展开、提取 公因式

In [2]:

```
from sympy import expand, factor,  pprint
from sympy.abc import x,y

expr = (x + y) ** 2

pprint(expr)

print('----------------------')
print('----------------------')

expr = expand(expr)
pprint(expr)
print('----------------------')
print('----------------------')
expr1 = y**2 +x**2 +2*x*y
expr1 =  factor(expr1)
pprint(expr1)
```

```
      2
(x + y)
----------------------
----------------------
 2          2
x  + 2·x·y + y
----------------------
----------------------
      2
(x + y)
```

分开与合并

In [3]:

```
from sympy import *
# from sympy.abc import x
from sympy import init_printing

init_printing(use_unicode=True)

expr = 1/(x**2 +2*x)
pprint(expr)
```

```
  1
──────
 2
x  + 2·x
```

In [4]:

```
expr1 = apart(expr)
pprint(expr1)
```

```
     1       1
- ──────── + ──
  2·(x + 2)  2·x
```

In [5]:

```
expr1 = together(expr)
pprint(expr1)
```

```
   1
──────
x·(x + 2)
```

Rational值

SymPy 具有用于处理有理数的Rational。
有理数是可以表示为两个整数（分子 p 和非零分母 q）的商或分数 p / q 的任何数字。

In [6]:

```
#!/usr/bin/env python

from sympy import Rational

r1 = Rational(1/10)
r2 = Rational(1/10)
r3 = Rational(1/10)

val = (r1 + r2 + r3) * 3
print(val.evalf())

# 注意，当不使用有理数时，输出中会有一个小错误。
val2 = (1/10 + 1/10 + 1/10) * 3
print(val2)
```

```
0.900000000000000
0.9000000000000001
```

SymPy pprint

pprint()用于在控制台上漂亮地打印输出。
LaTeX 可以达到最佳效果，例如 在 Jupyter 笔记本中。

In [7]:

```
#!/usr/bin/env python

from sympy import pprint, Symbol, exp, sqrt
from sympy import init_printing

init_printing(use_unicode=True)

x = Symbol('x')

a = sqrt(2)
pprint(a)
print(a)

print("------------------------")

c = (exp(x) ** 2)/2
pprint(c)
print(c)
```

```
√2
sqrt(2)
------------------------
 2·x
ℯ
────
 2
exp(2*x)/2
```

平方根sqrt（）

In [8]:

```
from sympy import sqrt, pprint, Mul

x = sqrt(2)
y = sqrt(2)

pprint(Mul(x,  y, evaluate=False))
print('equals to ')
print(x * y)
```

```
√2·√2
equals to
2
```

SymPy 符号 symbol() symbols() latex()

在sympy里进行符号运算之前，必须先定义sympy的符号，这样sympy才能识别该符号。
.init_printing(use_latex=True)开启时，在jupyter运行时，输出的是LaTeX的格式
使用：latex()函数，同样返回LaTeX的格式。

符号计算适用于符号，以后可以对其进行求值。

使用符号之前，必须在 SymPy 中定义符号。
可以使用symbols()方法定义多个符号
可以从sympy.abc模块导入符号。
它将所有拉丁字母和希腊字母导出为符号，因此我们可以方便地使用它们。

In [9]:

```
import sympy as sy


# 符号化变量
x = sy.Symbol('x')
y, z = sy.symbols('y z')

# 输出设置
sy.init_printing(use_latex=True)

# 输出结果
print("x:", type(x))
print("y:", type(y))
print(x**2+y+z)
print(sy.latex(x**2+y+z))
```

```
x: <class 'sympy.core.symbol.Symbol'>
y: <class 'sympy.core.symbol.Symbol'>
x**2 + y + z
x^{2} + y + z
```

符号的初始化分为两种形式：
单个符号的初始化：x = sympy.Symbol('x')
多个符号的初始化：x,y=sympy.symbol("x y")

In [10]:

```
from sympy import Symbol, symbols
from sympy.abc import x, y#导入

expr = 2*x + 5*y
print(expr)

a = Symbol('a')#也可以自定义
b = Symbol('b')
i, j = symbols('i j')#也可以一次性自定义
expr2 = a*b + a - b
print(expr2)

i, j = symbols('i j')
expr3 = 2*i*j + i*j
print(expr3)
```

```
2*x + 5*y
a*b + a - b
3*i*j
```

自定义表达式-lambdify（）

该函数有点类似于lambda（）,用于自己构造一个函数表达

In [11]:

```
from sympy import *
import numpy as np

x = Symbol('x')

a = np.arange(10)

expr = x**2

# 构造自己的函数
f = lambdify(x, expr, "numpy")

print(f(a))
```

```
[ 0  1  4  9 16 25 36 49 64 81]
```

