Programmierkurs Python I

Michaela Regneri

2009-11-05

(Folien basieren auf dem gemeinsamen Kurs mit Stefan Thater)

Übersicht

- Variablen
- Datentypen
- Werte
- Operatoren und Ausdrücke
- Kontrollstrukturen: if, while

Imperatives Programmieren

- Python ist im Kern eine imperative Programmiersprache
- Programme sind Schritt-für-Schritt-Sequenzen von Anweisungen
- Ausdrücke haben Werte
- Werte können Variablen zugewiesen werden
- Das wesentliche Werkzeug zur Steuerung des Programmablaufs sind Kontrollstrukturen

„Größte Zahl“ in Python

- Gegeben eine Liste `lis` von n natürlichen Zahlen; gesucht ist die größte Zahl in `lis`.

```
lis = [17,23,2,19]
max = lis[0]
i = 1
while i < len(lis):
  if lis[i] > max:
    max = lis[i]
  i = i + 1
```

Elemente imperativer Programme

- Gegeben eine Liste `lis` von n natürlichen Zahlen; gesucht ist die größte Zahl in `lis`.

```
lis = [17,23,2,19]
max = lis[0]
i = 1
while i < len(lis):
    if lis[i] > max:
        max = lis[i]
    i = i + 1
```

Variablen

Zuweisungen

Ausdrücke

Kontrollstrukturen

(Schleifen, Verzweigungen)

Variablen, Werte, Datentypen

- Werte in Python können verschiedenen Datentypen haben: Zahlen, Listen, Zeichenketten, ...
- Variablen zeigen auf Positionen im Hauptspeicher, an denen Werte gespeichert werden
- Dynamische Typisierung: Variablen haben keinen festen Datentyp
  - Der Datentyp einer Variablen ergibt sich aus dem Typ des zugewiesenen Wertes
  - Eine Variable kann während der Laufzeit Werte verschiedener Typen annehmen

Einige Datentypen

- Wahrheitswerte: `bool`
  (= Typ der Konstanten `True` und `False`)
- Zahlen: `int`, `long`, `float`, `complex`
- Zeichenketten: `str`, unicode
- Sammeltypen: `tuple`, `list`, `set`, `dict`
- [...]

Ausdrücke

- Ausdrücke sind Konstrukte, die einen Wert haben
- Wir können unterscheiden:
  - Literale: Audrücke, aus denen der Wert direkt abgelesen / hingeschrieben werden kann
  - Variablen
  - Zusammengesetzte Ausdrücke mit Operatoren
  - Funktions- bzw. Methodenaufrufe

Ganze Zahlen

- `int` (plain integers)
  - Wertebereich: -2^b, ..., $+2^{b-1}$, b ≥ 31 (Systemabhängig)
- `long` (long integers)
  - (in Python) beliegig große ganze Zahlen
- Ganzzahl-Literale (*i* = *3*)
  - denotieren Werte vom Typ `int`
  - Ausnahmen: Die Zahl ist zu groß für den zulässigen Wertebereich oder das Literal endet mit „L“

Ganzzahl-Literale

- „normal“ notierte Zahlen wie `17, 0, -23` im Quelltext werden als Dezimalzahlen (Basis 10) interpretiert
- Literale, die mit `0o` (oder `0`0) anfangen, werden als Oktalzahlen (Basis 8) interpretiert
  (Bsp.: `0o13` repräsentiert Wert 11)
- Literale, die mit 0x anfangen, werden als Hexadezimalzahl (Basis 16) interpretiert
  (Bsp.: `0x1ca` repräsentiert 458)

Fließkommazahlen

- Fließkommazahlen werden als Dezimalzahlen repräsentiert (1.1, 47.11)
- Wertebereich ist Systemabhängig
- Häufig können Dezimalzahlen intern nicht präzise dargestellt werden

```
>>> 0.1
0.1
>>> 0.1 *  10000000000000000000000000000000
1.0000000000000001e+31
```


Operatoren

- Elementare (Rechen-) Operationen werden mit Operatoren dargestellt.
- Rechenoperatoren (Auswahl):

```
a + b
a - b
a * b
a / b
a % b
```

(Modulo/Rest bei ganzzahliger Division)

- Wenn a, b nicht den gleichen Typ haben, liefern die Operationen einen Wert vom allgemeineren Typ.

