Abspann

Simon Clematide
simon.clematide@uzh.ch

Institut für Computerlinguistik
Universität Zürich

Programmiertechniken in die Computerlinguistik I

Danke an Rico Sennrich bzw. Thorsten Marek für Folienvorlagen.

Übersicht

Lernziele

- Verstehen von Zuweisung, Binding und Namen
- Verstehen der Parameterübergabe bei Funktionen und in for-Schlaufen
- Hohe Kunst des Sortierens bei Listen und Dictionaries

Namen, Zuweisung, Bindung und Objekte

Zuweisung (*Assignment*)

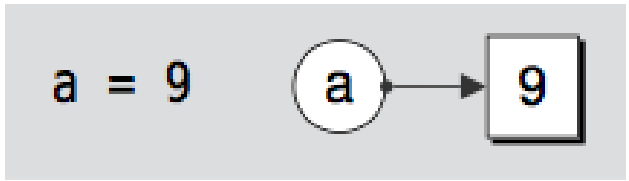
```
a = 5*8
```

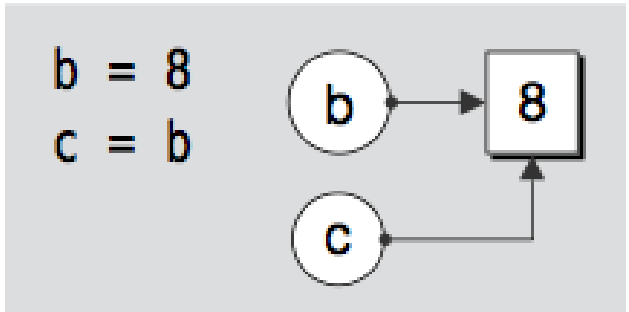
Was passiert beim Verarbeiten der Zuweisungsanweisung?

1. Evaluiere (*evaluate*) RHS-Ausdruck `5*8` zu einem Ganzzahl-Objekt. (RHS=*right-hand side*)
2. Binde (*binding*) das evaluierte Ganzzahl-Objekt an den Namen `a`.

Namen referieren auf Objekte

1. Rechteck = Objekte
2. Kreis = Referenz auf Objekt





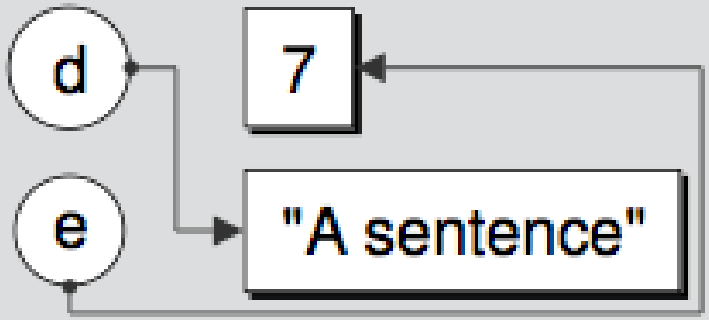
Quelle: [Summerfield 2008, 14]

Was passiert beim Statement `c = b`?

Mehrfaches Binden eines Namens (*rebinding*)

Was passiert, wenn derselbe Namen mehrfach zugewiesen wird? ▸1

```
d = 7
e = d
d = "A sentence"
```



Unreferenzierte Objekte und Müllsammlung

```
s = "Ein String"                    # s macht "Ein String" zugänglich
                                    # im nachfolgenden Programm.
s += " wird zusammengesetzt!" # Nach dieser Anweisung ist
                                    # "Ein String" nicht mehr
                                    # zugänglich via Name s.
print s                             # s referenziert ein neues Objekt.
Ein String wird zusammengesetzt!
```

- Unbenannte Objekte sind ausserhalb des Ausdrucks, in dem sie vorkommen, nicht mehr zugänglich und benutzbar: Sie sind Datenmüll (*garbage*).
- Nicht mehr zugängliche Objekte können periodisch gesammelt und entsorgt werden (*garbage collection*). Dadurch wird Speicherplatz frei.

Python weiss für jedes Objekt, wie viele Referenzen (Namen) darauf existieren. Das Modul `gc` ist eine Schnittstelle zur *garbage collection*.

Identität eines Objekts (`id()`) und *mutable data*

Identität bei veränderlichen Datenstrukturen

Verschiedene Namen können auf dasselbe Objekt referenzieren. Die eingebaute Funktion `id()` identifiziert jedes Objekt über eine Ganzzahl (entspricht ungefähr seiner Speicheradresse). Python garantiert, dass 2 verschiedene Objekte gleichzeitig nie dieselbe ID haben.

(Re-)Binding einer Variable

```
>>> l = ['a']
>>> id(l)
4300400112
>>> l = ['a']
>>> id(l)
4299634664
```

Weshalb?

Veränderliche Datenstrukturen

```
>>> l = ['a']
>>> id(l)
4300400112
>>> l[0] = 'b'
>>> id(l)
4300400112
```

Weshalb?

Binding in *for*-Konstrukten

Identität vs. Wertgleichheit (*equality*, Äquivalenz)

- `o1 == o2` testet, ob 2 Objekte/Variablen denselben Wert haben
- `o1 is o2` testet, ob 2 Objekte/Variablen dieselbe Identität haben, d.h. identisch sind, d.h. `id(o1) == id(o2)`

Was wird hier ausgegeben? ▸2

```
>>> l = [('der',1200),('die',1000),('das',900)]
>>> for i,e in enumerate(l):
        print e is l[i]
True
True
True
```

In for-Konstrukten werden bestehende Objekte an neue Namen gebunden!

Binding bei Funktionsparametern

Beim Funktionsaufruf werden die Parameternamen an die übergebenen Objekte gebunden (*binding*).

Was wird hier ausgegeben? ▸3

```
global_list = [('der',1200),('die',1000),('das',900)]

def del_first(l):
    print 'global_list is parameter l:', global_list is l
    del l[0]

del_first(global_list)
print global_list
```

Kopieren von Listen

Eine spannende Verbindung: Binding und Listen ▸4

Wie spielen Zuweisung von Listen-Namen und Veränderbarkeit zusammen?

Kopieren oder Binding?

```
l1 = [('der',1200),('die',1000)]
# Binding
l2 = l1
# Kopieren via Slicing
l3 = l1[:]
# Welche Listen werden modifiziert?
l1[0] = ('der',1201)
```

Kopieren via allgemeinem Modul zum Kopieren von Objekten

```
import copy
l4 = copy.copy(l1)
```

Sortieren und maximieren

Ordnung erzeugen bei Dictionaries

- `min()`, `max()`, `in`, `sorted()` etc. operieren über Schlüsseln.
- `dict.values()` ist Liste aller Werte.
- Höchster Schlüssel: `max(d)`
- Höchster Wert: `max(d.values())`
- Schlüssel mit höchstem Wert: `max(d,key=d.get)`
- Nach Schlüssel sortieren: `sorted(d)`
- Nach Werten sortieren: `sorted(d,key=d.get)`
- Umgekehrt nach Werten sortieren:
  `sorted(d, key=d.get, reverse=True)`

Vertiefung

- Pflichtlektüre: Kapitel 4.1 bis 4.2 aus NLTK-Buch

Liste der verlinkten Programme und Ressourcen I

Folie

Literaturangaben I

- Summerfield, Mark (2008).

  Rapid GUI programming with Python and Qt: the definitive guide to PyQt programming. Prentice Hall, Upper Saddle River, NJ.