**Запуск notebook**

ipython notebook

**Основы**

**Вывод красиво (в начале пишем)**

import sympy

sympy.init\_printing()

%pylab inline

**Числа в python**

В Python есть два типа для хранения чисел int и float

У int работает целочисленное деление (берется целая часть результата)

>>> type(3)

int

>>> type(3.14)

float

>>> 1/3

0

Встроенная функция sqrt может *вычислить* квадратный корень

>>> import math as mth

>>> mth.sqrt(9)

3.0

>>> mth.sqrt(8)

2.82842712475

Иногда дроби плохо считать в float , например, 1/3 + 2/3 это 1. Но если посчитать в виде десятичных дробей, то потеряем точность и не получим 1.

Хочется считать точно и не терять корни.

>>> import sympy

>>> sympy.sqrt(3)

sqrt(3)

>>> sympy.sqrt(8)

2\*sqrt(2)

%TODO% Резервированные константы (e, pi) и греческие буквы

DEA! Дальше считаем, что у нас работают функции из sympy

**Символы - symbols**

В python можно было первое использование переменной было ее созданием

x = 3

В sympy так нельзя

>>> x + 1

Traceback (most recent call last):

...

NameError: name 'x' is not defined

Traceback (most recent call last):

...

NameError: name 'x' is not defined

DEA! В sympy нужно **сначала определить переменные**

>>> x = symbols('x')

>>> x + 1

x + 1

или сразу **несколько переменных**

>>> x, y, z = symbols('x y z')

Имя переменной и символ может состоять из нескольких букв

>>> height = symbols('height')

Имя переменной может не совпадать с ее символом.

>>> crazy = symbols('unrelated')

>>> crazy + 1

unrelated + 1

**Нельзя делать символы Q,C,O,S,I,N и E**   
Они имеют специальные значения.

**Выражения**

Можно записать несколько переменных в выражение. Можно придумать любое имя выражению, например, expr или aaa.

>>> expr = 2\*x + 3\*x - sin(x) - 3\*x + 42

>>> expr

2\*x + 3\*x - sin(x) - 3\*x + 42

>>> aaa = x\*\*2 + 2\*x +1

>>> aaa

x\*\*2 + 2\*x +1

Еще выражений

>>> expr = x + 2\*y

>>> expr

x + 2\*y

>>> expr + 1

x + 2\*y + 1

>>> expr - x

2\*y

>>> x\*expr

x\*(x + 2\*y)

**Математические функции и константы**

DEA! Для работы с символом ∞ как oo не забудьте или писать sm.oo или

>>> from sympy import oo

|  |  |
| --- | --- |
| oo | ∞ |
| sqrt(x) | x \*\* 0.5 |
| root(8, 3) | 2 = корень 3 степени из 8 |
| root(x, n) | x \*\* (1.0/n) |
| factorial(n) | n! |
| sin(x) cos(x) tan(x) cot(x) | sin, cos, tangence, cotangence |
| log(x) | ln(x) |
| log(x, b) | logb(x) |

**simplify - упростить выражение**

>>> simplify(sin(x)\*\*2 + cos(x)\*\*2)

1

**expand, factor, collect - раскрыть скобки, сгруппировать, в форме полинома**

>>> expr = x + 2\*y

>>> expanded\_expr = expand(x\*expr)

>>> expanded\_expr

x\*\*2 + 2\*x\*y

>>> factor(expanded\_expr)

x\*(x + 2\*y)

>>> factor( x\*\*2-2\*x-8 )

(x - 4)\*(x + 2)

>>> expand( (x-4)\*(x+2) )

x\*\*2 - 2\*x - 8

>>> collect(x\*\*2 + x\*b + a\*x + a\*b, x)

x\*\*2 + (a+b)\*x + a\*b # collect terms for diff. pows of x

**тригонометрические функции trigsimp и expand\_trig**

>>> expand\_trig(sin(x + y))

sin(x)&#8901;cos(y) + sin(y)&#8901;cos(x)

>>> trigsimp(sin(x)\*cos(y) + sin(y)\*cos(x))

sin(x + y)

**subs - подстановка, evalf, n, N - вычислить**

>>> expr = sin(x) + cos(y)

>>> expr

sin(x) + cos(y)

>>> expr.subs({x:1, y:2})

sin(1) + cos(2)

>>> expr.subs({x:1, y:2}).n()

0.425324148260754

**apart, together - работа с дробями**

>>> 1/( (x+2)\*(x+1) )

1

---------------

(x + 1)\*(x + 2)

>>> apart(1/( (x+2)\*(x+1) ), x)

