Sage 1: Einführung, Programmieren, Zahlen

Grundlagen

Worksheets & Variablen

Einfache Ausgaben mit Worksheet-Boxen

Dieses Dokument ist ein Sage-Worksheet. In die Eingabeboxen werden Rechnungen/Befehle eingegeben, durch drücken auf "evaluate" oder strg+enter ausgeführt werden. Das Ergebnis wird unter der Box angezeigt.

Um erklärenden Text zwischen den Eingabeboxen hinzuzufügen, kann unter dem "Edit"-Button HTML-Code eingefügt werden. Mathematische Formeln im erklärenden Text können in

LaTeX-Syntax eingegeben werden, zum Beispiel: $\int_0^{\pi/2} \cos x \, dx = 1$.

Einfache Taschenrechner-artige Eingaben:

```
5+4
```

9

Bei mehreren Eingaben werden alle Werte berechnet, aber nur das Ergebnis der letzten Rechnung wird angezeigt:

```
5*3
```

```
5-4
```

1

Um Ergebnisse weiterverwenden zu können, kann man es einer Variable zuweisen (hier mit dem Namen "a"). Da das Ergebnis einer Zuweisung standardmäßig nicht angezeigt wird, schreiben wir noch eine Zeile, in der das Ergebnis von "a" (also sein Wert) angezeigt wird:

```
a = 5*2
```

```
a
```

10

Anders als z.B. in C muss man den Typ der Variablen nicht explizit angeben, sondern er ergibt sich aus dem Wert dessen, was zugewiesen wird. Im obigen Beispiel ist `int*int=int`, also ist `a` ein Integer. Man kann den Wert von `a` jetzt auch mit Werten eines anderen Types überschreiben; anders als in C ist der Typ von `a` nicht statisch, sondern dynamisch. Nach der folgenden Zuweisung ist der Typ von `a` eine Kommazahl (float):

```
a = 5.0/2.0
```

```
a
```

2.5000000000000000

Und jetzt ein String:

```
a = "asdf"
```

```
a
```

'asdf'

Neben der normalen Ausgabe (im gleichen Format wie die Eingabe) kann Sage Ergebnisse auch in LaTeX-Formatierung ausgeben (mit "show"). Ebenso kann man sich den LaTeX-Quellcode dazu anzeigen lassen (mit "latex"). Analog gibt es Darstellungen für HTML usw.

```
show(1/10)
latex(1/10+2/10)
```

$$\frac{1}{10}$$

```
\frac{3}{10}
```

Sage vs. Python

Neben den Grundrechenarten funktionieren auch viele weitere Operatoren, die man aus C und Python gewohnt ist, etwa:

```
5%2
```

1

```
5<<2
```

20

Die Datentypen und Operatoren in Sage sind aber etwas "mathematischer" als in C und Python. Der Datentyp des folgenden Ausdrucks beispielsweise ist "rationale Zahl" (Bruch), anstatt beispielsweise Integer mit Wert 0 (wie in C) oder Gleitkommazahl. Der Wert wird also exakt gespeichert und nicht durch eine Gleitkommazahl approximiert. In Binärdarstellung hat 1/10 nämlich keine exakte, endliche Darstellung wie 0.1 in dezimal, sondern unendlich viele, periodische Nachkommastellen: 0.00011001100110011... Eine Gleitkommadarstellung würde diese Kommastellen irgendwo abschneiden, und das Ergebnis wäre nur mehr gerundet. In Sage hingegen wird der Wert als "Bruch mit Nenner 10 und Zähler 1" gespeichert, und es wird damit exakt weitergerechnet. Ähnliches gilt auch für Wurzeln usw., sofern man sie nicht explizit in floats umwandelt (z.B. durch Multiplikation mit einem Float).

```
1/10
```

1/10

Wie oben im Bruchbeispiel schon gesehen, verhalten sich einige Operatoren anders, "mathematischer". Das liegt daran, dass eingegebene Zahlen automatisch einen Sage-Zahlen-Datentyp bekommen statt einen Python-Zahlen-Datentyp (z.B. Integer statt int), um beispielsweise eine bessere Genauigkeit zu erreichen, und eben auch die Operatoren passender belegen zu können. Ein weiteres Beispiel ist der Zirkumflex, der in Sage das tut, was man als Mathematiker erwartet, nämlich exponentieren:

```
5^2
```

25

Um die C- (und Python-)Verhaltensweise von / und ^ zu bekommen (nämlich Ganzzahldivision und bitweises XOR), gibt es separate Operatoren:

```
5//2
```

2

```
5^^2
```

7

Es funktioniert auch der Python-Exponential-Operator:

```
5**2
```

25

Die kombinierten Zuweisungsoperatoren +=, -= usw. aus C funktionieren auch (aber führen jeweils die Sage-Entsprechung aus, etwa bei /)


```
22838915, 56477245, 426406280327, 2132031401635, 469046908359
23452345417985]
```

Einige Funktionen erlauben sowohl die Schreibweise `var.fun()` als auch `fun(var)`, zum Beispiel:

```
factor(a) == a.factor()
```

```
True
```

Um die Faktoren weiter verwenden zu können, wandelt man das Ergebnis am besten in eine Liste um (die Liste enthält Tupel der Form (Primzahl, Vielfachheit)):

```
list(factor(a))
```

```
[(5, 1), (11, 1), (415253, 1), (1026859, 1)]
```

Ziffern (binär, dezimal, ...) einer Zahl:

```
print a.digits()
print len(a.digits())
print sum(a.digits())
print a.digits(base=2)
```

```
[5, 8, 9, 7, 1, 4, 5, 4, 3, 2, 5, 4, 3, 2]
```

```
14
```

```
62
```

```
[1, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 1, 1, 1, 1, 1, 0,
0, 1, 0, 0, 1, 1, 0, 1, 1, 0, 0, 0, 1, 0, 1, 0, 1, 0, 1]
```

Wenn nicht genau sicher ist, wie eine Funktion heißt, gibt es mit der Tabulator-Taste eine Autovervollständigung bzw. eine Liste passender Funktionen (z.B. `div+TAB`, `a.+TAB` für eine Liste der möglichen Funktionen von Variable `a`).

