Programmierkurs Python I

Michaela Regneri & Stefan Thater
Universität des Saarlandes
FR 4.7 Allgemeine Linguistik (Computerlinguistik)

Winter 2010/11

Übersicht

- Variablen
- Datentypen
- Werte
- Ausdrücke
- Operatoren
- Kontrollstrukturen

Imperatives Programmieren

- Programme sind (Schritt-für-Schritt) Folgen von Anweisungen
- Anweisungen können Teilkomponenten (Ausdrücke) haben
  - `a = 1 + 2`
- Ausdrücke haben Werte
  - Werte können Variablen zugewiesen werden
- Kontrollstrukturen steuern den Programmablauf

„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
```

Variablen

Zuweisungen

Ausdrücke

Kontrollstrukturen

(Schleifen, Verzweigungen)

Variablen, Werte, Datentypen

- Werte in Python können verschiedene Datentypen haben:
  - Zahlen, Listen, Zeichenketten, …
- Variablen verweisen auf Werte
  - Variablen heißen auch Namen
- 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

Duck Typing

- *When I see a bird that walks like a duck and swims like a duck and quacks like a duck, I call that bird a duck*
- Häufig interessiert uns der konkrete Typ eines Wertes / einer Variablen nicht
- Wichtig ist vor allem, dass der Wert alle Operationen unterstützt, die wir auf den Wert anwenden wollen

Einige Datentypen

- Wahrheitswerte: `bool`
  - `bool` ist der Typ der beiden Konstanten `True` und `False`
- Zahlen: `int`, `float`, `complex`
- Zeichenketten: `str`, `bytes`
- Sammeltypen: `tuple`, `list`, `set`, `dict`
- […]

Ausdrücke

- Ausdrücke sind Konstrukte, die einen Wert haben
- Wir unterscheiden:
  - Literale = Audrücke, aus denen der Wert direkt abgelesen werden kann (Zahlen, Strings)
  - Variablen
  - Zusammengesetzte Ausdrücke: Ausdrücke mit Operatoren, Funktions- bzw. Methodenaufrufe

Ganze Zahlen (`int`)

- `int` (integers)
  - beliebig große ganze Zahlen (Python 3)
- Python 2
  - Wertebereich: -2^b, ..., $+2^{b-1}$, $b \geq 31$ (Systemabhängig)
  - Automatische Konvertierung von `int` nach `long`, wenn eine Zahl nicht als `int` repräsentiert werden kann.
  - `long` = beliebig große ganze Zahlen

Ganze Zahlen: Literale

- Zahlen wie 1, 17, -23 werden als Dezimalzahlen (Basis 10) interpretiert
  - Dezimalzahlen dürfen nicht mit `0` beginnen
- Das Präfix `0o` kennzeichnent Oktalzahlen (Basis 8)
  - `0o13` repräsentiert Wert 11
- Das Präfix `0x` kennzeichnet Hexadezimalzahl (Basis 16)
  - `0x1ca` repräsentiert 458
- Das Präfix `0b` kennzeichnet Binärzahlen (Basis 2)
  - `0b1101` repräsentiert 13

Fließkommazahlen (`float`)

- Fließkommazahlen werden als Dezimalzahlen repräsentiert
  - 12.34
- Wertebereich ist Systemabhängig
  - typischerweise: 53 Bit Präzision
- Achtung: Häufig können Dezimalzahlen intern nicht präzise dargestellt werden!