1 1

- ----- + -----

x + 2 x + 1

>>> (x+1)/(x-1)

x + 1

-----

x - 1

>>> apart((x+1)/(x-1), x)

2

1 + -----

x - 1

together - привести к общему знаменателю

>>> together(1/x + 1/y + 1/z)

x\*y + x\*z + y\*z

---------------

x\*y\*z

**Limit**

limit(функция, переменная, значение) - найти предел функции при переменной стремящейся к значению

>>> limit(sin(x)/x, x, 0)

1

>>> limit(x, x, oo)

oo

>>> limit(1/x, x, oo)

0

>>> limit(x\*\*x, x, 0)

1

**series Разложить в ряд**

Функция .series(переменная, точка, порядок) - разложить функцию по переменной в окрестностях точки с точностью до порядка

>>> cos(x).series(x, 0, 10)

2 4 6 8

x x x x / 101 - — + — - — + ----- + O\x /

2 24 720 40320

>>> (1/cos(x)).series(x, 0, 10)

2 4 6 8

x 5\*x 61\*x 277\*x / 101 + — + ---- + ----- + ------ + O\x /

2 24 720 8064

**summation Сумма**

b

\_\_\_\_

\ `

summation(f, (i, a, b)) = ) f

/\_\_\_,

i = a

>>> summation(2\*i - 1, (i, 1, n))

2

n

>>> summation(1/2\*\*i, (i, 0, oo))

2

>>> summation(1/log(n)\*\*n, (n, 2, oo))

oo

\_\_\_

\ `

\ -n

/ log (n)

/\_\_,

n = 2

>>> summation(i, (i, 0, n), (n, 0, m))

3 2

m m m

-- + — + -

6 2 3

>>> summation(x\*\*n/factorial(n), (n, 0, oo))

x

e

**diff Производная**

diff(func, var) производная функции func по переменной var

>>> diff(sin(x), x)

cos(x)

>>> diff(sin(2\*x), x)

2\*cos(2\*x)

>>> diff(tan(x), x)

2

tan (x) + 1

diff(func, var, n) производная функции func по переменной var порядка n

>>> diff(sin(2\*x), x, 1)

2\*cos(2\*x)

>>> diff(sin(2\*x), x, 2)

-4\*sin(2\*x)

>>> diff(sin(2\*x), x, 3)

-8\*cos(2\*x)

%?% Производная от двух переменных

**integrate Интегрирование**

**Неопределенный интеграл**

>>> integrate(6\*x\*\*5, x)

6

x

>>> integrate(sin(x), x)

-cos(x)

>>> integrate(log(x), x)

x\*log(x) - x

>>> integrate(2\*x + sinh(x), x)

2

x + cosh(x)

**Определенный интеграл**

>>> integrate(x\*\*3, (x, -1, 1))

0

>>> integrate(sin(x), (x, 0, pi/2))

1

>>> integrate(cos(x), (x, -pi/2, pi/2))

2

>>> integrate(exp(-x), (x, 0, oo))

1

>>> integrate(log(x), (x, 0, 1))

-1

**solve - решение уравнений и систем уравнений**

Для решения уравнения вида f(x) = 0

>>> solve( x\*\*2 + 2\*x - 8, x)

[2, 4]

Сохраним корни уравнения в список (list)

>>> gen\_sol = solve( a\*x\*\*2 + b\*x + c, x)

>>> [ gen\_sol[0].subs({'a':1,'b':2,'c':-8}),

gen\_sol[1].subs({'a':1,'b':2,'c':-8}) ]

[2, -4]

Для решения системы уравнений x + y - 3=0 и 3\*x - 2\*y = 0 напишем список функций в виде [f(x), g(x)]

>>> solve([x + y - 3, 3\*x - 2\*y], [x, y])

{x: 6/5, y: 9/5}

**Ссылки**

* wakari / Gallery / Lecture 5 Sympy J. Robert Johansson

In[26]: integrate(x\*root(1-x,3), (x, 1, 9))

# получаем exception, нужно делать предположение, что 1-x не отрицательно. Заменим переменные.

In[27]: i = Integral(x\*root(1-x,3), (x, 1, 9))

Out[27]:Integral(x\*(-x + 1)\*\*(1/3), (x, 1, 9))

In[28]: u = symbols('u', positive=True)

iu = i.transform(1-x, u)

Out[28]:Integral(u\*\*(1/3)\*(-u + 1), (u, -8, 0))

In[29]: iu.doit()

Out[29]:468\*(-1)\*\*(1/3)/7