Crashkurs Python und Sage

U23 Krypto-Mission

florob
Simon

Chaos Computer Club Cologne e.V.
http://koeln.ccc.de

4. Oktober 2015

Übersicht

Sage Computer-Algebra-System, wie MatLab oder Mathematica
sagemath.org

Python interpretierte Programmiersprache mit einfacher Syntax

Objekte und Variablen

Objekte sind Werte, haben einen Typ
Variablen sind Namen, die auf Objekte zeigen

```
1, True, ['a', 'b', 'c']
# Kommentar
ding = 5 # Typ: Integer
anderes_ding = "fuenf" # Typ: string
ding = anderes_ding # Typ: string!
```

Felder und Methoden sind Namen, die zu einem Objekt gehören

```
person.vorname
17.is_prime()
```

Operatoren

Arithmetisch	`+, -, *, /, //, **, ^, %`
Logisch	`and, or, not, >, >=, ==, !=, <=, <`
Bitweise	`&, |, ^^, ~, <<, >>`
Mengen	`in, not in`
Index, Slice	`[], [::]`
Funktionsaufruf	`()`

Kontrollstrukturen

Zeilen, die gleich weit eingerückt sind, sind ein *Block*

While-Schleife `while <condition>:`

For-Schleife `for <var> in <sequence>:`

Verzweigung `if <condition>:`

```
for i in range(17):
    if i == 4:
        print '4 gefunden'
    else:
        do_nothing()
    something_else(i)
```

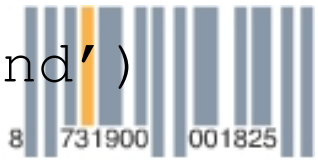
Funktionen

Funktion ist ein Block mit Namen
Funktionen können Parameter/Argumente haben und einen Wert zurückgeben
Argumente können Standardwerte haben

```
def greet(name, greeting='hello'):
    return greeting + ' ' + name
```

Beim *Aufruf* einer Funktion *müssen* Parameter ohne Standardwert angegeben werden. Parameter mit Standardwert *können* angegeben werden

```
greet('earthling')
greet('meine Damen und Herren', 'Guten Abend')
greet() # Fehler
```



Funktionen

Beim Aufruf einer Funktion *können* Parameter auch direkt benannt werden

```
greet(greeting='Guten Morgen',
      name='Sonnenschein')
```

Funktionen ohne explizites `return` geben `None` zurück.

Funktionen

Lambdas sind Funktionen ohne Namen

```
def f(x):
    return 2*x + 1
```

```
f = lambda x: 2*x + 1
```

Unterschied: Lambdas enthalten einen Ausdruck, Funktionen einen ganzen Block

```
lambda x: x^2 # Lambdas sind Objekte
sort(anwesenheitsliste,
     key = lambda person: person.nachname)
```

Literale: Schreibweisen für komplexe Objekte

Liste `[eins, zwei, drei, ...]`

```
l = [1, 2, 3]
l[0] # Lesezugriff
l[2] = 'text' # Schreibzugriff
len(l)
l.append('unsinn')
l.remove(17)
```

Slicing erzeugt eine Teilliste

```
l[start:stop]
l[start:stop:step]
l[::-1]
```

Literale: Schreibweisen für komplexe Objekte

Zuordnung `{key1:eins, key2:zwei, key3:drei, ...}`

```
d = {'one': 1, 'two': 2}

# Lesezugriff:
d['one']
d['nix'] # KeyError
'two' in d # True, Schluessel vorhanden
d.keys()
d.values()

# Schreibzugriff
d['slice'] = l[5:-7]
del d['two']
```

Literale: Schreibweisen für komplexe Objekte

Menge `{eins, zwei, drei, ...}`

```
s = {'a', 'b', 'c', 'a'}
s == {'a', 'b', 'c'}

# Lesezugriff:
for i in s: pass
if 'v' in s: pass

# "Schreibzugriff"
s | {4} # Vereinigung
s & {4} # Schnitt
s - {4} # Differenz
```

List Comprehension

```
quads = map(lambda x: x**2, range(50))
l = []
for i in quads:
    if i % 2 == 0:
        l.append(i)

l = [i for i in quads if i % 2 == 0]
```

Zusammenfassung

Namen	Objekt, Variable, Feld, Methode
Struktur	Schleife, Verzweigung, Funktion
Funktionen	Parameter, Argumente, benannte Argumente, Rückgabewert, Lambdas
Literale	Zahlen, Strings, Listen, Mengen, Zuordnungen, List Comprehension

Kommandozeile

*R*ead-*E*valuate-*P*rint-*L*oop

```
sage: def f(i):
....:     return i^2
....:
sage: f
<function f at 0x7f13d5cb96e0>
sage: f(4)
16
```

Alternativ: `notebook()`

Navigation

<tab> Felder und Methoden vorschlagen
Achtung: geht nur bei konstanten Ausdrücken!

```
sage: l = []
sage: l.<tab>
sage: l.r<tab>
```

Pfeiltasten Letzte Befehle wiederholen

Unterstriche _, __, ___ sind das letzte bis drittletzte Ergebnis

magic commands `%ed <name>` für mehrzeiliges Bearbeiten
Vim geht mit Strg+C, `:wq`, <Enter> wieder weg :)

Hilfe

Fragezeichen-Operator Hilfe zum Objekt anzeigen

Achtung: geht nur bei konstanten Ausdrücken!

```
sage: l = {}
sage: l?
sage: help(l)
sage: FiniteField?
```

Doppeltes Fragezeichen zeigt Quellcode an (wenn verfügbar).

Online-Hilfe `manual()` für Befehlsreferenz

`tutorial()` für Anleitungen

Konstruktoren

Sage bringt eine Menge Know-How mit, z.B.

- Symbolische Berechnungen
  ```
  solve([0 == x^3 + 486662*x^2 + x], x)
  ```
- Plots in 2D und 3D
- Mathematische Strukturen
  ```
  FiniteField(431)['x'].quotient(x^2 + x + 1)
  ```
- Alles, was Python kann
- uvm: `sage.<tab>`

Ende

Fragen?

Dann wirds jetzt praktisch.