Python voor Natuur- en Sterrenkundigen Week 1

Kristian Rietveld

http://liacs.leidenuniv.nl/~rietveldkfd/courses/pmpy2015/

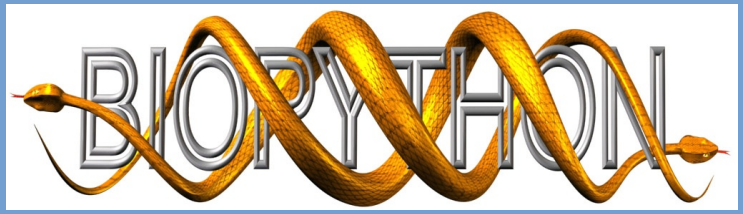
Universiteit Leiden
The Netherlands

Wat is Python & Waarom Python?

- "Scripttaal", ontworpen door Guido van Rossum eind jaren '80 / begin jaren '90.
- Eenvoudig & portable.
- Complexe bewerkingen in maar enkele regels code -- hierdoor een ultiem gereedschap!
- Zeer populair geworden in de laatste tien jaar.

Waarom zo populair?

- Zeer uitgebreide standaard bibliotheek.
- Eenvoudig om uitbreidingen te schrijven.
- Er zijn vele modules ontwikkeld voor het doen van numeriek rekenwerk en maken van plots.
- Hierdoor zeer populair in verschillende wetenschappelijke disciplines.



Opzet van het college

- **College 1**: Introductie, getallen, print, strings, control flow
- **College 2**: Datastructuren, functies en files
- **College 3**: NumPy en matplotlib
- **College 4**: Vervolg NumPy, modules maken en gebruiken, en hoe meer leren over Python?

- Alle informatie (en dictaat) via de website:

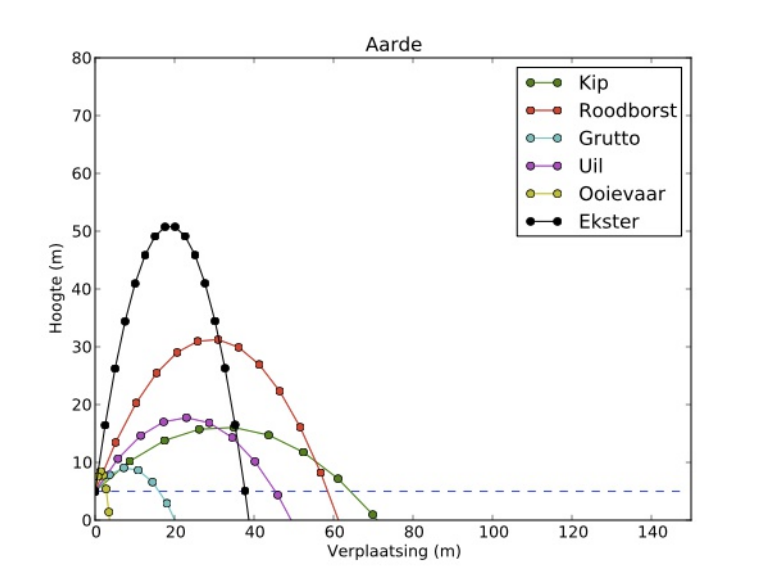
  http://liacs.leidenuniv.nl/~rietveldkfd/courses/pmpy2015/

