Programmeermethoden NA

Week 6: Lijsten

Kristian Rietveld

http://liacs.leidenuniv.nl/~rietveldkfd/courses/prna2016/

Universiteit Leiden
The Netherlands

Getal opbouwen

Stel je leest losse karakters (waaronder cijfers) en je moet daar een getal van maken? Hoe doe je dat?

Hint: decimaal getalstelsel.

```
if kar >= "0" and kar <= "9":
    getal = (getal * 10) + (ord(kar) - ord("0"))
else:
    # ...
    pass
```

```
qwerty7392abc
```

`getal` is 73 en `kar` is `"9"`.

Cijfer voor cijfer

Gevraagd: zet de cijfers van een getal op aparte regels, het laatste cijfer eerst:

```
def cijfers(getal):
    """Zet alle cijfers van getal op aparte
    regels op het scherm, laatste cijfer eerst."""

    while getal != 0:
        print getal % 10
        getal /= 10
```

Ontbinden

Ontbind een getal in factoren, bijv.

```
84 = (2 ** 2) * (3 ** 1) * (7 ** 1):
```

```
def ontbinden(getal):
    """Ontbind getal in factoren."""
    teller = 0  # hoe vaak past deler in getal?
    deler = 2   # kandidaatdeler
    while getal != 1:
        if getal % deler == 0:
            teller = 0
            while getal % deler == 0:
                getal /= deler
                teller += 1
            print "{0} ** {1}".format(deler, teller)
        deler += 1
```

Op papier, opgave 3

```
def f(x, y):
    x -= 1
    return x * y

def g(a, b):
    x = 3
    b += x
    a -= 1
    a = f(a, b) + f(a, a)
    print x, a, b,
    return a + x - 2
```

a. Neem aan globale variabelen x en y bij binnenkomst g 6 resp. 16. Wat gebeurt er bij `print g(x, y), x, y`?

Op papier, opgave 3

```
def f(x, y):
    x -= 1
    return x * y

def g(a, b):
    x = 3
    b += x
    a -= 1
    a = f(a, b) + f(a, a)
    print x, a, b,
    return a + x - 2
```

b. Geef een eenvoudige functie `def G(a, b)` die `g(a, b)` uitrekent.

Op papier, opgave 3

```
def f(x, y):
    x -= 1
    return x * y

def g(a, b):
    global x    # <--
    x = 3
    b += x
    a -= 1
    a = f(a, b) + f(a, a)
    print x, a, b,
    return a + x - 2
```

c. We voegen aan `g` als eerste regel toe `global x`. Beantwoord opnieuw a. Verandert de uitvoer als `g` meerdere keren wordt aangeroepen?

Lijsten

- Een *lijst* is een geordend rijtje van variabelen.

```
a = [1.0, 53, False, "hallo"]
```

- We kunnen de elementen individueel uitlezen: `a[0]`, `a[1]`, `a[2]`, `a[3]`.
- `a[3]` is een element van de lijst en `3` is de index of subscript.

Lijsten initialiseren

- Verschillende elementen:

```
a = [11, 12, 13, 33, 44, 55, 66]
```

- Middels een functie-aanroep:

```
b = range(10, 110, 10)
```

- *n*-keer hetzelfde element:

```
c = [0] * 10
d = [0 for i in range(10)]
e = [i ** 2 for i in range (10)]
```

- Lege lijst (lengte 0):

```
f = []
g = list()
```

Lege lijsten

Let op dat je een lege lijst niet zomaar kunt indexeren!

```
>>> a = []
>>> a[4] = "test!"
Traceback (most recent call last):
File "<stdin>", line 1, in <module> IndexError: list
assignment index out of range
```

Het toevoegen van elementen moet *expliciet* gebeuren.

