PYTHON SYMPY 이해하기

Moon Yong Joon

PYTHON SYMPY 심벌 처리

Moon Yong Joon

Symbolic mathematics

sympy 모듈 이란

SymPy는 symbolic mathematics을위한 python 라이브러리입니다.

SymPy는 가능한 한 간단하게 코드를 유지하는 것은 이해하기 쉽게 확장 할 수있는 a full-featured computer algebra system (CAS)처리

sympy 구조

SymPy는 symbolic mathematics를 처리하기 위해 다양한 객체들로 구성해서 평가해 수학처럼 문제를 푸는 체계

```
from sympy import *
print(pi, type(pi))
print(pi.n(10))
print(pi.evalf())
```

```
(pi, <class 'sympy.core.numbers.Pi'>)
3.141592654
3.14159265358979
```

Sympy는 정의한 것은 pi도 하나의 인스턴스 객체 이므로 이 값을 처리를 위해서는 평가(evalf)를 해야 함

6

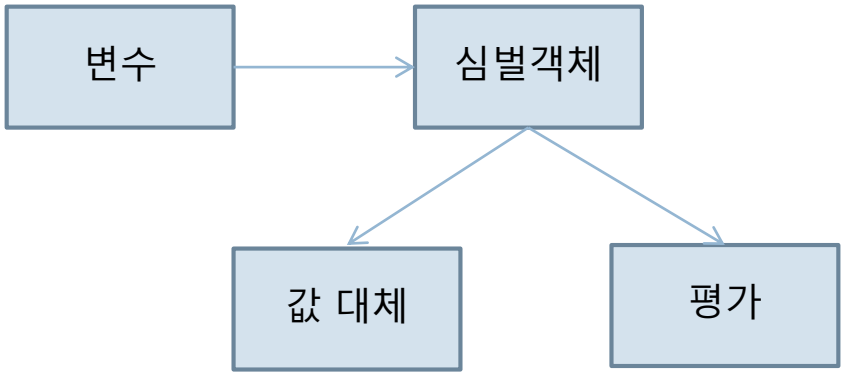
변수와 심벌 구분하기

python과 sympy 모듈 차이

Python은 참조변수와 객체를 가지지면 sympy는 참조변수와 수학의 표현식 객체와 수 객체를 별도로 구성

sympy는 심벌객체 처리

sympy는 값으로 전환 처리



Sympy : 심볼릭 처리

Sympy는 심볼 객체에 대해 값을 대체하고 평가해야 python 기초처럼 실행됨

```
from sympy import Mul, symbols
x = 100

y = symbols('y')


print(x, type(x))
print(y, type(y))
print(y.subs({y:100}).evalf())
```

```
(100, <type 'int'>)
(y, <class 'sympy.core.symbol.Symbol'>)
100.000000000000
```

Symbol 객체 이해

sympy 객체

Sympy객체를 변수에 할당해서 사용하지만 별도의 객체로 역할 함

```
from sympy import *
a = symbols('a', integer=True)
print(a.is_integer)
print(a.subs(a,10))


x, y, z = symbols('x,y,z', real=True)
print(x.is_real and y.is_real and z.is_real)

print(a.name, a.n)
print(x.name)
```

```
True
10
True
('a', <bound method Symbol.evalf of a>)
x
```

Symbol로 정의할 때 들어간 문자열의 실제 name이고 할당된 참조변수는 symbol 객체가 할당된 변수라 이름이 달라도 됨

sympy 구조

SymPy는 symbolic mathematics를 처리하기 위해 다양한 객체들로 구성되며 이를 평가해 수학처럼 문제를 푸는 체계

```
from sympy import *
print(pi, type(pi))
print(pi.n(10))
print(pi.evalf())
```

```
(pi, <class 'sympy.core.numbers.Pi'>)
3.141592654
3.14159265358979
```

Sympy는 정의한 것은 pi도 하나의 인스턴스 객체 이므로 이 값을 처리를 위해서는 평가(evalf)를 해야 함

Sympy symbol 정의

sympy.abc 사용하기

sympy.abc 모듈 내에 기본적인 영어 알파벳과 라틴어 등이 명확히 symbol로 정의되어 있는 모듈

```
from sympy.abc import x, y

print(x,y)
```

```
(x, y)
```

```
from sympy import symbols
x, y = symbols('x y')
print(x,y)
```

```
(x, y)
```

symbol 처리

Sympy 는 명확히 수학적인 심벌을 표시

x = Symbol('x')는 심벌을 하나 정의

a, b, c = symbols('a, b, c')는 심벌을 여러 개 정의

```
from sympy import *
x = Symbol('x')
y = Symbol('y')
print x+y+x-y
print (x+y)**2
a, b, c = symbols('a, b, c')
print (a+b+c)**2
print ((a+b+c)**2).expand()
```

```
2*x
(x + y)**2
(a + b + c)**2
a**2 + 2*a*b + 2*a*c + b**2 + 2*b*c + c**2
```

Symbol에 속성 추가

symbol 처리 : 변수에 타입주기

symbols 정의시 이 변수에 들어갈 데이터 타입을 부여할 수 있음

```
from sympy import *
a = symbols('a', integer=True)
print(a.is_integer)
print(a.subs(a,10))


x, y, z = symbols('x,y,z', real=True)
print(x.is_real and y.is_real and z.is_real)
```

```
True
10
True
```

symbol 처리 : 함수명

symbols 정의시 cls 파라미터에 Function을 부여하면 함수 변수가 됨

```
from sympy import *
x, y, z, t = symbols('x y z t')
k, m, n = symbols('k m n', integer=True)
f, g, h = symbols('f g h', cls=Function)

print(type(x))
print(type(f))

f = x**2
print(f)
print(type(f))
```

```
<class 'sympy.core.symbol.Symbol'>
<class 'sympy.core.function.UndefinedFunction'>
x**2
<class 'sympy.core.power.Pow'>
```

Sympy symbol 평가

symbol 평가: evalf

symbol를 정의하면 하나의 객체로 생성되고 값이 있을 경우 이 값을 평가해야 결과로 출력됨

```
from sympy import evalf,pi

pi = pi
print(pi)
print(pi.evalf())
```

```
pi
3.14159265358979
```

```
expr = x + 2*y
print(expr.__class__)
print(expr.args)
print(expr.subs({x:1, y:2}))
print(expr.subs({x:1, y:2}).evalf())
```

```
<class 'sympy.core.add.Add'>
(x, 2*y)
5
5.00000000000000
```

PYTHON SYMPY 출력 처리

Moon Yong Joon

출력세팅

init_printing: 역순 프린트

기본이 차원이 높은 순으로 표시. 단 sympy 표현이어야 하고 pprint로 출력해야 정확히 출력됨

```
from sympy import sympify,pprint, init_printing, symbols
y = symbols('y')
init_printing(order='rev-lex')
pprint(y**4 +  y**3  )
init_printing(order='grevlex')
pprint(y**2 +  y**3 )
```

```
 3    4
y  + y
 3    2
y  + y
```

출력방법

pprint: 수학 표현처럼 출력

출력을 수학 산식처럼 표현

```
from sympy import symbols, pprint
x,y = symbols('x,y')
a = 2*x**2 + 2*x*y + y**2
print(a)
print(" ====== math ====")
pprint(a)
```

```
2*x**2 + 2*x*y + y**2
 ====== math ====
   2             2
2·x  + 2·x·y + y
```

```
from sympy import *
x, t, z, nu = symbols('x t z nu')

init_printing(use_unicode=True)
pprint(diff(sin(x)*exp(x), x))

pprint(x**2+2*x+y**2)
```

```
 x            x
ℯ ·sin(x) + ℯ ·cos(x)
 2    2
x  + y  + 2·x
```

```
pprint(integrate(sin(x**2), (x, -oo, oo)))
pprint(solve(x**2 - 2, x))
```

```
√2·√π
─────
  2
[-√2, √2]
```

구조 출력하기

print_tree: 객체 구조 출력

sympy 객체 구조에 대해 tree 구조로 출력

```
from sympy import symbols, print_tree
x,y = symbols('x,y')

a = x+y

print_tree(a)
```

```
Add: x + y
+-Symbol: x
| commutative: True
+-Symbol: y
  commutative: True
```

PYTHON SYMPY 숫자 처리

Moon Yong Joon

수 타입 체계

정수 : sympy

정수(整數, integer)는 자연수(1, 2, 3, ...)와 이들의 음수(−1, −2, −3, ...)와 0으로 이루어진 수 체계이다.

```
from sympy import *
a = Integer(1)

print(a)
print(a.p)
print(a.q)

b = Integer("10")
print(b)
print(b.p, b.q)
```

```
1
1
1
10
(10, 1)
```

```
from sympy import *

x,n = symbols('x, n')

x = Integer(10.5)
print(x)
print(x == 10)
print(x is 10)
```

```
10
True
False
```

유리수 : sympy

유리수(有理數, rational number)를 분자(p)와 분모(q)로 관리

```
from sympy import *
a = Rational(1,2)

print(a)
print(a.p)
print(a.q)

b = Rational(".1")
print(b)
print(b.p, b.q)

c = Rational(1,3)
print(b)
print(b.p, b.q)
print(c.evalf(10))
```

```
1/2
1
2
1/10
(1, 10)
1/10
(1, 10)
0.3333333333
```

실수 1 : sympy

실수 [實數, real number]

```
from sympy import *
a = Float(1,2)

print(a)


b = Float(".1")
print(b)


c = Float(1/3)
print(b)

print(c.evalf(10))
```

```
1.0
0.100000000000000
0.100000000000000
0.3333333333
```

```
from sympy import *

x,n = symbols('x, n')

x = Float(10.5)
print(x, type(x))
print(x == 10.5)
print(x is 10.5)
```

```
(10.5000000000000, <class 'sympy.core.numbers.Float'>)
True
False
```

실수 2 : sympy

Sympify로 sympy 숫자 타입을 확인

```
from sympy import *

r = sympify(111/333)
print(r, type(r))

print(" integer ",sympify(2).is_integer)
print(" real ",sympify(2).is_real)
print(" real ",sympify(2.0).is_real)
print(" real ",sympify("2.0").is_real)
print(" real ",sympify("2e-45").is_real)
```

```
(0.333333333333333, <class 'sympy.core.numbers.Float'>)
(' integer ', True)
(' real ', True)
(' real ', True)
(' real ', True)
(' real ', True)
```

복소수 : sympy

Sympify로 sympy 의 복소수는 I(대문자 I)로 표시함
Python complex와 호환처리 됨

```
from sympy import *

x,n = symbols('x, n')

print((x*I + 1).subs({x:10}))
print(complex(1,10))

print( complex(1,10) == (x*I + 1).subs({x:10}))
```

```
1 + 10*I
(1+10j)
True
```

```
from sympy import *
import cmath
a = complex(1,2)

b = complex(2,2)
print(a/b)
print(a.__truediv__(b))
c = a*a.conjugate()
d = b*a.conjugate()
e = c/d
print(c)
print(d)
print(e)
```

```
(0.75+0.25j)
(0.75+0.25j)
(5+0j)
(6-2j)
(0.75+0.25j)
```

무한대 처리

무한대/pi를 심벌 처리

sympy는 oo로 무한대, pi로 파이 상수를 관리

```
from sympy import *
print pi
print pi**2
print pi.evalf()
print oo > 1e8
print oo + 1
```

```
pi
pi**2
3.14159265358979
True
oo
```

PYTHON SYMPY 제곱근 처리하기

Moon Yong Joon

제곱근 처리

무리수 : sympy

무리수(제곱근) 처리에 대한 산식 표현

```
from sympy import *
x = Symbol('x')

print(sqrt(3))
print(sqrt(3).evalf())

print(sqrt(8))
```

```
sqrt(3)
1.73205080756888
2*sqrt(2)
```

유리화 처리

radsimp : 유리화

무리수(제곱근) 처리에 대한 산식이 분수일 경우 분모에 대한 유리화 처리

```
from sympy import radsimp, sqrt, I, simplify
print(radsimp(1/(I + 1)))
print(simplify((((1) *(I-1))/((I + 1) *(I - 1)))))

print(radsimp(1/(2 + sqrt(2))))
print(simplify((1/(2 + sqrt(2)) * ((2 - sqrt(2))/(2-sqrt(2))))))
```

```
(1 - I)/2
1/2 - I/2
(-sqrt(2) + 2)/2
-sqrt(2)/2 + 1
```

PYTHON SYMPY 표현식 값 구하기

Moon Yong Joon

Numerical evaluation

evalf 메소드

숫자 값을 가지는 심볼릭 변수에 대해 평가를 하고 파라미터로 숫자를 넣으면 총 숫자의 자리까지 표시

```
from sympy import *

x,n = symbols('x, n')

x = Float(10.5)

print(x)
print(x.evalf(1))
print(x.evalf(2))
print(x.evalf(3))
```

```
10.5000000000000
1.e+1
11.
10.5
```

evalf 메소드 2

수식이 있을 경우 subs 메소드는 값만 대치하므로 평가하려면 evalf 메소드를 다시 사용

```
from sympy import *

x,y = symbols('x, y')
y = (x + pi)**2

print(y.subs(x, 1.5))
print(N(y.subs(x, 1.5)))
print(y.subs(x, 1.5).evalf())
```

```
(1.5 + pi)**2
21.5443823618587
21.5443823618587
```

