2025_12_05_sage_3

December 5, 2025

Mehr über Listen

[1]:
```
l=[0..9]
l
```

[1]:
```
[0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
```

Hier wird die Liste selbst verändert

[2]:
```
l.append(10)
```

[3]:
```
l
```

[3]:
```
[0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
```

Hier wird l selbst nicht verändert, das Resultat wird als neues Objekt im Output zurückgegeben.

[4]:
```
l+[4,7,32,5]
```

[4]:
```
[0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 4, 7, 32, 5]
```

[5]:
```
l*2
```

[5]:
```
[0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
```

Zugriff auf Teillisten ähnlich wie in Matlab. Achtung: obere Grenze ist nicht inkludiert.

[6]:
```
l[2:7]
```

[6]:
```
[2, 3, 4, 5, 6]
```

[7]:
```
l[2:10:2]
```

[7]:
```
[2, 4, 6, 8]
```

Teillisten können auch direkt überschrieben werden.

[8]:
```
l[2:10:2]=[42]*4
```

[9]:
```
l
```

[9]:
```
[0, 1, 42, 3, 42, 5, 42, 7, 42, 9, 10]
```

[10]:
```
[42]*4
```

[10]:
```
[42, 42, 42, 42]
```

Funktionale Programmierung (basierend auf Listen und Funktionen): map, filter, reduce

[11]:
```
l=[0..9]
l
```

[11]:
```
[0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
```

map(f,[x1,...,xn) werdet die Funktion f auf jedes Element der Liste an. Ergebnis: [f(x1),...,f(xn)]. Das wird allerdings nicht direkt als Liste zurückgegeben, sondern als map Objekt. Darüber kann man iterieren, aus Effizienzgründen werden die einzelnen Elemente erst dann bestimmt, wenn sie tatsächlich benötigt werden ("lazy evaluation").

[12]:
```
l2=map(factorial,l)
l2
```

[12]:
```
<map object at 0x79a4deabb8b0>
```

[13]:
```
for x in l2:
    print(x)
```

```
1
1
2
6
24
120
720
5040
40320
362880
```

Über das map-Objekt kann nur einmal iteriert werden, danach ist es leer.

[14]:
```
for x in l2:
    print(x)
```

[15]:
```
l
```

[15]:
```
[0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
```

Das map-Objekt kann man auch einfach direkt in eine Liste umwandeln.

[16]:
```
l2=list(map(factorial,l))
l2
```

[16]:
```
[1, 1, 2, 6, 24, 120, 720, 5040, 40320, 362880]
```

filter(f,[x1,...,xn]) wählt aus der Liste jene Elemente aus, für die die Funktion f zu True evaluiert.

[17]:
```
l3=filter(is_prime,l)
l3
```

[17]:
```
<filter object at 0x79a4deaba1d0>
```

Geliefert wird wieder ein iterierbares Objekt. Hier sieht man wieder: dieses wird nur einmal durchlaufen, einmal aufgerufene Objekte sind danach nicht mehr vorhanden.

[18]:
```
next(l3)
```

[18]:
```
2
```

[19]:
```
list(l3)
```

[19]:
```
[3, 5, 7]
```

[20]:
```
list(l3)
```

[20]:
```
[]
```

[21]:
```
l
```

[21]:
```
[0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
```

List comprehensions

List comprehensions bieten eine angenehme, intuitive und leicht lesbare Möglichkeit, dieselbe Funktionalität wie map und filter zu erreichen.

[22]:
```
list(map(factorial,l))
```

[22]:
```
[1, 1, 2, 6, 24, 120, 720, 5040, 40320, 362880]
```

Dasselbe als list comprehension. Die Bedeutung des Codes sollte hier intuitiv klar sein.

[23]:
```
[factorial(n) for n in l]
```

[23]:
```
[1, 1, 2, 6, 24, 120, 720, 5040, 40320, 362880]
```

Filter als list comprehension.

[24]:
```
[n for n in l if is_prime(n)]
```

[24]:
```
[2, 3, 5, 7]
```

Beides kombiniert.

[25]:
```
[factorial(p) for p in l if is_prime(p)]
```

[25]:
```
[2, 6, 120, 5040]
```

Dasselbe mit map und filter: deutlich schwerer lesbar.

[26]:
```
list(map(factorial,filter(is_prime,l)))
```

[26]:
```
[2, 6, 120, 5040]
```

[27]:
```
l
```

[27]:
```
[0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
```

reduce(f,l) verschmilzt die Elemente der Liste paarweise, wobei iterativ auf je zwei Elemente die Funktion f angewandt wird. Z.B. reduce(f,[x1,x2,x3,x4]) liefert f(f(f(x1,x2),x3),x4)

[28]:
```
reduce(operator.add,l)
```

[28]:
```
45
```

[29]:
```
1+2+3+4+5+6+7+8+9
```

[29]:
```
45
```

[30]:
```
add(l)
```

[30]:
```
45
```

[31]:
```
reduce(max,l)
```

[31]:
```
9
```

Teste Klammerung mit nicht-kommutativer Funktion

[38]:
```
var('x,y,z')
```

[38]:
```
(x, y, z)
```

[39]:
```
reduce(pow,[x,y,z])
```

[39]:
```
(x^y)^z
```

Weitere Datentypen

[41]:
```
L=[0..4]*2
L
```

[41]:
```
[0, 1, 2, 3, 4, 0, 1, 2, 3, 4]
```

Mengen: wie Listen, aber ohne Wiederholungen und Ordnung irrelevant

[57]:
```
S=set(L)
S
```

[57]:
```
{0, 1, 2, 3, 4}
```

[58]:
```
parent(S)
```

[58]:
```
<class 'set'>
```

[59]:
```
S1=set([4,5,7,8])
```

[60]:
```
S-S1
```

[60]:
```
{0, 1, 2, 3}
```

[61]:
```
S1={4,5,7,8}
```

[62]:
```
S1
```

[62]:
```
{4, 5, 7, 8}
```

[63]:
```
parent(S1)
```

[63]:
```
<class 'set'>
```

[64]:
```
[1,2]==[2,1]
```

[64]:
```
False
```

[65]:
```
{1,2}=={2,1}
```

[65]:
```
True
```

[66]:
```
{1,2}=={1,2,2}
```

[66]:
```
True
```

Vereinigung

[67]:
```
S.union(S1)
```

[67]:
```
{0, 1, 2, 3, 4, 5, 7, 8}
```

Heute in der Vorlesung hat das nicht so gut funktioniert, hier ist die Erklärung dafür:

Die Klassen Set und set sind unterschiedlich. set ist die Python-Mengenklasse, Set eine Sage-Klasse mit leicht anderen Funktionen.