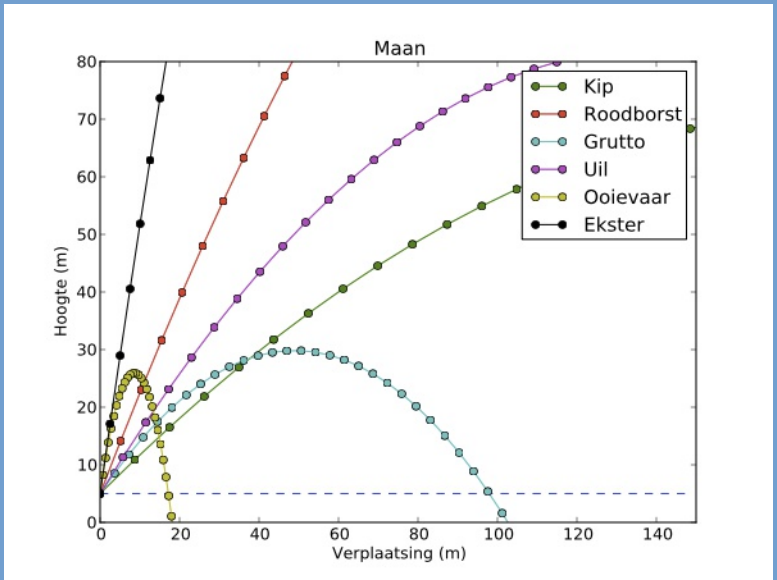
Werkcolleges

- Donderdagen (4x) van 11:15 tot 13:00.
- Aanwezigheidscontrole.
- Zalen 303-308 in het Snelliusgebouw.
- Met de assistenten hebben jullie al kennisgemaakt.

De eindopdracht

- De eindopdracht: Angry Birds.
- Kogelbanen plotten met verschillende lanceersnelheid en lanceerhoek, op verschillende planeten.
- **Deadline: vrijdag 11 december, 17:00 uur.**
- Voor een complete beschrijving: zie de website.





Bepaling eindcijfer

- Bij deze collegereeks hoort een eindcijfer, dat zal tellen als het cijfer voor de vierde programmeeropdracht van Programmeermethoden.
- De eindopdracht telt voor **70%** mee.
- De andere **30%** (3 van de 10 punten) via de eerste 3 werkcolleges:
  - **0.5 punt** voor aanwezigheid.
  - **0.5 punt** voor het voltooien van 1 verplichte opgave tijdens het werkcollege.

Compileren vs. interpreteren

- C++ is een gecompileerde taal.

```
gedit programma.cc
g++ -Wall -o programma programma.cc
./programma
```

- Python is een geinterpreteerde taal.

```
gedit programma.py
python programma.py
```

Compileren vs. interpreteren (verv.)

- Python wordt ook wel een scripttaal genoemd, net als bijvoorbeeld Perl, Ruby en PHP. De programma's noemen we vaak "scripts".
- Geen compilatieslag, dus sneller testen.
- Nadeel: minder fouten worden van te voren ontdekt.

De Python interpreter

- (Demo)

Python installeren

- Python is "open source" en gratis te verkrijgen.
- Linux of Mac machine: standaard geïnstalleerd.
  - (Mac >= 10.9: ook standaard NumPy & matplotlib).
- Windows: zelf downloaden en installeren.
  - Probeer een distributie: Enthought of Python(x,y).
- Zie dictaat voor details.

Een eerste Python programma

```
print "Dit komt op het scherm."
exit(0)
```

Een eerste Python programma (verv.)

```
# dit is een simpel programma
getal = 42 # een variabele declareren en initialiseren
print "Geef een geheel getal ..",
getal = int(raw_input())
print "Kwadraat is:", getal * getal
exit(0)
```

Een eerste Python programma (verv.)

```
# Dit is een regel met commentaar ...
import math # voor de "pi" constante
print "Geef straal, daarna Enter ..",
straal = float(raw_input())
if straal > 0:
    print "Oppervlakte:",
    print math.pi * straal * straal
else:
    print "Niet zo negatief ..."
print "Einde van dit programma."
exit(0)
```

Een eerste Python programma (verv.)

```
# Dit is een regel met commentaar ...
import math # voor de "pi" constante
print "Geef straal, daarna Enter ..",
straal = float(raw_input())
if straal > 0:
    print "Oppervlakte:",
    print math.pi * straal * straal
else:
    print "Niet zo negatief ..."
print "Einde van dit programma."
exit(0)
```