수식값 부여 상세

수식평가 : +/-

수식을 만들고 subs, evalf, replace로 평가하기

```
from sympy import *

x = symbols('x')
expr = x + 1
print(expr, type(expr))
print(expr.subs(x, 2))
print(expr.evalf(subs={x:3}))
print(expr.replace(x,4))
```

```
(x + 1, <class 'sympy.core.add.Add'>)
3
4.00000000000000
5
```

```
from sympy import *

x = symbols('x')
expr = x - 1
print(expr, type(expr))
print(expr.subs(x, 2))
print(expr.evalf(subs={x:3}))
print(expr.replace(x,4))
```

```
(x - 1, <class 'sympy.core.add.Add'>)
1
2.00000000000000
3
```

수식평가 : *, /

수식을 만들고 subs, evalf, replace로 평가하기

```
from sympy import *

x = symbols('x')
expr = x * 1
print(expr, type(expr))
print(expr.subs(x, 2))
print(expr.evalf(subs={x:3}))
print(expr.replace(x,4))
```

```
(x, <class 'sympy.core.symbol.Symbol'>)
2
3.00000000000000
4
```

```
from sympy import *

x = symbols('x')
expr = x / 1
print(expr, type(expr))
print(expr.subs(x, 2))
print(expr.evalf(subs={x:3}))
print(expr.replace(x,4))
```

```
(x, <class 'sympy.core.symbol.Symbol'>)
2
3.00000000000000
4
```

PYTHON SYMPY 연산자 타입 처리

Moon Yong Joon

Add

Add 타입

표현식이 Add로 분리가 필요한 경우 Add 타입으로 묶는다

```
from sympy import Add, symbols
x,y = symbols('x,y')

a = Add(x,y, 10)
print(a)

print(a.as_coeff_add())

b = x+y+10

print(b == a)
print(type(b))
```

```
x + y + 10
(10, (x, y))
True
<class 'sympy.core.add.Add'>
```

Mul

Mul 타입

표현식이 Mul로 분리가 필요한 경우 Mul타입으로 묶는다

```
from sympy import Mul, symbols
x,y = symbols('x,y')

a = Mul(x,y, 10)
print(a)

print(a.as_coeff_mul())

b = x*y*10

print(b == a)
print(type(b))
```

```
10*x*y
(10, (x, y))
True
<class 'sympy.core.mul.Mul'>
```

PYTHON SYMPY 표현식 처리

Moon Yong Joon

표현식 : 연산자 단위

두 변수의 덧셈 표현식

Mul 타입으로 파라미터로 두 변수의 합(뺄셈 포함)을 표현

```
from sympy import Add, symbols
x,y = symbols('x,y')
print(Add(x, y))
print(Add(x, -1*y))
```

```
x + y
x - y
```

두 변수의 곱셈 표현식

Mul 타입으로 파라미터로 두 변수의 곱(나눗셈 포함)을 표현

```
from sympy import Mul, symbols
x,y = symbols('x,y')
print(Mul(x, y))
print(Mul(x, 1/y))
```

```
x*y
x/y
```

숫자 표현식

Integer를 이용해서 숫자를 생성

```
from sympy import Integer, symbols
x,y = symbols('x,y')
x = Integer(1)
y = Integer(-1)
print(x, y)
```

```
(1, -1)
```

거듭제곱 표현

Pow를 이용해서 거듭제곱에 표현(지수)을 생성

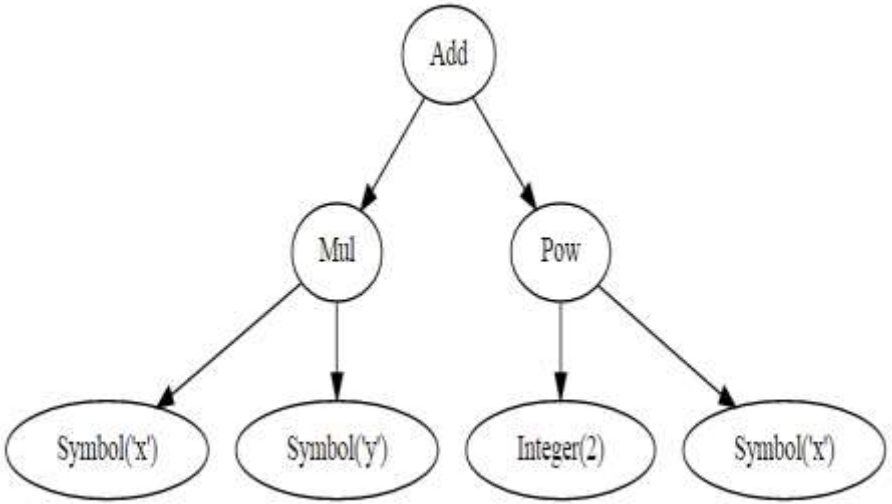
```
from sympy import Pow, symbols
x,y = symbols('x,y')

print(Pow(x,2))
print(Pow(x, 1/2))
```

```
x**2
x**0.5
```

표현식 : 연산자통합

표현식이 구조

표현식의 구조를 확인해 보면 심벌을 연산자의 각 타입에 맞춰 표현됨



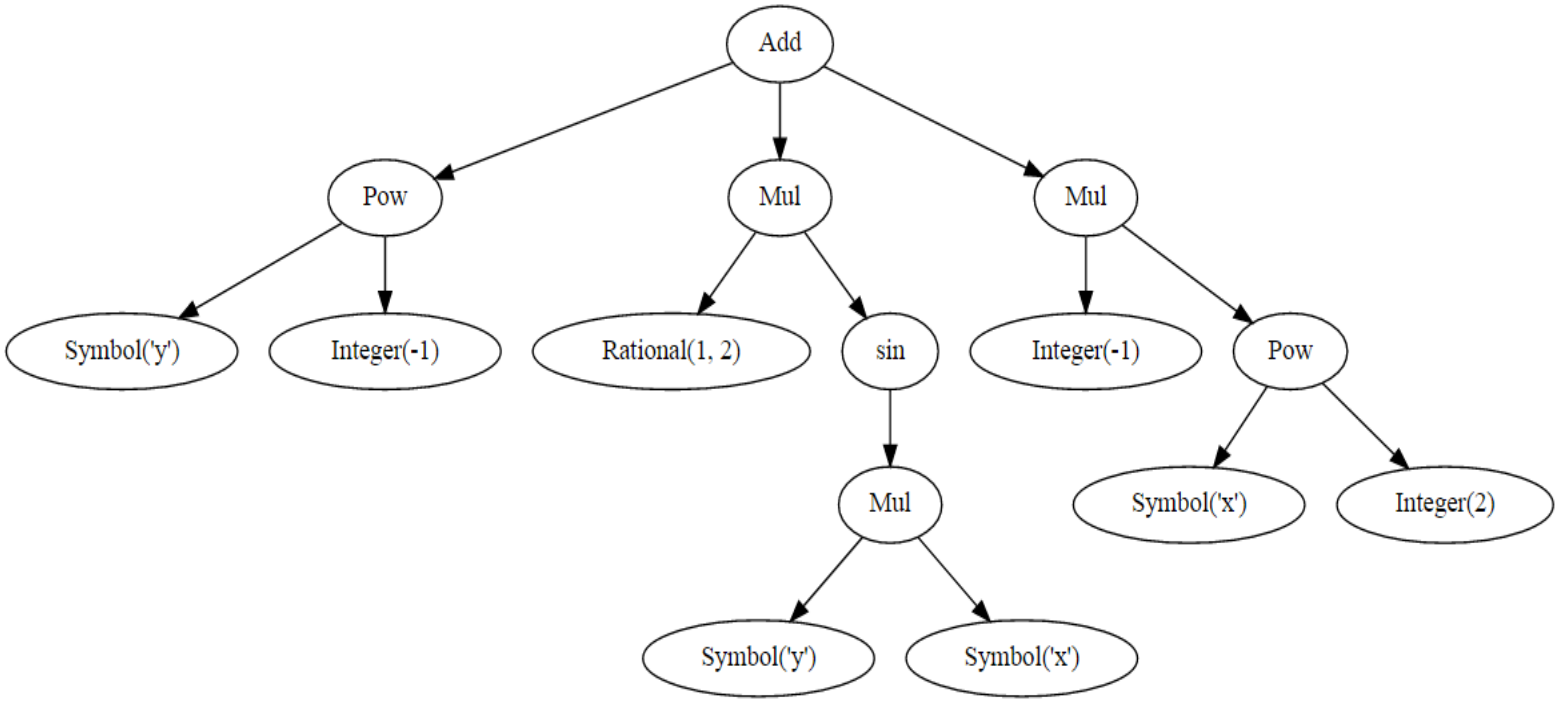

```
from sympy import Integer, symbols
x,y = symbols('x,y')
expr = x*y + x**2
print(expr)
```

```
x**2 + x*y
```

표현식 : srepr

표현식이 구조 : 그래프

표현식의 구조를 확인해 보면 심벌을 연산자의 각 타입에 맞춰 표현됨



표현식이 구조 : srepr

표현식의 구조를 확인해 보면 심벌을 연산자의 각 타입에 맞춰 표현됨

```
from sympy import symbols, srepr
x,y = symbols('x,y')
expr = sin(x*y)/2 - x**2 + 1/y
print(srepr(expr))
```

```
Add(Mul(Integer(-1), Pow(Symbol('x'), Integer(2))), Mul(Rational(1, 2), sin(Mul(Symbol('x'), Symbol('y')))), Pow(Symbol('y'), I
nteger(-1)))
```

PYTHON SYMPY 표현식 구성요소 확인

Moon Yong Joon

표현식 : 최종 타입 확인

표현식 : func

표현식이 최상위 타입에 대한 구성을 확인하기

```
from sympy import symbols, srepr
x,y = symbols('x,y')

expr = 3*y**2*x
print(expr.func)

print(type(expr))
print(srepr(expr))
```

```
<class 'sympy.core.mul.Mul'>
<class 'sympy.core.mul.Mul'>
Mul(Integer(3), Symbol('x'), Pow(Symbol('y'), Integer(2)))
```

표현식 : 자동 구조 변환

표현식 : 타입 자동 변환

표현식에서 같은 변수를 더하면 표현식 기준으로는 같은 문자의 곱으로 변하므로 데이터 타입이 Add에서 Mul로 자동 변환이 됨 가장 최소 단위를 확인하기

```
from sympy import symbols, srepr
x,y = symbols('x,y')
expr = Add(x, 1)
print(expr.func)
print(srepr(expr))

expr = Add(x, x)
print(expr.func)
print(srepr(expr))
```

```
<class 'sympy.core.add.Add'>
Add(Symbol('x'), Integer(1))
<class 'sympy.core.mul.Mul'>
Mul(Integer(2), Symbol('x'))
```

표현식 : func 확인

표현식 : func

표현식에 .func을 표시하면 데이터 타입이 나오고 args를 받아서 실행하면 expr이 표현

```
from sympy import symbols, srepr
x,y = symbols('x,y')

expr = 3*y**2*x
print(expr.func)

print(type(expr))
print(srepr(expr))

print(expr.func(*expr.args))

print(expr == expr.func(*expr.args))
```

```
<class 'sympy.core.mul.Mul'>
<class 'sympy.core.mul.Mul'>
Mul(Integer(3), Symbol('x'), Pow(Symbol('y'), Integer(2)))
3*x*y**2
True
```

표현식 :args 확인

표현식 : args

표현식에서 가장 최소 구성 단위를 확인하기

```
from sympy import symbols, srepr
x,y = symbols('x,y')

expr = 3*y**2*x
print(expr.func)

print(expr.args)
```

```
<class 'sympy.core.mul.Mul'>
(3, x, y**2)
```

표현식 :atoms 확인

표현식 : atoms

표현식에서 가장 최하위 단위를 확인하기

```
from sympy import Integer, symbols
x,y = symbols('x,y')
expr = x*y + x**2
print(expr)
print(expr.atoms())
help(expr.atoms)
```

```
x**2 + x*y
set([2, x, y])
Help on method atoms in module sympy.core.basic:

atoms(self, *types) method of sympy.core.add.Add instance
    Returns the atoms that form the current object.

    By default, only objects that are truly atomic and can't
    be divided into smaller pieces are returned: symbols, numbers,
    and number symbols like I and pi. It is possible to request
    atoms of any type, however, as demonstrated below.
```

표현식 :args/atoms 확인

표현식 : args 내부 확인

표현식을 더 세부 구성요소로 세분화하려면 args내의 args를 사용해서 처리

```
from sympy import Mul, symbols
x,y = symbols('x,y')
expr = y**2*3*x
print(expr.args)
print(expr.args[0].args)
print(expr.args[2])
print(expr.args[2].args)
print(expr.args[2].atoms())
```

```
(3, x, y**2)
()
y**2
(y, 2)
set([2, y])
```

PYTHON SYMPY 표현식 계수 처리

Moon Yong Joon

계수 추출

계수 추출 1

add, mul에 대한 계수를 추출해서 보여 주기

```
from sympy import symbols, oo
A, B = symbols('A B', commutative=False)
x, y = symbols('x y')
expr = -2+ 10*x*B*A*y
print(expr.as_coeff_Add())
print(expr.as_coeff_add())
```

```
(-2, 10*x*y*B*A)
(-2, (10*x*y*B*A,))
```

```
from sympy import symbols, oo
A, B = symbols('A B', commutative=False)
x, y = symbols('x y')
expr = 10*x*B*A*y
print(expr.as_coeff_Mul())
print(expr.as_coeff_mul())
```

```
(10, x*y*B*A)
(10, (x, y, B, A))
```