Eigentlich wollte ich die Python-Variante set vorstellen, habe aber versehentlich S=Set(L) verwendet.

Hier sind ein paar Demonstrationen dazu:

[84]:
```
S1={1,2,3}
S2=Set([3,4,5])
```

[85]:
```
parent(S1), parent(S2)
```

[85]:
```
(<class 'set'>, <class 'sage.sets.set.Set_object_enumerated_with_category'>)
```

Man kann set mit Set vereinigen, das Ergebnis ist ein set

[86]:
```
S1.union(S2)
```

[86]:
```
{1, 2, 3, 4, 5}
```

[87]:
```
parent(_)
```

[87]:
```
<class 'set'>
```

Man kann aber nicht Set nur mit Set vereinigen, und nicht mit set

[88]:
```
S2.union(Set(S1))
```

[88]:
```
{1, 2, 3, 4, 5}
```

[89]:
```
S2.union(S1)
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
Cell In[89], line 1
----> 1 S2.union(S1)

File /opt/sagemath/sage-10.7/src/sage/sets/set.py:1161, in Set_object_enumerated.
 ↪union(self, other)
   1145 """
   1146 Return the union of ``self`` and ``other``.
   1147
   (...)
   1158     [0, 1, 2, 3, c, c + 1, c^2, c^2 + 1, c^2 + c, c^2 + c + 1]
   1159 """
   1160 if not isinstance(other, Set_object_enumerated):
-> 1161     return Set_object.union(self, other)
   1162 return Set_object_enumerated(self.set().union(other.set()))

File /opt/sagemath/sage-10.7/src/sage/sets/set.py:248, in Set_base.union(self, X)
    246         return self
    247     return Set_object_union(self, X)
--> 248 raise TypeError("X (=%s) must be a Set" % X)
```

```
TypeError: X (={1, 2, 3}) must be a Set
```

Addition als Vereinigung funktioniert für set nicht, aber für Set schon.

[90]:
```
Set(S1)+S2
```

[90]:
```
{1, 2, 3, 4, 5}
```

[91]:
```
S1+set(S2)
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
Cell In[91], line 1
----> 1 S1+set(S2)

TypeError: unsupported operand type(s) for +: 'set' and 'set'
```

[ ]:
```

```

[92]:
```
L=[4,3,2,1]*2
L
```

[92]:
```
[4, 3, 2, 1, 4, 3, 2, 1]
```

Elegante Methode, um mehrfache Einträge einer Liste zu löschen (und die Liste "kanonisch" zu ordnen)

[93]:
```
L=list(set(L))
```

[94]:
```
L
```

[94]:
```
[1, 2, 3, 4]
```

Tupel

Wie Listen, aber unveränderbar.

[95]:
```
T=(3,4,5)
T
```

[95]:
```
(3, 4, 5)
```

Einträge eines Tupels können direkt Variablen zugewiesen werden.

[96]:
```
a,b,c=T
```

[97]:
```
a
```

[97]: 3

[98]:
```
b
```

[98]: 4

[99]:
```
c
```

[99]: 5

Das ist sehr praktisch und wird z.B. verwendet, wenn Funktionen mehrere Werte zurückgeben. Hier: Division mit Rest.

[100]:
```
q,r=21.quo_rem(4)
```

[101]:
```
q
```

[101]: 5

[102]:
```
r
```

[102]: 1

Vertauschen zweier Variablen.

Klassisch, mit einer zusätzlichen Buffervariable:

[103]:
```
a=2
b=1
```

[104]:
```
c=b
b=a
a=c
```

[105]:
```
a
```

[105]: 1

[106]:
```
b
```

[106]: 2

In einer Zeile mittels Tupeln:

[107]:
```
a,b=b,a
```

[108]:
```
a
```

[108]: 2

[109]:
```
b
```

[109]:
```
1
```

Zugriff auf Einträge eines Tupels in Iterationen:

[110]:
```
L=[(i,i^2,i^3) for i in [4,7,5,9]]
L
```

[110]:
```
[(4, 16, 64), (7, 49, 343), (5, 25, 125), (9, 81, 729)]
```

Klassisch:

[111]:
```
for T in L:
    print(T[0],T[2]-T[1])
```

```
4 48
7 294
5 100
9 648
```

Kürzer und leichter lesbar: Einträge gleich in der Iteration direkt benennen.

[112]:
```
for i,s,c in L:
    print(i,c-s)
```

```
4 48
7 294
5 100
9 648
```

Der wesentliche Unterschied zwischen Listen und Tupeln ist, dass Listen im Nachhinein verändert werden können (“mutable”), Tubel aber nicht (“immutable”).

[113]:
```
L=[1,2,3]
L
```

[113]:
```
[1, 2, 3]
```

[114]:
```
L[1]=pi
L
```

[114]:
```
[1, pi, 3]
```

[115]:
```
T=(1,2,3)
T
```

[115]:
```
(1, 2, 3)
```

[116]:
```
T[1]=pi
T
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
Cell In[116], line 1
----> 1 T[Integer(1)]=pi
      2 T

TypeError: 'tuple' object does not support item assignment
```

Dictionaries

Listen haben Einträge indiziert mit 0,1,2,...

Die Einträge von dictionaries können durch beliebige Objekte indiziert werden.

Geschrieben mit {...} als Menge von Paaren der Form k:v, wobei k der Schlüssel/Index und v der zugehörige Wert ist.