Een eerste Python programma (verv.)

```
# Dit is een regel met commentaar ...
import math # voor de "pi" constante
print "Geef straal, daarna Enter ..",
straal = float(raw_input())
if straal > 0:
    print "Oppervlakte:",
    print math.pi * straal * straal
else:
    print "Niet zo negatief ..."
print "Einde van dit programma."
exit(0)
```

Een eerste Python programma (verv.)

```
# Dit is een regel met commentaar ...
import math # voor de "pi" constante
print "Geef straal, daarna Enter .",
straal = float(raw_input())
if straal > 0:
    print "Oppervlakte:",
    print math.pi * straal * straal
else:
    print "Niet zo negatief ..."
print "Einde van dit programma."
exit(0)
```

Een eerste Python programma (verv.)

```
# Dit is een regel met commentaar ...
import math # voor de "pi" constante
print "Geef straal, daarna Enter ..",
straal = float(raw_input())
if straal > 0:
    print "Oppervlakte:",
    print math.pi * straal * straal
else:
    print "Niet zo negatief ..."
print "Einde van dit programma."
exit(0)
```

Een eerste Python programma (verv.)

```
# Dit is een regel met commentaar ...
import math # voor de "pi" constante
print "Geef straal, daarna Enter ..",
straal = float(raw_input())
if straal > 0:
    print "Oppervlakte:",
    print math.pi * straal * straal
else:
    print "Niet zo negatief ..."
print "Einde van dit programma."
exit(0)
```

Variabelen in Python

- In C++ moesten variabelen vooraf worden gedeclareerd als een bepaald type en kan dit type niet meer veranderen.
- Dit is in Python niet nodig, we maken variabelen met een toekenningsstatement (assignment).
- Toekenning op een al bestaande naam overschrijft de oude waarde.

Variabelen in Python (verv.)

```
a = 4
b = "testje!"
a = "overschrijven"  # oude waarde van variabele
a wordt overschreven
d = a + g
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
NameError: name 'g' is not defined
```

Variabelen in Python (verv.)

- Elke variabele in Python heeft een type.

```
>>> a, b, c = 9, 3.14, "strrrr"
>>> type(a)
<type 'int'>
>>> type(b)
<type 'float'>
>>> type(c)
<type 'str'>
>>> a = "strrr2" # oude waarde van a wordt overschreven
>>> type(a)
<type 'str'>
```

Getallen

- *int*: Integers, meestal 4 bytes groot.

  Bereik -2^{31} tot $2^{31} - 1$.

- *long*: Long integers, zo groot als maar past in het geheugen van de computer. Zeer grote getallen mogelijk!
- *bool*: True of False.
- *float*: Benaderingen (!) van reele getallen. In Python altijd double precision, komt overeen met double in C++.
- *complex*: Complexe getallen. Ingebouwd!

Complexe getallen

```
>>> z = 6+9j # "j" is de imaginaire
eenheid, in de wiskunde i geheten
>>> type(z)
<type 'complex'>
>>> z.real
6.0
>>> z.imag
9.0
```

Operaties op getallen

```
a, b = 3, -5
getal = a + b    # getal wordt -2
a = a + 7        # a wordt 10
b += 1           # Python kent geen ++ operator
a -= 1
getal += a
a = 19 / 7       # Integer deling: a wordt 2
b = 19 % 7       # Rest bij deling (mod): b wordt 5
# Optelling complexe getallen: resultaat (10+11j)
q = (6+9j) + (4+2j)
q = (6+9j) * 2   # Resultaat: (12+18j)
```

Integer vs. floating point

```
i = 9 / 5           # Geeft 1, i wordt een integer
x = 9 / 5.0         # Geeft 1.8, x wordt een float
x = float(9 / 5)    # Geeft 1.0, 9 / 5 geeft een
                    # integer resultaat dat wordt
                    # omgezet naar een float.
x = 9 / float(5)    # Geeft 1.8, x wordt een float
x = 9.0 // 5.0      # Geeft 1.0, // is delen met
                    # integer-afronding
m = 3 ** 4  # Python heeft een ingebouwde operator
       # voor machtsverheffing, het resultaat is 81
```