계수 추출 2

add, mul에 대한 계수를 추출해서 dict 타입으로 보여 주기

```
from sympy import E,symbols, oo
from sympy.abc import a, x

print((3*x + a*x + 4).as_coefficients_dict())

print((3*a*x).as_coefficients_dict())
```

```
defaultdict(<type 'int'>, {1: 4, x: 3, a*x: 1})
defaultdict(<type 'int'>, {a*x: 3})
```

교환법칙

가환 commutative

개의 대상이 결합한 결과가 결합하는 순서와 관계가 없을 때의 조작, 즉 교환법칙이 성립할 때를 말함.
A,B는 가환이 안 되므로 표현식에서 바뀌면 False 처리 됨

```
from sympy import symbols, oo
A, B = symbols('A B', commutative=False)
x, y = symbols('x y')

print((-2*x*y) == (x*-2*y))
print((-2*x*A*B*y) == (-2*x*B*A*y))
print(type(A))
print(type(x), type(y))
```

```
True
False
<class 'sympy.core.symbol.Symbol'>
(<class 'sympy.core.symbol.Symbol'>, <class 'sympy.core.symbol.Symbol'>)
```

args_cnc 메소드

표현식 내에 가환 변수와 비가환 변수를 표시하는 메소드(앞에는 가환, 뒤에는 비가환을 표시)

```
from sympy import symbols, oo
A, B = symbols('A B', commutative=False)
x, y = symbols('x y')

print((-2*x*A*B*y).args_cnc())
print((-2*x*B*A*y).args_cnc())
print((-2*x*A*B*y) == (-2*x*B*A*y))
```

```
[[-1, 2, x, y], [A, B]]
[[-1, 2, x, y], [B, A]]
False
```

PYTHON SYMPY
PYTHON 문자열/수식을 SYMPY 수식으로 변환

Moon Yong Joon

객체 변환하기

Sympify 함수 사용

python 정수 타입을 sympy 정수타입으로 변환

```
print(type(2))

print(type(sympify(2)))
```

```
<type 'int'>
<class 'sympy.core.numbers.Integer'>
```

수식변환하기

Sympify 함수 사용

문자열로 입력된 것을 sympy 객체로 변환하기

```
from sympy import sympify
s = "2*x"
print(s, type(s))
expr = sympify(s)
print(expr, type(expr))
print(expr.subs({x:10}))
```

```
('2*x', <type 'str'>)
(2*x, <class 'sympy.core.mul.Mul'>)
20
```

Sympify 처리 에러

문자열로 작성시 파이썬 산식에 맞아야 한다. 산식 2x는 파이썬 표현으로는 2*x 이므로 오류 발생함

```
from sympy import sympify
s = "2x"
print(s, type(s))
expr = sympify(s)
print(expr, type(expr))
print(expr.subs({x:10}))
```

```
('2x', <type 'str'>)

---------------------------------------------------------------------------
SympifyError                                Traceback (most recent call last)
<ipython-input-292-814ae388be32> in <module>()
      2 s = "2x"
      3 print(s, type(s))
----> 4 expr = sympify(s)
      5 print(expr, type(expr))
      6 print(expr.subs({x:10}))
```

PYTHON SYMPY 유리수 표현식의 단순화

Moon Yong Joon

유리수 표현의 분리

apart 함수

유리수 표현식으로 분리하기 위해서는 apart 함수를 이용해서 분모(denomimator)와 분자(numerator)를 통분해서 처리

```
from sympy import factor,apart, symbols
x,y = symbols('x,y')

print(factor(x**2+x-2))
print(apart((x**2+x+4)/(x+2)))
```

```
(x - 1)*(x + 2)
x - 1 + 6/(x + 2)
```

apart 와 simplify 비교

apart 함수는 유리수 식을 분리하지만 simpify 는 단순화 대상이 존재할 때만 처리

```
from sympy import *
x,y = symbols('x, y')
f1 = 1/((x+1)*(x+2))

print(apart(f1))
print(simplify(f1))

f2 = (y**2 -1)/(y-1)
print(apart(f2))
print(simplify(f2))
```

```
-1/(x + 2) + 1/(x + 1)
1/((x + 1)*(x + 2))
y + 1
y + 1
```

PYTHON SYMPY ALGEBRAIC MANIPULATIONS

Moon Yong Joon

다항식 polynomial

다항식 polynomial

수식에 차원이 다른 항이 많은 경우를 다항식이라 함. 전개식(expand), 인수분해(factor)를 이용해서 정리

```
from sympy import symbols, expand, factor
x,y = symbols('x,y')

p = (x-1)*(x-2)*(x-3)
print(p.func)
print(p)
print(expand(p))
print(factor(expand(p)))
```

```
<class 'sympy.core.mul.Mul'>
(x - 3)*(x - 2)*(x - 1)
x**3 - 6*x**2 + 11*x - 6
(x - 3)*(x - 2)*(x - 1)
```

다항식 전개: multinomial=True

expand 내의 파라미터로 multinomial을 사용하면 다항식에 대해 전개

```
from sympy import *
x, y, z = symbols('x, y, z')

expr = (x + y + z)**3
print expr, '=', expr.expand()
print expr, '=', expr.expand(multinomial=True)
```

```
(x + y + z)**3 = x**3 + 3*x**2*y + 3*x**2*z + 3*x*y**2 + 6*x*y*z + 3*x*z**2 + y**3 + 3*y**2*z + 3*y*z**2 + z**3
(x + y + z)**3 = x**3 + 3*x**2*y + 3*x**2*z + 3*x*y**2 + 6*x*y*z + 3*x*z**2 + y**3 + 3*y**2*z + 3*y*z**2 + z**3
```

다항식 전개

expand 메소드 : 전개식 처리

표현식도 별도의 타입을 가지므로 expand메소드를 사용하면 표현식이 전개됨

```
from sympy import *
x, y, z = symbols('x, y, z')
expr = y*(x + z)
print expr, '=', expr.expand(mul=True)
print(help(expr.expand))
expr = x + y
print expr, '=', expr.expand(complex=True)
expr = exp(x + y)
print expr, '=', expr.expand(power_exp=True)
expr = 2**(x + y)
print expr, '=', expr.expand(power_exp=True)
```

```
y*(x + z) = x*y + y*z
Help on method expand in module sympy.core.expr:

expand(*args, **kwargs) method of sympy.core.mul.Mul instance
    Expand an expression using hints.

    See the docstring of the expand() function in sympy.core.function for
    more information.

None
x + y = re(x) + re(y) + I*im(x) + I*im(y)
exp(x + y) = exp(x)*exp(y)
2**(x + y) = 2**x*2**y
```

expand 함수 : 전개식 처리

표현식도 별도의 타입을 가지므로 expand메소드를 사용하면 표현식이 전개됨

```
from sympy import *
print(expand((x*4)**2))
print(expand((x*4)**2, power_base = True, force=True))
print(expand((x+y)**3))

expr = (x*y)**z
print(expand(expr))
print(expand(expr, power_base = True))
print(expand(expr, power_base = True, force=True))
```

```
16*x**2
16*x**2
x**3 + 3*x**2*y + 3*x*y**2 + y**3
(x*y)**z
(x*y)**z
x**z*y**z
```

곱셈 전개시 동일 항에 대한 곱셈부분도 전개가 필요하며 force=True 처리

복소수 전개

expand 메소드 : 전개식 처리

복소수에 대한 전개가 필요할 경우에는 complex=True로 표시해야 함

```
from sympy import *
x,y = symbols('x, y')
print((x + y).expand())
print((x + y).expand(complex=True))

print(cos(x).expand(trig=True))
print(cos(x).expand(complex=True))
```

```
x + y
re(x) + re(y) + I*im(x) + I*im(y)
cos(x)
-I*sin(re(x))*sinh(im(x)) + cos(re(x))*cosh(im(x))
```

인수분해: 메소드

factor 메소드

다항식 수식 객체 내의 factor 메소드를 이용해서 인수분해 처리

```
from sympy import symbols, expand, factor
x,y = symbols('x,y')

p = (x-1)*(x-2)*(x-3)
print(p.func)
print(p)
print(p.expand(x))
print(p.factor())
```

```
<class 'sympy.core.mul.Mul'>
(x - 3)*(x - 2)*(x - 1)
x**3 - 6*x**2 + 11*x - 6
(x - 3)*(x - 2)*(x - 1)
```

인수분해: 함수

factor 함수

인수분해는 factor 함수를 사용하고 전개는 expand 함수를 사용하고 해는 solve함수를 사용함

```
from sympy import *
x, y, z = symbols("x y z")

print(simplify((x + 1)**2))
print(solve(x**2 + 2*x + 1))
print(factor(x**2 + 2*x + 1))
print(expand((x + 1)**2))
print(factor(x**2 - 1))
```

```
(x + 1)**2
[-1]
(x + 1)**2
x**2 + 2*x + 1
(x - 1)*(x + 1)
```

factor 함수 : modulus

인수분해를 위한 파라미터를 modulus로 사용하면 인수분해 불가함 방정식을 조건에 따라 인수분해 함

```
f = x**2+2
print(factor(f))
print(factor(f, modulus=1))
print(factor(f, modulus=2))
print(factor(f, modulus=3))
print(expand(factor(f, modulus=3)))
```

```
x**2 + 2
0
x**2
(x - 1)*(x + 1)
x**2 - 1
```

factor 함수 :가우시안정수

가우스 정수(Gauß整數, Gaussian integer)는 실수부와 허수부가 모두 정수인 수이다.

$(a+bi)(a-bi)=a^2+b^2$ 로 표현이 가능한 수

```
from sympy import *
x, y, z = symbols("x y z")

print(factor(x**2 - 1))
print(primitive(x**2 -1))
print(factor(x**2 + 1, modulus=2))
print(factor(x**2 + 1, gaussian=True))
print(factor(x**2 - 2, extension=sqrt(2)))
```

```
(x - 1)*(x + 1)
(1, x**2 - 1)
(x + 1)**2
(x - I)*(x + I)
(x - sqrt(2))*(x + sqrt(2))
```

X2+1은 인수분해가 안되어서 가우시안 수로 인수분해**

= (0+i)(0 -i)
= -i*I
= -(i)**2
= 1

factor 함수 : extension

인수분해를 위한 값을 배정해서 처리

```
from sympy import *
x, y, z = symbols("x y z")

print(factor(x**2 - 1))
print(primitive(x**2 -1))
print(factor(x**2 + 1, modulus=2))
print(factor(x**2 + 1, gaussian=True))
print(factor(x**2 - 2, extension=sqrt(2)))
```

```
(x - 1)*(x + 1)
(1, x**2 - 1)
(x + 1)**2
(x - I)*(x + I)
(x - sqrt(2))*(x + sqrt(2))
```

방정식 check

Simplify 함수

두개의 방정식을 가지고 0 값이 나오려면 simplify 함수를 이용해서 결과값이 0이 나오는지를 확인함

```
from sympy import symbols, simplify
x,y = symbols('x,y')

p = (x-5)*(x+5)
q = x**2 - 25

print(p == q)

print(simplify(p-q) == 0)
```

```
False
True
```

Simplify 함수 : 삼각함수

삼각함수의 방정식을 이용해서 방정식이 값이 정당하게 처리되는지 비교

```
from sympy import symbols, simplify, sin, cos
x,y = symbols('x,y')

p = (x-5)*(x+5)
q = x**2 - 25

print(sin(x)**2+ cos(x)**2 == 1)

print(simplify(sin(x)**2+ cos(x)**2-1) == 0)
```

```
False
True
```

공통요소 제거

cancel 함수 : 공통요소 제거

분모와 분자의 공통요소를 제거하는 함수는 cancel로 처리

```
from sympy import *
x, y, z = symbols("x y z")

print(cancel((x**2 - 4)/(x**2 + 4*x + 4)))
print(primitive((x**2 - 4)/(x**2 + 4*x + 4)))
print(factor((x**2 - 4)/(x**2 + 4*x + 4)))
```

```
(x - 2)/(x + 2)
(1, x**2/(x**2 + 4*x + 4) - 4/(x**2 + 4*x + 4))
(x - 2)/(x + 2)
```

수식 단순화

trigsimp함수 : 단순화

수식에서 제거될 수 있는 부분을 정리하는 trigsimp 함수

```
from sympy import trigsimp, cancel, simplify, cos, sin, solve,pi
a = (x + x**2)/(x*sin(y)**2 + x*cos(y)**2)
b = sin(y)**2 + cos(y)**2
print("b = ", b.subs({y:pi/2}))
print(" trigsimp ",trigsimp(a))

print(cancel(trigsimp(a)))
```

```
('b = ', 1)
(' trigsimp ', (x**2 + x)/x)
x + 1
```

simplify함수 : 단순화

수식에서 제거될 수 있는 부분을 정리하는 trigsimp/cancel를 하나의 simplify 함수로 처리

```
from sympy import trigsimp, cancel, simplify
a = (x + x**2)/(x*sin(y)**2 + x*cos(y)**2)
print("simplify", simplify(a))
```

```
('simplify', x + 1)
```