Die Dokumentation zu einer Funktion (kurze Beschreibung, mögliche Parameter, Beispiele) erhält man mit `?`:

```
gcd?
```

File: /usr/local/sage-6.3/local/lib/python2.7/site-packages/sage/rings/arith.py

Type: <type 'function'>

Definition: gcd(a, b=None, **kwargs)

Docstring:

The greatest common divisor of a and b, or if a is a list and b is omitted the greatest common divisor of all elements of a.

INPUT:

- a, b - two elements of a ring with gcd or
- a - a list or tuple of elements of a ring with gcd

Additional keyword arguments are passed to the respectively called methods.

OUTPUT:

The given elements are first coerced into a common parent. Then, their greatest common divisor *in that common parent* is returned.

EXAMPLES:

```
sage: GCD(97, 100)
```

```
1
```

```

sage: GCD(97*10^15, 19^20*97^2)
97
sage: GCD(2/3, 4/5)
2/15
sage: GCD([2,4,6,8])
2
sage: GCD(srange(0,10000,10)) # fast !!
10

```

Note that to take the gcd of n elements for $n \neq 2$ you must put the elements into a list by enclosing them in `[..]`. Before #4988 the following wrongly returned 3 since the third parameter was just ignored:

```

sage: gcd(3,6,2)
Traceback (click to the left of this block for traceback)
...

```

Für Modulo-Rechnung kann man in Sage natürlich den normalen Modulo-Operator verwenden:

```

5 % 3
-5 % 3

```

```
1
```

Möchte man längere Rechnungen modulo einer fixen Zahl ausführen, definiert man sich am besten den dazugehörigen "Modulo-Rechnungs-Ring", zum Beispiel für Rechnung modulo 7:

```

Z7 = Integers(7)
a = Z7(4)
-a; a^(-1); a * a^(-1); 5*a; parent(a)

```

```
3
```

```
2
```

```
1
```

```
6
```

```
Ring of integers modulo 7
```

Z7 und a bringen auch jede Menge weitere nützliche Funktionen mit - einfach ausprobieren:

```
list(Z7); Z7.random_element()
```

```
[0, 1, 2, 3, 4, 5, 6]
```

```
4
```

Brüche und exakte Ausdrücke

Brüche von Ganzzahlen werden weder abgerundet (wie Ganzzahldivision in C) noch mit Gleitkommazahlen approximiert, sondern exakt gespeichert. Das gleiche gilt auch für viele Funktionsauswertungen:

```

3/5
sin(pi/3)

```

```
1/2*sqrt(3)
```

```
parent(3/5); parent(sin(pi/3)); parent(6/2)
```

```
Rational Field
```


9

Komplexe Zahlen

Für die Darstellung von komplexen Zahlen $z \in \mathbb{C}$ sind die Konstanten I und i auf die imaginäre Einheit vordefiniert; in Python (und notfalls auch in Sage) kann auch die Notation mit j verwendet werden:

```
print sqrt(-1)
z = 3 + 4*i
z2 = 3 + 4j
print z
print z2
parent(z); parent(z2)
```

```
I
4*I + 3
3.0000000000000000 + 4.0000000000000000*I
Symbolic Ring
Complex Field with 53 bits of precision
```

```
z3 = z2 / (4 + 2*I)
z3 in CC
```

```
True
```

Um separat auf Imaginärteil, Realteil, Absolutbetrag usw. einer komplexen Zahl zugreifen zu können, gibt es diverse Hilfsfunktionen:

```
z3; imag(z3); real(z3); abs(z3)
1.0000000000000000 + 0.5000000000000000*I
0.5000000000000000
1.0000000000000000
1.11803398874989
```

Betrag (abs) und Winkel (arg) für Polarkoordinaten:

```
print abs(z)
print arg(z)
print arg(z).n()
```

```
5
arctan(4/3)
0.927295218001612
```

Vorsicht, man kann I und i (genau wie andere Konstanten wie π , ...) auch versehentlich überschreiben:

```
I = 5
3 + 4*I
```

```
23
```

Wahrheitswerte (Bool)

Vergleiche und wahr-falsch-Fragen haben als Ergebnis-Datentyp Wahrheitswerte (Bool):

```
parent(5 in ZZ)
parent(3 < 5)
```

```
parent(4 * 5 == 20)
```

```
<type 'bool'>
```

Boolesche Werte und Bedingungen kann man mit "and, or, not" (entspricht in C &&, ||, !) verknüpfen:

```
5 in ZZ and not 3 < 5
```

```
False
```

Anders als in C sind auch mehrere Vergleiche hintereinander möglich:

```
-5 < -3 < -1
```

```
True
```

Bei Vergleichen mit exakten Ausdrücken ist es teilweise notwendig, das Ergebnis der Gleichung explizit in einen Wahrheitswert zu konvertieren:

```
sin(pi/3) == sqrt(3)/2  
bool(sin(pi/3) == sqrt(3)/2)
```

```
True
```

```
0.1*10 == 1.0
```

```
True
```