SAGE-Tutorium 04 im SoSe 2009

Lars Fischer*

20.05.2009

Inhaltsverzeichnis

*WWW: `http://w3.countnumber.de/fischer`, EMail: vorname.nachname (bei der) uni-siegen.de

1 Wiederholung

- Rekursion Teil 2
- Verzweigung
- Boolesche Ausdrücke, insbesondere Auswertung bei UND und ODER Ketten
- Ordnung und Primitivwurzel
- `(a^t % m)` für großes a und t und m dauert ewig vor allem wenn man es eine Million-mal aufruft.
- Im Vergleich dazu ist `Mod(a,m)^t` wesentlich schneller. Genauso ein `for a in IntegerModRing(m): a^t ....` , weil dann `a` automatisch eine Restklasse ist, die während der Potenzierung immer wieder reduziert werden kann.

2 Listen, Generatoren und Multi-Ranges

Mit `mrange` und `xmrange` laufen wir über alle möglichen Paare, Tripel, ...

```
mrange?
```

```
print len(mrange([2,3,4]))
print mrange([2,3,4])
```

```
Obst=["Apfel", "Erdbeer", "Kirsch"]
Dessert=["kuchen", "pudding"]
Sahne=["","mit Sahne"]

mrange( [len(Obst), len(Dessert)] )
```

```
for o,d,s in mrange([len(Obst), len(Dessert), len(Sahne) ] ):
    print "Lecker", Obst[o]+Dessert[d]+Sahne[s]
```

```
for x,y in mrange([10,10]):
    p=x^2+y^2
    if is_prime(p):
        print x,y,p, t%4
```

2.1 Generatoren/Iteratoren

Angenommen wir suchen in einer riesigen Liste ein Element mit bestimmten Eigenschaften. Wir sind mit dem ersten Element, das die Eigenschaften hat, zufrieden.

`mrange` berechnet zuerst die gesamte Liste. Das dauert und verbraucht Speicher. Die Funktion `xmrange` erzeugt einen Iterator. Dieser berechnet immer nur **das nächste** Element. Das geht schnell und verbraucht nur wenig Speicher.

Genauso `range` und `xrange`.

Da ein Generator immer nur das nächste Element liefert kann folgendes passieren, wenn der Generator erschöpft ist:

```
G=( i for i in [1..10]) # wie List Comprehension, nur mit einem
                        # Generator als Ergebnis

print "erster Durchlauf"
for k in G:
    print k,
print "\nzweiter Durchlauf:"
for k in G:
    print k,
```

3 Alles ist Referenz

Bisher habe ich den Eindruck vermittelt, dass ein Name und die Daten, die unter diesem Namen gespeichert sind, ein und dasselbe sind. Das ist nicht richtig, Name und Daten sind zwei verschiedene Dinge, die voneinander zu trennen sind. (Bild malen, Namensraum und Pfeile auf Speicherbereiche.)

```
a=1
b=a
print a, b
print id(a), id(b)
```

Hier ist b nur ein andere Name, um auf die gleiche Speicherstelle (Ausgabe von `id`) zuzugreifen. Bei `a=1` wird ein neues Zahl-Objekt erzeugt. Der Name `a` wird damit verbunden. Und `b` ist nur ein anderer Name für diesen Speicherbereich.

```
b+=1 # hier passiert etwas
print a,b
print id(a), id(b)
```

Bei dem Inkrement wird nun endlich ein weiteres Zahl-Objekt angelegt und der Name `b` zeigt anschließend darauf.

Generell gilt die Faustregel, dass eine Zuweisung mit `=`, immer eine Referenz und niemals eine Kopie anlegt. Also wird durch das `=` nur eine Verbindung zwischen einem Datenbereich und einem Namen erstellt. Manchmal, abhängig von der rechten Seite, wird im Datenbereich ein neues Objekt erstellt.

Diese Details sind oft unwichtig; sehr oft, genau das, was man möchte. Nur manchmal möchte man wirklich mit einer Kopie arbeiten. Dazu gibt es die Methoden `copy.copy` und `copy.deepcopy`.

Besonders lehrreiche Beispiele ergeben sich bei Listen:

```
L1=[0,1,2]                    # neue Liste
L2=["Null", "Eins", "Zwei"]   # neue Liste
L3= L1                        # nur ein anderen Name
print id(L1), id(L3), id(L2)
L3.append("neu") # das verändert auch L1
print L1
```

```
import copy
L4=copy.copy(L1)
print id(L1), id(L3), id(L4)
L4[3] = "alt"
print L1, L3, L4
```

Das nächste Beispiel zeigt den Unterschied zwischen `copy` und `deepcopy`:

```
# Liste innerhalb einer Liste
L0=[0,1,2]
L1=[L0, 1]
L2= copy.copy(L1) # eine flache Kopie, engl. shallow-copy
print L1, L2
print id(L1), id(L2), id(L0)
L2.append(2)
print L1,L2
# Die Elementen INNERHALB sind immer noch die gleichen.
print id(L1[0]), id(L2[0])
L3=copy.deepcopy(L1) # eine aufwendige tiefe Kopie
# erst nach einer tiefen Kopie sind diese verschieden
print id(L1[0]), id(L3[0])
```