동일변수로 다항식 통합

collect 함수 : 통합

수식을 특정 변수로 통합해서 단순화

```
from sympy import *
from sympy.abc import a, b, c, x, y

print(collect(a*x**2 + b*x**2 + a*x - b*x + c, x))
```

```
c + x**2*(a + b) + x*(a - b)
```

collect와 cancel 차이점

수식을 변수와 계수에 맞춰 처리하는 함수.

simplify 함수와 다른 점은 cancel 처리를 하지 않는다.

```
from sympy import factor, expand, collect
from sympy.abc import a, b, c, x, y, z

print(factor( x **2 -2* x -8 ))

print(expand( (x -4)*( x +2) ))

print(collect(x **2 +  b*x + a*x + a*b , x))
print(collect((x**2-1)/(x+1), x))

print(simplify(x **2 +  b*x + a*x + a*b ))
print(simplify((x**2-1)/(x+1)))
```

```
(x - 4)*(x + 2)
x**2 - 2*x - 8
a*b + x**2 + x*(a + b)
(x**2 - 1)/(x + 1)
a*b + a*x + b*x + x**2
x - 1
```

PYTHON SYMPY 방정식 해구하기

Moon Yong Joon

해 구하기

Solve 함수 : 해 구하기

표현식을 solve 함수에 넣으면 해를 구해 줌

```
from sympy import *
x, y = symbols('x, y')
pprint( solve( x**2 - 1, x) )
print(solve(x**2+ 2*x+ 1, x))
```

```
[-1, 1]
[-1]
```

방정식 풀기

2차 방정식을 전개는 expand 함수로 처리하고 해는 solve 함수로 구함

```
from sympy import *

x = symbols('x')
expr = (x + 1) **2
print(expand(expr))
print(solve(expr))
```

```
x**2 + 2*x + 1
[-1]
```

PYTHON SYMPY 함수

Moon Yong Joon

함수에 대한 전개

expand : 함수 전개

표현식에서 expand 파라미터로 func=True를 넣으면 전개

```
from sympy import *
x, y, z = symbols('x, y, z')

expr = gamma(x + 1)
print expr, '=', expr.expand()
print expr, '=', expr.expand(func=True)
```

```
gamma(x + 1) = gamma(x + 1)
gamma(x + 1) = x*gamma(x)
```

PYTHON SYMPY 함수 그래프

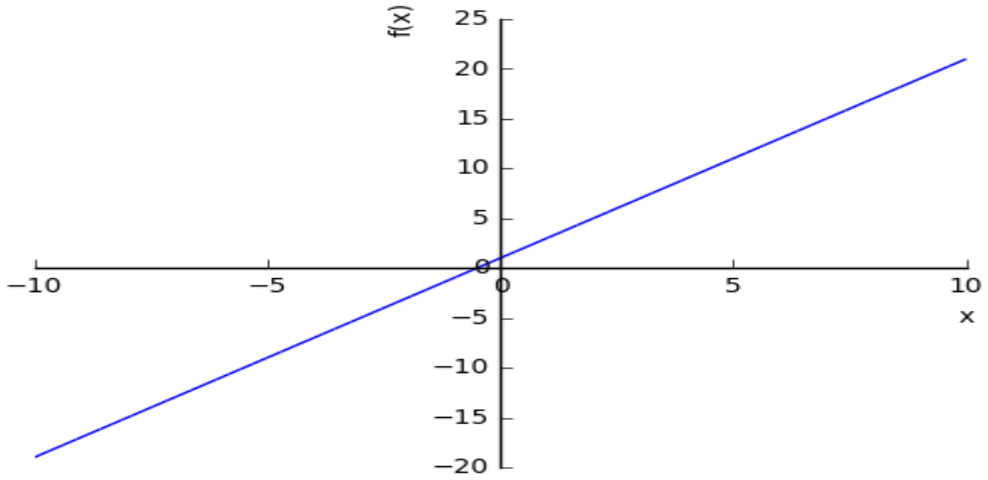
Moon Yong Joon

plotting

1차 함수 그래프

1차 함수를 만들고 plot 함수로 그래프 그리기

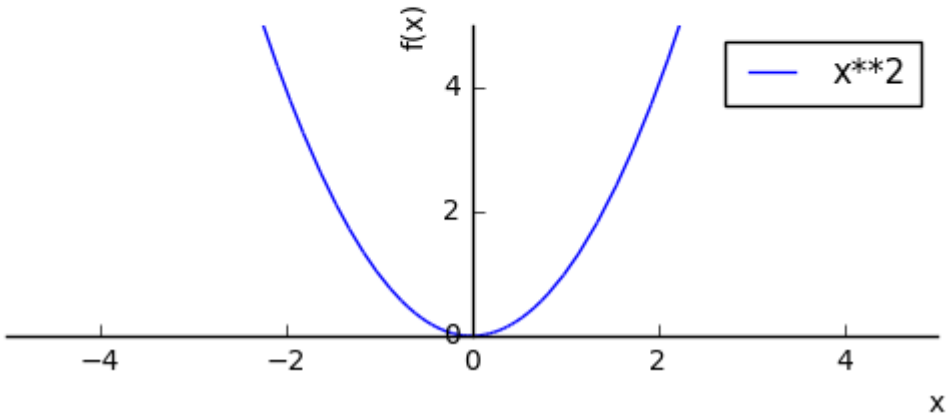
```
import sympy.plotting as plt
from sympy import Symbol

a = Symbol('x')
expr = 2*a +1

plt.plot(expr)
```



2차 함수 그래프

2차 함수를 만들고 plot 함수로 그래프 그리기

```
import sympy.plotting as plt
from sympy import Symbol

x = Symbol('x')
expr = x*x

plt.plot(expr, (x,-5,5), ylim=(-5, 5),legend=True)
```

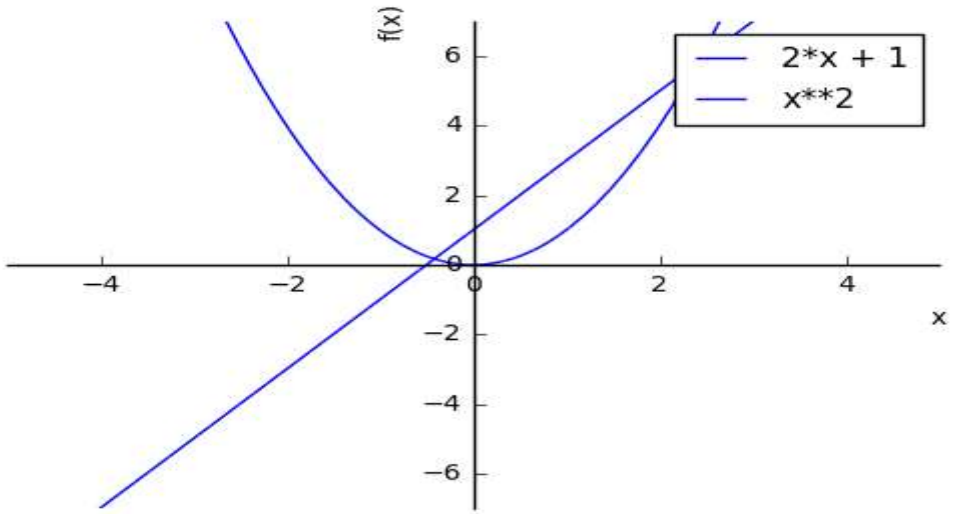


여러 개 그래프 그리기

두개 함수 그래프

1차,2차 함수를 만들고 plot 함수로 그래프 그리기

```
import sympy.plotting as plt
from sympy import Symbol

x = Symbol('x')
expr_2 = x*x


expr_1 = 2*x +1

plt.plot(expr_1,expr_2, (x,-5,5), ylim=(-7, 7),legend=True)
```

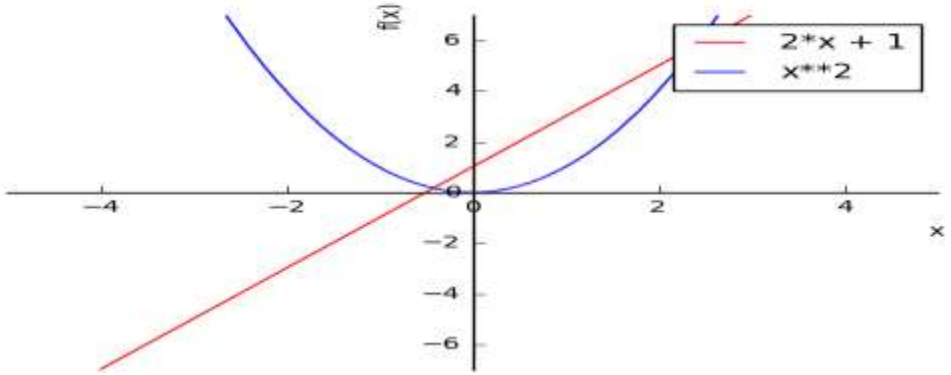


그래프 선 색상 변경하기

plot 함수 결과에 line_color 변수에 색상을 할당

```
import sympy.plotting as plt
from sympy import Symbol

x = Symbol('x')
expr_2 = x*x


expr_1 = 2*x +1

p = plt.plot(expr_1,expr_2, (x,-5,5), ylim=(-7, 7),legend=True,show=False)
print(p)
p[0].line_color = 'r'
p.show()
```

```
Plot object containing:
[0]: cartesian line: 2*x + 1 for x over (-5.0, 5.0)
[1]: cartesian line: x**2 for x over (-5.0, 5.0)
```

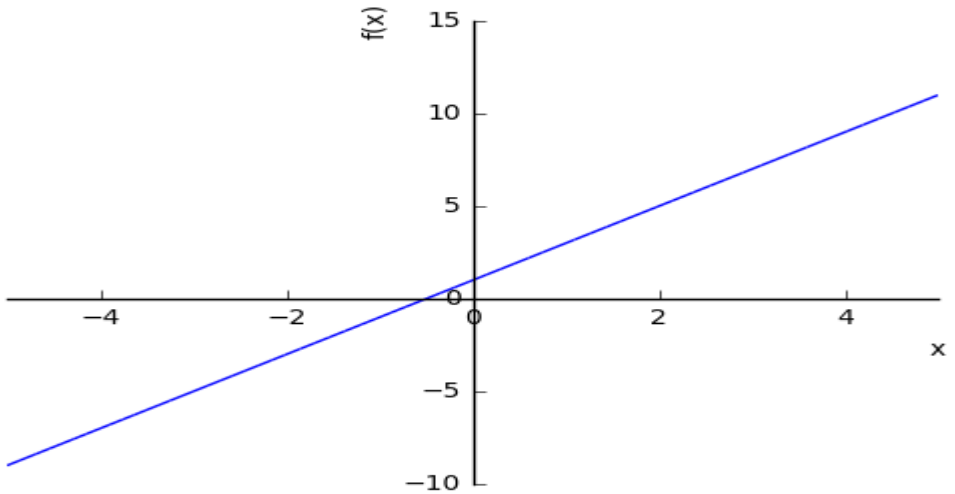


x축 범위

1차 함수 그래프 : x축 범위

1차 함수를 만들고 plot 함수에서 (x,-5,5)를 제공해서 x축 범위를 제한

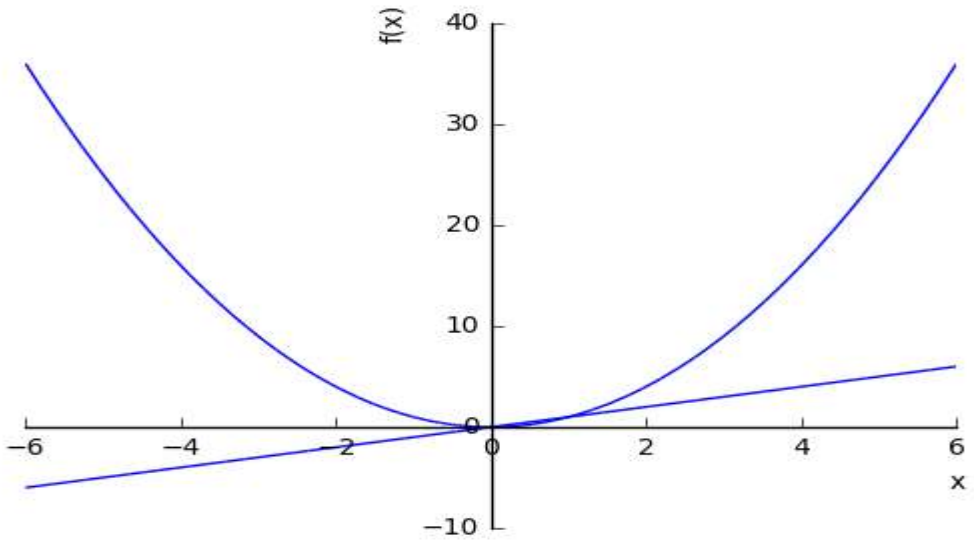
```
import sympy.plotting as plt
from sympy import Symbol

x = Symbol('x')
expr = 2*x +1

plt.plot(expr, (x,-5,5))
```