Conversie van getallen

- `float()` is een type conversie. Accepteert ook strings: `float("3.14")`.
- Andere typeconversies: `int()`, `complex()`, `str()`.
- Niet hetzelfde als een C++ cast, die kan bijvoorbeeld niet van (een ouderwetse) string naar float!
- Operatie op twee verschillende typen resulteert in een impliciete conversie: type coercion.

print statement

- `print` zet data op het scherm.
- Keyword `print`, gevolgd door een lijst van expressies.
- Impliciete conversie naar strings.
- Spaties ingevoegd tussen uitvoeren van verschillende expressies.

print statement

```
>>> a = 110
>>> b = 12
>>> print "Test:", "a =", a, "b =", b, "en samen
maakt dat", a + b
Test: a = 110 b = 12 en samen maakt dat 122
```

Uitvoerformattering

- Nieuwe stijl, oude stijl staat omschreven in dictaat.
- Met een format field specificeren we hoe een variabele moet worden afgedrukt.
- `{0:8.4f}`
- `{1:>10s}`
- `print "{0:6d} {1:8.4f} {2:20s}".format(a, b, "test")`
- `print "{een} {twee}".format(een=a, twee=b)`

Strings

- Een string is een object, net als de C++ "string" klasse.
  - Python heeft geen `char` type.
- Op string-objecten kun je operaties uitvoeren.

```
>>> woord = "De."
>>> len(woord)
3
>>> woord == "test"
False
>>> woord == "De."
True
```

Strings - Indexing en slicing

- In een ouderwetse C-string kunnen we met een index een individueel array element uitlezen.
- In Python kunnen we objecten van het type `str` ook "indexen".
  - `woord[1]`
- Je mag ook een start en eind index geven, bijvoorbeeld om een substring uit te lezen. We noemen dit slicing.
  - `zin[3:14]`
  - De eind-index telt **niet** mee!

Strings - Indexing en slicing

```
>>> s = "een lange test string"
>>> s[2]
'n'
>>> s[-4]
'r'
>>> s[3:8]
' lang'
>>> s[6:]
'nge test string'
```

Strings - Andere operaties

```
>>> s = "aaa bbb ccc eee fff ggg"
>>> "aaa" in s
True
>>> "zzz" in s
False
>>> f = "testbestand.txt"
>>> f.endswith(".txt")
True
>>> s + f
'aaa bbb ccc eee fff gggtestbestand.txt'
```

Strings - Andere operaties

```
>>> s + 12
File "<stdin>", line 1, in <module>
TypeError: cannot concatenate 'str' and
'int' objects
>>> s + str(12)
"aaa bbb ccc eee fff ggg12"

>>> a = "aaa"
>>> b = "bbb"
>>> a * 3
'aaaaaaaaa'
>>> a * 3 + b * 3
'aaaaaaaaabbbbbbbbb'
```

Lijsten

- Een `list`-object is een geordende lijst van variabelen.
- Verschillen met C++ arrays:
  - De variabelen hoeven **niet** van hetzelfde type te zijn.
  - De grootte van de lijst staat niet vast, je kan eenvoudig elementen toevoegen en verwijderen (zien we volgende week).
- Er wordt vaak over lijsten gesproken als compound data type of sequence type.

Lijsten (verv.)