[117]:
```
D={'a':1, 'b':2, 25:15, ZZ:QQ}
D
```

[117]:
```
{'a': 1, 'b': 2, 25: 15, Integer Ring: Rational Field}
```

[118]:
```
D['b']
```

[118]:
```
2
```

[119]:
```
D[25]
```

[119]:
```
15
```

[120]:
```
D[4]
```

```
---------------------------------------------------------------------------
KeyError                                  Traceback (most recent call last)
Cell In[120], line 1
----> 1 D[Integer(4)]

KeyError: 4
```

Zuweisung neuer Werte.

[121]:
```
D[4]=72
```

[122]:
```
D
```

[122]:
```
{'a': 1, 'b': 2, 25: 15, Integer Ring: Rational Field, 4: 72}
```

Iteration über Tupel: geht über die Schlüssel

[123]:
```
for v in D:
    print(v)
```

```
a
b
25
Integer Ring
4
```

[124]:
```
list(D.keys())
```

[124]: ['a', 'b', 25, Integer Ring, 4]

[125]:
```
list(D.values())
```

[125]: [1, 2, 15, Rational Field, 72]

[126]:
```
for v in D:
    print(D[v])
```

```
1
2
15
Rational Field
72
```

Beispiel: mitspeichern von Funktionswerten. Wir haben gereits gesehen, dass der @cached_function Dekorator das automatisch macht. Jetzt können wir es auch sehr einfach händisch programmieren.

Automatisch (bereits gesehen, zu bevorzugen):

[127]:
```
@cached_function
def fib(n):
    if n<2:
        return 1
    return fib(n-1)+fib(n-2)
```

Händisch mit Dictionary:

[128]:
```
D={}
def fib(n):
    global D
    if n in D:
        return D[n]
    if n<2:
        D[n]=1
    else:
        D[n]=fib(n-1)+fib(n-2)
    return D[n]
```

[129]:
```
fib(4)
```

[129]:
```
5
```

[130]:
```
fib(31)
```

[130]:
```
2178309
```

Rechnen mit symbolische Ausdrücken

[131]:
```
reset('x,y,z')
```

[132]:
```
x
```

[132]:
```
x
```

[133]:
```
parent(x)
```

[133]:
```
Symbolic Ring
```

[134]:
```
x^2+4
```

[134]:
```
x^2 + 4
```

x ist automatisch als (mathematische) Variable definiert. Alle anderen Variablen müssen erst als solche definiert werden.

