Python als praktischer Helfer

Ulrich Schumann

01.02.2016

IEEE Student Branch Magdeburg

Geschichte

Geschichte

- Entwicklung gestartet durch Guido van Rossum

Guido van Rossum

Geschichte

- Entwicklung gestartet durch Guido van Rossum
- Version 1.0 im Januar 1994
- Version 2.0 im Oktober 2000
- Version 3.0 im Dezember 2008

Warum Python?

Warum Python?

- Interpretierte Scriptsprache
- Leicht zu lesen
- Umfangreiche Standardbibliothek
- Hohe Verbreitung
- Open source
- Durchgehend Objektorientiert

Warum Python nicht?

- Interpretierte Scriptsprache
- Keine statische Typisierung
- Verfügbarkeit Microcontroller/Smartphones

Anwendung

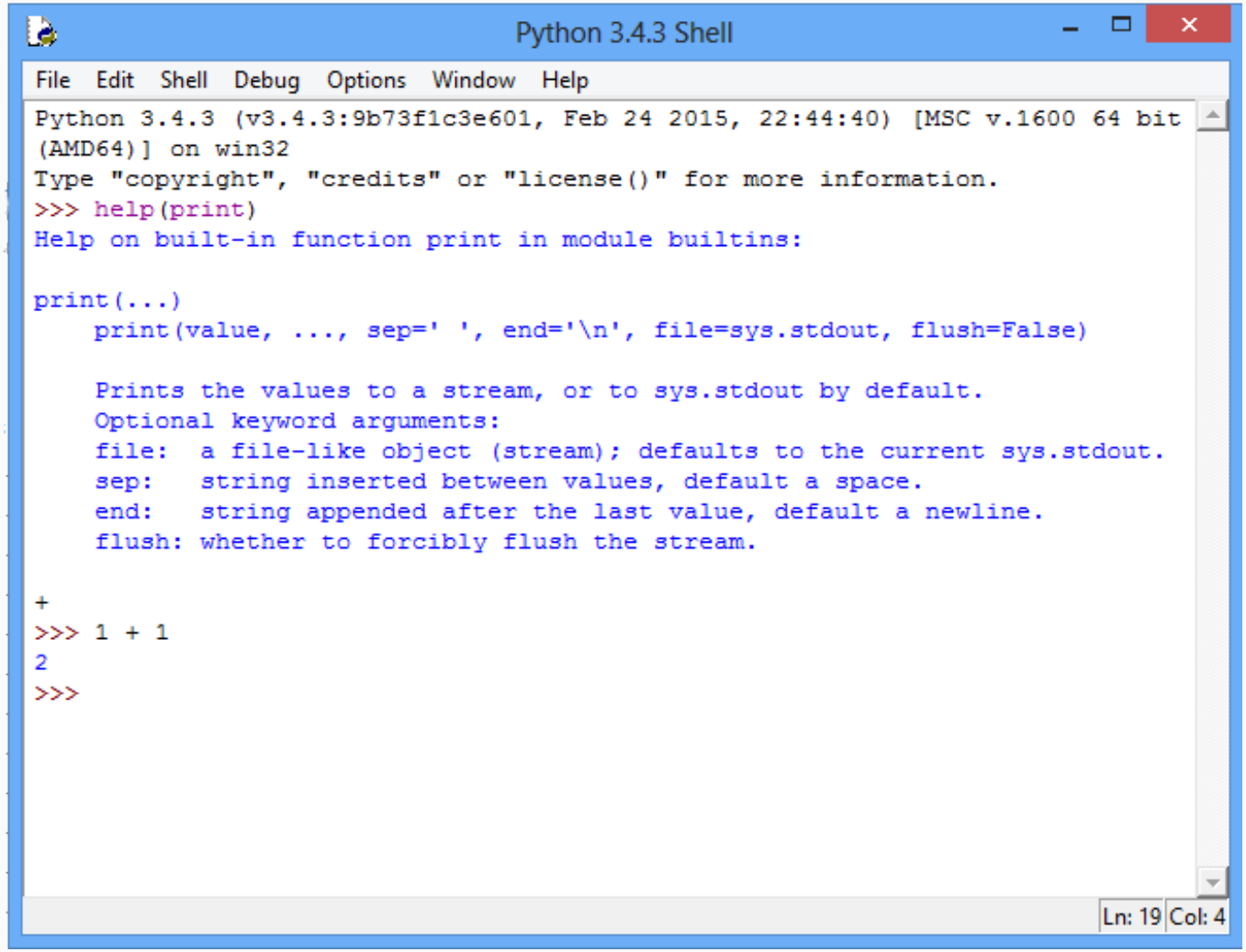
Version

- 2.7 und 3.x parallel
- Versionen inkompatibel
- Konverter 2.x -> 3.x und umgekehrt
- Viele Bibliotheken für beide Versionen
- Empfehlung: 3.x