SymPy 规范表达形式

SymPy 会自动将表达式转换为规范形式。
SymPy 仅执行廉价操作； 因此，表达式可能无法求值为最简单的形式。

In [12]:

```
from sympy.abc import a, b

expr = b*a + -4*a + b + a*b + 4*a + (a + b)*3

print(expr)
```

```
2*a*b + 3*a + 4*b
```

SymPy 扩展代数表达式

使用expand()，我们可以扩展代数表达式；
即该方法尝试消除幂和乘法。

In [13]:

```
from sympy import expand, pprint
from sympy.abc import x

expr = (x + 1) ** 2

pprint(expr)

print('----------------------')
print('----------------------')

expr = expand(expr)
pprint(expr)
```

```
      2
(x + 1)
----------------------
----------------------
 2
x  + 2·x + 1
```

将字符串变为sympy的表达式-sympify（）

不要混淆了sympify()函数与 simplify()函数，前者是转化，后者是简化。

In [14]:

```
from sympy import *

string = "x**2+2*y + z/2"

# 转化
expr = sympify(string)

print("类型： ", type(expr))
print("表达式： ", expr)
```

```
类型：  <class 'sympy.core.add.Add'>
表达式：  x**2 + 2*y + z/2
```

Type *Markdown* and LaTeX: α^2

化简

In [15]:

```
from sympy import *
from sympy.abc import x


expr = (2*x+Rational(1,3)*x+4)/x
pprint(expr)
print('////////////////////')
expr = simplify(expr)
pprint(expr)
```

```
7·x
─── + 4
 3
──────────
   x
////////////////////
7   4
─ + ─
3   x
```

SymPy 简化表达式

可以使用simplify()将表达式更改为更简单的形式。

In [16]:

```
from sympy import sin, cos, simplify, pprint
from sympy.abc import x

expr = sin(x) / cos(x)

pprint(expr)

print('----------------------')

expr = simplify(expr)
pprint(expr)
```

```
sin(x)
──────
cos(x)
----------------------
tan(x)
```

SymPy 比较表达式

SymPy 表达式与equals()而不是==运算符进行比较。
我们用equals()比较两个表达式。 在应用该方法之前，SymPy 尝试简化表达式。

In [17]:

```
from sympy import pprint, Symbol, sin, cos

x = Symbol('x')

a = cos(x)**2 - sin(x)**2
b = cos(2*x)

print(a.equals(b))

# we cannot use == operator
print(a == b)
```

```
True
False
```

SymPy 求值表达式 数值计算-evalf()

相当于python自带的eval()函数，只是进行的是float浮点数运算。
可以通过替换符号来求值表达式。
该示例将 pi 值求值为 30 个位。

In [18]:

```
from sympy import pi

print(pi.evalf(30))
```

```
3.14159265358979323846264338328
```

（2）对于符号表达式的运算

In [19]:

```
from sympy import *

# 符号化
x = Symbol('x')

# 进行计算
expr = x**2+3
result = expr.evalf(subs={x: 2})

print(result)
```

```
7.00000000000000
```

替换符号-subs(old,new)

sub是Substitution的简称，也就是替换，其有两个作用：
语法是：expr.sub(old,new)
数值替换，用数值替换符号，进行带入计算。 符号替换，用一些符号替换符号。
subs()函数不改变原表达式，并且返回一个修改的表达式。
2）替换多个表达式

当需要替换多个表达式时，可以在subs()里使用列表
如：subs([(x,2), (y, 3), (z, 4)])
表示：将x替换成2，y替换成3，z替换成4

In [20]:

```
from sympy.abc import a, b
from sympy import pprint
from sympy import *

expr = b*a + -4*a + b + a*b + 4*a + (a + b)*3

print(expr.subs([(a, 3), (b, 2)]))



# 符号化变量
x, y, z = symbols('x y z')

expr = x**2+1

# 数值替换
result = expr.subs(x, 2)
print("原式：", expr)
print("数值计算的结果：", result)

# 符号替换
new_expr = expr.subs(x, y+z)
print("符号替换的结果：", new_expr)
```

```
29
原式： x**2 + 1
数值计算的结果： 5
符号替换的结果： (y + z)**2 + 1
```

SymPy 求解方程

用solve()或solveset()求解方程。 solve()的第一个参数是公式。
该公式以适合 SymPy 的特定形式编写；
即x**2 - x代替x**2 = x。
第二个参数是我们需要解决的符号。

In [21]:

```
from sympy import Symbol, solve

x = Symbol('x')

sol = solve(x**2 - x, x)

print(sol)
#[0, 1]
# Py
# 该方程式有两个解：0 和 1。
```

```
[0, 1]
```

Eq用于公式求解方程

等号（=）是赋值运算符，不表示相等。
如果你想要 (x = y)，则使用 Eq(x，y) 表示相等。
或者，所有表达式都假设等于零，因此您只要减去一边，用 x - y。
等号的正确用法是将表达式赋值给变量。