[135]:
```
y
```

```
---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)
Cell In[135], line 1
----> 1 y

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

[136]:
```
var('y')
```

[136]:
```
y
```

[137]:
```
parent(y)
```

[137]:
```
Symbolic Ring
```

[138]:
```
x+y
```

[138]:
```
x + y
```

[139]:
```
(x+y)^3
```

[139]:
```
(x + y)^3
```

Symbolische Ausdrücke werden nicht automatisch umgeformt/ausmultipliziert.

expand: ausmultiplizieren

[140]:
```
f=expand((x+y)^3)
```

[141]:
```
f
```

[141]:
```
x^3 + 3*x^2*y + 3*x*y^2 + y^3
```

factor: zusammenfassen

[142]:
```
factor(f)
```

[142]:
```
(x + y)^3
```

[143]:
```
factor(4562)
```

[143]:
```
2 * 2281
```

Obwohl beide obige Funktionen factor() heißen, geschehen hier verschiedene Dinge. Die globale Funktion factor() ruft im ersten Fall die Methode Expression.factor() auf, im zweiten Fall die Methode Integer.factor()

Viele mathematische Funktionen sind als symbolische Ausdrücke vordefiniert.

[144]:
```
sin(x)
```

[144]:
```
sin(x)
```

[145]:
```
parent(_)
```

[145]:
```
Symbolic Ring
```

Ableiten ist aufgrund der diversen Ableitungsregeln ein sehr mechanischer Prozess. Sage kennt die Ableitungen elementarer Funktionen

[146]:
```
diff(sin(x),x)
```

[146]:
```
cos(x)
```

Unter Anwendung der Ableitungsregeln können auch komplizierte Kombinationen aus diesen Funktionen zuverlässig abgeleitet werden.

[147]:
```
diff(exp(cos(x^5+3*x))/sin(x),x)
```

[147]:
```
-(5*x^4 + 3)*e^(cos(x^5 + 3*x))*sin(x^5 + 3*x)/sin(x) - cos(x)*e^(cos(x^5 +
3*x))/sin(x)^2
```

[148]:
```
show(_)
```

$$-\frac{\left(5\,x^4+3\right)e^{\left(\cos\left(x^5+3\,x\right)\right)}\sin\left(x^5+3\,x\right)}{\sin\left(x\right)}-\frac{\cos\left(x\right)e^{\left(\cos\left(x^5+3\,x\right)\right)}}{\sin\left(x\right)^2}$$

