Programmeermethoden NA

Week 3: Controlestructuren

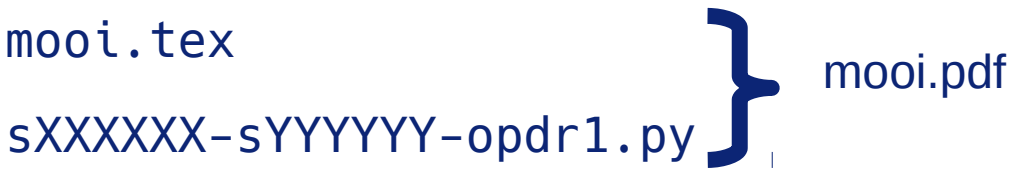
Kristian Rietveld

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

Universiteit
Leiden

Inleveren opdracht 1

- Lever digitaal “`sXXXXXXX-sYYYYYYY-opdr1.py`” en “`sXXXXXXX-sYYYYYYY-opdr1.pdf`” in via Blackboard.
  - Vermeld ook studentnummers en namen bij het inleveren.
- Een print van het verslag geschreven in LaTeX, zie de uitgebreide uitleg bij het derde werkcollege.



Controlestructuren

- Met behulp van controlestructuren kunnen we het verloop van een programma beïnvloeden.
- De belangrijkste controlestructuren in Python zijn:
  - *if* voor het maken van keuzes.
  - *for* voor een vast aantal herhalingen.
  - *while* voor een onbekend aantal herhalingen.

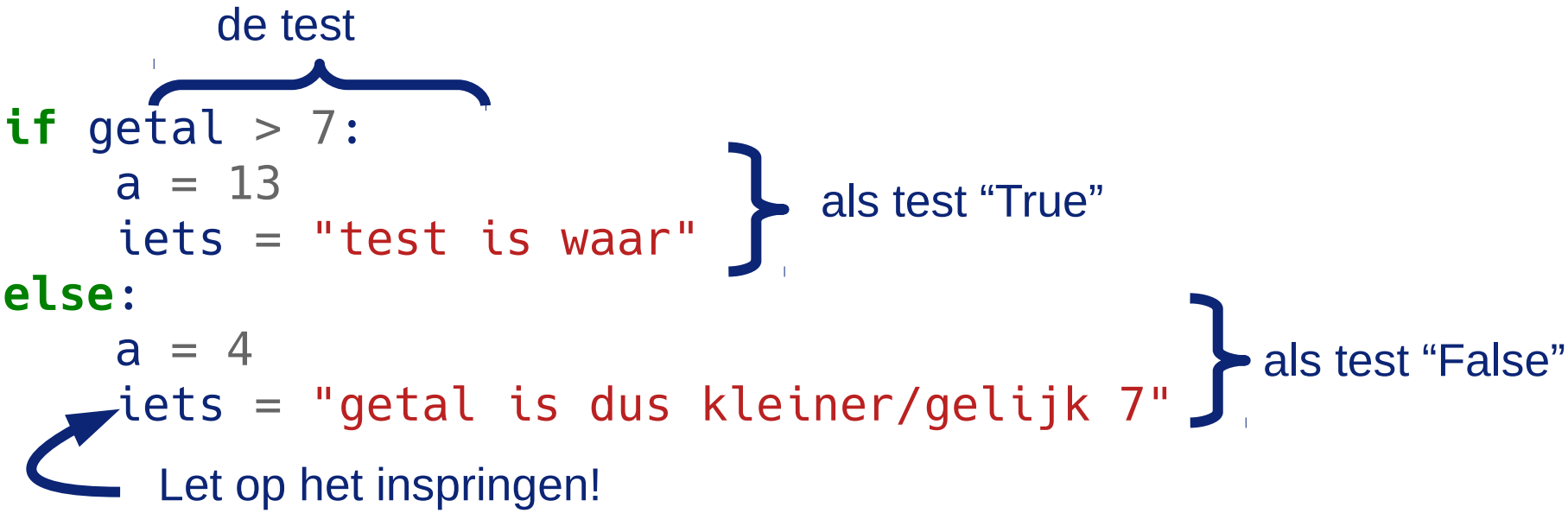
- Iets als goto's of labels bestaat in Python niet!

if-statement

- Een if-statement gaat informeel als volgt:

```
als een of andere test:
    de test is waar: doe zus en zo
anders:
    de test is niet waar: dit en dat
```

if-statement (2)



- In de tests gebruiken we predicaten als == (gelijk), != (ongelijk), < (kleiner dan), > (groter dan), <= (kleiner dan of gelijk aan) en >= (groter dan of gelijk aan).
- Uitgebreidere tests kunnen worden gemaakt met Boolese expressies, zie vorige week.

if-statement (3)

- Als je meerdere 'gevallen' in 1 if-statement wilt testen, maak je gebruik van “elif” (kort voor “else if”).

```
if test > 7:
    a = 13
    iets = "test is waar"
elif test < 7:
    a = 10
    iets = "we kwamen langs else if"
else:
    a = 4
    iets = "test is dus 7"
```

if-statement (4)

- Waar hoort het laatste else-blok bij?

```
if x > 0:
    if y > 0:
        print "Beide groter dan nul."
    else:
        print " ??? "
```

- Dit wordt altijd bepaald door hoe ver er is ingesprongen. De layout geeft altijd uitsluitsel.
- Dus in dit geval hoort het else-blok bij `if y > 0`.

for loops

- Een *for* loop gaat informeel als volgt:

```
voor elk element in een rijtje:
    doe iets met element
```

- De Python-notatie scheelt niet eens zo heel veel:

```
for variabele in rij:
    statementblok
```

- In feite wordt een iteratie van een rijtje elementen (getallen, variabelen, …) uitgedrukt.
- De *iteratievariabele* (“element”) neemt opeenvolgend de verschillende waarden van het rijtje aan.

for loops (2)

- Het volgende voorbeeld drukt de eerste 10 getallen af met hun kwadraat:

```
for i in range(10):
    print i, "--", i * i
```

- De loop wordt uitgevoerd voor `i` met waarden `0` tot en met 9.