Plot 함수에서 그래프 통합 축 제한

1차함수와 2차함수의 x축이 제한을 통합해서 제한하려면 튜플로 하나만 표시

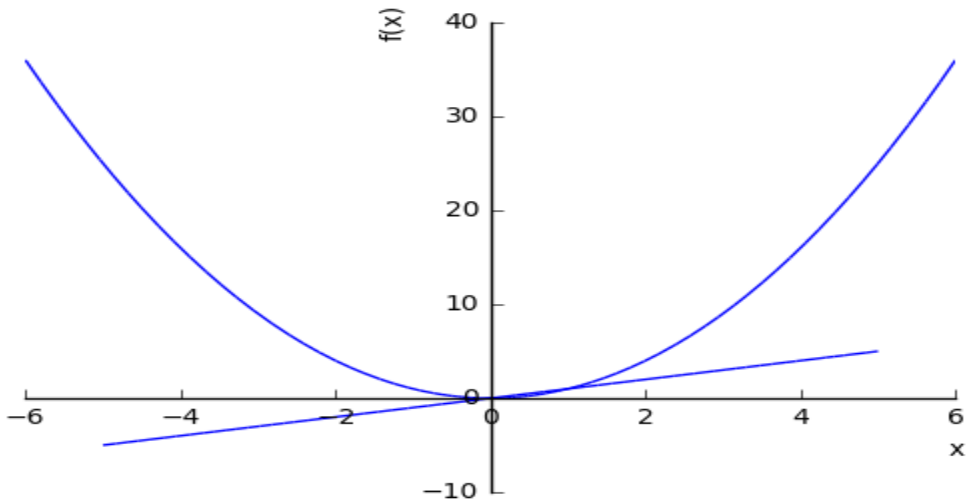
```
from sympy import symbols
from sympy.plotting import plot
x = symbols('x')
plot(x**2,x, (x, -6, 6))
```



Plot 함수에서 각 그래프별 축 제한

1차함수와 2차함수의 x축이 제한이 다르게 표시하려면 각 그래프별로 x축에 대한 제한을 별도로 넣어줘야 함

```
from sympy import symbols
from sympy.plotting import plot
x = symbols('x')
plot((x**2, (x, -6, 6)), (x, (x, -5, 5)))
```

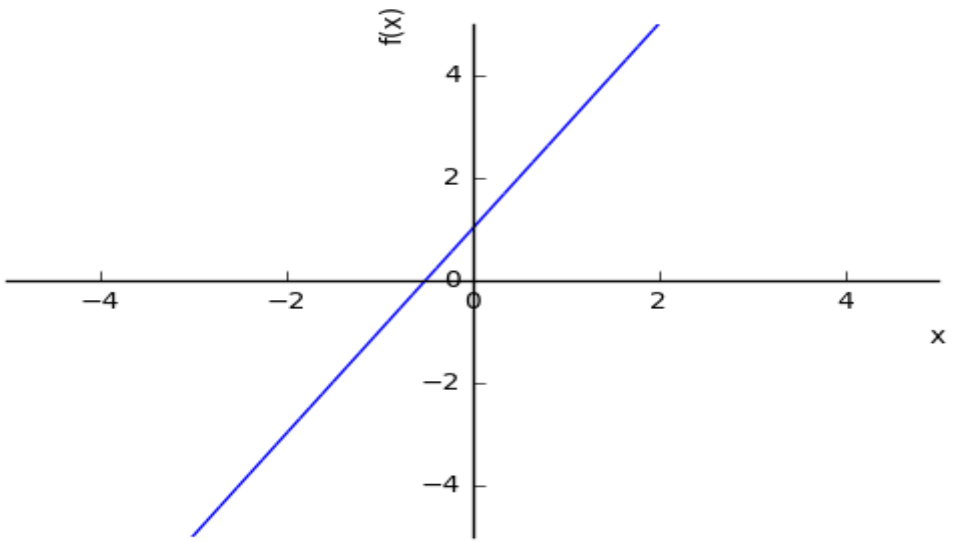


y축 범위

1차 함수 그래프 : y축 범위

1차 함수를 만들고 plot 함수에서 ylim=(y,-5,5)를 제공해서 y축 범위를 제한

```
import sympy.plotting as plt
from sympy import Symbol

x = Symbol('x')
expr = 2*x +1

plt.plot(expr, (x,-5,5), ylim=(-5, 5))
```

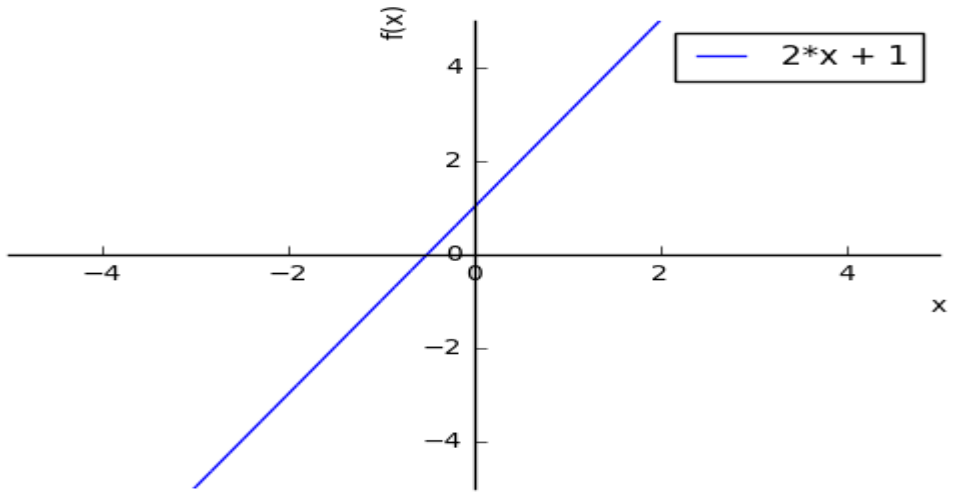


그래프 꾸미기

그래프 설명자료:legend

그래프에 대한 설명자료 legend

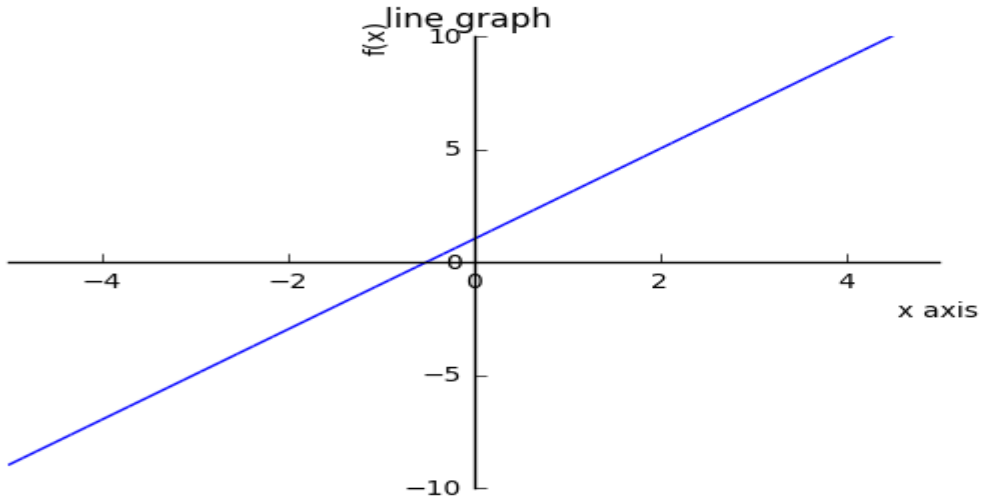
```
import sympy.plotting as plt
from sympy import Symbol

x = Symbol('x')
expr = 2*x +1

plt.plot(expr, (x,-5,5), ylim=(-5, 5),legend=True)
```



1차 함수 그래프 꾸미기

그래프 내의 텍스트 정보(title, xlabel, ylabel)를 제공해서 그래프 꾸미기

```
import sympy.plotting as plt
from sympy import Symbol

x = Symbol('x')
expr = 2*x +1

plot(expr, (x,-5, 5), ylim=(-10, 10),title= "line graph ",xlabel=" x axis ", ylable="y axis")
```



자동 show 차단하기

자동 그래프 표시 제한

sympy는 plot함수만 작동해도 그래프가 그려진다. show=False 파라미터를 제공해서 자동 그리기를 제한

```
import sympy.plotting as plt
from sympy import Symbol

x = Symbol('x')
expr_2 = x*x


expr_1 = 2*x +1

p = plt.plot(expr_1,expr_2, (x,-5,5), ylim=(-7, 7),legend=True,show=False)
print(p)
```

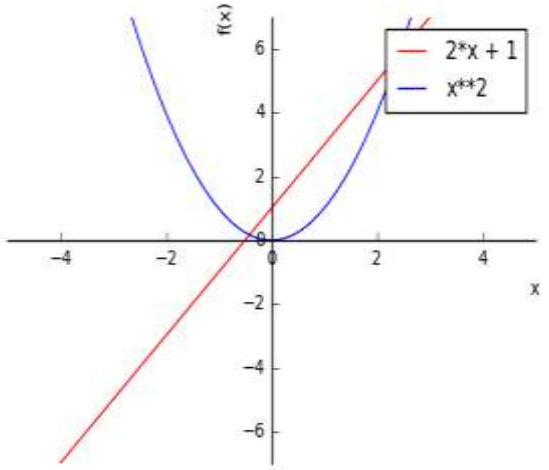
```
Plot object containing:
[0]: cartesian line: 2*x + 1 for x over (-5.0, 5.0)
[1]: cartesian line: x**2 for x over (-5.0, 5.0)
```

Show 메소드 실행하기

show메소드 사용

sympy는 plot함수만 작동해도 그래프가 그려진다. show=False 파라미터를 제공해서 자동 그리기를 제한

```
import sympy.plotting as plt
from sympy import Symbol

x = Symbol('x')
expr_2 = x*x


expr_1 = 2*x +1

p = plt.plot(expr_1,expr_2, (x,-5,5), ylim=(-7, 7),legend=True,show=False)
p[0].line_color = 'r'
p.show()
```



Multi plot 사용

다중 plot 사용하기 1

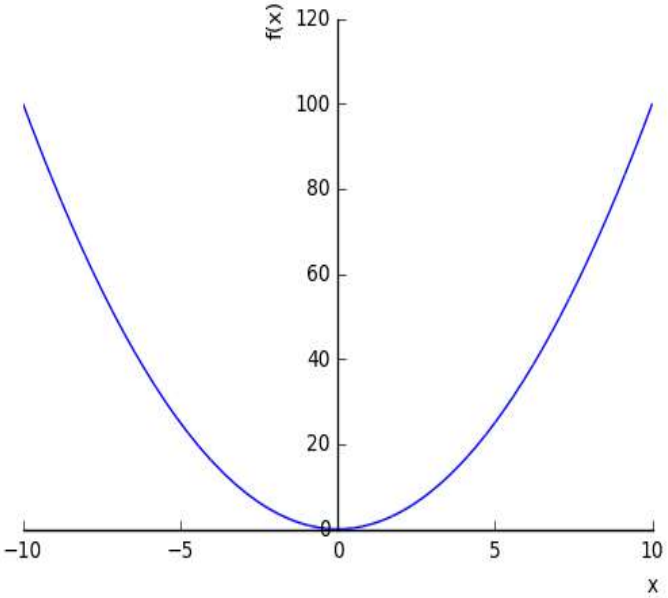
여러 plot 함수로 그래프를 그리면 각 plot별로 그래프가 그려짐

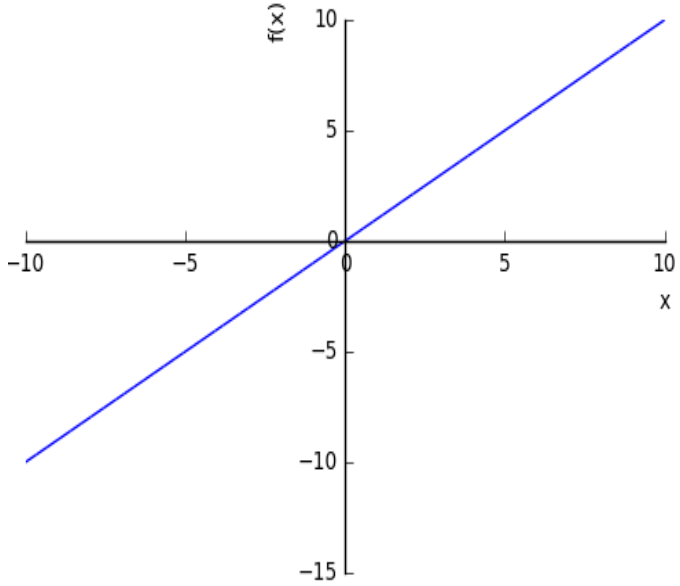
```
from sympy import symbols
from sympy.plotting import plot
x = symbols('x')
p1 = plot(x*x, show=False)
p2 = plot(x,show=False)

print(p1,p2)
p1.show()
p2.show()
```

```
(<sympy.plotting.plot.Plot object at 0x7fdac98eec90>, <sympy.plotting.plot.Plot object at 0x7fdac9697550>)
```