Übersicht Anwendungsarten

- Interaktiv

Interaktiv


Python 3.4.3 Shell
File Edit Shell Debug Options Window Help
Python 3.4.3 (v3.4.3:9b73f1c3e601, Feb 24 2015, 22:44:40) [MSC v.1600 64 bit (AMD64)] on win32
Type "copyright", "credits" or "license()" for more information.
>>> help(print)
Help on built-in function print in module builtins:
print(...)
print(value, ..., sep=' ', end='\n', file=sys.stdout, flush=False)
Prints the values to a stream, or to sys.stdout by default.
Optional keyword arguments:
file: a file-like object (stream); defaults to the current sys.stdout.
sep: string inserted between values, default a space.
end: string appended after the last value, default a newline.
flush: whether to forcibly flush the stream.
+
>>> 1 + 1
2
>>>
Ln: 19 Col: 4

Übersicht Anwendungsarten

- Interaktiv
- Interpretiertes Script

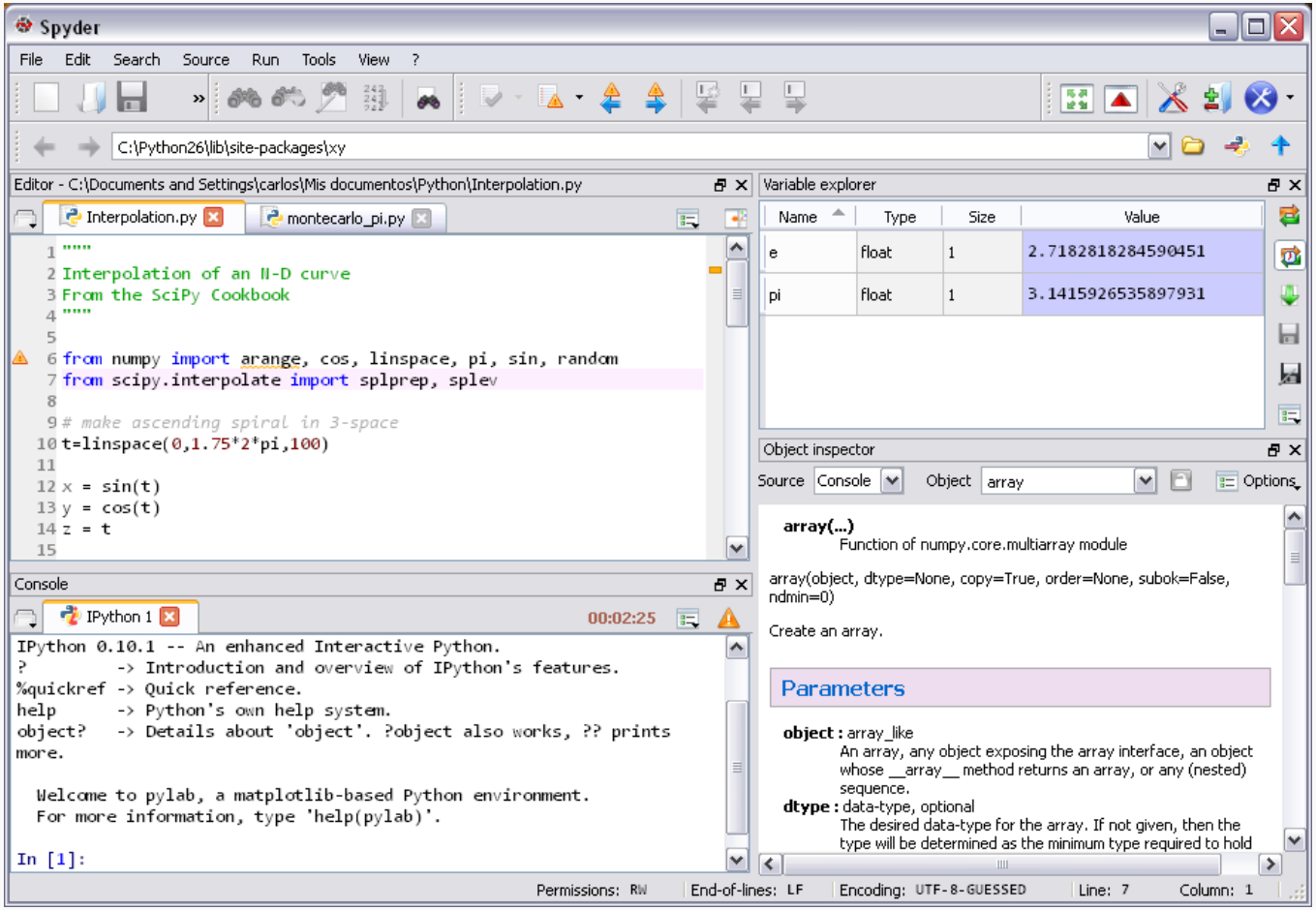
Interpretiertes Script

```python
#!/usr/bin/env python

from utils import parseCSV

import sys
sys.path.append("..")

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

# sample data
data = parseCSV("data/s11", startRow=1, steps=10)
val1 = data[:, 1] + data[:, 2] * 1j

data = parseCSV("data/s22", startRow=1, steps=10)
val2 = data[:, 1] + data[:, 2] * 1j

line = np.array([0.7 + 0.2j, 0.7 + 1.8j, 0.3 + 1.8j, 2])

# plot data
plt.figure(figsize=(8, 8))

ax = plt.subplot(1, 1, 1, projection='smith', axes_norm=50)
plt.plot(val1, markevery=5, label="S11")
plt.plot(val2, markevery=5, label="S22")
ax.plot_vswr_circle(0.3 - 0.7j, real=1, solution2=True, label="Re(Z)->1")

plt.plot(line, path_interpolation=0, label="Polyline")

plt.legend(loc="lower right")
plt.title("Matplotlib Smith Chart Projection")

plt.show()
```

