Einführung in Sage
Kurzreferenz

Über die Kurzreferenz In der Kurzreferenz werden an vielen Stellen Platzhalter benutzt. Diese sind durch spitze Klammern gekennzeichnet. Zum Beispiel `<Ausd>`. Wenn in der Referenz steht, dass Zuweisungen in der Form

```
<Bezeichner> = <Ausdruck>
```

gemacht werden, dann ist ein Beispiel für einen solchen Code: Vier = 2 + 2

Überlebensregeln

- Das Auswerten eines Blocks erfolgt mit <SHIFT>+<ENTER>.
- Ein neues Eingabefeld erhält man durch Klicken auf den blauen, horizontalen Balken oder das Plussymbol unter jeder Zelle.
- Bei Python (und damit auch Sage) ist das Einrücken von Codezeilen von Bedeutung. Es werden damit die bei anderen Programmiersprachen üblichen Klammern ersetzt.

Nützliches

- Löschen aller eigenen Variablen und Zurücksetzen auf den Anfangsstatus: reset()
- Aktivieren des Feldes *Typeset* lässt alle Ausgaben von LaTeX rendern.
- Kommentare werden mit # eingeleitet.
- Dokumentation im Notebook mit HTML und LaTeX-Formeln: Durch Klick auf die Sprechblase neben dem Plussymbol unter jeder Zelle startet einen WYSIWYG Editor.
- Publish: Im Notebook kann durch Klicken des *Publish*-Reiters das Notebook für alle offen gelegt werden.

Hilfefunktionen

- Autovervollständigung : Mit der <TAB>-Taste erhält man alle möglichen Funktions- und/oder Variablen-Namen im gegebenen Kontext.
  Dies gilt insbesondere auch für Objektfunktionen (<Objekt>.<Funktion()>).
- <Befehl>? : Gibt ausführliche Hilfe zu Befehl an.
- <Befehl>?? : Gibt den Quellcode von Befehl an.
- help(<Befehl>) : Öffnet ein Hilfefenster zu Befehl.
- search_doc('<Begriff>') : Sucht in der Hilfe nach <Begriff>.
- Dokumentation:
  - Sage (lokal): `file:///usr/local/sage-5.5/devel/sage-main/doc/output/html/en/index.html`
  - Sage (Hauptseite): `http://www.sagemath.org/doc/index.html`
  - Python: `http://docs.python.org/`

Objektorientierung Herausfinden der Klasse eines Objekts:

```
type(<object>)
```

Datentypen Liste/Tuple: list(), tuple()

- Konstruktion

```
<Bez> = [<Wert1>,<Wert2>,...] #Liste
<Bez> = (<Wert1>,<Wert2>,...) #Tupel
```

Dictionaries: dictionaries

- Deklarieren eines Dictionaries:

```
<Bez> = {<Index1>:<Wert1>,...}
```

- Beispiel:

```
Auto = {'Marke':'VW','Typ':'Up','Km':150000}
Auto['Marke']       # gibt 'VW' aus
```

map() und map_threaded(): map()

(Rekursive) Auswertung der einstelligen Funktion auf eine (verschachtelte) Liste.

```
map_threaded(<Funktion>,<Menge oder Liste>)
```

Beispiel:

```
map(sqrt,[4,144,16])            #[2,12,4]
map_threaded(sqrt,[[25,16],9]   #[[5,4],3]
```

filter: filter()

Filtert nach Wahrheitswert der übergebenen Funktion.

```
filter(<Funktion>,<Menge oder Liste>)
```

Zahlen

- Zahlenmengen/Körper:

ZZ	Ganze Zahlen
QQ	Rationale Zahlen
RR	Reelle Zahlen
CC	Komplexe Zahlen
GF(2)	Körper mit zwei Elementen

- Einige wichtige Funktionen für Zahlen:

abs	Absolutbetrag	ceil	Aufrunden
sign	Vorzeichen	floor	Abrunden
arg	Argument	round	Runden
sqrt	Wurzel	n	num. Näherung

- Annahmen: assume()

```
assume(<Annahme>)
```

Achtung: Annahmen werden mit reset() nicht wieder zurückgesetzt. Dafür gibt es den Befehl forget().