다중 plot 사용하기 : 2

여러 plot 함수로 그래프 그리기

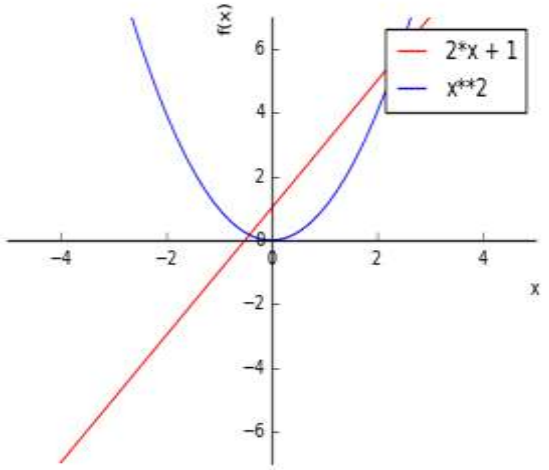




하나의 plot에 여러 그래프 넣기

하나의 그래프에 산식을 여러 개 넣으면 하나의 plot에 그래프가 통합해서 그려짐

```
import sympy.plotting as plt
from sympy import Symbol

x = Symbol('x')
expr_2 = x*x


expr_1 = 2*x +1

p = plt.plot(expr_1,expr_2, (x,-5,5), ylim=(-7, 7),legend=True,show=False)
p[0].line_color = 'r'
p.show()
```



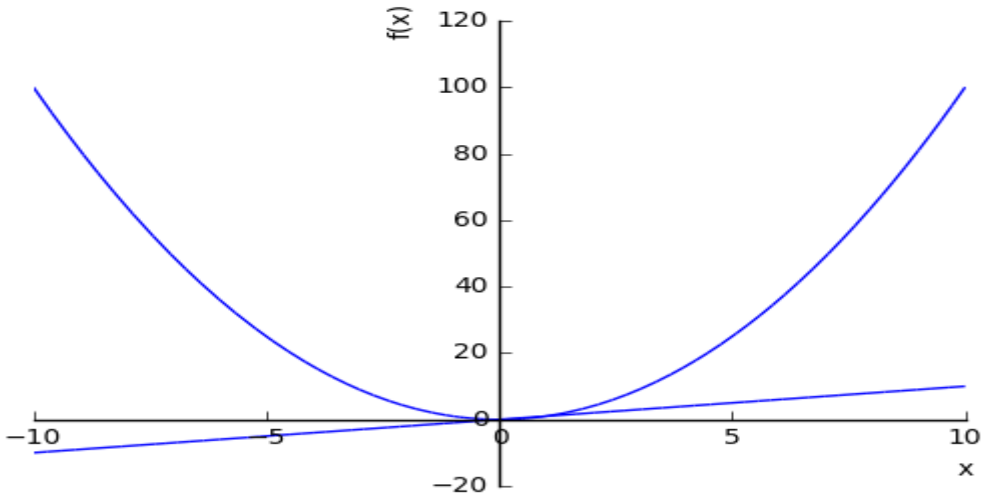
Multi plot 병합하기

append로 plot 추가

append 메소드로 plot 처리를 하나로 합치기

```
from sympy import symbols
from sympy.plotting import plot
x = symbols('x')
p1 = plot(x*x, show=False)
p2 = plot(x,show=False)
p1.append(p2[0])
print(p1)
p1.show()
```

```
Plot object containing:
[0]: cartesian line: x**2 for x over (-10.0, 10.0)
[1]: cartesian line: x for x over (-10.0, 10.0)
```

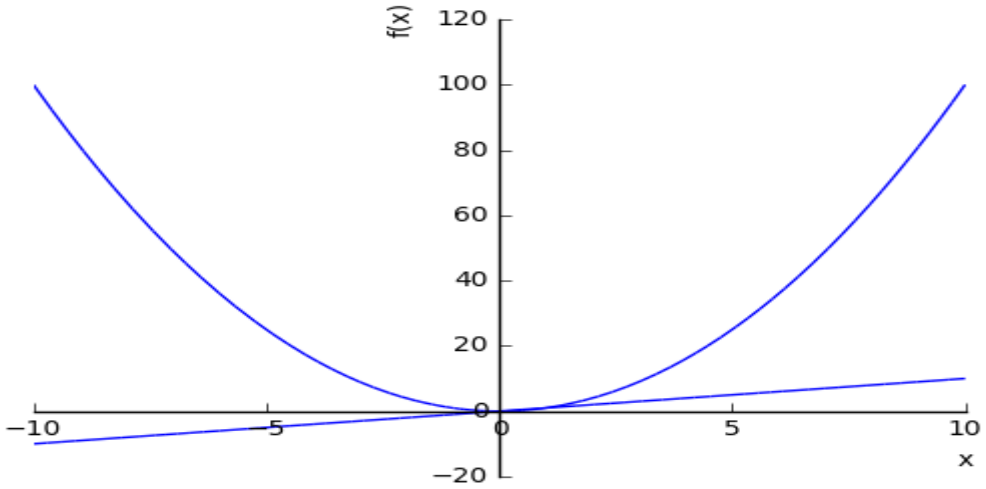


extend로 plot 추가

extend 메소드로 plot 처리를 하나로 합치기

```
from sympy import symbols
from sympy.plotting import plot
x = symbols('x')
p1 = plot(x*x, show=False)
p2 = plot(x,show=False)
p1.append(p2[0])
print(p1)
p1.show()
```

```
Plot object containing:
[0]: cartesian line: x**2 for x over (-10.0, 10.0)
[1]: cartesian line: x for x over (-10.0, 10.0)
```



PYTHON SYMPY 3D 그래프

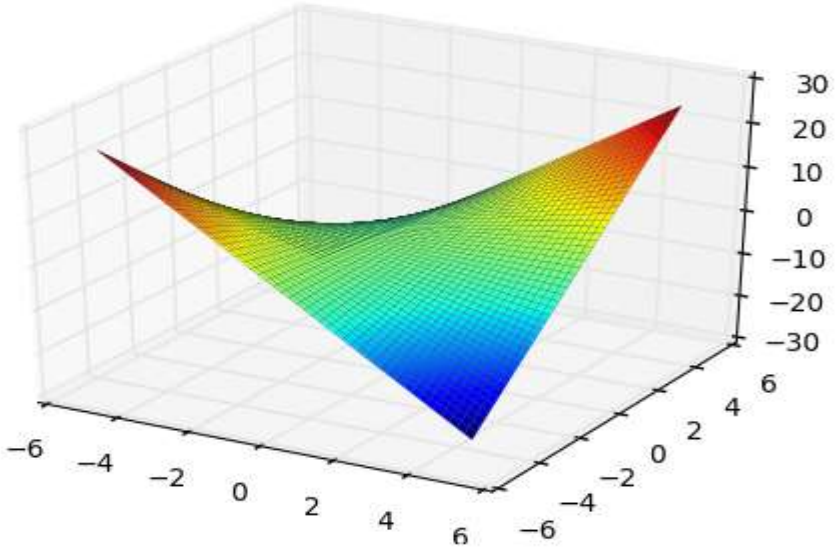
Moon Yong Joon

3d 그래프

plot3d: 하나의 산식 처리

산식에 대해 3D 그래프 그리기

```
from sympy import symbols
from sympy.plotting import plot3d
x, y = symbols('x y')
plot3d(x*y, (x, -5, 5), (y, -5, 5))
```

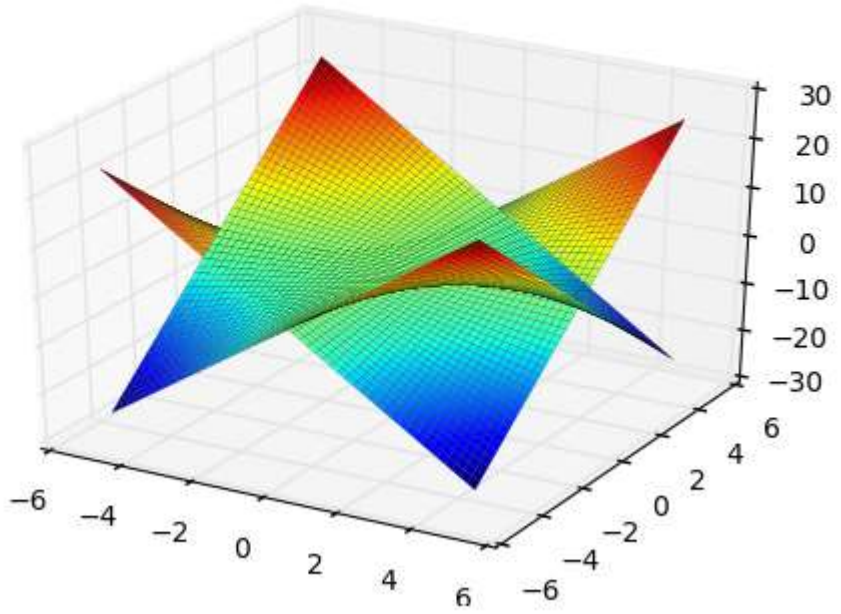



```
<sympy.plotting.plot.Plot at 0x7fdac9d8eed0>
```

plot3d: 2개의 산식처리

산식에 대해 3D 그래프 그리기

```
from sympy import symbols
from sympy.plotting import plot3d
x, y = symbols('x y')
plot3d(x*y, -x*y, (x, -5, 5), (y, -5, 5))
```



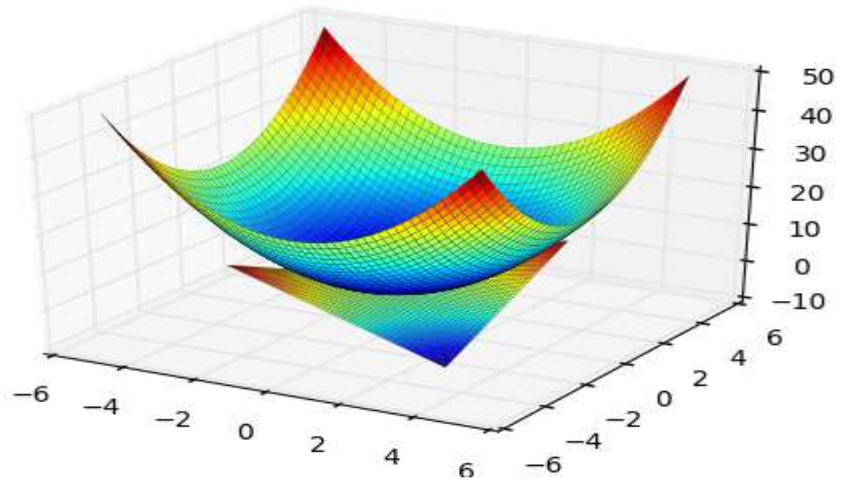

```
<sympy.plotting.plot.Plot at 0x7fdac9c6e610>
```

3D 축 제한하기

plot3d: 산식별로 축 제한하기

산식에 별도의 축을 제한해서 3D 그래프 그리기

```
from sympy import symbols
from sympy.plotting import plot3d
x, y = symbols('x y')
plot3d((x**2 + y**2, (x, -5, 5), (y, -5, 5)),
       (x*y, (x, -3, 3), (y, -3, 3)))
```




```
<sympy.plotting.plot.Plot at 0x7fdac98e4550>
```

PYTHON SYMPY 지수/로그함수 처리

Moon Yong Joon

Pow : e를 제외한 지수

Pow함수

base**a을 표시하는 Pow 함수를 정의해서 다양한 base에 대한 거듭제곱 구하기

```
from sympy import Pow,Symbol
x = Symbol('x', positive=True)
y = Symbol('y', positive=True)

print(Pow(x,y))
print(Pow(x,y).subs({x:10, y:2}))
```

```
x**y
100
```