Rijtjes van getallen maken

- We gebruiken de functie `range()` om rijtjes van getallen te maken.
  - `range(stop)`
  - `range(start, stop)`
  - `range(start, stop, step)`
- De gegeven stop-waarde doet **niet** mee!

Rijtjes van getallen maken

```
>>> range(6)
[0, 1, 2, 3, 4, 5]
>>> range(3,6)
[3, 4, 5]
>>> range(0, 50, 5)
[0, 5, 10, 15, 20, 25, 30, 35, 40, 45]
>>> range(20, 50, 5)
[20, 25, 30, 35, 40, 45]
>>> for i in range(10):
...     print i,
...
0 1 2 3 4 5 6 7 8 9
```

Geneste loops

- Je mag ook loops binnen loops gebruiken. We spreken dan over geneste loops.
- Hier een voorbeeld van een dubbele for loop:

```
for i in range(1, 6):
    print "{0}:".format(i),
    for j in range(1, i+1):
        print i * j,
    print
```

Uitvoer:

```
1: 1
2: 2 4
3: 3 6 9
4: 4 8 12 16
5: 5 10 15 20 25
```

while loops

- Informeel ziet een *while*-statement er als volgt uit:

```
zolang een of andere test waar is:
    doe iets
```

- Een *while*-statement bestaat dus uit een testexpressie en een “loop body”.
- Er is standaard *geen* rijtje en *geen* iteratievariabele.

- (Python kent **geen** `do-while`.)

while loops (2)

Stel je wilt de eerste n positieve gehele getallen en hun kwadraten afdrukken:

```
i = 1
n = 10
while i <= n:
    print i, "--", i * i
    i += 1
```

“i” heeft hier de functie van “teller” en kun je de iteratievariabele noemen.

while loops (3)

- Nog wat voorbeelden van while-loops.

```
while n != 0:       # zolang n niet 0 is
    n -= 1          # verlaag n met 1
```

- Let er op dat je beter `n > 0` als test kunt gebruiken:

```
x = 1
while x < 100:
    print x
    x += 2
```

- Deze loop drukt de oneven getallen `< 100` af, maar niet als de test `x != 100` zou zijn, want dan stopt het programma niet!
  - Mocht je zo'n fout maken, denk aan de CTRL-C om een proces te stoppen.

while loops (4)

- Bij de iteratie van een rijtje, zoals in for loops, staat het aantal "doorgangen" in principe vast. Bij een while loop is dit aantal doorgangen van te voren niet bekend of lastig te bepalen.

```
x = 1
while x < 1000:
    x = 2 * x
# Nu is x gelijk aan 1024
```

- Het Collatz-probleem (3x+1 vermoeden) zegt dat het volgende programma voor iedere positieve gehele x stopt:

```
# 13->40->20->10->5->16->8->4->2->1
while x != 1:
    if x % 2 == 0:  # is x even?
        x = x / 2
    else:
        x = 3 * x + 1
```

Als dit programma stopt, is x na afloop gelijk aan 1.

while loops (5)

Na

```
while x > 1:
    x = x / 2
```

geldt dat `not (x > 1)`, dus `x <= 1`.

En na

```
while x % 2 == 0:
    x = x / 2
```

zijn alle factoren `2` uit de "oude" `x` gehaald; `x` is nu oneven.

while loops (6)

```
while True:
    print "Goedendag"
```

- Oneindige loop / infinite loop.
- Of met een "empty statement" (computer lijkt te hangen):

```
while True:
    pass
```

Welke loop?

- Alle for loops kunnen ook als while loop worden geschreven, sommige while loops ook als for loop.
- Welke loop kiezen we nu?
- Vuistregel:
  - Is het aantal herhalingen van te voren bekend? Itereer je een rijtje van dingen? Kies voor de for loop.
  - In alle andere gevallen is het aantal herhalingen niet bekend. Dan kies je de while loop.

Inspringregels

- Correct inspringen is een must, fout inspringen wordt bestraft met een `IndentationError`.
- Wanneer inspringen?
  - Om blokken (groepjes) van statements te vormen.
  - We hebben dit nodig bij:
  - if-statements, loops en definiëren van functies.
  - Vuistregel: na een dubbele punt wordt een ingesprongen blok verwacht!
- Er kan geen verwarring zijn: de layout (het inspringniveau) is altijd leidend.

Inspringregels (verv.)

- Binnen eenzelfde blok moet er op elke regel op dezelfde manier worden ingesprongen.
- De eerste regel die anders wordt ingesprongen maakt geen deel meer uit van dat blok.
- Advies: altijd 4 spaties, vermijd tabs.

Inspringregels (verv.)

- In het voorbeeld van de geneste loop vinden we 3 niveau's terug:

```
for i in range(1, 6):
    print "{0}:".format(i),
    for j in range(1, i+1):
        print i * j,
    print
```

pass statement

- Om “lege” blokken van statements te maken, heeft Python een speciaal “empty statement”.
  - Want zonder statement, is het moeilijk inspringen …
- `pass`-statement:

```
x = 10
if x > 0:
    # niets
print "test"
if x > 0:
    pass
print "test"
```

Afsluitend

- Rond de eerste programmeeropgave af!
  - Deadline maandag 25 september 2017!
- Huiswerk:
  - Lees het collegedictaat t/m hoofdstuk 5.

- Website:

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

Programmeermethoden NA
Week 3

Universiteit
Leiden