Integration ist viel schwieriger als ableiten. Einfache Integrale sind kein Problem für Sage

[149]:
```
integrate(sin(x),x)
```

[149]:
```
-cos(x)
```

Viele elementare Funktionen (Funktionen gebaut aus Polynomen, exp(), log() und trigonometrischen Funktionen mittels +,-,*,/ und Hintereinanderausführung) haben keine elementare Stammfunktion. Einige davon kann Sage dennoch integrieren, weil es entsprechende spezielle Funktionen kennt, mit denen sich die Stammfunktion ausdrücken lässt.

[150]:
```
integrate(exp(x^2),x)
```

[150]:
```
-1/2*I*sqrt(pi)*erf(I*x)
```

[151]:
```
show(_)
```

$$-\frac{1}{2}i\,\sqrt{\pi}\,\mathrm{erf}\left(i\,x\right)$$

[ ]:
```
erf?
```

Das stößt aber schnell an seine Grenzen

[153]:
```
integrate(sin(exp(sqrt(x))),x)
```

[153]:
```
integrate(sin(e^sqrt(x)), x)
```

[154]:
```
show(_)
```

$$\int \sin\left(e^{\sqrt{x}}\right)\,dx$$

Prinzipiell gibt es einen Algorithmus von Risch (1968), der zu jeder elementaren Funktion, die eine elementare Stammfunktion hat, diese berechnet, und sonst ausgibt, dass die Funktion keine elementare Stammfunktion hat. Dieser Algorithmus ist jedoch extrem kompliziert und (meines Wissens) noch in keinem Computeralgebrasystem vollständig implementiert. Stattdessen verwenden Cmputeralgebrasysteme bei komplizierteren Integralen oft heuristische Methoden, die leider auch falsche Ergebnisse liefern können.

Das folgende Integral wird inzwischen richtig berechnet. In Sage 8 lieferte es noch das falsche Ergebnis -1.

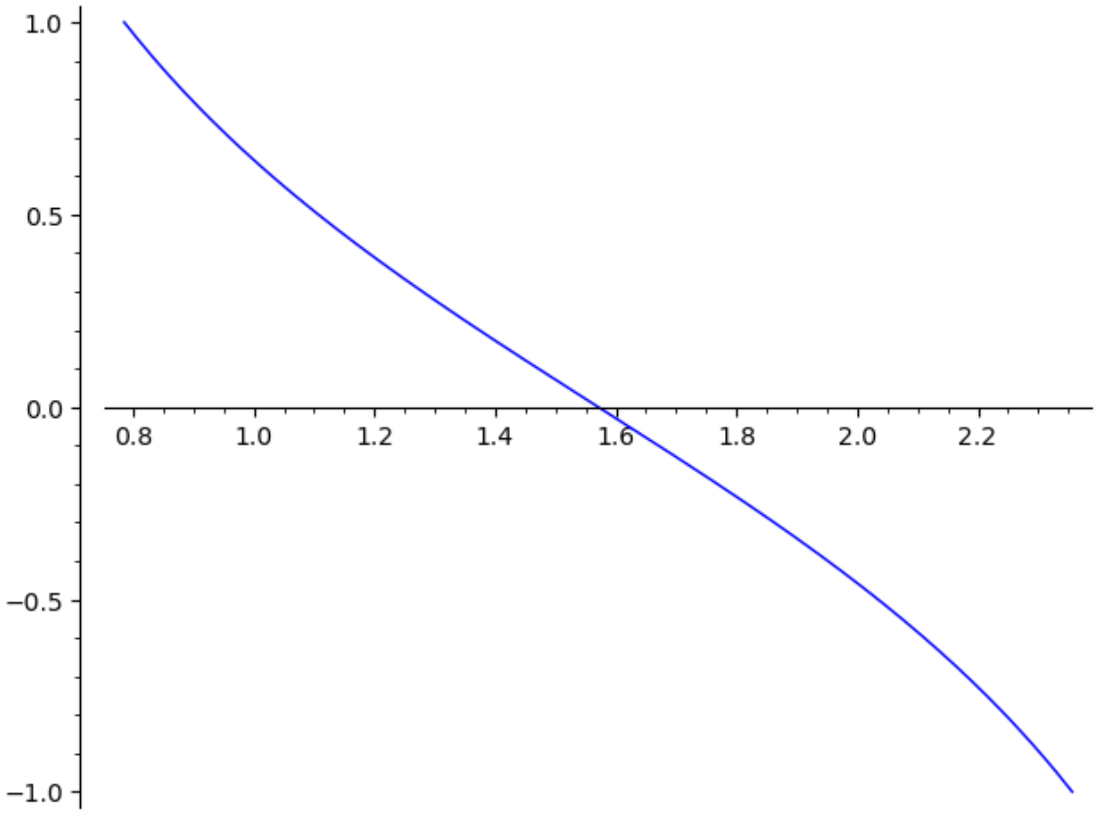
[156]:
```
integrate(abs(cos(x)),x,0,pi)
```

[156]:
```
2
```