Die Dokumentation des copy-Moduls:

```
copy?
```

3.1 Ein Beispiel wo eine Kopie nötig ist

Meistens macht es SAGE/Python mit den Referenzen genau richtig und man muss sich keine Gedanken machen. Hier mal eine Beispiel wo ausnahmsweise eine Kopie nötig ist:

```
# zusammengesetzte Zahlen aussortieren
L=range(100)
i=0
while i < len(L):
    if not is_prime(L[i]):
        del L[i]
        # wenn wir z.B. die 8 löschen, rutscht die 9 auf ihren Platz
        # vor, im nächsten Schritt wird aber schon die 10 betrachtet
    i+=1
print L
```

```
# zusammengesetzte Zahlen aussortieren
L=range(100)
i=0
ArbeitsKopie = L  # geht auch schief, nur Referenzierung
#ArbeitsKopie= copy.copy(L) # Hier ist eine Kopie angebracht
#ArbeitsKopie= L[:] # Abkürzung für Listenkopie
while i < len(L):
    if not is_prime(L[i]):
        del ArbeitsKopie[ArbeitsKopie.index(i)]
    i+=1
print ArbeitsKopie, len(ArbeitsKopie)
```

Aufgabe 1: Wie muss die ursprüngliche Variante verändert werden, so dass

1. sie funktioniert,
2. ohne Kopie auskommt?

4 Batteries included

Man sagt von Python, dass es schon die Batterien enthält. Damit bringt man zum Ausdruck, dass es für fast jeden Wunsch etwas in der Standardbibliothek gibt. (Siehe `http://docs.python.org/library/`.) In SAGE wird diese Bibliothek um sehr viel Mathematik erweitert.

5 Funktionen höherer Ordnung – eine experimentelle Annäherung an die Arithmetischen Funktionen

5.1 Funktionen höherer Ordnung

Eine Funktion höherer Ordnung ist ein Funktion, die eine Funktion als Argument erhält oder eine Funktion als Ergebnis zurückgibt.

5.1.1 Beispiele

Ein einfaches Beispiel, das niemanden überaschen wird:

```
def f( a ):
    return a^2

#print integral( sin, 0 , pi/2)
#print integral( cos, 0 , pi/2)
print numerical_integral( f, 0 , 5)
```

`numerical_integral` ist eine Funktion die eine richtige Funktion aus dem Lebensraum der Programmiersprache bekommt. (Programmiert das mal in einer statisch getypten Programmiersprache.)

Wir können ebenso eine Funktion schreiben, die eine richtige Funktion zurückgibt:

```
def createAddFunction( plusWhat ):
    def f(argument):
        return argument+plusWhat

    return f

Plus2= createAddFunction(2)
Plus3= createAddFunction(3)
print type(Plus2), Plus2(1), Plus2(x)
print type(Plus3), Plus3(1), Plus3(x)

G=plot(Plus3, [0,5])
G+=plot(Plus2,[0,5], rgbcolor=(1,0,0))
G.show(ymin=-1)
```

5.2 Arithmetische Funktionen

Die arithmetischen Funktionen sind Funktionen $a : \mathbb{Z}_{\geq 1} \to \mathbb{C}$. In der Analysis haben wir so etwas eine Folge komplexer Zahlen genannt. In der Zahlentheorie sind die multiplikativen arithmetischen Funktionen von besonderem Interesse.

Definition. *Wir nennen eine arithmetische Funktion $f : \mathbb{Z}_{\geq 1} \to \mathbb{C}$* ***multiplikativ****, falls gilt:*

1. $f(mn) = f(m)f(n)$ *für alle* m, n *mit* $\mathrm{ggT}(m, n) = 1$
2. $f \not\equiv 0$, *d.h.* f *ist nicht die Funktion* $z \mapsto 0$

Wenn außerdem $f(mn) = f(m)f(n)$ *für alle* m, n *gilt, dann hießt* f ***stark multiplikativ****.*

Für eine multiplikative Funktion f gilt:

1. $f(1) = 1$
2. $f(n) = \prod_{p^\alpha || n} f(p^\alpha)$

Die zweite Eigenschaft zeigt, die Bedeutung der multiplikativen Funktionen: Sie sind durch die Werte auf den Primzahlpotenzen bereits eindeutig bestimmt.

Definition. *Es sei* f *eine arithmetische Funktion. Wir nennen*

$$F(n) := \sum_{d|n} f(d)$$

die ***Summatorische Funktion von*** f*.*

Definition. *Es seien* f, g *zwei arithmetische Funktionen. Dann nennen wir*

$$(f * g)(n) := \sum_{d|n} f(d) \cdot g\left(\frac{n}{d}\right) = \sum_{d_1 d_2 = n} f(d_1) \cdot g(d_2)$$

das ***Dirichlet-Produkt von*** f ***und*** g*.*

Es bezeichne C die arithmetische Funktion $z \mapsto 1$, dann ist $(C * f)$ die Summatorische Funktion von f.

