Практическое занятие 1

Алгебра

https://docs.sympy.org/latest/tutorial/intro.html

```
In [ ]:
#Вначале для простоты будем подключать модуль sympy целиком
from sympy import *
```

Действия с числами, числовые выражения

Об основных типах данных Python 3.8 читайте здесь: https://docs.python.org/3/reference/datamodel.html#index-19

Сейчас нам понадобятся типы:

int - целое число (насколько большим оно может быть, зависит только от объема доступной памяти)

float - вещественное число (double precision floating point numbers)

Пример 1.

Произведение чисел 3000 и 50000

```
In [14]:
3000*50000
```

```
Out[14]: 150000000
```

Пример 2.

Целая часть от деления 8 на 5 и остаток от деления

```
In [5]:
print(8//5, 8%5)
display(8//5, 8%5)
```

```
1 3

1

3
```

Пример 3.

Частное 8 и 5 - тип float

```
In [4]:
8/5, type(8/5)
```

```
Out[4]: (1.6, float)
```

Пример 4.

Далее будем пользоваться функцией display() для красивого вывода математических формул там, где это не получается автоматически. Сравнения результат с print.

Посмотрим, как выглядят значения некоторых математических функций из sympy:

```
In [9]:
display(sqrt(8), sin(pi/3), atan(0))
print(sqrt(8), sin(pi/3), atan(0))
```

$$2\sqrt{2}$$

$$\frac{\sqrt{3}}{2}$$

$$0$$

```
2*sqrt(2) sqrt(3)/2 0
```

```
In [6]:
float(sqrt(8)), float(sin(9)), float(atan(0))
```

```
Out[6]: (2.8284271247461903, 0.4121184852417566, 0.0)
```

Символы, символьные выражения

Для аналитических преобразований в sympy используется класс Symbol https://docs.sympy.org/latest/modules/core.html?highlight=symbol#module-sympy.core.symbol В этом классе есть метод Symbol для создания одного символа.

Пример 5.

Создадим символ x и используем его для составления выражения

```
In [7]:
x = Symbol('x')
expr1 = x + 2**x
expr2 = sin(x)/x
expr1*expr2
```

Out[7]:

$$\frac{(2^x + x)\sin(x)}{x}$$

Можно создавать несколько символов сразу.

Пример 6.

Создадим символы y и z и упростим выражение с ними. Обратим внимание, что используется symbols, а не Symbol!

```
In [8]:
y, z = symbols('y, z')
simplify(y**2 + 1 - z*y**2 + 5*z**3 - 3*y**2)
```

Out[8]:

$$-y^2z - 2y^2 + 5z^3 + 1$$

Пример 7.

Создадим символы $x_0, \dots x_9$ и составим выражение с ними

Здесь используется цикл for со счетчиком i, значения счетчика берутся из диапазона range(start, end, step). По умолчанию start равен 0, step единица, так что если передать только один аргумент, получится последовательность всех целых чисел от 0 до end-1.

ВАЖНО!!! end не включается в range.

len(X) - длина X, в данном случае число элементов X.

```
In [17]:
X = symbols('x:10')
display(X)
expr7 = 0
for i in range(len(X)):
    expr7 += X[i]**i
display(expr7)
```

```
(x0, x1, x2, x3, x4, x5, x6, x7, x8, x9)
```

$$x_1 + x_2^2 + x_3^3 + x_4^4 + x_5^5 + x_6^6 + x_7^7 + x_8^8 + x_9^9 + 1$$

Пример 8.

Создадим символы x10,...x21. Обратите внимание, что послений индекс не учитывается, так что пишем x10:22, а не x10:21.

Для красивого вывода используем display, для поэлементного вывода используем $*$ перед именем X10_21 (X10_21 - tuple, состоящий из отдельных символов).

Сравните результат с $*$ и без.

```
In [12]:
X10_21 = symbols('x10:22')
display(X10_21)
display(*X10_21)
```

```
(x10, x11, x12, x13, x14, x15, x16, x17, x18, x19, x20, x21)
```

x_{10}

x_{11}

x_{12}

x_{13}

x_{14}

x_{15}

x_{16}

x_{17}

x_{18}

x_{19}

x_{20}

x_{21}

Операции с символьными выражениями: expand,factor,collect

expand

используется для раскрытия скобок

factor

для разложения на множители

collect

для группировки по степеням переменной

Пример 9.

Создадим выражение $(x-3)^2$ и раскроем скобки. К полученному выражению применим factor. Выражение $x + 5xy - x^2y^3 - x^2$ сгруппируем по степеням x.

```
In [13]:
expr9 = (x - 3)**2
expr9_1 = expand(expr9)
display(expr9_1)
expr9_2 = factor(expr9_1)
display(expr9_2)
expr9_3 = x - 5*x*y - x**2*y**3 - x**2
collect(expr9_3, x)
```

$$x^2 - 6x + 9$$

$$(x-3)^2$$

Out[13]:

$$x^2\left(-y^3 - 1\right) + x(1 - 5y)$$

```
In [14]:
factor(x**2 + 2*x*y)
```

Out[14]:

$$x(x + 2y)$$

Списки и кортежи

Нам понадобятся два итерируемых типа

списки list

изменяемый тип

кортежи tuple

неизменяемый тип

Пример 10

создадим кортеж символов:

```
In [11]:
tup = symbols('c:f')
display(tup)
display(tup[1])
```

```
(c, d, e, f)
```

d

Пример 11

Создадим список символов и заменим в нем второй символ на символ x

```
In [16]:
listt = symbols(['m', 'p', 'q', 'r'])
display(listt)
listt[2] = x
listt
```

```
[m, p, q, r]
```

```
Out[16]: [m, p, x, r]
```