Es gibt immer noch viele Integrale, die falsch berechnet werden.

[158]:
```
plot(cot(x),x,pi/4,3*pi/4)
```

[158]:

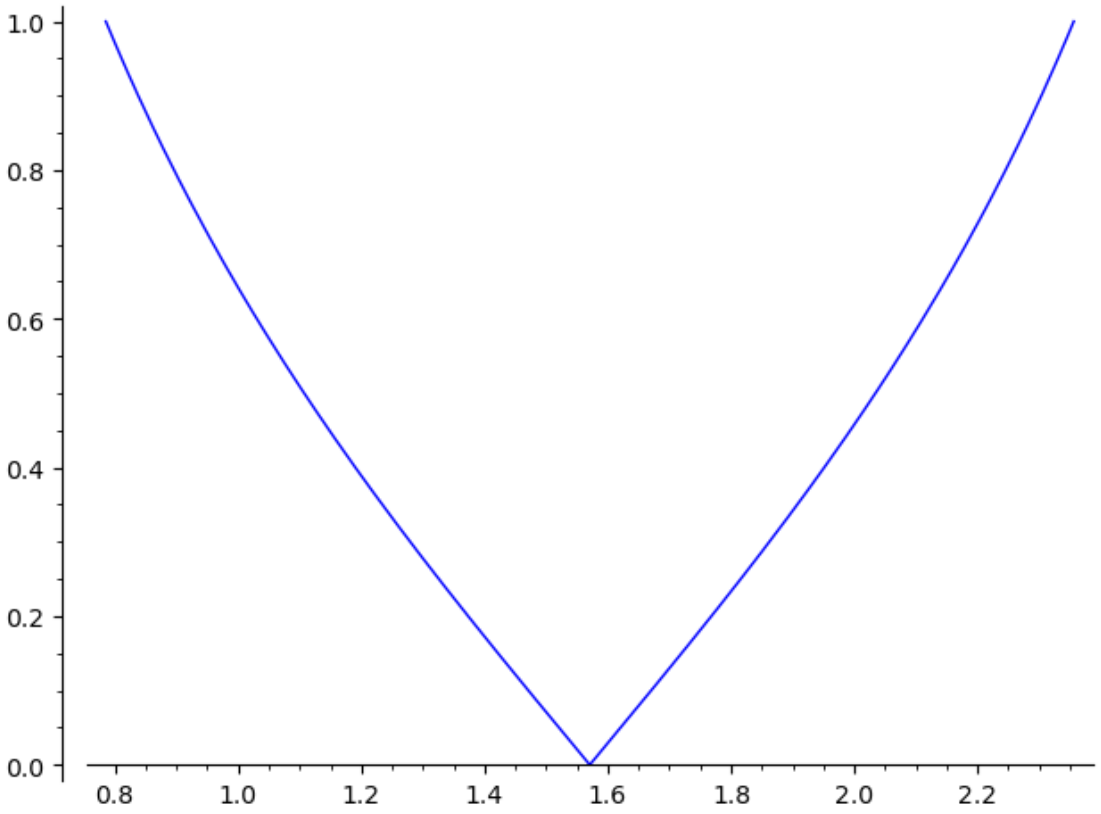


[159]:
```
integrate(cot(x),x,pi/4,3*pi/4)      # richtig
```

[159]: 0

[160]:
```
plot(abs(cot(x)),x,pi/4,3*pi/4)
```

[160]:

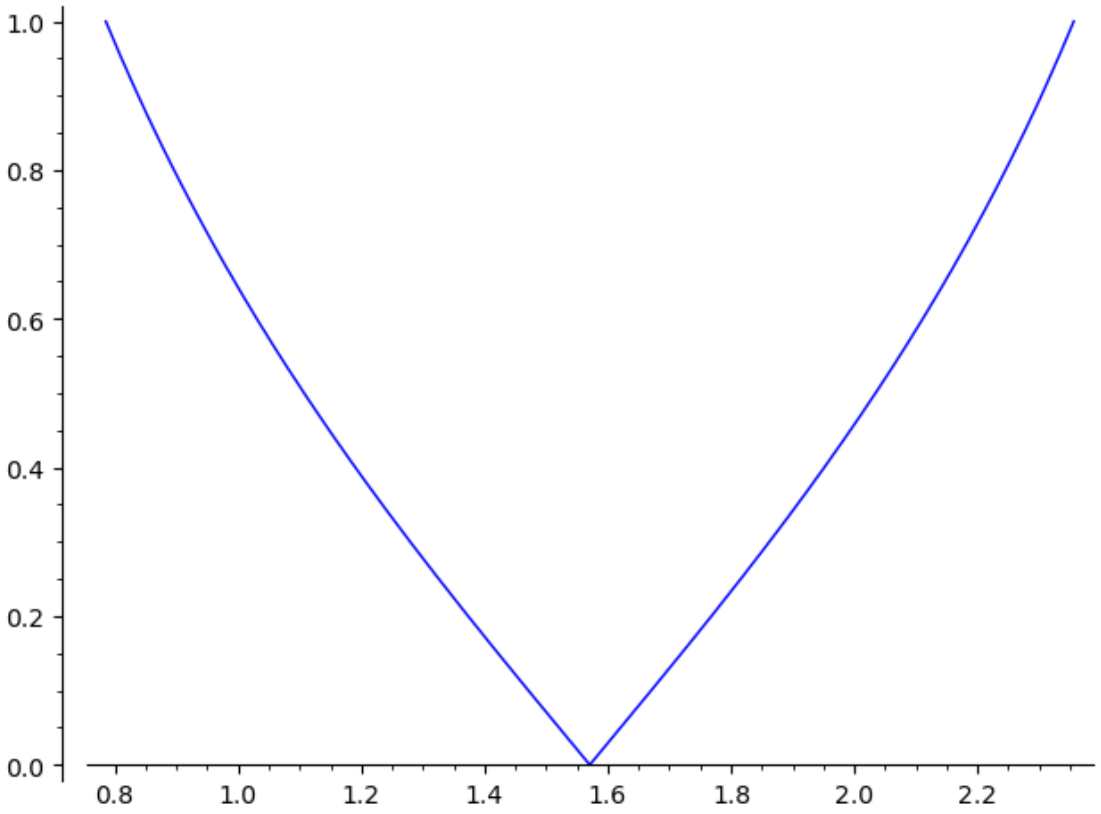


[162]:
```
integrate(abs(cot(x)),x,pi/4,3*pi/4)    # immer noch richtig
```

[162]:
```
-2*log(1/2*sqrt(2))
```

[163]:
```
plot(sqrt(cot(x)^2),x,pi/4,3*pi/4)    # gleiche Funktion anders geschrieben
```

[163]:



Obwohl es mathematisch dieselbe Funktion ist, wie vorher, wird hier das offensichtlich falsche Ergebnis 0 ausgegeben!

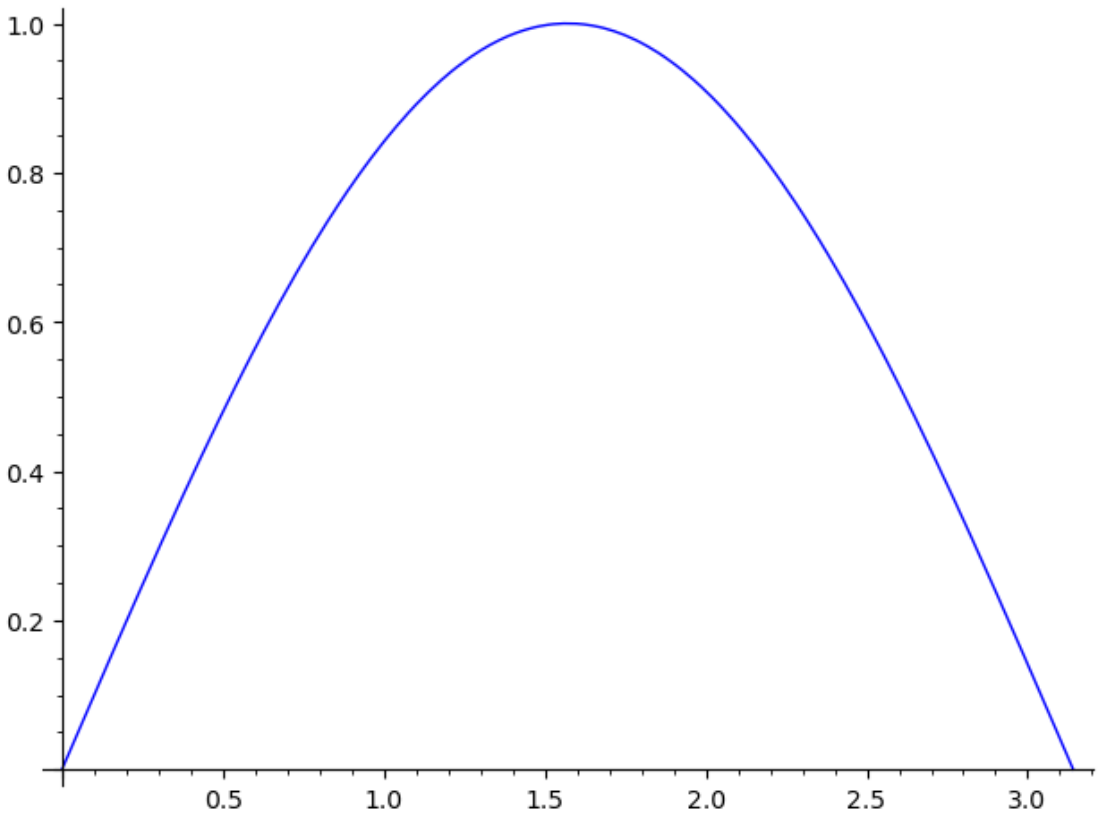
[165]:
```
integrate(sqrt(cot(x)^2),x,pi/4,3*pi/4)
```

[165]: 0

Einfache Grafik

[166]:
```
G=plot(sin(x),x,0,pi)
G
```

[166]:

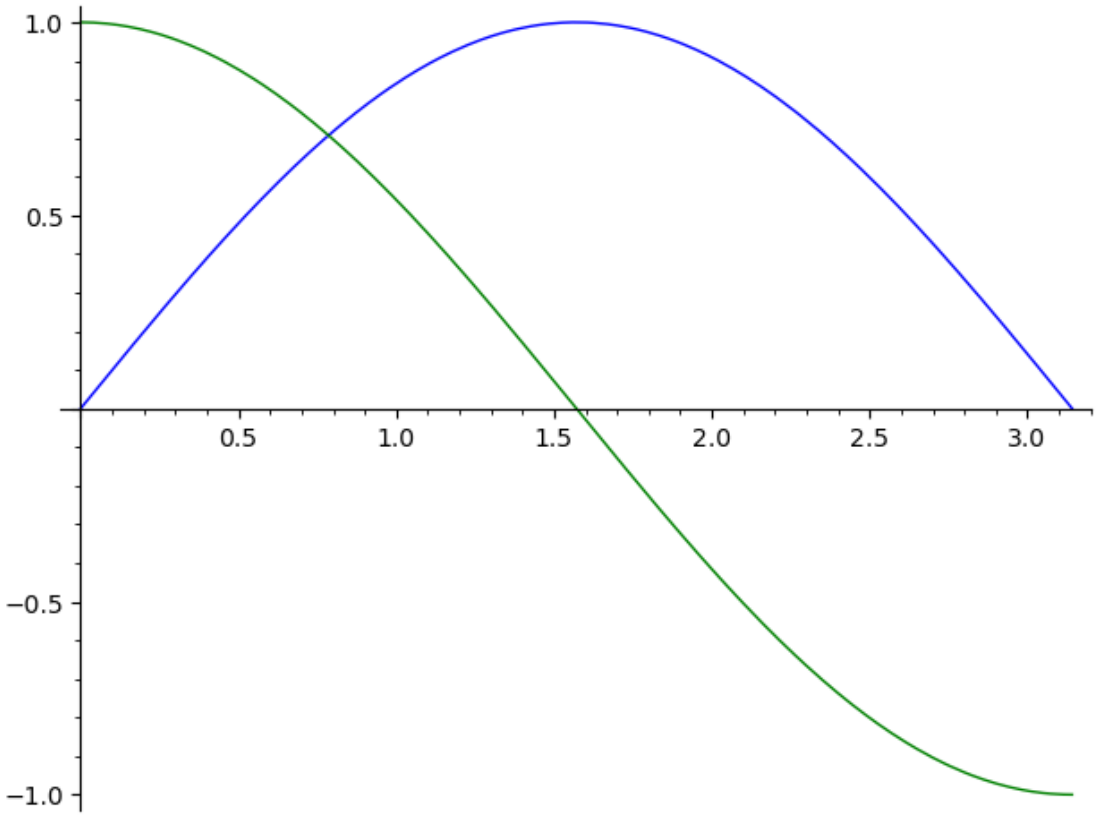


[167]:
```
parent(G)
```

[167]:
```
<class 'sage.plot.graphics.Graphics'>
```

[168]:
```
G += plot(cos(x),x,0,pi,color='green')
G
```

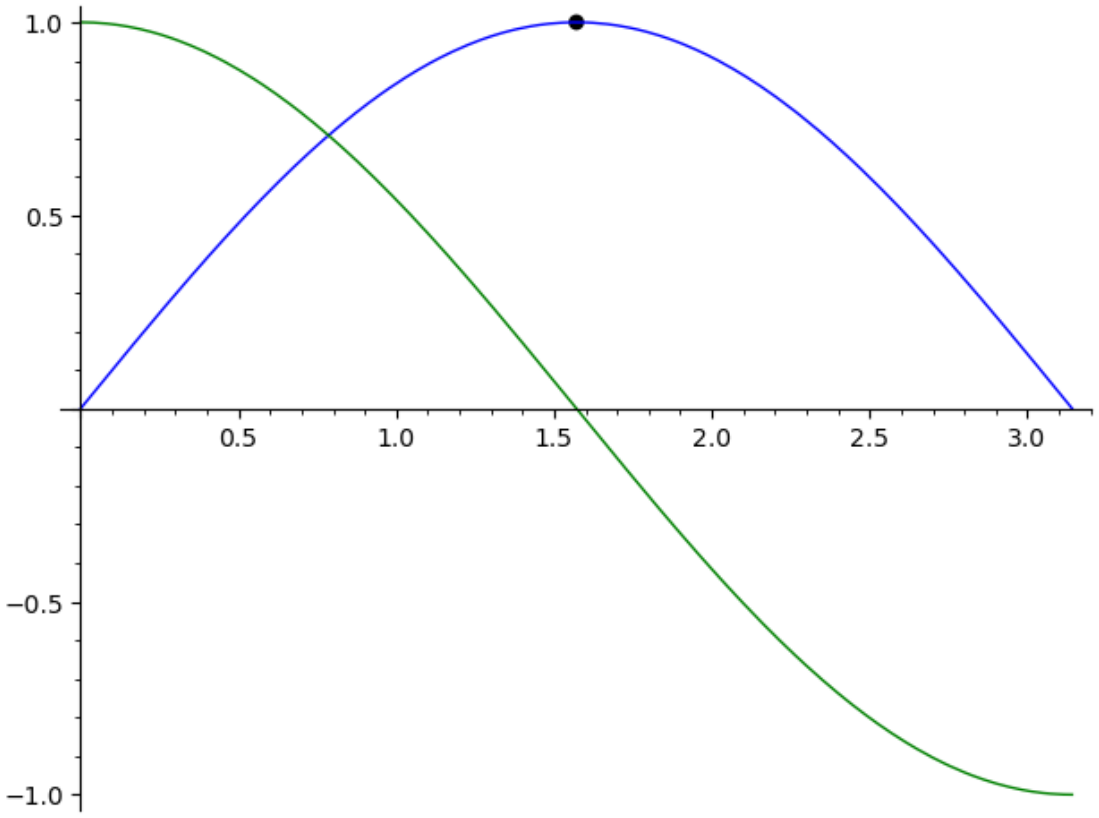
[168]:



[169]:
```
G += point((pi/2,1), color='black', size=40)
```

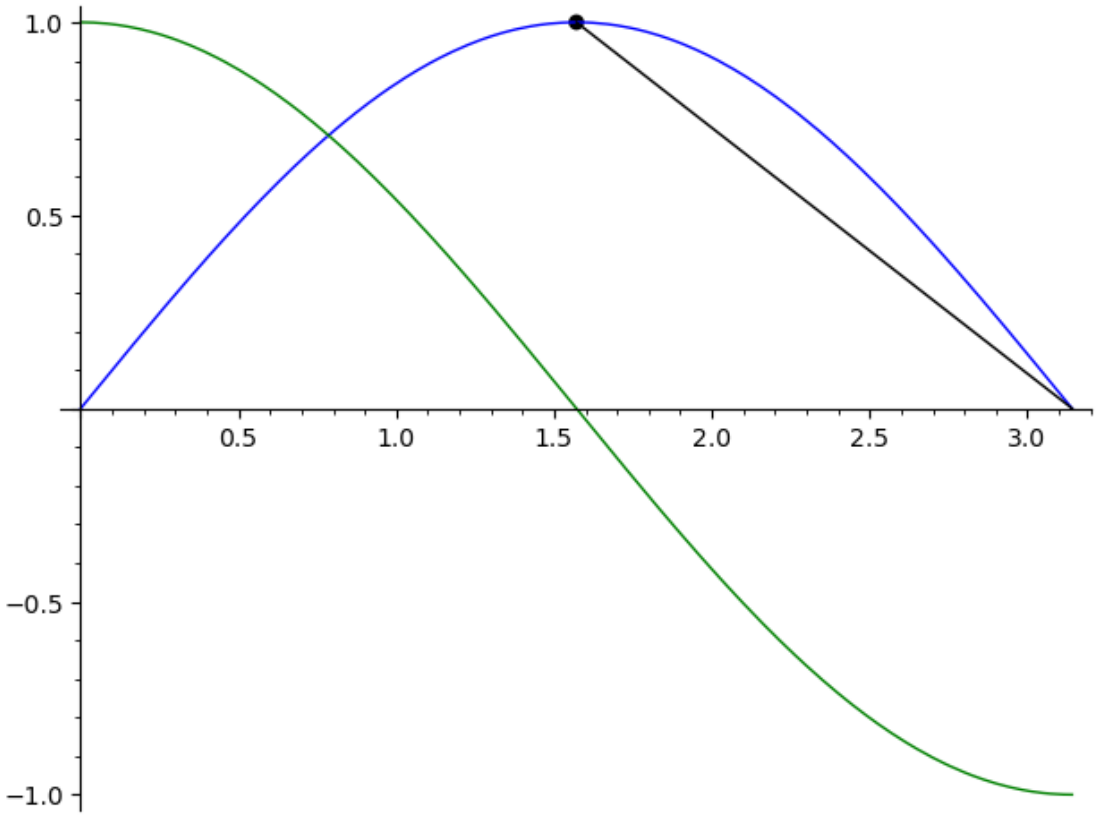
[170]:
```
G
```

[170]:



[171]:
```
G += line([(pi/2,1),(pi,0)],color='black')
G
```

[171]:



Lösen von Gleichungen

Quadratische Lösungsformel

[172]:
```
solve(x^2-2==0,x)
```

[172]:
```
[x == -sqrt(2), x == sqrt(2)]
```

[173]:
```
parent(x^2-2==0)
```

[173]:
```
Symbolic Ring
```

Kubische Lösungsformel von Cardano

[174]:
```
solve(x^3-3==0,x)
```

[174]:
```
[x == 1/2*I*3^(5/6) - 1/2*3^(1/3),
 x == -1/2*I*3^(5/6) - 1/2*3^(1/3),
 x == 3^(1/3)]
```

[175]:
```
show(_[0])
```

$$x = \frac{1}{2} i \cdot 3^{\frac{5}{6}} - \frac{1}{2} \cdot 3^{\frac{1}{3}}$$