- Lijsten kunnen worden aangemaakt met blokhaken:

```
a = [1, 2, 3, 4, 5]
b = [1.0, 2.5, 3.4]
c = [1, "test", 4.5, False]
```

Lijsten - Indexing en slicing

```
>>> a = [0, 1, 2, 3, 4, 5, 6, 7]
>>> len(a)
8
>>> a[6]
6
>>> a[2:5]
[2, 3, 4]
>>> a[3:]
[3, 4, 5, 6, 7]
>>> a[:6]
[0, 1, 2, 3, 4, 5]
>>> a[4] = 'ha!'
>>> a
[0, 1, 2, 3, 'ha!', 5, 6, 7]
```

Controlestructuren

- De belangrijkste controlestructuren in Python zijn:
  - `if` voor het maken van keuzes.
  - `for` voor een vast aantal herhalingen.
  - `while` voor een onbekend aantal herhalingen.

if - then - else

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

Boolean expressies

- De predicaten die je kent uit C++ werken gewoon: ==, !=, <, >=, enzovoort.
- In plaats van `!`, && en `||` gebruiken we `not`, `and` en `or`.
- Als je zowel `and` als `or` in een expressie gebruikt: gebruik haakjes om verwarring te voorkomen!
- Ook in Python wordt short-circuiting toegepast.

```
- (x != 0 and y / x == 7)
```

Boolean expressies

```
if y >= 3 and y <= 7: ...
if not (y < 3 or y > 7): ...
if y>= 3 and (x == 4 or x == 5): ...
if s == "hello": ...
if y >= 3 and (x == 4 or x == 5) and \
   z == 12 and (q >= 10 or q <= -10): ...
```

for loops

- `for`-loops in Python een stuk eenvoudiger.
- We drukken een iteratie van een lijst uit.
- De iteratievariabele neemt opeenvolgend de verschillende waarden van de lijst aan.

for loops

```
for karakter in ['a', 'b', 'c', 'd', 'e']:
    print karakter,
# drukt af: a b c d e
```

```
for i in [1, 2, 3, 4, 5]:
    print i
```

`range()` functie

- Voor grote aantallen herhalingen wil je niet met de hand zo'n lijst schrijven.
- Met `range()` kunnen getallenreeksen worden gemaakt.
- `range(start, stop, step)`
- De gegeven stop-waarde doet **niet** mee!

range() functie

```
>>> 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
```

Van C++ naar Python

```
for ( i = 3; i <= 17; i = i + 2)
  cout << i << "-";

for i in range(3, 17+1, 2):
    print i, "-",
```

Geneste loops

```
for i in range(1, 5+1):
    print "{0}:".format(i),
    for j in range(1, i+1):
        print i * j,
    print
```

while loops

- Het `while`-statement bestaat uit een testexpressie en een loop body.
- Er wordt in dit geval geen lijst afgelopen.
- Python kent **geen** `do-while`.

while loops

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

Inspringregels

- In C++ kun je slordig zijn met de layout van je code, Python is daar echter veel strikter in.
- Correct inspringen is een must, fout inspringen wordt bestraft met een `IndentationError`.
- Wanneer inspringen?
  - Om blokken van statements te vorm.
  - `if`-statements, loops en definiëren van functies.
  - In C++ plaats je bij bijna al deze gevallen accolades!

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, 5+1):
    print "{0}:".format(i),
    for j in range(1, i+1):
        print i * j,
    print
```

pass statement

- In C++ konden we een accolade openen en direct weer sluiten, zonder statements er in. Een leeg blok!
- Dat wordt met inspringen een beetje lastig ...
- Oplossing: `pass`-statement.

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

Dangling else in Python

```
if ( x > 0 )
  if ( y > 0 )
    cout << "Beide groter dan nul.";
  else                  // waar hoort deze bij?
    cout << " x positief, y negatief (of 0)  ";
```

Layout

- "Zorg ervoor dat de layout klopt - de compiler kijkt er niet naar."
- Maar in Python wordt er door de interpreter **juist wel** naar de layout gekeken!
- Er kan geen verwarring zijn: de layout (het inspringniveau) is leidend.

Volgende week

- Meer over lijsten
- Functies
- Files lezen & schrijven

Morgen werkcollege

- 11:15 - 13:00 uur
- Snelliusgebouw: zalen 303-308
- Aanwezigheidscontrole
- 1 verplichte opdracht voor 0.5 punt

http://liacs.leidenuniv.nl/~rietveldkfd/courses/pmpy2015/