Einführung in Python

David A. Clarke

Computerphysik

15 Apr 2020

Eine Geschichte

Neugier und Unschuld: `DOG.BAS`

```
10 PRINT "WOOF"
GOTO 10
```

Der Tod der Unschuld: `dog.cpp`

```
#include <iostream>
using namespace std;
int main() {
  while(true) {
    cout << "woof..." << endl;
  }
  return 0;
}
```

Auferstehung: `dog.py`

```
while True:
  print("Woof!")
```

Warum Python?

Wie wir gesehen haben ist Python **sehr lesbar**.

Python ist **sehr beliebt**: Zum Beispiel laut `codeburst.io` ist Python die zweitpopulärste Sprache für Entwickler. Dieses Resultat basiert auf Umfragen von StackOverflow, Github Aktivitäten und Google Suchen.

Wie jede andere beliebte Sprache hat sie eine **große Community**.

Außerdem hat man mit einer populären Programmiersprache bessere Chancen auf dem Arbeitsmarkt!

Warum Python?

Python ist **sehr flexibel**.

Aber vor allem ist sie **gut für die Naturwissenschaften geeignet**:

- Man hat Zugriff auf Libraries, die für das naturwissenschaftliche Rechnen nützlich sind.
  - `numpy`: Elementweise Array-Operationen, lineare Algebra.
  - `scipy`: Optimierung, Kurven fitten.
  - Sonstiges: Google Suche, Excel Tabellen manipulieren, Machine Learning.
- Python ist anspruchsvoll genug, um Funktionen und Klassen zu unterstützen.
- Python ist gut um Graphen zu plotten.
  - Erstelle mit `matplotlib` professionell aussehende Plots.

Wann sollte ich Python benutzen?

Wann man Python NICHT verwenden sollte:

- Python ist eine Scriptsprache, also man kompiliert sie nicht.
- Daher ist sie für **rechenintensive Arbeit** zu langsam.
- Sie ist auch nicht für **speicherintensive Arbeit** geeignet.

Wann man verwenden sollte:

- Ich persönlich benutze sie für **fast alles andere**.

Beispiel. In meiner Physikgruppe benutzen wir Python hauptsächlich für Datenanalyse, Fitten, Interpolation, etc. Das Generieren der Daten hingegen ist sehr rechenintensiv, und es verlangt z.B. mindestens $\mathcal{O}(10\,000) \times \mathcal{O}(10\,000)$ Multiplikationen von 3×3 Matrizen. Dafür verwenden wir C++.

erste Schritte

Für dieses Lernprogramm benötigen wir:

- Anaconda (Python 3.x)
- Jupyter

Vielleicht finden Sie auch die folgenden Ressourcen hilfreich:

- Python CheatSheet (Jupyter Notebook)
- Python documentation
- numpy and scipy documentation
- StackOverflow
- Jupyter interactive notebooks

Types

Es gibt verschiedene Types von Variablen:

- string: “Wörter”
- integer: Zahlen
- float[1]: reele Zahlen
- boolean: `True` oder `False`

[1] Double Precision. Wenn man Single Precision braucht muss man z.B. `numpy` benutzen.

Terminal Auslesen

print ist der primäre Befehl für die Terminalausgabe. Der Code

```
print("Hello, world!")
print(99, "Luftballons")
print("Baby you're a...\t firework\n")
print("%11.5f is the loneliest number." % 1.0)
print("%11.5e can be as bad as %11.5e." % (2.0, 1.0))
```

ergibt

```
Hello, world!
99 Luftballons
Baby you're a...   firework

    1.00000 is the loneliest number.
2.00000e+00 can be as bad as 1.00000e+00.
```

Variablen und Kommentare

Man deklariert keine Types für Variablen. Python findet heraus was sie sind.

```
one   = 1                          # an integer
two   = 2.0                        # a float
three = two + one                  # cast as float
city  = "bielefeld"
thing = city + "Conspiracy"

""" Can also comment this way """
print(one, two, three, thing)
```

Ausgabe:

```
1 2.0 3.0 bielefeldConspiracy
```

WARNUNG: Wenn eine Funktion einen **float** erwartet aber einen **int** bekommt kann sie den **int** in einen **float** umwandeln.