Quartische Lösungsformel von Ferrari

[176]:
```
solve(x^4+x^3-x^2-2,x)
```

[176]:
```
[x == -1/12*sqrt(3)*sqrt((4*(3*sqrt(821)*sqrt(3) - 100)^(2/3) +
11*(3*sqrt(821)*sqrt(3) - 100)^(1/3) - 92)/(3*sqrt(821)*sqrt(3) - 100)^(1/3)) -
1/2*sqrt(-1/3*(3*sqrt(821)*sqrt(3) - 100)^(1/3) +
5/2*sqrt(3)/sqrt((4*(3*sqrt(821)*sqrt(3) - 100)^(2/3) + 11*(3*sqrt(821)*sqrt(3)
- 100)^(1/3) - 92)/(3*sqrt(821)*sqrt(3) - 100)^(1/3)) +
23/3/(3*sqrt(821)*sqrt(3) - 100)^(1/3) + 11/6) - 1/4,
 x == -1/12*sqrt(3)*sqrt((4*(3*sqrt(821)*sqrt(3) - 100)^(2/3) +
11*(3*sqrt(821)*sqrt(3) - 100)^(1/3) - 92)/(3*sqrt(821)*sqrt(3) - 100)^(1/3)) +
1/2*sqrt(-1/3*(3*sqrt(821)*sqrt(3) - 100)^(1/3) +
5/2*sqrt(3)/sqrt((4*(3*sqrt(821)*sqrt(3) - 100)^(2/3) + 11*(3*sqrt(821)*sqrt(3)
- 100)^(1/3) - 92)/(3*sqrt(821)*sqrt(3) - 100)^(1/3)) +
23/3/(3*sqrt(821)*sqrt(3) - 100)^(1/3) + 11/6) - 1/4,
 x == 1/12*sqrt(3)*sqrt((4*(3*sqrt(821)*sqrt(3) - 100)^(2/3) +
11*(3*sqrt(821)*sqrt(3) - 100)^(1/3) - 92)/(3*sqrt(821)*sqrt(3) - 100)^(1/3)) -
1/2*sqrt(-1/3*(3*sqrt(821)*sqrt(3) - 100)^(1/3) -
5/2*sqrt(3)/sqrt((4*(3*sqrt(821)*sqrt(3) - 100)^(2/3) + 11*(3*sqrt(821)*sqrt(3)
- 100)^(1/3) - 92)/(3*sqrt(821)*sqrt(3) - 100)^(1/3)) +
23/3/(3*sqrt(821)*sqrt(3) - 100)^(1/3) + 11/6) - 1/4,
 x == 1/12*sqrt(3)*sqrt((4*(3*sqrt(821)*sqrt(3) - 100)^(2/3) +
11*(3*sqrt(821)*sqrt(3) - 100)^(1/3) - 92)/(3*sqrt(821)*sqrt(3) - 100)^(1/3)) +
1/2*sqrt(-1/3*(3*sqrt(821)*sqrt(3) - 100)^(1/3) -
5/2*sqrt(3)/sqrt((4*(3*sqrt(821)*sqrt(3) - 100)^(2/3) + 11*(3*sqrt(821)*sqrt(3)
- 100)^(1/3) - 92)/(3*sqrt(821)*sqrt(3) - 100)^(1/3)) +
23/3/(3*sqrt(821)*sqrt(3) - 100)^(1/3) + 11/6) - 1/4]
```

[177]:
```
show(_[0])
```