5.3 Einige Beispiele

Damit das ganze nicht so abstrakt wirkt, übertragen wir die Beispiele aus dem Skript nach SAGE.

```
# Anzahl der Teiler
def d(n): return len(divisors(n))

#sigma und sigma_k gibt es schon:
#sigma?

def LiouvilleLambda(n): return (-1)^(len(prime_divisors(n)))

# Konstant 1
```

```
def C(n): return 1

# Neutrales Element der Gruppe(A^*, *)
def NeutralesElement(n):
    if n==1: return 1
    else: return 0

def identitaet(n): return n
```

In der Vorlesung gibt es eine Tabelle, die für verschiedene g die Summatorische Funktion G auflistet. Es ist z.B. `d` die Summatorische Funktion von `C`. Wir wollen die Tabelle überprüfen, dazu benötigen wir eine Vergleichsmethode.

```
# eine simple VergleichsFunktion
def vergleiche(f1, f2, MaxN=100):
    for n in [1..MaxN]:
        if f1(n) != f2(n):
            return False
    return True



def SummatoricFunction (f):
    return lambda n : sum( [ f(d) for d in divisors(n) ] )



Tabelle = [ (C, d),
            (identitaet, sigma),
            (lambda n: n^2, lambda n: sigma(n,2) ),
            (lambda n: n^3, lambda n: sigma(n,3) ),
            (euler_phi, identitaet )
            ]

for g, G in Tabelle:
    print vergleiche(SummatoricFunction(g),G)
```

Die Tabelle im Skript scheint also richtig zu sein.[1]

Wir prüfen nun experimentell, dass die Summatorische Funktion einer arithmetischen Funktion f das Dirichlet-Produkt mit C ist:[2]

```
def DirichletProdukt( f, g):
    def P(n):
        return sum( [ f( d ) * g( n/d ) for d in divisors( n ) ] )
    return P

for f1,f2 in Tabelle:
    vergleiche( DirichletProdukt( f1 , C),
                SummatoricFunction( f1 ) )
    vergleiche( DirichletProdukt( f2 , C),
                SummatoricFunction( f2 ) )
```

Schlussendlich prüfen wir noch die Möbius'sche Umkehrfunktion:

```
for g,G in Tabelle:
    # G ist die Summatorische Funktion von g (aus der Tabelle
    # übernommen),
    # (G * moebius ) ist wieder g:
    vergleiche(g, DirichletProdukt(G, moebius))
```

6 Fragen von mir zum Tutorium

- Wie kann das Tutorium besser werden? Habt Ihr konkrete Vorschläge oder Wünsche?
- Haben die Lehrer Lust und Vorschläge für interaktive Worksheets, die am Ende des Semester vorgestellt werden?
- Welche weiteren Themen interessieren Euch:
  - OOP
  - LaTeX
  - SQL und Datenbanken
  - GUI-Programmierung

[1] Die Vergleichsfunktion ersetzt natürlich keinen mathematischen Beweis. Sie könnte aber eine Vermutung bestätigen oder widerlegen.

[2] Das könnt Ihr ausnahmsweise im Kopf schneller beweisen.

- Wie werde ich (also Ihr) SAGE-Entwickler?
- Kryptographie
- Primzahltests
- Wie beschleunigt man langsamen Code? Profiling, Cython
- ...

7 Aufgaben

Aufgabe 2: Eine Zahl n heißt **abundante Zahl**, falls die Summe ihrer echten Teiler ($d|n$ und $d \neq n$), größer ist als n selber. Finde alle abundanten Zahlen kleiner 1000. Was fällt Euch dabei auf?

Aufgabe 3: Zwei natürliche Zahlen a, b heißen **befreundet**, falls gilt

1. a ist die Summe der echten Teiler von b,
2. b ist die Summe der echten Teile von a.

Wieviele Paare befreundeter Zahlen mit $a, b <$500 gibt es?

Hinweis: da die kleiner Zahl abundant ist, könnt Ihr die Liste aus der vorigen Aufgabe benutzen.

Aufgabe 4: Finde die Summatorische Funktion von $\mathbb{E} : n \mapsto \begin{cases} 1 & n = 1 \\ 0 & sonst \end{cases}$, entweder durch Vergleichen mit den oben definierten Arithmetischen Funktionen oder durch Überlegen.

8 Nächstes Mal

- Quadratische Reziprozität
- ...

9 Quellcode

Das gesamte Worksheet ist als Text-Datei in dem PDF eingebettet.

- Im Acrobat-Reader lässt es sich unter dem Büroklammer-Symbol in der linken Leiste herunterladen.
- Okular zeigt es im File-Menu als Embeded Files an.

- Unter Linux kann man die Text-Datei auch mit
  pdftk Tutorium04.pdf unpack_files
  aus dem PDF herauslösen.

Anschließend lässt sich die Text-Datei mit der Upload-Funktion des SAGE-Notebooks hochladen.