Exp : e

exp함수

e에 대한 지수 함수로 e**x에 대한 표현식 처리

```
from sympy import E, exp,Symbol
import math
x = Symbol('x', positive=True)
y = Symbol('y', positive=True)

print(exp(x))
print(exp(x).subs({x:10}))
print(exp(x).subs({x:10}).evalf())

print(math.exp(10))
```

```
exp(x)
exp(10)
22026.4657948067
22026.4657948
```

log 함수 base:e 기준

log 함수 : base(e)

sympy는 기본 base가 e 이므로 지수값을 구하려면 exp(x)가 나온 결과를 log에 반영하면 실제 지수값을 구함

```
from sympy import E,log,exp,simplify,symbols
x,y = symbols('x,y')

print(log(x))
print(log(x).subs({x:E}).doit())
print(log(x).subs({x:E}).evalf())

# e**2 = a
a = exp(2).evalf()
print(a)

# ln a = 2
print(log(x).subs({x:a}).evalf())
```

```
log(x)
1
1.00000000000000
7.38905609893065
2.00000000000000
```

log 함수 base:10 기준

log 함수 : base(10)

sympy는 기본 base가 e 이므로 10진수로 base를 하기 위해서는 변경하려면 log정의 시 x,y 심벌을 정의하고 x에는 값, y에는 base를 입력해서 처리

```
from sympy import E,log,simplify,symbols
x,y = symbols('x,y')

print(log(x,y))
print(log(x,y).subs({x:100, y:10}).doit())
print(log(x,y).subs({x:100, y:10}).evalf())
```

```
log(x)/log(y)
log(100)/log(10)
2.00000000000000
```

log 함수 : base(10진수)

log 표현식 내의 evaluate=False로 정의해고 x, 와 base를 지정한 후 simplipy로 처리한 후에 로그 계산하면 실제 값이 나옴

```
from sympy import E,log,simplify,symbols
x,y = symbols('x,y')


print(simplify(log(x, 10, evaluate=False)))
a = simplify(log(x, 10, evaluate=False))
print(a.subs({x:100}))
print(a.subs({x:100}).doit())
print(a.subs({x:100}).evalf())
```

```
log(x)/log(10)
log(100)/log(10)
log(100)/log(10)
2.00000000000000
```

log함수 전개

log 함수 : x*y 전개

log 내부 심벌이 positive일 경우 log내의 파라미터(x*y)를 expand 메소드 처리시 2개의 log 함수의 덧셈으로 처리

Expand메소드 내의 log=True로 정의해도 전개

```
from sympy import E,log,exp,simplify,Symbol
x = Symbol('x', positive=True)
y = Symbol('y', positive=True)

print(log(x*y))
print(log(x*y).expand())
```

```
log(x*y)
log(x) + log(y)
```

```
from sympy import *

x, y = symbols('x, y', positive=True)
expr = log(x**2*y)
print expr, '=', expr.expand(log=True)
```

```
log(x**2*y) = 2*log(x) + log(y)
```

log 함수 : x/y 전개

log 내부 심벌이 positive일 경우 log내의 파라미터(x*y)를 expand 메소드 처리시 2개의 log 함수의 뺄셈으로 처리

```
from sympy import E,log,exp,simplify,Symbol
x = Symbol('x', positive=True)
y = Symbol('y', positive=True)

print(log(x/y))
print(log(x/y).expand())
```

```
log(x/y)
log(x) - log(y)
```

log 함수 : 복잡한 전개

log 함수에 대한 전개 내에 force=True를 정하면 복잡한 산식도 전개

```
from sympy import *
x, y, z = symbols('x, y, z')
expr = log(x**2*y)
print expr, '=', expr.expand()
print expr, '=', expr.expand(log=True)
print expr, '=', expr.expand(log=True, force=True)
```

```
log(x**2*y) = log(x**2*y)
log(x**2*y) = log(x**2*y)
log(x**2*y) = 2*log(x) + log(y)
```

expand_log함수 전개

expand_log 함수 : 복잡한 전개

symbol에 positive일 경우만 log 함수에 대한 전개 함수

```
from sympy import *
x = Symbol('x', positive= True)
y = Symbol('y', positive= True)
n = Symbol('m', positive= True)
t = Symbol('t', positive= True)
z = Symbol('t', positive= True)

print(ln(x))
print(expand_log(log(x*y)))
print(expand_log(log(x/y)))
print(expand_log(log(x**2)))
print(expand_log(log(x**n)))

print(expand_log(log(z*t)))
```

```
log(x)
log(x) + log(y)
log(x) - log(y)
2*log(x)
m*log(x)
2*log(t)
```

expand_log 함수 : 복잡한 전개

symbol에 positive일 아닐 경우에는 force=True를 지정해서 전개

```
from sympy import *
z = Symbol('z')
print(expand_log(log(z**2)))
print(expand_log(log(z**2), force=True))
```

```
log(z**2)
2*log(z)
```

logcombine함수 통합

logcombine함수 : 복잡한 통합

symbol에 positive일 경우는 log 함수에 대해 통합하는 함수, force=True는 지정되지 않는 심볼도 통합함

```
from sympy import *
x = Symbol('x', positive= True)
y = Symbol('y', positive= True)
a = Symbol('a', extended_real=True)

f = log(x)
g = log(y)
print(logcombine(f+g))
print(logcombine(a*log(x) + log(y) - log(z), force=True))
```

```
log(x*y)
log(x**a*y/z)
```

PYTHON SYMPY 삼각함수 기초

Moon Yong Joon

삼각함수 전개

expand 메소드 : 전개식 처리

삼각함수 전개가 필요할 경우에는 trig=True로 표시해야 함

```
from sympy import *
x,y = symbols('x, y')
print(expand(sin(x+y)))

# trig :  Do trigonometric expansions.
print(expand(sin(x+y), trig=True))
```

```
sin(x + y)
sin(x)*cos(y) + sin(y)*cos(x)
```

삼각함수

Sin 함수 처리

sin 함수 두개의 각을 더할 경우 아래의 공식으로 계산

$$\sin(\alpha+\beta)=\sin\alpha\cos\beta+\cos\alpha\sin\beta$$

```
import math
from sympy import *
x, y, z, t = symbols('x y z t')
f, g, h = symbols('f g h', cls=Function)

x= pi/4
y= pi/4
f = sin(x+y)

print(f.evalf())
print(math.sin(x+y))
```

```
1.00000000000000
1.0
```

```
import math
from sympy import *
x, y, z, t = symbols('x y z t')
f, g, h = symbols('f g h', cls=Function)

x= pi/4
y= pi/4
f = sin(x)*cos(y)+cos(x)*sin(y)

print(f.evalf())
print(math.sin(x+y))
```

```
1.00000000000000
1.0
```

Sin 함수 뺄셈 예시

sin 함수 두개의 각을 뺄 경우 아래의 공식으로 계산

$$\sin(\alpha-\beta)=\sin\alpha\cos\beta-\cos\alpha\sin\beta$$

```
import math
from sympy import *
x, y, z, t = symbols('x y z t')
f, g, h = symbols('f g h', cls=Function)

x= pi/2
y= pi/4
f = sin(x)*cos(y)-cos(x)*sin(y)

print(f.evalf())
print(math.sin(x-y))
```

```
0.707106781186548
0.707106781187
```

PYTHON SYMPY 집합 기초

Moon Yong Joon

집합 생성

FiniteSet : 집합생성

sympy FiniteSet을 이용해서 유한 집합을 생성

```
from sympy import FiniteSet
A = FiniteSet(1, 2, 3, 4)

print(A, type(A))
print(3 in A)
```

```
({1, 2, 3, 4}, <class 'sympy.sets.sets.FiniteSet'>)
True
```

FiniteSet : list 이용 생성

list로 생성된 객체는 unpack 처리한 인자로 전달해야 함

```
from sympy import FiniteSet

members = [1,3]
B = FiniteSet(*members)
print(B, type(B))
print(3 in B)
```

```
({1, 3}, <class 'sympy.sets.sets.FiniteSet'>)
True
```

부분집합

FiniteSet : 공집합

원소가 하나도 없는 집합을 공집합이라고 함

```
from sympy import S, FiniteSet
A = FiniteSet()
print(A)
print(S.EmptySet)
```

```
EmptySet()
EmptySet()
```

FiniteSet : 부분집합

부분집합의 is_subset 메소드는 자기가 속한 집합을 넣고 확인, is_superset 메소드의 인자는 부분집합을 전달해서 확인

```
from sympy import FiniteSet, ProductSet

 #'is_proper_subset',
 # 'is_proper_superset',
A = FiniteSet(1, 2, 3, 4)
members = [1,3]
B = FiniteSet(*members)

print(B.is_subset(A))
print(A.is_superset(B))
print(B.is_proper_subset(A))
print(A.is_proper_superset(B))
```

```
True
True
True
True
```

집합 연산

FiniteSet : 합집합

집합 A와 집합B의 모든 원소들 중에 중복되지 않는 것을 만듬

```
from sympy import FiniteSet, Union
A = FiniteSet(1, 2, 3, 4)
members = [1,3]
B = FiniteSet(*members)

C = Union(A,B)
print(C)
```

```
{1, 2, 3, 4}
```

FiniteSet : 교집합

집합 A와 집합B의 모든 원소들 중에 중복된 것만을 원소로 만듬

```
from sympy import FiniteSet, Intersection
A = FiniteSet(1, 2, 3, 4)
members = [1,3]
B = FiniteSet(*members)

C = Intersection(A,B)
print(C)
```

```
{1, 3}
```

FiniteSet : 차집합

집합 A의 원소와 집합B의 원소가 같을 경우 제거한 후에 집합 A에만 있는 원소로 집합을 만듬

```
from sympy import FiniteSet,Complement
A = FiniteSet(1, 2, 3, 4)
members = [1,3]
B = FiniteSet(*members)

C = Complement(A,B)

print(C)
print(2 in C)
```

```
{2, 4}
True
```

FiniteSet : product

집합 A와 집합B의 모든 원소을 순서쌍으로 만듬

```
from sympy import FiniteSet, ProductSet
A = FiniteSet(1, 2, 3, 4)
members = [1,3]
B = FiniteSet(*members)

C = ProductSet(A,B)

print(C)
print((1,1) in C)
```

```
{1, 2, 3, 4} x {1, 3}
True
```

PYTHON SYMPY 수열 기초

Moon Yong Joon

Summation(급수)

Sum : 합의 법칙

연속적인 수열의 합을 구하기

```
from sympy import *
S = Sum(i, (i, 1, n)).doit()
print(S)
print(S.subs({n:10}))
```

```
n**2/2 + n/2
55
```

Product : 곱의 법칙

연속적인 수열의 곱을 구하기

```
from sympy import *
S = Product(i, (i, 1, n)).doit()
print(S)
print(S.subs({n:3}))
```

```
factorial(n)
6
```

선형대수 (LINEAR ALGEBRA) 기초

Moon Yong Joon

행렬식

행렬식(行列式, determinant 은 정사각행렬에 수를 대응시키는 함수의 하나이다.

```
from sympy import *
m11, m12, m21, m22 = symbols("m11, m12, m21, m22")
A = Matrix([[m11, m12],[m21, m22]])
B = Matrix([[2, 0],[0, 2]])
print(A)
print('matrix determinant ',A.det())
print(B)
print('matrix determinant ',B.det())
```

```
Matrix([[m11, m12], [m21, m22]])
('matrix determinant ', m11*m22 - m12*m21)
Matrix([[2, 0], [0, 2]])
('matrix determinant ', 4)
```

역행렬 구하기

행렬과 역행렬을 dot 연산으로 단위행렬이 나오는 행렬

```
from sympy import *

m11, m12, m21, m22 = symbols("m11, m12, m21, m22")
A = Matrix([[m11, m12],[m21, m22]])
B = Matrix([[2, 0],[0, 2]])
print(A)

print(A.inv())
print(A.inv()*A)

print'matrix inv ',(B.inv())
print'matrix  ',(B.inv()*B)
```

```
Matrix([[m11, m12], [m21, m22]])
Matrix([[1/m11 + m12*m21/(m11**2*(m22 - m12*m21/m11)), -m12/(m11*(m22 - m12*m21/m11))], [-m21/(m11*(m22 - m12*m21/m11)), 1/(m22
 - m12*m21/m11)]])
Matrix([[m11*(1/m11 + m12*m21/(m11**2*(m22 - m12*m21/m11))) - m12*m21/(m11*(m22 - m12*m21/m11)), m12*(1/m11 + m12*m21/(m11**2*(m
22 - m12*m21/m11))) - m12*m22/(m11*(m22 - m12*m21/m11))], [0, m22/(m22 - m12*m21/m11) - m12*m21/(m11*(m22 - m12*m21/m11))]])
matrix inv  Matrix([[1/2, 0], [0, 1/2]])
matrix   Matrix([[1, 0], [0, 1]])
```

PYTHON SYMPY 확률/통계 기초

Moon Yong Joon

206

확률

Die: 유한확률변수 생성

유한확률변수를 생성하는 함수

```
from sympy.stats import Die, density
D6 = Die('D6', 6) # Six sided Die
print(D6,  type(D6))
print(density(D6).dict)
```

```
(D6, <class 'sympy.stats.rv.RandomSymbol'>)
{1: 1/6, 2: 1/6, 3: 1/6, 4: 1/6, 5: 1/6, 6: 1/6}
```

```
help(Die)
```

```
Help on function Die in module sympy.stats.frv_types:

Die(name, sides=6)
    Create a Finite Random Variable representing a fair die.

    Returns a RandomSymbol.
```

given : 특정 사건을 배정

특정 확률변수의 집합에서 표현식에 맞는 부분 집합을 만드는 함수

```
from sympy.stats import given, density, Die

X = Die('X', 6)
print(density(X).dict)

Y = given(X, X > 3)
print(Y, type(Y))
print(density(Y).dict)
```

```
{1: 1/6, 2: 1/6, 3: 1/6, 4: 1/6, 5: 1/6, 6: 1/6}
(X, <class 'sympy.stats.rv.RandomSymbol'>)
{4: 1/3, 5: 1/3, 6: 1/3}
```

확률밀도함수

density : 이산형 데이터

이산 데이터에 대해 확률분포에 대한 데이터를 산출해서 dict으로 값을 나타냄

```
from sympy.stats import DiscreteUniform, density
from sympy import symbols

X = DiscreteUniform('X', symbols('a b c'))
print(density(X).dict)

Y = DiscreteUniform('Y', list(range(5)))
print(density(Y).dict)
```

```
{c: 1/3, b: 1/3, a: 1/3}
{0: 1/5, 1: 1/5, 2: 1/5, 3: 1/5, 4: 1/5}
```

density : 연속형 데이터

연속 데이터는 lambdas 즉 함수로 나타냄

```
from sympy.stats import density, Die, Normal
from sympy import symbols
x = Symbol('x')
X = Normal(x, 0, 1)

print(density(X))
print(density(X)(x))
```

```
NormalDistribution(0, 1)
sqrt(2)*exp(-x**2/2)/(2*sqrt(pi))
```