Lijsten manipuleren

- `append(obj)`: voeg `obj` toe achteraan de lijst.
- `insert(idx, obj)`: zet `obj` *voor* plek `idx`.

```
>>> a = []
>>> a.append("een")
>>> a.append("twee")
>>> a.append("drie")
>>> a.insert(0, "nul")
>>> a
["nul", "een", "twee", "drie"]
```

Lijsten manipuleren (2)

- `remove(obj)`: haal `obj` uit de lijst.
- `pop()`: geef en verwijder laatste element.

```
>>> a
["nul", "een", "twee", "drie"]
>>> a.remove("een")
>>> a.pop()
"drie"
>>> a
["nul", "twee"]
>>> a.pop()
"twee"
```

Lijsten manipuleren (3)

- `pop(idx)`: geef en verwijder het element op index `idx`.
- `del lijst[i]`: verwijder `lijst[i]`.
- `del lijst[i:i+10]`: verwijder `lijst[i:i+10]`.

```
>>> b = range(10, 15)  # 10, 11, 12, 13, 14
>>> b.pop(2)  # 10, 11, 13, 14
12
>>> del b[1]  # 10, 13, 14
>>> del b[2]  # 10, 13
>>> b
[10, 13]
```

Lijsten kopieren

Beschouw:

```
>>> a = [1, 2, 3, 4]
>>> b = a
>>> a.remove(2)
>>> a
[1, 3, 4]
```

Vraag: wat is de waarde van b?

Lijsten kopieren (2)

```
>>> a = [1, 2, 3, 4]
>>> b = a
>>> a.remove(2)
>>> a
[1, 3, 4]
>>> b
[1, 3, 4]
```

Belangrijk: Toekenning maakt **geen** kopie! Door `b = a` gaat b naar dezelfde lijst refereren als `a`!

Lijsten kopieren (3)

```
>>> b
[1, 3, 4]
>>> b.remove(4)
>>> b
[1, 3]
>>> a
[1, 3]
```

Lijsten kopieren (4)

Hoe bereiken we wel dat we een kopie krijgen? Maak een nieuwe lijst, initialiseer deze met een andere lijst.

```
>>> a = [1, 2, 3, 4]
>>> b = list(a)
>>> a.remove(2)
>>> a
[1, 3, 4]
>>> b
[1, 2, 3, 4]
```

Gebruik lijsten

Ga uit van een lijst bestaande uit integers. Vermenigvuldig elk element met 5.

```
a = range(10, 110, 10)
for i in range(len(a)):
    a[i] = a[i] * 5
```

Vraag 1: waarom werkt het volgende niet?

```
a = range(10, 110, 10)
for el in a:
    el = el * 5
```

Gebruik lijsten (2)

Ga uit van een lijst bestaande uit integers. Vermenigvuldig elk element met 5.

```
a = range(10, 110, 10)
for i in range(len(a)):
    a[i] = a[i] * 5
```

Vraag 2: waarom werkt het volgende niet?

```
a = range(10, 110, 10)
a = a * 5
```

Lijsten en functies

Lijsten kun je zonder problemen doorgeven als functie-argument:

```
def sommeer(lijst):
    som = 0
    for l in lijst:
        som += l
    return som

reeks = range(10, 110, 10)
s = sommeer(reeks)
```

LET OP: de lokale variabele `lijst` wijst naar dezelfde lijst als `reeks`. Er wordt **geen** kopie van de lijst gemaakt. Veranderingen gemaakt in `lijst` zijn zichtbaar in `reeks`.