$$x = -\frac{1}{12}\sqrt{3}\sqrt{\frac{4\left(3\sqrt{821}\sqrt{3}-100\right)^{\frac{2}{3}}+11\left(3\sqrt{821}\sqrt{3}-100\right)^{\frac{1}{3}}-92}{\left(3\sqrt{821}\sqrt{3}-100\right)^{\frac{1}{3}}}} - \frac{1}{2}\sqrt{-\frac{1}{3}\left(3\sqrt{821}\sqrt{3}-100\right)^{\frac{1}{3}}+\frac{5\sqrt{3}}{2\sqrt{\frac{4\left(3\sqrt{821}\sqrt{3}-100\right)^{\frac{2}{3}}+11\left(3\sqrt{821}\sqrt{3}-100\right)^{\frac{1}{3}}-92}{\left(3\sqrt{821}\sqrt{3}-100\right)^{\frac{1}{3}}}}}+\frac{23}{3\left(3\sqrt{821}\sqrt{3}-100\right)^{\frac{1}{3}}}+\frac{11}{6}} - \frac{1}{4}$$

Für Polynome 5. und höheren Grades lassen sich die Lösungen im Allgemeinen nicht mehr als Wurzelausdrücke schreiben. (Das heißt nicht nur, dass noch niemand eine solche Formel gefunden hat. Der Satz von Abel-Ruffini zeigt, dass es keine quintische Lösungsformel geben kann.)

[178]:
```
solve(x^5-x+2==0,x)
```

[178]:
```
[0 == x^5 - x + 2]
```

Gleichungen mit transzendenten Funktionen sind noch schwieriger. Im Allgemeinen ist es nicht möglich, die Lösungen geschlossen auszudrücken. Selbst bei sehr einfachen Gleichungen mit trigonometrischen Funktionen kann es schon vorkommen, dass Computeralgebrasysteme wie Sage Lösungen übersehen.

[179]:
```
solve(sin(x)==0,x)
```

[179]:
```
[x == 0]
```

Hier wird ein anderer Lösungsalgorithmus erzwungen, nun werden alle Lösungen gefunden. Die Variable z... in der Lösung gibt einen ganzzahligen Parameter an.

[180]:
```
solve(sin(x)==0,x,to_poly_solve='force')
```

[180]:
```
[x == pi*z14043]
```

Auch mehrmaliges Anschreiben derselben Gleichung führt hier zu allen Lösungen. Fazit: Gleichungslösen in Computeralgebrasystemen ist mit Vorsicht zu genießen!

[181]:
```
solve([sin(x)==0,sin(x)==0],x)
```

[181]:
```
[[x == pi*z14082]]
```