Übersicht Anwendungsarten

- Interaktiv
- Interpretiertes Script
- Integrierte Entwicklungsumgebung

Integrierte Entwicklungsumgebung



Übersicht Anwendungsarten

- Interaktiv
- Interpretiertes Script
- Integrierte Entwicklungsumgebung
- Kompiliertes Binary
- Als Scripstsprache in eigenen Programmen

Distributionen

- Python.org
- PyPy
- WinPython
- Anaconda
- Python(x,y)
- Enthought Canopy

„Hello World“ etc.

Hello World

Python 3.x

```
print ("Hello, World!")
```

Python 2.x

```
print "Hello, World!"
```

Variablen und Operatoren

Numerische Datentypen

- `int` und `long` für ganze Zahlen
- `float` für Gleitkommazahlen
- `complex` für komplexe Zahlen
- `bool` für boolsche Werte

Sequenzielle Datentypen

- `str` und unicode für Zeichenketten
- `list` beliebig und änderbar
- `tuple` beliebig und fest

Variablen und Operatoren

Arithmetische Operatoren

- `x + y` Addition
- `x - y` Subtraktion
- `x * y` Multiplikation
- `x / y` Division
- `x % y` Rest ganzzaliger Division
- `x // y` Ganzzahliger Anteil Division
- `x ** y` Potenzieren
- `-x` Negatives Vorzeichen
- Speziell für Typ `bool` : `not`, `and`, `or`

Vergleichsoperatoren

- Anwendbar auf `int`, `long`, `float`, `bool`
- == Gleich
- != Ungleich
- < Kleiner
- <= Kleiner gleich
- > Größer
- >= Größer gleich

Operatoren sequenzielle Typen

- `x in s` Ist x in s?
- `x not in s` Ist x nicht in s?
- `s + t` Verkettung von s und t
- `s += t` s um Element t verlängern
- `s * n` Neue Sequenz mit n-fach s
- `s[i]` i-tes Element
- `s[i:j]` Auschnitt Elemente i bis j
- `s[i:j:k]` Ausschnitt jedes k-te Element i bis j
- `len(s)` Anzahl der Elemente
- `min(s)` Kleinstes Element
- `max(s)` Größtes Element

Python praktisch

Variableneingabe

```
zahl = float(input("Zahl eingeben: "))

ganzzahl = int(input("Ganzzahl eingeben: "))

text = input("Text eingeben: ")


Zahl eingeben: 1.5

Ganzzahl eingeben: 2

Text eingeben: Puderquaste
```

Variablenausgabe

```
print ("Zahl =", zahl)

print ("Zahl * 2 =", zahl * 2)

print ("Zahl * Ganzzahl =", zahl * ganzzahl)

print ("Text ist", text)

print ("Datentyp Text ist", type(text))

print ("Text * Ganzzahl ist", text * ganzzahl)
```

```
Zahl = 1.5

Zahl * 2 = 3.0

Zahl * Ganzzahl = 3.0

Text ist Puderquaste

Datentyp Text ist <class'str'>

Text * Ganzzahl ist PuderquastePuderquaste
```

WHILE-Schleife

```
1 a = 0
2 while a < 5:
3   a = a + 1
4   print(a)
```

1

2

3

4

5

FOR-Schleife I

```
1 zahlenliste = range(1, 11)
2 for zaehler in zahlenliste:
3   print(zaehler)
```

```
1
2
3
4
5
6
7
8
9
10
```

FOR-Schleife II

```
1 liste = ['Test', 13, 14, 'Puderquaste', 'Pony']
2 for element in liste:
3   print("Element ist:", element)
```

```
Element ist: Test

Element ist: 13

Element ist: 14

Element ist: Puderquaste

Element ist: Pony
```