Matrix: matrix()

- Deklaration

```
matrix(<Koerper>,[[a11,...],[a21,..],..])
```

Dabei ist die Angabe des Körpers/Gruppe meist optional.
Beispiel:

```
matrix([[1,2],[3,4]])
```

Einige Funktionen für Matrizen:

det	Determinante
eigenvalues	Eigenwerte
inverse	Inverse berechnen
rank	Rang der Matrix bestimmen
right_kernel	Kern der Matrix bestimmen

Vektor: vector()

- Deklaration

```
vector([v1,v2,..])
```

Vektorräume: vectorspace()

- Deklaration

```
vectorspace(<Koerper>,<Dimension>)
```

- Lineare Hülle:

```
span([<Vec1>,<Vec2>,...],<Koerper>)
```

Abfragen: if

- Syntax:

```
if <Boolscher Ausdruck>:    #z.B. x==2
    <Code-Block>
else:
    <Code-Block>
```

Schleifen

- Einzeilige for-Schleife

```
[<Ausd(Bez)> for <Bez> in <Liste> if <Bed>]
```

- for-Schleife: **for**

```
for <Bezeichner> in <liste>:
    <Code-Block>
```

- while-Schleife: **while()**

```
while <Bedingung>:
    <Code-Block>
```

Funktionen

- Mathematische Funktionen (Ausdrücke)

```
<Bez>(<Arg1>,<Arg2>,...) = <Ausdruck>
```

- einzeilige Deklaration: **def**

```
def <Bez> (<Arg1>,..): return <Rückgabewert>
```

- normale Deklaration:

```
def <Bez><(<arg1>,<arg2>,..)>:
    <Code-Block>
    return <Rückgabewert>
```

Beispiele:

```
Summe(x,y) = x+y

def Summe(x,y): return x+y

def Summe(x,y):
    s = x+y
    return s
```

Grafik: plot() / plot3d()

- 2D/3D Plot

```
plot(<Funktion>,(x,a,b),<Optionen>,...)
plot3d(<Funk>,(x,a,b),(y,c,d),<Optionen>,...)
```

Einige mögliche Optionen:

color	Farbe z.B. 'red', '#FF0000', (1,0,0)
plot_points	Bildauflösung
opacity	Transparenz (bei 3D Plots)
aspect_ratio	Seitenverhältnis der Achsen

Beispiel:

```
plot(x^2,(x,-2,2),color='red')
```

Summen: sum()

- Aufaddieren von Zahlen:

```
add([<Summand1>,<Summand2>,..])
```

- Symbolischer Summenausdruck

```
sum(<Ausdr>,<Var>,<Start>,<Stop>)
```

Achtung: Der symbolische Summenausdruck kann von Sage nicht immer in einen Zahlwert umgewandelt werden. Der symbolische Summenoperator kann auch Reihen vereinfachen. Beispiel:

```
sum(x^(-2),x,1,oo)      #1/6*pi^2
```

Grenzwerte: limit()

- Verhalten von Funktionen an Grenzwerten:

```
limit(<Ausdr>,<Variable>=<Grenzwert>,dir=<
    Richtung>)
```

Beispiel:

```
limit(e^(-1/x), x=0, dir='right')
```

Differentiation: diff()

- Ableitungen:

```
diff(<Ausdruck>,<var1>,<var2>,<var3>,...)
diff(<Ausdruck>,<var>,<anzahl>)
```

Beispiele:

```
diff(x^2*y^2,x,y)    #6*x^2*y
diff(x^10,x,3)       #720*x^7
```

Taylorformel: taylor()

- Taylorapproximation:

```
taylor(<funktion>,<var>,<stelle>,<grad>)
```

Gleichungen:

- Exaktes Lösen von Gleichungen: **solve()**

```
solve([<Gleichung1>,<Gleichung2>,...],<Var>)
```

Bei nur einer Gleichung, kann die Liste auch weggelassen werden. Beispiel:

```
S=solve(x^2-4 == 0,x) #Ergebnis: [x==2,x==-2]
```

Zugreifen auf die Lösung:

```
S[0].rhs() #Ergebnis: 2
S[1].rhs() #Ergebnis: -2
```

- Numerisches Lösen: **find_root()**

```
find_root(<Gleichung>,<uG>,<oG>,<Toleranz>)
```

Findet Lösungen im Intervall $[< uG >, < oG >]$.
Beispiel:

```
find_root(cos(x)==sin(x),0,2)
```

Integrale: integrate()

- bestimmte/unbestimmte Integrale:

```
integrate(<funktion>,<var>,[<uG>,<oG>])
```

Strings/Zeichenketten und Ausgabe: string

- Deklaration:

```
<Bezeichner> = '<Inhalt>'
```

- Zu Strings konvertieren: **str()**

```
str(<vorher kein String>)
```

- Stringformatierung: **format**

```
print ("Text {<format>} und {<format>}... ".
    format(x,y,...))
```

wichtigsten Formate:

- :d : integer (Ganze Zahl)
- :f : Nachkommastellen-Notation
- :e : Exponential-Notation