Если попытаемся заменить элемент кортежа, получим сообщение об ошибке

```
In [13]:
tup[1] = 1
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
<ipython-input-13-d516279dfb61> in <module>
----> 1 tup[1] = 1

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

Вложенные списки

Создадим аналог двумерного массива:

```
In [18]:
list2d = [[1, 2], [3, 4]]
display(list2d)
list2d[1][0] = 5
list2d
```

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

```
Out[18]: [[1, 2], [5, 4]]
```

Матрицы

В sympy есть объекты Matrix

https://docs.sympy.org/latest/tutorial/matrices.html?highlight=matrix

К матрицам можно применять метод det().

У матриц есть свойство shape.

Матрицу можно построить, например, на основе списка:

```
In [19]:
M = Matrix(list2d)
display(M)
display(M.det())
M.shape
```

$$\begin{bmatrix} 1 & 2 \\ 5 & 4 \end{bmatrix}$$

$$-6$$

```
Out[19]: (2, 2)
```

Создание матриц определенного вида

Можно создавать матрицы из нулей, единиц и единичные матрицы с помощью встренных функций

```
In [20]:
display(zeros(2), ones(3), eye(4))
```

$$\begin{bmatrix} 0 & 0 \\ 0 & 0 \end{bmatrix}$$

$$\begin{bmatrix} 1 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \end{bmatrix}$$

$$\begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Можно создать диагональную матрицу:

```
In [21]:
R = diag(1, 2, ones(3))
R
```

Out[21]:

$$\begin{bmatrix} 1 & 0 & 0 & 0 & 0 \\ 0 & 2 & 0 & 0 & 0 \\ 0 & 0 & 1 & 1 & 1 \\ 0 & 0 & 1 & 1 & 1 \\ 0 & 0 & 1 & 1 & 1 \end{bmatrix}$$

В матрице можно выделить строку или столбец:

```
In [22]:
R.row(1)
```

Out[22]:

$$\begin{bmatrix} 0 & 2 & 0 & 0 & 0 \end{bmatrix}$$

```
In [23]:
R.col(0)
```

Out[23]:

$$\begin{bmatrix} 1 \\ 0 \\ 0 \\ 0 \\ 0 \end{bmatrix}$$

Можно использовать нумерацию "с хвоста":

```
In [24]:
R.col(-4)
```

Out[24]:

$$\begin{bmatrix} 0 \\ 2 \\ 0 \\ 0 \\ 0 \end{bmatrix}$$

Можно удалить или добавить строку или столбец:

```
In [25]:
Q = diag(2, 3, 4, 5, 6)
display(Q)
Q.col_del(3)
Q
```

$$\begin{bmatrix} 2 & 0 & 0 & 0 & 0 \\ 0 & 3 & 0 & 0 & 0 \\ 0 & 0 & 4 & 0 & 0 \\ 0 & 0 & 0 & 5 & 0 \\ 0 & 0 & 0 & 0 & 6 \end{bmatrix}$$

Out[25]:

$$\begin{bmatrix} 2 & 0 & 0 & 0 \\ 0 & 3 & 0 & 0 \\ 0 & 0 & 4 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 6 \end{bmatrix}$$

```
In [26]:
G = ones(2)
display(G)
G.row_insert(1, zeros(2))
```

$$\begin{bmatrix} 1 & 1 \\ 1 & 1 \end{bmatrix}$$

Out[26]:

$$\begin{bmatrix} 1 & 1 \\ 0 & 0 \\ 0 & 0 \\ 1 & 1 \end{bmatrix}$$

Внимание!

col_insert и row_insert НЕ изменяют матрицу, к которой они применяются!!! col_insert и row_insert возвращают измененную матрицу в качестве результата.

```
In [27]:
D = diag(7, 8, 9, 0)
display(D)
T = D.col_insert(1,Matrix([3, 4, 5, 6]))
display(T, D)
```

$$\begin{bmatrix} 7 & 0 & 0 & 0 \\ 0 & 8 & 0 & 0 \\ 0 & 0 & 9 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix}$$

$$\begin{bmatrix} 7 & 3 & 0 & 0 & 0 \\ 0 & 4 & 8 & 0 & 0 \\ 0 & 5 & 0 & 9 & 0 \\ 0 & 6 & 0 & 0 & 0 \end{bmatrix}$$

$$\begin{bmatrix} 7 & 0 & 0 & 0 \\ 0 & 8 & 0 & 0 \\ 0 & 0 & 9 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix}$$

Пригодится

Числа π и e есть в Sympy:

```
In [28]:
display(pi, E)
```

π

e

Можем посмотреть на их приближенные значения:

```
In [29]:
round(pi, 3), round(E, 3)
```

```
Out[29]: (3.142, 2.718)
```