```
>>> 0.1
0.10000000000000001
>>> 0.1 * 4
0.40000000000000002
```


Operatoren

- Elementare (Rechen-) Operationen werden mit Operatoren dargestellt.
- Rechenoperatoren (Auswahl):
  - a + b
  - a - b
  - a * b
  - a / b, a // b (Division, ganzzahlige Division)
  - a % b (Rest bei ganzzahliger Division, „Modulo“)
  - a ** b (Potenz, a^b)
- Wenn a, b nicht den gleichen Typ haben, liefern die Operationen einen Wert vom allgemeineren Typ.

Präzedenz

- Ein Ausdruck 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: Punkt-vor-Strich-Regel
  - 2 * 3 + 4 = (2 * 3) + 4
- Bessere Lesbarkeit: manchmal ist es sinnvoll, Klammern auch dann zu verwenden, wenn sie redundant sind.
- Bei irrelevanter Präzedenz gilt das natürlich 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, ungleich)
- Das Resultat einer Vergleichsoperation ist ein Wahrheitswert (`bool`)

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


Wahrheitswerte

- `bool` ist der Typ der 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`

Wahrheitswerte

- Die Argumente der Operatoren `and` und `or` werden von links nach rechts ausgewertet.
- Kurzschluss-Auswertung:
  - die Auswertung bricht ab, sobald das Ergebnis feststeht
  - `False and x` ⇒ `False`
    - `x` wird nicht ausgewertet

String-Literale

- Strings sind Folgen von Zeichen (characters)
  - `'Das ist ein String.'`
  - `"Das auch."`
  - `"Er sagte \"Hallo\"."`
  - `'Er sagte "Hallo".'`
- Kein separater Typ für Zeichen
- Wenn Strings Umlaute enthalten (sollen):
  - Quelltextkodierung ist UTF-8, oder
  - Explizite Angabe der Kodierung in der ersten Zeile:
    `# -*- coding: latin-1 -*-`

String-Operatoren (Auswahl)

- Konkatenation:
  - `'Hallo' + 'Welt' ⇒ 'HalloWelt'`
- Zugriff auf einzelne Zeichen: wie mit Listen
  - `'Hallo'[0] ⇒ 'H'`
- 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`
  - `float(123) ⇒ 123.0`

Variablen

- Variablen sind „Platzhalter“ oder Namen für Werte
  - Man kann Variablen den Wert eines Ausdrucks zuweisen.
- Variablen können ausgewertet werden, um ihren Wert in einem anderen Ausdruck zu verwenden.
  - `print(zahl)` gibt den Wert eines Ausdrucks (`zahl`) aus

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


Variablen

- Variablennamen (allgemeiner: alle Bezeichner) müssen mit einem Buchstaben oder „_“ beginnen.
  - Die restlichen Zeichen dürfen auch Ziffern enthalten.
  - Sonderzeichen (Umlaute etc.) sind erlaubt (Python3)
- Der Name darf kein Schlüsselwort (`if`, `while`, ...) sein
- Groß/Kleinschreibung wird unterschieden
- Beispiele:
  - `Foo`, `foo12`, `_Foo` (ok)
  - `12foo`, `if` (Fehler)

Zuweisungen

- `var = expr`
  - Der Ausdruck `expr` wird ausgewertet
  - Dann wird der Wert der Variablen `var` zugewiesen
- var_1 `=` var_2 `= ... =` `expr`
  - Allen var_i wird der Wert von `expr` zugewiesen.
- var_1`, ...,` var_n `=` $expr_1$`, ...,` $expr_n$
  - Die $expr_i$ werden ausgewertet
  - Dann werden die entsprechenden Werte den Variablen var_i zugewiesen
  - Beispiel: `a, b = b, a` (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)

- Python-Programm sind Folgen von Anweisungen
- Bisher kennen wir: Zuweisungen
  - `print` ist keine Anweisung, sondern eine Funktion
- Eine Anweisung entspricht einem Schritt im Algorithmus
- Anweisungen werden durch Zeilen getrennt
  - pro Zeile steht (normalerweise) genau eine Anweisung
- Man kann auch mehrere (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 $expr_1$ zu wahr auswertet, wird $anweisung_1$ ausgeführt.
- Ansonsten wird $anweisung_2$ 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

- Die Ausdrücke $expr_i$ 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
- Leerzeichen:
  - Space, Tabulator

```
...
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
  - Der Rest wird im nächsten Schritt zum neuen Divisor
  - 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
```

- Ergebnis: 21

Größter gemeinsamer Teiler in Python

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

```
x = <eingabe>
y = <eingabe>
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 Blocks
  - wertet die Bedingung neu aus
  - und setzt ggf. die Schleife fort.

while – break – else

- Schleifen können einen `else` Block haben.
- Der `else` Block wird ausgewertet sobald die Bedingung der Schleife zu Falsch auswertet, ...
- aber nicht, wenn die Schleife mit `break` verlassen wurde.

```
while bedingung:
   ...
   if irgendwas:
     break
   ...
else:
   ...
```

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 Typen.
- 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