PYTHON SYMPY 극한 기초

Moon Yong Joon

극한

limit 함수 : 극한

극한을 처리하는 함수limit

```
from sympy import limit, sin, Symbol, oo
from sympy.abc import x

# limit 함수 파라미터 : 함수, 변수, limit 값 , dir은 극한값으로 접근방향
print(limit(sin(x)/x, x, 0))
# 우 극한값으로 접근, x>0 크면서 접근
print(limit(1/x, x, 0, dir="+"))
#좌 극한값으로 접근, x <0 작으면서 접근
print(limit(1/x, x, 0, dir="-"))

print(limit(1/x, x, oo))
```

```
1
oo
-oo
```

PYTHON SYMPY 미분 기초

Moon Yong Joon

미분

diff 함수 : 미분

diff 함수로 미분을 풀이

```
from sympy import *
x, y, z = symbols("x y z")
f, g, h = symbols('f g h', cls=Function)
f = expand((x + 1)**20)

g = diff(f, x)
print(g)
print(factor(g))
```

```
20*x**19 + 380*x**18 + 3420*x**17 + 19380*x**16 + 77520*x**15 + 232560*x**14 + 542640*x**13 + 1007760*x**12 + 1511640*x**11 + 1
847560*x**10 + 1847560*x**9 + 1511640*x**8 + 1007760*x**7 + 542640*x**6 + 232560*x**5 + 77520*x**4 + 19380*x**3 + 3420*x**2 + 3
80*x + 20
20*(x + 1)**19
```

상수미분 : sympy

상수의 미분은 실제 0이 됨

```
from __future__ import division
from sympy import *

print(diff(2, x))
```

```
0
```

```
help(diff)
```

```
Help on function diff in module sympy.core.function:

diff(f, *symbols, **kwargs)
    Differentiate f with respect to symbols.

    This is just a wrapper to unify .diff() and the Derivative class; its
    interface is similar to that of integrate().  You can use the same
    shortcuts for multiple variables as with Derivative.  For example,
    diff(f(x), x, x, x) and diff(f(x), x, 3) both return the third derivative
    of f(x).

    You can pass evaluate=False to get an unevaluated Derivative class.  Note
    that if there are 0 symbols (such as diff(f(x), x, 0), then the result will
    be the function (the zeroth derivative), even if evaluate=False.
```

실수배 미분 : sympy

c라는 실수배를 가진 함수의 미분에서 실수배는 미분에 영향을 미치지 않는다

y=cf(x)

$$y' = \lim_{\Delta x \to 0} \frac{cf(x+\Delta x) - cf(x)}{\Delta x} = c \lim_{\Delta x \to 0} \frac{f(x+\Delta x) - f(x)}{\Delta x} = cf'(x)$$

```
from __future__ import division
from sympy import *

x  = symbols('x')
print(diff(2*x**2, x))
```

```
4*x
```

함수의 미분 : sympy

n 이 자연수일 경우 x^n의 미분은 nx^{n-1}

```
from __future__ import division
from sympy import *

x  = symbols('x')
n  = symbols('n', integer=True)
n=10
print(diff(x**n,x))
```

```
10*x**9
```

함수의 합 미분

합과차의 미분 : sympy

두 함수 합에 대한 미분은 각 함수의 미분에 합과 같다

$$
\begin{aligned}
(f(x)+g(x))' &= \lim_{\Delta x \to 0} \frac{f(x+\Delta x)+g(x+\Delta x)-(f(x)+g(x))}{\Delta x} \\
&= \lim_{\Delta x \to 0} \frac{f(x+\Delta x)-f(x)+g(x+\Delta x)-g(x)}{\Delta x} \\
&= \lim_{\Delta x \to 0} \frac{f(x+\Delta x)-f(x)}{\Delta x} + \lim_{\Delta x \to 0} \frac{g(x+\Delta x)-g(x)}{\Delta x} \\
&= f'(x)+g'(x)
\end{aligned}
$$

```
from __future__ import division
from sympy import *
x, y, z, t = symbols('x y z t')
k, m, n = symbols('k m n', integer=True)
f, g, h = symbols('f g h', cls=Function)

n = 2
f = x**n
g = x**n
h = f + g

print(diff(f,x))
print(diff(h,x))
```

```
2*x
4*x
```

함수의 곱 미분

곱의 미분

두 함수 곱에 대한 미분은 f(x) 함수 미분과 g(x) 함수의 곱 + f(x)함수와 g(x)함수 미분의 곱과의 합과 같다

$$(f(x)g(x))' = \lim_{\Delta x \to 0} \frac{f(x+\Delta x)g(x+\Delta x) - (f(x)g(x))}{\Delta x}$$

$$= \lim_{\Delta x \to 0} \frac{f(x+\Delta x)g(x+\Delta x) - f(x)g(x+\Delta x) + f(x)g(x+\Delta x) - (f(x)g(x))}{\Delta x}$$

$$= \lim_{\Delta x \to 0} \frac{f(x+\Delta x) - f(x)}{\Delta x} \lim_{\Delta x \to 0} g(x+\Delta x) + \lim_{\Delta x \to 0} \frac{g(x+\Delta x) - g(x)}{\Delta x} \lim_{\Delta x \to 0} f(x)$$

$$= f'(x)g(x) + f(x)g'(x)$$

x*x 를 미분하면
즉 f(x) = x,
g(x)= x 이므로
= 1*x + x*1
= x + x
= 2x

곱의 미분 : sympy

두 함수 곱에 대한 미분은 f(x) 함수 미분과 g(x) 함수의 곱 + f(x)함수와 g(x)함수 미분의 곱과의 합과 같다

```
from __future__ import division
from sympy import *
x, y, z, t = symbols('x y z t')
k, m, n = symbols('k m n', integer=True)
f, g, h = symbols('f g h', cls=Function)

n = 2
f = x**n
g = x**n
h = f * g

print(diff(f,x))
print(diff(g,x))
print(diff(h,x))
print(diff(f,x)*g+ f*diff(g,x))
```

```
2*x
2*x
4*x**3
4*x**3
```

미분 공식

상수와 1차 함수 미분

단순 함수의 미분

$$\frac{d}{dx}c = 0$$

$$\frac{d}{dx}x = 1$$

$$\frac{d}{dx}|x| = \frac{x}{|x|} = \operatorname{sgn} x, \qquad x \neq 0$$

```
: from __future__ import division
from sympy import *
x, y, z, t = symbols('x y z t')
k, m, n = symbols('k m n', integer=True)
f, g, h = symbols('f g h', cls=Function)

n = 2
f = 10
g = x

h = abs(x)
print(h)

print(diff(f,x))
print(diff(g,x))
print(diff(h,x))
```

```
Abs(x)
0
1
(re(x)*Derivative(re(x), x) + im(x)*Derivative(im(x), x))/Abs(x)
```

분수와 제곱근 미분 :sympy

단순 함수의 미분

$$\frac{d}{dx}x^c = cx^{c-1}$$

$$\frac{d}{dx}\sqrt{x} = \frac{1}{2\sqrt{x}}$$

$$\frac{d}{dx}\left(\frac{1}{x}\right) = -\frac{1}{x^2}$$

```
from __future__ import division
from sympy import *
x, y, z, t = symbols('x y z t')
k, m, n = symbols('k m n', integer=True)
f, g, h = symbols('f g h', cls=Function)

n = 2
f = 1/x
g = sqrt(x)
h = x**n

print(diff(f,x))
print(diff(g,x))
print(diff(h,x))
```

```
-1/x**2
1/(2*sqrt(x))
2*x
```

반복 미분

미분을 2번할 경우 :sympy

미분을 여러 번 할 경우 미분변수 다음에 차수를 숫자로 표시하면 그 수 만큼 미분을 처리함

```
from sympy import *
x,y = symbols('x, y')

print(diff(y**2,y))
print(diff(diff(y**2,y)))
print(diff(y**2,y,2))
```

```
2*y
2
2
```

삼각함수 미분

삼각함수 :sympy

삼각함수에 대한 미분 구하기

함수 $f(x)$	도함수 $f'(x)$
$\sin x$	$\cos x$
$\cos x$	$-\sin x$
$\tan x$	$\sec^2 x$
$\cot x$	$-\csc^2 x$
$\sec x$	$\sec x \tan x$
$\csc x$	$-\csc x \cot x$

```
from sympy import *
x = Symbol('x')

print(sin(x).diff(x))
print(cos(x).diff(x))
print(tan(x).diff(x))
print(sec(x).diff(x))
print(csc(x).diff(x))
print(cot(x).diff(x))
print("integrate ")
print(sin(x).diff(x).integrate(x))
print(cos(x).diff(x).integrate(x))
print(tan(x).diff(x).integrate(x))
print(sec(x).diff(x).integrate(x))
print(csc(x).diff(x).integrate(x))
print(cot(x).diff(x).integrate(x))
```

```
cos(x)
-sin(x)
tan(x)**2 + 1
tan(x)*sec(x)
-cot(x)*csc(x)
-cot(x)**2 - 1
integrate
sin(x)
cos(x)
sin(x)/cos(x)
1/cos(x)
1/sin(x)
cos(x)/sin(x)
```

편미분

편미분 :sympy

미분에 해당되지 않는 변수는 상수로 인지하고 diff 편미분시 미분대상 변수를 지정해 주고 미분계수를 표시하면 됨

```
from sympy import *
x,y = symbols('x, y')
print(diff(y**2, x))
print(diff(y**2,y))

print(diff(y**2,y,2))

print(diff(x**2 * y**2,x,1,y,2))
```

```
0
2*y
2
4*x
```

적분(INTEGRATION) 기초

Moon Yong Joon

적분 공식

일반 적분 공식 : sympy

미분은 nx^{n-1} 이므로 X에 대한 적분은 일단 상수를 반대로 제거하고 지수를 올리는 방식으로 처리1/n+1*x**n+1로 처리

```
from sympy import integrate, log, exp, oo

x = symbols('x')
print(integrate(x, x))
```

```
x**2/2
```

$\frac{1}{2} * x^2$ 을 미분하면
$= 2 * 1/2 \; x^{2-1}$
$= x$

상수와 함수: sympy

일반 적분 공식

$$\int af(x)\,dx = a\int f(x)\,dx \qquad (a \text{ constant}) \qquad \int k\,dx = kx + C$$

```
from sympy import integrate, log, exp, oo

a = symbols('k m n', integer=True)
f, g, h = symbols('f g h', cls=Function)
x = symbols('x')
f=x**2
print(integrate(f,x))
print(integrate(2*f, x))
```

```
x**3/3
2*x**3/3
```

n차 함수: sympy

일반 적분 공식

$$\int x^n \, dx = \frac{x^{n+1}}{n+1} + C \qquad (\text{for } n \neq -1)$$

```
from sympy import integrate, log, exp, oo

a = symbols('a', integer=True)
f, g, h = symbols('f g h', cls=Function)
x,n = symbols('x n')
f=x**n


print(integrate(f, x))
```

```
Piecewise((log(x), Eq(n, -1)), (x**(n + 1)/(n + 1), True))
```

함수의 곱: sympy

일반 적분 공식

$$\int f(x)g(x)\,dx = f(x)\int g(x)\,dx - \int \left[f'(x)\left(\int g(x)\,dx\right)\right]dx$$

```
from sympy import integrate, log, exp, oo

a = symbols('k m n', integer=True)
f, g, h = symbols('f g h', cls=Function)
x = symbols('x')
f=x**2
g=x
h = f*g

print(integrate(h, x))
print(f*integrate(g,x)-integrate(diff(f,x)*integrate(g,x)))
```

```
x**4/4
x**4/4
```

도함수와 함수 곱: sympy

일반 적분 공식

$$\int f'(x)f(x)\,dx = \frac{1}{2}[f(x)]^2 + C$$

```
from sympy import integrate, log, exp, oo

a = symbols('k m n', integer=True)
f, g, h = symbols('f g h', cls=Function)
x = symbols('x')
f=x**2


print(diff(f,x))
print(diff(f,x)*f)
print(integrate(diff(f,x)*f, x))
```

```
2*x
2*x**3
x**4/2
```

삼각함수 적분

삼각함수 :sympy

삼각함수에 대한 적분 구하기

함수 $f(x)$	도함수 $f'(x)$
$\sin x$	$\cos x$
$\cos x$	$-\sin x$
$\tan x$	$\sec^2 x$
$\cot x$	$-\csc^2 x$
$\sec x$	$\sec x \tan x$
$\csc x$	$-\csc x \cot x$

```
from sympy import *
x = Symbol('x')

print(sin(x).diff(x))
print(cos(x).diff(x))
print(tan(x).diff(x))
print(sec(x).diff(x))
print(csc(x).diff(x))
print(cot(x).diff(x))
print("integrate ")
print(sin(x).diff(x).integrate(x))
print(cos(x).diff(x).integrate(x))
print(tan(x).diff(x).integrate(x))
print(sec(x).diff(x).integrate(x))
print(csc(x).diff(x).integrate(x))
print(cot(x).diff(x).integrate(x))
```

```
cos(x)
-sin(x)
tan(x)**2 + 1
tan(x)*sec(x)
-cot(x)*csc(x)
-cot(x)**2 - 1
integrate
sin(x)
cos(x)
sin(x)/cos(x)
1/cos(x)
1/sin(x)
cos(x)/sin(x)
```