Präzedenz

- EinAusdruck kann mehrere Operatoren beinhalten:
  2 * 3 + 4

- Die Reihenfolge, in der Operatoren ausgewertet werden. heißt *Präzedenz*

- Mit Klammern kann die Präzedenz direkt angegeben werden:

```
>>> (2 * 3) + 4
10
>>> 2 * (3 + 4)
14
```


Präzedenz

- Ohne Klammern wendet Python die Punkt-vor-Strich-Regel an
  `2 * 3 + 4 = (2 * 3) + 4`

- Stilfrage: manchmal ist es sinnvoll, Klammern auch dann zu verwenden, wenn sie redundant sind (mehr Leserlichkeit)

- Bei irrelevanter Präzedenz gilt das nicht:
  `2 + 3 + 4` ist besser als `2 + (3 + 4)`

Vergleichsoperatoren

- Vergleichsoperatoren:

`a < b`	`a > b`	(größer)
`a <= b`	`a >= b`	(größer-gleich)
`a == b`	`a != b`	(gleich bzw. ungleich)

- Das Resultat einer Vergleichsoperation ist ein Wahrheitswert (bool)

```
>>> 3 > 2
True
>>> (2 * 3) + 4 != 2 * 3 + 4
False
```


Wahrheitswerte

- Der Typ `bool` repräsentiert die beiden Wahrheitswerte `True` und `False`.
- Operationen:

`not a`	(Negation)
`a and b`	(Konjunktion)
`a or b`	(Disjunktion)

- Präzedenz: `not` ⋙ `and` ⋙ `or`
  `a and not b or c = (a and (not b)) or c`
- Kurzschluss-Auswertung: die Auswertung bricht ab, sobald das Ergebnis feststeht (z.B.: `True or irgendwas`).

String-Literale

```
'Das ist ein String.'
"Das auch."
"Er sagte \"Hallo\"."
'Er sagte "Hallo".'
```

- Beachte: Strings dürfen keine Umlaute etc. enthalten, wenn keine Kodierung angegeben ist.
- Kodierung am Anfang des Quelltexts spezifizieren:

```
# -*- coding: utf-8 -*-
# -*- coding: latin-1 -*-
```


String-Operatoren (Auswahl)

- Konkatenation:

```
>>> 'Hallo' + 'Welt'
'HalloWelt'
```

- Zugriff auf einzelne Zeichen: wie mit Listen

```
>>> 'Hallo'[0]
'H'
```

```
>>> 'Hallo'[1]
'a'
```

- Testen, ob ein Teilstring vorkommt:

```
>>> 'Ha' in 'Hallo'
True
```

```
>>> 'Ha' in 'Hello'
False
```

String-Operatoren (Auswahl)

- Länge:

```
>>> len('Hallo')
5
```

- Konvertieren in einen anderen Datentyp (Zahl):

```
>>> int('123')
123
```

```
>>> float('123')
123.0
```


Variablen

- Variablen kann der Wert eines Ausdrucks zugewiesen werden
- Variablen können ausgewertet werden, um ihren Wert in einem Ausdruck zu verwenden.

```
>>> zahl = 123
>>> zahl = zahl + 2
>>> print(zahl)
125
```

- `print(zahl)` ist eine Anweisung, die den Wert eines Ausdrucks (hier: `zahl`) ausgibt

Variablen

- Variablennamen (allgemeiner: alle Bezeichner) müssen mit einem Buchstaben oder „_“ beginnen. Die restlichen Zeichen dürfen auch Ziffern enthalten.
- Umlaute etc. sind nicht erlaubt (ASCII-Codierung)
- Der Name darf kein Schlüsselwort (`if`, `while`, etc.) sein
- Groß/Kleinschreibung wird unterschieden.
- Beispiele:
  - ✓ OK: `Foo, foo12, _Foo`
  - ✗ Falsch: `12foo, Übertrag, if`

Zuweisungen

- `var = expr` Der Ausdruck expr wird ausgewertet, dann wird der Wert in var gespeichert.
- $\texttt{var}_1 = \texttt{var}_2 = \ldots = \texttt{expr}$
  Allen $\texttt{var}_i$ wird der Wert von `expr` zugewiesen.
- $\texttt{var}_1, \ldots, \texttt{var}_n = \texttt{expr}_1, \ldots, \texttt{expr}_n$
  - Die $\texttt{expr}_i$ werden ausgwertet; dann werden die entsprechenden Werte den $\texttt{var}_i$ zugewiesen.
  - Beispiel: `a, b = ‘a‘, ‘b‘` (Werte tauschen)

Zuweisungen

- Zuweisungen der Form `x = x + y`, in der eine Variable x nur mit einem anderen Wert kombiniert und gleich wieder zugewiesen wird, sind sehr häufig.
- Abkürzende Syntax:

```
x += expr
x -= expr
x *= expr
x /= expr
x %= expr
```


Anweisungen (Statements)

- Ein Python-Programm ist aus einer Sequenz von Anweisungen aufgebaut
- Bisher kennen wir: Zuweisungen, `print`
- Eine Anweisung entspricht etwa einem Schritt im Algorithmus
- Anweisungen werden durch Zeilen getrennt: pro Zeile steht (normalerweise) genau eine Anweisung
- Man kann auch (kurze) Anweisungen per Semikolon trennen (und auf eine Zeile schreiben)