Minimum

Ga weer uit van een lijst bestaande uit integers. Wat is de minimum-waarde in een lijst?

```
def minimum(lijst):
    klein = lijst[0]
    for el in lijst:
        if el < klein:
            klein = el
    return klein

# Test
a = [47, 54, 52, 35, 84, 69, 99, 77, 48, 6,
     46, 75, 29, 67, 63, 13, 30, 41, 86, 97]
print minimum(a)
```

Lineair zoeken

```
def lineairzoeken(lijst, getal):
    """Zoek getal in lijst volgens methode
    van lineair zoeken. Returnwaarde:
    index waar getal is gevonden, anders -1"""
    index = 0
    gevonden = False
    while not gevonden and (index < len(lijst)):
        if getal == lijst[index]:
            gevonden = True
        else:
            index += 1
    if gevonden:
        return index
    else:
        return -1
```

Slicing

Bij strings maakten we al gebruik van slicing om substrings te isoleren. Slicing kent ook een stapgrootte. De volledige notatie is:

```
start : eind : stap
```

Spelregels:

- Eind-index telt niet mee.
- Elk van de delen mag worden weggelaten.
- Bij lijsten mag je ook toekenningen doen aan de slice (bij strings niet).

Slicing (2)

```
>>> a = range(10, 110, 10)
>>> a[2:5]
[30, 40, 50]
>>> a[2:]
[30, 40, 50, 60, 70, 80, 90, 100]
>>> a[:5]
[10, 20, 30, 40, 50]
>>> a[2:8:2]
[30, 50, 70]
>>> a[::2]
[10, 30, 50, 70, 90]
>>> a[::3]
[10, 40, 70, 100]
```

Slicing (3)

```
>>> a = range(10)
>>> a[0:5] = ["a", "b", "c", "d", "e"]
>>> a
["a", "b", "c", "d", "e", 5, 6, 7, 8, 9]
>>> a[5:5] = ["x", "y", "z"]
>>> a
["a", "b", "c", "d", "e", "x", "y", "z", 5, 6, 7, 8, 9]
>>> a[10:] = range(100, 110)
>>> a
["a", "b", "c", "d", "e", "x", "y", "z", 5, 6, 100, 101,
102, 103, 104, 105,106, 107, 108, 109]
>>> a[0:10] = []
>>> a
[100, 101, 102, 103, 104, 105, 106, 107, 108, 109]
>>> a[:] = []
>>> a
[]
```

Andere operaties op lijsten

- `x in lijst` operator: zit `x` in `lijst`?
- `lijst.count(obj)`: hoe vaak komt `obj` in de lijst voor?
- `lijst.index(obj)`: op welke index kunnen we `obj` vinden? (Lineair zoeken).

Oh ja, sommeer en minimum? Ook ingebouwd: `sum(lijst)`, `min(lijst)`.

Sorteren (simpelsort)

Hoe sorteer je de elementen van een lijst oplopend? Een eerste idee is om herhaaldelijk het kleinste element vooraan te zetten.

```
def simpelsort(lijst):
    for voorste in range(len(lijst)):
        # Zoek kleine element in ongesorteerde stuk [i:]
        plaatskleinste = voorste
        kleinste = lijst[voorste]

        for k in range(voorste + 1, len(lijst)):
            if lijst[k] < kleinste:
                kleinste = lijst[k]
                plaatskleinste = k

        if plaatskleinste > voorste:
            # Wissel om
            lijst[plaatskleinste], lijst[voorste] \
                = lijst[voorste], lijst[plaatskleinste]

## Test
l = [47, 10, 7, 3, 31, 75, 18, 21, 48, 79]
simpelsort(l)
print l
```

Sorteren (simpelsort)

Een voorbeeld van de werking van simpelsort:

```
0  1  2  3  4  5  6   (len(lijst) = 7)

3  8  7  5  2  4  9
2| 8  7  5  3  4  9
2  3| 7  5  8  4  9
2  3  4| 5  8  7  9
2  3  4  5| 8  7  9
2  3  4  5  7| 8  9
2  3  4  5  7  8| 9
2  3  4  5  7  8  9|
```

Sorteren (bubblesort)

Nog een sorteermethode:

```
def bubblesort(lijst):
    for i in range(1, len(lijst)):
        for j in range(0, len(lijst) - i):
            if lijst[j] > lijst[j + 1]:
                lijst[j], lijst[j + 1] = \
                    lijst[j + 1], lijst[j]
```