IF THEN ELSE

```
1 zahl = float(input("Zahl eingeben: "))
2 if zahl % 2 == 0:
3   print(int(zahl), "ist gerade.")
4 elif zahl % 2 == 1:
5   print(int(zahl), "ist ungerade.")
6 else:
7   print(zahl, "ist keine Ganzzahl.")
```

```
Zahl eingeben: 5

5 ist ungerade.

Zahl eingeben: 2

2 ist gerade.

Zahl eingeben: 2.3

2.3 ist keine Ganzzahl.
```

Umgang mit Listen

```
1  demoliste = ['1', 2, 'drei', '44']
2  print(demoliste)
3  print("Laenge der Liste:", len(demoliste))
4  demoliste.append("Neu")
5  print(demoliste)
6  print("Index von 2 =", demoliste.index(2))
7  print("Element an Index 1 =", demoliste[1])
8  if "drei" in demoliste:
9    print("drei ist in demoliste")
10 del demoliste[2]
11 print(demoliste)
```

Umgang mit Listen

```
['1', 2, 'drei', '44']

Laenge der Liste: 4

['1', 2, 'drei', '44', 'Neu']

Index von 2 = 1

Element an Index 1 = 2

drei ist in demoliste

['1', 2, '44', 'Neu']
```

Funktionen

```
1 def fibonacci(n):
2   a, b = 0, 1
3   while a < n:
4     print(a, end=' ')
5     a, b = b, a+b
6   print()
7 fibonacci(1000)
```

```
0 1 1 2 3 5 8 13 21 34 55 89 144 233 377 610 987
```

Funktionen und Variablen

```
1  a = 10
2  b = 15
3  e = 25
4
5  def funktion(a):
6    print("a in funktion =", a);
7    b = 100 + a
8    d = 2 * a
9    print("b in funktion =", b);
10   print("d in funktion =", d);
11   print("e in funktion =", e);
12   return b + 10
13
14 c = funktion(b)
15
16 print("a =", a)
17 print("b =", b)
18 print("c =", c)
19 print("d =", d)
```

Funktionen und Variablen

```
a in funktion = 15

b in funktion = 115

d in funktion = 30

e in funktion = 25

a = 10

b = 15

c = 125


NameError: name 'd' is not defined
```

Bibliotheken einbinden I

```
1 import calendar
2 jahr = int(input("Jahreszahl angeben: "))
3 calendar.prcal(jahr)
```

```
1 from calendar import prcal
2 jahr = int(input("Jahreszahl eingeben: "))
3 prcal(jahr)
```

Bibliotheken einbinden II

```
1 import math
2 math.e
3 math.pi
4 math.ceil(4.5)
5 math.fabs(-44)
```

```
2.718281828459045

3.141592653589793

5

44.0
```

GUI und Grafik

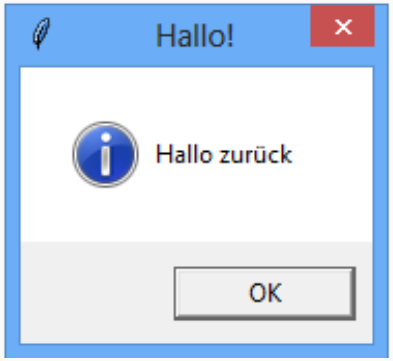
Graphical User Interface

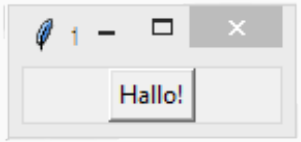
- Standardbibliothek Tkinter von Tk
- Anbindung an populäre GUI-Bibliotheken
  - WxWidgets - WxPython
  - Qt - PyQt, PySide
  - KDE - PyKDE
  - Gtk - PyGtk
  - .Net/Mono - IronPython
- Python Frontend mit GUI

GUI mit TkInter

```
1 from tkinter import *
2 def antwort():
3       messagebox.showinfo('Hallo!','Hallo zurueck')
4
5 root=Tk()
6 but=Button(root,text="Hallo!",command=antwort)
7 but.pack()
8 root.mainloop()
```

GUI mit TkInter





Grafik mit Python

- Canvas Widget von Tkinter
- graphics.py
- PyOpenGL für OpenGL Applikationen
- turtle.py für Turtle-Grafik

Grafik mit TkInter

```
1  from tkinter import *
2
3  master = Tk()
4
5  window = Canvas(master, width=200, height=200)
6  window.pack()
7
8  window.create_line(20, 20, 150, 50, fill="red")
9  window.create_line(150, 50, 50, 170, fill="yellow")
10 window.create_line(50, 170, 20, 20, fill="blue")
11
12 window.create_rectangle(100, 150, 190, 195, fill="
       orange")
13 mainloop()
```

Grafik mit TkInter