Kontrollstrukturen

- Manchmal will man Anweisungen mehrfach ausführen, oder nur unter bestimmten Bedingungen
- Das ist die Funktion von Kontrollstrukturen
  - Bedingungen: if
  - Schleifen: while, for

if – else

- Wenn $\texttt{expr}_1$ zu wahr auswertet, wird $\texttt{anweisung}_1$ ausgeführt.
- Ansonsten wird $\texttt{anweisung}_2$ ausgeführt.
- Als falsch gelten: `False`, `0`, der leere String, leere Listen, leere Mengen, …
- Alle anderen Werte gelten als wahr.

```
if expr1:
  anweisung1
[else:
  anweisung2]
```

if – elif – else

- Ausdrücke werden in angegebener Reihenfolge ausgewertet, bis einer zutrifft.
- Dann wird die entsprechende Anweisung ausgeführt.
- Wenn keiner der Ausdrücke zutrifft, wird die else-Anweisung ausgeführt.

```
if expr1:
  anweisung1
[elif expr2:
  anweisung2]
...
[else:
  anweisungk]
```


Einrückungen

- Leerzeichen sind wichtig: die Anweisungen in einer if-Anweisung müssen eingerückt werden!

```
if a < b:
  if a < c:
    print(”bla”)
  else:
    print(”blub“)
```

```
if a < b:
  if a < c:
    print(”bla”)
else:
  print(”blub“)
```

Blöcke

- Mehrere Anweisungen können zu einem Block gruppiert werden, indem die entsprechenden Anweisungen gleich eingerückt werden
- Anweisungen des gleichen Blocks müssen mit der gleichen Anzahl des gleichen Typs Leerzeichen eingerückt sein

```
if a < 10:
  print(”bla”)
  a = a + 1
```


while

1. Der Ausdruck expr wird ausgewertet.
2. Trifft er zu, wird block ausgeführt. Danach gehe zu 1.
3. Sonst wird der Programmfluss hinter der Schleife fortgesetzt.

```
while expr:
  block
```

Größter gemeinsamer Teiler

- Der größte gemeinsame Teiler zweier ganzer Zahlen m und n ist die größte natürliche Zahl, durch die sowohl m als auch n ohne Rest teilbar sind.
- Euklidischen Algorithmus: in aufeinanderfolgenden Schritten wird jeweils eine Division mit Rest durchgeführt, wobei der Rest im nächsten Schritt zum neuen Divisor wird.
- Der Divisor, bei dem sich Rest 0 ergibt, ist der größte gemeinsame Teiler der Ausgangszahlen.

Größter gemeinsamer Teiler

- Beispiel: Berechnung des größten gemeinsamen Teilers von 1071 und 1029

```
1071 / 1029 = 1,  Rest: 42
1029 / 42   = 24, Rest: 21
  42 / 21   = 2,  Rest: 0
```

- Somit ist 21 der größte gemeinsame Teiler von 1071 und 1029.

Größter gemeinsamer Teiler in Python

- Die Variablen x und y enthalten die Eingabezahlen
- Am Ende der Berechnung enthält die Variable g den größten gemeinsamen Teiler von x und y.

```
g = y
while x > 0:
  g = x
  x = y % x
  y = g
```


`break & continue`

- Die `break`-Anweisung verlässt die aktuelle Schleife, ohne die Bedingung auszuwerten
- Die `continue`-Anweisung überspringt den Rest des aktuellen Durchlaufs, wertet die Bedingung neu aus und setzt ggf. die Schleife fort

while – else

- Schleifen können eine `else`-Anweisung haben.
- Die `else`-Anweisung wird ausgewertet sobald die Bedingung der Schleife zu Falsch auswertet, ...
- aber nicht, wenn die Schleife mit `break` abgebrochen wurde.

Beispiel: Primzahlen von 2 ... 100

```
n = 2
while n < 100:
   m = 2
   while m < n:
      if n % m == 0:
         break
      m += 1
   else:
      print(n, 'ist eine Primzahl')
   n += 1
```

Zusammenfassung

- Ausdrücke sind Konstrukte, die einen Wert haben.
- Werte haben Datentypen.
- Variablen sind Ausdrücke, denen Werte zugewiesen werden können.
- Mit dem if-Statement kann man zur Laufzeit entscheiden, welche Teile eines Programms ausgeführt werden sollen.
- Mit while-Schleifen kann eine Anweisung mehrere male ausgeführt werden.

Kommandozeilenargumente

- Python-Programmen können Kommandozeilenargumente übergeben werden.
- Auf diese kann man mit `sys.argv[i]` zugreifen; die Zählung beginnt bei 1; der Wert ist ein String.

```
# programm: echo.py
import sys
print(sys.argv[1])
```

- python echo.py bla blub ⇒ Ausgabe: bla