Sorteren (bubblesort)

Simpelsort (links) vs. bubblesort (rechts)

0	1	2	3	4	5	6
3	8	7	5	2	4	9
2 |	8	7	5	3	4	9
2	3 |	7	5	8	4	9
2	3	4 |	5	8	7	9
2	3	4	5 |	8	7	9
2	3	4	5	7 |	8	9
2	3	4	5	7	8 |	9
2	3	4	5	7	8	9 |

0	1	2	3	4	5	6
3	8	7	5	2	4	9
3	7	5	2	4	8	|9
3	5	2	4	7	|8	9
3	2	4	5	|7	8	9
2	3	4	|5	7	8	9
2	3	|4	5	7	8	9
2	|3	4	5	7	8	9

Analyse

Bubblesort doet bij een rij met n elementen

$$(n - 1) + (n - 2) + \ldots + 3 + 2 + 1 = n(n - 1)/2$$

vergelijkingen tussen elementen. Het is een O(n^2) algoritme, en dat is niet zo fijn.

Dezelfde analyse geldt voor "simpelsort" (selection sort).

Sorteren en zoeken kan veel sneller (= beter)! Zie bijv. http://www.sorting-algorithms.com/

(en The Sound of Sorting?)

Lijsten nesten

- Een element van een lijst mag elk type variabele zijn, dus ook weer een lijst. Hiermee kun je lijsten van lijsten maken.

- Je kunt dan over meerdere niveau's indexeren:

```
a[i][j][k]
```

Lijsten nesten (2)

```
>>> a = [[1, 2, 3, 4, 5], ["a", "b", "c"], [], ["x"]]
>>> for lijst in a:
...     print len(lijst),
...
5 3 0 1
>>> a[0][1]
2
>>> a[1][2]
'c'
```

Lijsten nesten (3)

```
>>> a = [[1, 2, 3, 4, 5], ["a", "b", "c"], [], ["x"]]
>>> a[2][4]
Traceback (most recent call last):
File "<stdin>", line 1, in <module>
IndexError: list index out of range
>>> b = [[1, 2, 3], 591243, ["a", "b", "c"]]
>>> b[1][3]
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
TypeError: 'int' object has no attribute '__getitem__'
```

Tuples

- Een tuple is een geordende reeks.
- Tuples kunnen **niet** worden veranderd.
- Nesten, indexing, slicing mogelijk!

```
a = (1, 2, 3, 'a', 'b', 'c')
a[4]
a[4:8]
b = (a, 4, ('q', 'z'), 6)
```

Dictionaries

- Soms is het handig om een 'lijst' te indexeren met iets anders dan een geheel getal.
- Dit kan in Python met een 'dictionary'.
- Vaak bekend als: associatieve array of hash table.

Dictionaries (2)

```
>>> d = dict()
>>> d["walter"] = "071-5270000"
>>> d["kris"] = "06-12345678"
>>> d["joop"] = "0123-524513"
# Value ophalen uit de dictionary
# aan de hand van een key
>>> d["kris"]
"06-12345678"
```

Dictionaries (3)

```
>>> k = {}
>>> k[4,3] = "rood" # We maken hier gebruik van een tuple!
>>> k[1,2] = "blauw"
>>> k[9,4] = "zwart"
>>> len(k)
3
>>> k
{(1, 2): "blauw", (9, 4): "zwart", (4, 3): "rood"}
>>> k[5,2]
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
KeyError: (5, 2)
>>> k[1,2]
"blauw"
```

Tot slot

- Uiteraard weer werkcollege.
  - Uitwerking van oefening “op papier” van vorige week te vinden op de website van het vak.
- De eerste opdracht is nagekeken, vraag naar het nagekeken werk bij het werkcollege.
- Denk aan die tweede programmeeropdracht, de deadline is al volgende week!