In [22]:

```
from sympy import pprint, Symbol, Eq, solve

x = Symbol('x')

eq1 = Eq(x + 1, 4)
pprint(eq1)

sol = solve(eq1, x)
print(sol)
```

```
x + 1 = 4
[3]
```

SymPy 序列

序列是其中允许重复的对象的枚举集合。
序列可以是有限的或无限的。
元素的数量称为序列的长度。
与集合不同，同一元素可以在序列中的不同位置出现多次。
元素的顺序很重要。 本示例创建一个由数字 1、2，…，10 组成的序列。 我们计算这些数字的总和。

In [23]:

```
from sympy import summation, sequence, pprint
from sympy.abc import x

s = sequence(x, (x, 1, 10))
print(s)
pprint(s)
print(list(s))

print(s.length)

print(summation(s.formula, (x, s.start, s.stop)))
# print(sum(list(s)))
```

```
SeqFormula(x, (x, 1, 10))
[1, 2, 3, 4, ...]
[1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
10
55
```

SymPy 极限

极限是函数（或序列）“接近”作为输入（或索引）“接近”某个值的值。

在示例中，我们具有1/x功能。 它具有左侧和右侧限制。
我们计算1/x的极限，其中 x 接近正无穷大。

In [24]:

```
from sympy import sin, limit, oo#oo表示无穷大。
from sympy.abc import x

l1 = limit(1/x, x, oo)
print(l1)

l2 = limit(1/x, x, 0)
print(l2)
```

```
0
oo
```

In [ ]:

```

```

SymPy 矩阵

在 SymPy 中，我们可以使用矩阵。
矩阵是数字或其他数学对象的矩形数组，为其定义了运算（例如加法和乘法）。
矩阵用于计算，工程或图像处理。
该示例定义了两个矩阵并将它们相乘。

In [25]:

```
from sympy import Matrix, pprint

M = Matrix([[1, 2], [3, 4], [0, 3]])
print(M)
pprint(M)

N = Matrix([2, 2])
pprint(N)
print("--------------------------")
print("M * N")
print("--------------------------")

pprint(M*N)
```

```
Matrix([[1, 2], [3, 4], [0, 3]])
⎡1  2⎤
⎢    ⎥
⎢3  4⎥
⎢    ⎥
⎣0  3⎦
⎡2⎤
⎢ ⎥
⎣2⎦
--------------------------
M * N
--------------------------
⎡6 ⎤
⎢  ⎥
⎢14⎥
⎢  ⎥
⎣6 ⎦
```

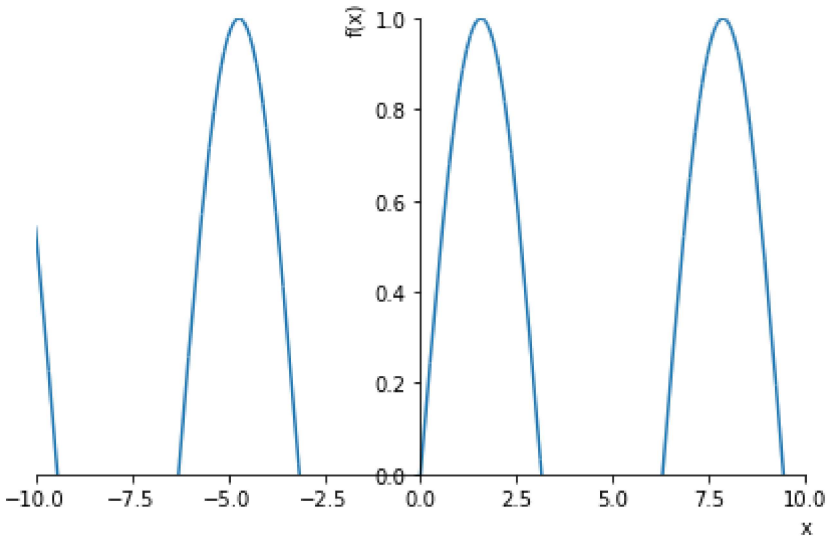
SymPy 绘图

In [26]:

```
import sympy
from sympy.abc import x
from sympy.plotting import plot

plot(sin(x))
```

```
c:\users\administrator\appdata\local\programs\python\python37\lib\site-packages\sym
py\plotting\plot.py:1065: MatplotlibDeprecationWarning:
The set_smart_bounds function was deprecated in Matplotlib 3.2 and will be removed t
wo minor releases later.
  self.ax[i].spines['left'].set_smart_bounds(True)
c:\users\administrator\appdata\local\programs\python\python37\lib\site-packages\sym
py\plotting\plot.py:1066: MatplotlibDeprecationWarning:
The set_smart_bounds function was deprecated in Matplotlib 3.2 and will be removed t
wo minor releases later.
  self.ax[i].spines['bottom'].set_smart_bounds(False)
```



Out[26]:

```
<sympy.plotting.plot.Plot at 0x22e924a3848>
```

In [ ]:

In [ ]: