Programmeren voor Taal en Tekst 2011: Python

Erik Tjong Kim Sang

16 mei 2011

Onderwerpen

- Week 1: basisdatastructuren en controlstatements (`for`, `if`, `while`)
- Week 2: verwerking van strings en reguliere expressies (`sub`, `search`)
- **Deze week**: complexe datastructuren (lijsten) en functies

Aanvulling op de vorige week (1)

Lezen van regels uit een bestand:

```
from sys import stdin       # laad functie stdin uit pakket sys
for regel in stdin:         # lees steeds 1 regel van standard input
   regel = regel.rstrip()   # verwijder newline van regel
   print(regel)             # print de regel
```

Dit programma verwerkt de regels van een bestand als het wordt aangeroepen als: `python3 prog.py < bestand`

Let op: de instructie `stdin` geeft wel een newline terug in de string
Deze kan worden verwijderd met het commando `rstrip()`

Aanvulling op de vorige week (2)

Het teken \ (backslash) wordt in Python gebruikt als startteken (escape) voor bijzondere tekens in strings:

```
>>> print("\110\305\n")
HÅ
```

Door een "r" voor de string te plaatsen, geef je aan dat backslashes in de string niet als escape moeten worden beschouwd:

```
>>> print(r"\110\305\n")
\110\305\n
```

De extra "r" staat voor raw

Feedback op opdrachten week 2

- kies betekenisvolle variabelenamen
- gebruik constantedefinities voor getallen
- combineer herhaalde code in een loop met `while` of `for`
- test de programma's ook op de practicummachines

(inloggen van thuis: zie mededelingen op Nestor)

COMPLEXE DATASTRUCTUREN

Lijsten en woordenboeken

Een lijst is een geördende verzameling:

```
lijst = ['a','b','xyz',11,37]
```

Elementen van een lijst kunnen worden aangewezen met een indexwaarde:

```
print(lijst[0])   # levert op: 'a' (eerste element)
print(lijst[3])   # levert op: 11 (vierde element)
print(lijst[-1])  # levert op: 37 (laatste element)
print(lijst[0:2]) # levert op: ['a','b'] (1e tot 3e element)

lijst[1] = 'd'    # verander 1 element van de lijst
```


Het verwerken van lijstelementen

Vaak moet een programma een taak uitvoeren met elk element van een lijst. Hier zijn twee voorbeelden van hoe dat kan:

```
# doe iets met alle elementen van een lijst
for element in lijst):
   print(element)

# als boven maar dan met gebruik van de indexwaarden
for index in range(len(lijst)):
   print(index,lijst[index])
```

waarbij `range(len(lijst))` gelijk is aan `[0,1,2,...,len(lijst)-1]`

Operaties op lijsten

```
>>> lijst = ['a','b','c']
>>> lijst+lijst    # combineren
['a', 'b', 'c', 'a', 'b', 'c']
>>> lijst*3        # kopieren
['a', 'b', 'c', 'a', 'b', 'c', 'a', 'b', 'c']
>>> 'a' in lijst   # testen
True
```

`lijst.append('a')`	voeg element 'a' toe aan het einde van de lijst
`lijst.pop(n)`	verwijder n-de element uit lijst (geteld vanaf 0)
`lijst.insert(p,'a')`	plaats element 'a' op positie p (geteld vanaf 0)
`lijst.sort()`	sorteer de lijst

Strings zijn lijsten met tekens

```
>>> string = 'dit is een test'
>>> string[0]      # geef het eerste teken van de string
'd'
>>> string[4:6]    # geef tekens 4 tot 6 van de string
'is'
```

Met het commando `split()` zet je een string om in een lijst van deelstrings, gegeven een scheidingssymbool:

```
>>> string.split()    # verdeel strings in deelstrings met ' '+
['dit', 'is', 'een', 'test']
>>> string.split('e') # idem met scheidingssymbool 'e'
['dit is ', '', 'n t', 'st']
```


Andere stringfuncties

`lstrip(), rstrip(), strip()`: verwijder spaties, tabs, newline van het begin (l), einde (r) of op beide plekken in de string

`find()`: geef de index van de eerste positie waarop een substring in een string zit (resultaat bij niet gevonden: -1)

`count()`: tel hoe vaak een substring in een string zit

Voorbeelden:

```
>>> string.find('een')
11
>>> string.count('e')
3
```


Gebruik van lijsten: verzamelen en verwerken

Vaak gebruik je lijsten om elementen in op te slaan en daarna alle elementen in 1 keer te verwerken:

```
lijst = []
for ...:
   if ...:
      lijst.append('a') # voeg element toe aan einde van lijst

for e in lijst:
   print(e) # verwerk element uit lijst
```


Gebruik van lijsten: stacks

Een stack is een wachtrij. Er zijn twee varianten:

First-in, First-out (FIFO):

```
lijst.append('a')       # voeg element toe aan einde van lijst
lijst.append('b')       # voeg element toe aan einde van lijst
e = print(lijst.pop(0)) # verwijder eerste element van lijst
```

Last In, First Out (LIFO):

```
lijst.append('a')       # voeg element toe aan einde van lijst
lijst.append('b')       # voeg element toe aan einde van lijst
e = print(lijst.pop())  # verwijder laatste element van lijst
```


Woordenboeken (dictionaries)

Voor een frequentielijst wil je strings/woorden koppelen aan getallen

In Python kan hiervoor de datastructuur dictionary worden gebruikt: een lijst die strings gebruikt als indices:

```
dict = {}                  # defineer een lege dictionary
dict['woord'] = frequentie # plaats een element in dictionary
dict = { 'de':500, 'het':350, 'een':290 }    # vul dictionary
```

(de, het, een) zijn sleutels/attributen; (500, 350, 290) zijn waarden

Functies voor woordenboeken

`dict.keys()`	geeft lijst met sleutels van dict
`dict.values()`	geeft lijst met waarden van dict
`del dict[s]`	verwijdert element van dict met sleutel s
`s in dict`	test of element met sleutel s in dict zit

Voorbeeld van gebruik:

```
dict = { 'a':1, 'b':2 }
for s in dict.keys():   # voor iedere sleutel in dict
   print(s,dict[s])     # verwerk het bewuste element van dict
```


Bouwen van een frequentielijst

In een woordenboek kan je woordfrequenties opslaan:
gebruik de woorden als sleutels en plaats de frequenties in de waarden

```
freq = {}             # begin met lege frequentielijst
for w in alleWoorden: # voor ieder woord:
   if not w in freq:  # als het woord niet in frequentielijst:
      freq[w] = 1     # eerste keer dat we woord tegenkomen
   else:              # anders: het woord is eerder gezien:
      freq[w] += 1    # verhoog de frequentie met 1
```

Resultaat: `freq` bevat de woorden met de bijbehorende frequenties

FUNCTIES

Functies

Een functie of subroutine is een stuk code met een naam

De naam maakt het mogelijk om de functie vanaf verschillende plekken in het programma aan te roepen en uit te laten voeren

Voorbeeld:

```
>>> def hoi(naam): print('Hoi ',naam,'!',sep='')
...
>>> hoi('Erik')
Hoi Erik!
```


Communiceren met functies

Functies hebben parameters met namen die je in de functie als variabelen kan gebruiken (zie vorige slide met parameter `naam`)

In de functie kunnen met het commando `return()` waarden worden teruggegeven aan de plek van de functieaanroep:

```
def fibonacci(getal1,getal2):
   getal3 = getal1 + getal2
   return(getal2,getal3)
```

Deze functie heeft twee parameters (`getal1, getal2`) en geeft twee getallen terug aan het aanroepende niveau (`getal2, getal3`)

De hoofdfunctie `main()`

De belangrijkste code van een Pythonprogramma wordt meestal in een functie `main()` geplaatst:

```
# definitie van hoofdtaak
def main():
   print('dit programma voert een belangrijke taak uit')

main()   # voer hoofdtaak uit
```

Dit heeft voordelen bij het uitvoeren van het programma via de interactieve Pythonshell: het kan na het laden nogmaals worden aangeroepen als `bestand.main()`

Het verwerken van programma-argumenten

Het is handig om de werking van een programma te kunnen beïnvloeden door programma-argumenten op te geven bij de aanroep:

```
$ python3 telop.py 2 3
```

Deze argumenten worden opgeslagen in de lijst `argv` van het pakket `sys`:

```
from sys import argv          # maak argv uit sys beschikbaar

def main(argv): print(argv)   # hoofdfunctie: toon argumenten
                              # noot: laat ook aanroep zien
main(argv)                    # roep hoofdfunctie aan
```


Scope van variabelen

Variabelen zijn alleen bekend in de functie waarin ze zijn gedefinieerd

Pythonvariabelen van het hoofdprogramma zijn overal bekend

```
def main():
   i = 1              # lokale variabele
   print(i)           # resultaat: 1
   test()

def test(): print(i) # resultaat: 2

i = 2                 # variabele uit het hoofdprogramma
main()
```


PROGRAMMEERVOORBEELD

Taak

Schrijf een programma dat 1 of meer woorden accepteert als programma-argument en vervolgens de relatieve frequentie van deze woorden berekent op basis van een tekst die op de standaardinvoer wordt aangeboden

Plan

```
# lees regel en herhaal voor elke regel
   # tokeniseer regel
   # herhaal voor elk woord op de regel
      # verhoog totaal aantal woorden
      # herhaal voor elk gevraagd woord
         # als woord van regel == gevraagd woord
            # verhoog teller
# voor elk gevraagd woord
   # laat aantal getelde woorden zien
```


Uitwerking (1)

```
def main(gezochteWoorden): # definitie hoofdfunctie
   # lees regel en herhaal voor elke regel
   for regel in stdin:
      # tokeniseer regel
      woorden = tokenize(regel)
      # herhaal voor elk woord op de regel
      for woord in woorden:
         # verhoog totaal aantal woorden
         aantalWoorden += 1
         ...
```


Uitwerking (2)

```
         # herhaal voor elk gevraagd woord
         for gezocht in gezochteWoorden:
            # als woord van regel == gevraagd woord
            if woord == gezocht:
               # verhoog teller
               if woord in teller: teller[woord] += 1
               else: teller[woord] = 1

   # voor elk gevraagd woord
   for gezocht in gezochteWoorden:
      # laat aantal getelde woorden zien
      if not gezocht in teller: print(0,gezocht)
      else: print(teller[gezocht]/aantalWoorden,gezocht)
```


Initialisatie en subroutine

```
from re import sub # laad benodigde functies
from sys import argv, stdin

def tokenize(regel): # definitie functie tokenize()
   regel = sub("([^a-zA-Z0-9 ]+)",r" \1 ",regel)
   return(regel.split())

def main(gezochteWoorden): # definitie hoofdfunctie
   teller = {}
   aantalWoorden = 0
   ...

main(argv[1:]) # aanroep hoofdfunctie, verwijder programmanaam
```


Literatuur

John Zelle, *Python Programming*. Franklin, Beedle & Associates, tweede editie, 2010, secties 1.6, 5.1, 5.2, 5.3, 5.5, 5.6, 6.3, 6.5, 11.2, 11.6, 11.7

Python Software Foundation, *The Python Standard Library*. Versie 3.2, 2011, `http://docs.python.org/py3k/library/re.html`

THE END