Arithmetik

```
x = 1
y = 2
z = 2.0
print(x, y, z, x/y) # x/y gives 0.5 rather than 0

x += 1                # i.e. xnew = xold + 1
y += 2.0              # Recasts y as float
z -= 1
print(x, y, z, y/z, y**2)

y *= 2
z /= 2
print(y, z)
```

```
1 2 2.0 0.5
2 4.0 1.0 4.0 16.0
8.0 0.5
```

Lists

Das grundlegende arrayartige Objekt in Python ist die List. Das erste Element in der List wird mit 0 indiziert.

```
emptyList   = []
integerList = [1, 2, 3]
floatList   = [4.0, 5.0, 6.0]
vocabList   = ["Springfield", "embiggen"]

print( len(integerList) )
print( floatList[1] )

vocabList.append("cromulent")

print(vocabList)
```

```
3
5.0
['Springfield', 'embiggen', 'cromulent']
```

Dictionaries

Eine Dictionary ist eine List, die mit Keys "indiziert" wird. Für jeden Key gibt es eine entsprechende Value.

```
emptyDictionary = {}
englishToGerman = { "I"      : "ich",
                    "you"    : ["du", "Sie", "ihr"],
                    1        : "eins",
                    2        : "zwei",
                    "police" : "Polizei" }

englishToGerman["she"] = "sie"

print( englishToGerman["I"])
print( englishToGerman["you"][0] )
print( englishToGerman[1] )
```

```
ich
du
eins
```

Bedingte Ausdrücke

Führende Leerzeichen[2] bestimmen den Scope (Bereich). Bitte achten Sie deshalb auf führende Leerzeichen!

```
if 1 > 2.0:
  print("That doesn't seem right.")
if 1 == 1.0:
  print("That makes sense.")
if (0.0 < 1) and (1 < 2.0):
  print("That also makes sense.")
if (-1.0 > 1) or (-1.0 < 1):
  print("Well it has to lie somewhere...")
if not (1 < 3):
  print("Python is bad at math.")
```

```
That makes sense.
That also makes sense.
Well it has to lie somewhere...
```

[2] Man kann auch Tabs benutzen, aber ich empfehle Leerzeichen.

Das **in** Keyword

Das **in** Keyword wird verwendet, um Dinge in Lists, Dictionaries, Dateien, etc. zu finden.

```
planets = ["Mercury","Venus" ,"Earth" ,"Mars"    ,
           "Jupiter","Saturn","Uranus","Neptune"]

if "Earth" in planets:
  print("Earth is a planet.")

if not "Pluto" in planets:
  print("You heard about Pluto? That's messed up,
      right?")
```

```
Earth is a planet.
You heard about Pluto? That's messed up, right?
```

Kontrolle, Scope und Loops

Python benutzt **in** in **for** Loops.

```
i = 0
while i < 3:
  print(i)
  i += 1
  j = 4
print(j)              # Note that Python remembered j

for i in range(3): # Pythonic integer loop
  print(i)
```

```
0
1
2
4
0
1
2
```

Mehr Kontrolle

Verwenden Sie **continue**, um zur nächsten Iteration einer Loop überzugehen.

```
objects = ["A table", "Paper", "The earth",
           "A postcard"]

for item in objects:

  if item == "The earth":
    continue

  print(item,"is flat.")
```

```
A table is flat.
Paper is flat.
A postcard is flat.
```

Funktionen

Verwenden Sie Funktionen, um allgemeine Aufgaben zu kapseln. Durch `*args` kann man beliebig viele Argumente haben.

```
def sum2(a,b):
  return a+b

def sumGeneral(*args):
  result = 0.
  for x in args:
    result += x
  return result

print( sum2(1,3.) )
print( sumGeneral(1,7,4.,1e-3) )
```

```
4.0
12.001
```

Libraries

Ein Großteil der Kraft von Python kommt von seinen robusten Libraries. Sie können einfach durch das Keyword `import` importiert werden. Für diese Klasse sind die interessantesten `numpy` und `matplotlib`:

```
import numpy as np
import matplotlib.pyplot as plt
```

numpy

`numpy` enthält viele Funktionen, die nützlich für Naturwissenschaftliche Rechnungen sind. Rechnungen durch `numpy` Funktionen sind i.A. relativ schnell.

```
import numpy as np

pi = np.pi
print( pi )
print( np.sin(pi) )
print( np.cos(pi) )
print( np.exp(1j*pi) ) # e^i*pi
```

```
3.141592653589793
1.2246467991473532e-16
-1.0
(-1+1.2246467991473532e-16j)
```

Datei einlesen und auslesen

PROBLEM: Ich habe eine Datei `numbers.d`, die wie folgt aussieht:

```
1 2
3 4
5 6
```

Ich möchte

- von `numbers.d` einlesen,
- Spalte 1 und Spalte 2 addieren
- Ergebnis in `theirSum.d` schreiben.

