
>>> True and True
True
>>> False and False
False
>>> False or True
True
>>> (weight > 2.0) and (length <= 25)
True
```

```
>>> elevator_can_operate = \
    not ((number_persons > 6) or (person_weight_sum > 200))
>>> elevator_can_operate = \
    (number_persons <= 5) and (person_weight_sum <= 200)
```

Komplexere Abfragen (1/2)

Verschachtelt (nested):

```
if production_process == "A":
    if weight > 2.6:
        presstakte = 5
    else:
        presstakte = 4
else:
    presstakte = 1
```

production_process	weight	presstakte
"A"	>2.6	5
"A"	≤2.6	4
sonst		1

Komplexere Abfragen (2/2)

```
if production_process == "A":
    configuration = 1
else:
    if production_process == "B":
        configuration = 2
    else:
        configuration = 3
```

kann verkürzt werden zu

```
if production_process == "A":
    configuration = 1
elif production_process == "B":
    configuration = 2
else:
    configuration = 3
```

Herbst / Winter

AUFGABE

Angenommen, die aktuelle Temperatur in Grad steht in der Variable `temperature`.

Schreibe ein Programm, das berechnet, ob man eine Jacke anziehen muss.

- Über 20 Grad: "Keine Jacke"
- Nur über 10 Grad: "Herbstjacke"
- Unter 10 Grad: "Winterjacke"

Verwende `print` für Ausgaben.

Variante: Speichere das Ergebnis in einer Variable.
Variante: Berücksichtige auch die Windgeschwindigkeit.

Positiv / Negativ

AUFGABE

Schreibe ein Programm, dass durch eine `print`-Ausgabe angibt, ob eine Variable positiv, 0 oder negativ ist.

Lohnverrechnung

AUFGABE

Schreibe ein Programm für eine Lohnverrechnung. Die Variable `hours` enthält die Anzahl der gearbeiteten Stunden für eine Woche.

Berechne den Wochenlohn wie folgt:

- Für die ersten 40 Stunden: 20 Euro pro Stunde
- Alle darüberliegenden Stunden mit 50% Überstundenzuschlag

Versuche noch Varianten einzubauen (48 Stunden Limit, Abzug wenn nur unter 8 Stunden, ...).

Zahlen, die sich ändern können, können auch am Anfang definiert werden (Stundenlohn, 48 Stunden Limit), um sie später leicht anpassen zu können.

Wahrheitswerte von nicht `bool`-Werten (1/2)

- Konvertierungsfunktion `bool()`
- Bei `if x:` wird intern `if bool(x):` aufgerufen
- "leer" bedeutet `False`

```
>>> bool("")
False
>>> bool("a")
True
>>> bool(" ")
True
>>> bool(" ".strip())
False
```

```
if value.strip():
    print("value contains non-whitespace characters")
else:
    print("value contains only whitespace characters")
```

Wahrheitswerte von nicht `bool`-Werten (2/2)

- Null bedeutet `False`
- Explizite Vergleiche oft sinnvoll (Lesbarkeit)

```
>>> bool(0)
False
>>> bool(-1)
True
>>> bool(1)
True
>>> bool(0.0)
False
>>> bool(0.0000001)
True
```

Kommentare (1/2)

- Alles nach `#` wird ignoriert
- Gebrauch für Dokumentation

```
if length >= 5.0 and wood_type == "Fichte":
    # As described in http://www.wood.com/fichte,
    # Fichte develops property A if it's long
    presstakte = presstakte + 1
```

Vermeide nichtssagende Kommentare:

```
# We need to increase the presstakte
presstakte = presstakte + 1
```

Vor dem Kommentarschreiben: Versuche den Code so zu formulieren, dass Kommentare nicht notwendig sind

Kommentare (2/2)

Missbrauch um Code kurz zu deaktivieren

```
a = f(b, c)
# g(a)
c = h(a)
# print(a)
```

Mehrzeilige Kommentare

```
"""Längere Kommentare können
mit dreifachen Anführungsstrichen
begonnen und beendetet werden"""
```

Eigentlich ist das ein mehrzeiliger String:

```
a = """Lorem ipsum dolor sit amet,
consectetur adipisici elit,
sed eiusmod tempor incidunt
ut labore et dolore magna aliqua."""
print(a)
```