Wie können wir das in Python machen?

Datei einlesen und auslesen

LÖSUNG: Verwenden Sie `loadtxt` und `savetxt`.

```
import numpy as np

inData  = np.loadtxt('numbers.d',unpack=True)
outData = []
col0    = inData[0]
col1    = inData[1]

for i in range(len(col0)):
  outData.append( col0[i] + col1[i] )

np.savetxt('theirSum.d',outData,fmt='%10.5f')
```

`theirSum.d`:

```
   3.00000
   7.00000
  11.00000
```

Plotten

Man kann einfach plotten durch `pyplot`. WARNUNG: Achten Sie auf die Ordnung der Befehle wenn Sie `matplotlib` benutzen!

```
import matplotlib.pyplot as plt
import numpy as np

x = np.linspace(0, 1, 101)
y = np.exp(x)
z = np.exp(2*x)
plt.xlabel('x')
plt.yscale('log')
plt.title('Plot Example (log scale)')

plt.plot(x,y,label='exp(x)')
plt.plot(x,z,label='exp(2x)')

plt.legend(loc='lower right')

plt.show()
```

Viel Spaß!

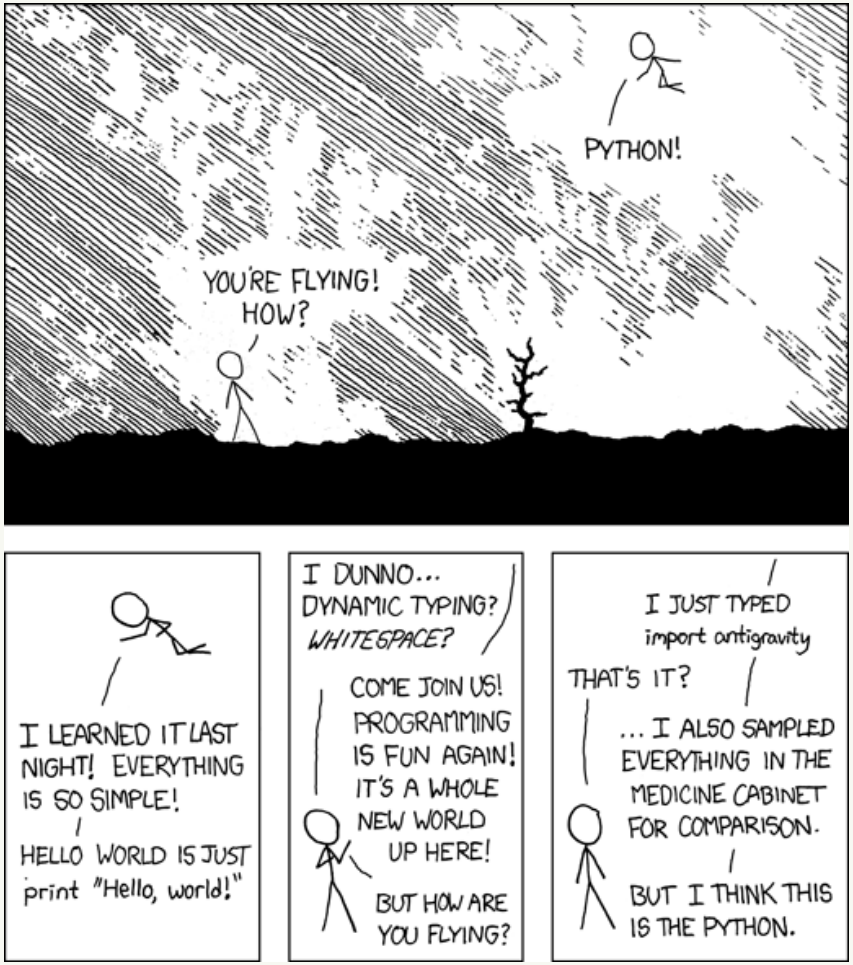

image from XKCD

Danke für Ihre Aufmerksamkeit!