

0.1 Базовый синтаксис

1) `math.exp(x)` – e^x
`math.factorial(x)` – $x!$
`math.gamma(x)` – гамма-функция
`math.pi`, `math.e` – числа π и e

2) `scipy.special.binom(m,n)` – биномиальный коэффициент
или
`scipy.special.comb(m,n)`

Т.е. получается так
`import scipy.special`
`scipy.special.binom(10, 5)`
252.0
`scipy.special.comb(10, 5)`
252.0

получим одинаковые результаты, однако, если вам интересна разница, то можно посмотреть тут
<https://qa-help.ru/questions/kakaya-raznicza-mezhdu-scipyspecialbinom-i-scipymisccomb>
<https://stackoverflow.com/questions/26560726/python-binomial-coefficient>

3) `scipy.special.gamma` - Гамма функция
Хорошее описание в документации библиотеки `scipy`
<https://docs.scipy.org/doc/scipy/reference/generated/scipy.special.gamma.html>

4) Цикл `While`.
Выполняет тело цикла до тех пор, пока условие цикла истинно.

```
>i = 5  
>while i < 15:  
... print(i)  
... i = i + 2  
...  
5  
7  
9  
11  
13
```

5) Цикл `for`.
Этот цикл проходится по любому итерируемому объекту (например строке или списку), и во время каждого прохода выполняет тело цикла.

```
>for i in 'hello world':  
... print(i * 2, end="")  
...  
hheelllloo wwoorrlldd
```

6) Оператор `continue`
Оператор `continue` начинает следующий проход цикла, минуя оставшееся тело цикла (`for` или `while`)
`>for i in 'hello world':`
`... if i == 'o':`
`... continue`

```
... print(i * 2, end="")
...
hheellll wwrrlldd
```

7) Оператор break

Оператор break досрочно прерывает цикл.

```
>for i in 'hello world':
... if i == 'o':
... break
... print(i * 2, end="")
...
hheellll
```

8) Else.

Слово else, примененное в цикле for или while, проверяет, был ли произведен выход из цикла инструкцией break, или же “естественным” образом.

Блок инструкций внутри else выполнится только в том случае, если выход из цикла произошел без помощи break.

```
>for i in 'hello world':
... if i == 'a':
... break
... else:
... print('Буквы а в строке нет')
...
Буквы а в строке нет
```

9) Пример построения графика <https://jenyay.net/Matplotlib/Matplotlib>

```
import math
import pylab
from matplotlib import mlab
```

Будем рисовать график этой функции

```
def func (x):
if x == 0:
return 1.0
return math.sin (x) / x
```

Интервал изменения переменной по оси X

```
xmin = -20.0
xmax = 20.0
```

Шаг между точками

```
dx = 0.01
```

!!! Создадим список координат по оси X на отрезке [-xmin; xmax], включая концы

```
xlist = mlab.frange (xmin, xmax, dx)
```

Вычислим значение функции в заданных точках

```
ylist = [func (x) for x in xlist]
```

!!! Нарисуем одномерный график
pylab.plot (xlist, ylist)

!!! Покажем окно с нарисованным графиком
pylab.show()

Еще пример <http://profitraders.com/Math/Norm.html>

```
import math
def gaussian(x, mu, sigma):
    return math.exp(-0.5*((x-mu)/sigma)**2) / sigma /
    math.sqrt(2*math.pi)

import numpy as np
import matplotlib.pyplot as plt

xs = np.arange(-8, 8, 0.05) # Сетка значений по оси абсцисс.
p1, = plt.plot(xs, [gaussian(x, 0, .5) for x in xs], label='σ = 0.5')
p2, = plt.plot(xs, [gaussian(x, 0, 1) for x in xs], label='σ = 1')
p3, = plt.plot(xs, [gaussian(x, 0, 2) for x in xs], label='σ = 2')
plt.legend()
plt.show()
```

0.1.1 Решение уравнений

Краткое описание -
<http://matandlife.blogspot.com/2018/06/python.html>

- Функция `linalg.solve()` -
<https://docs.scipy.org/doc/numpy/reference/generated/numpy.linalg.solve.html>
- Функция `linalg.lstsq()` -
<https://docs.scipy.org/doc/numpy/reference/generated/numpy.linalg.lstsq.html>
- Функция `roots()` -
<https://docs.scipy.org/doc/numpy/reference/generated/numpy.roots.html>
- Библиотека `Optimize` -
<https://docs.scipy.org/doc/scipy/reference/tutorial/optimize.html>

0.1.2 Символьные вычисления

Описание пакета для символьных вычислений SymPy.

<http://inp.nsk.su/~grozin/python/python7.html>

Pdf версия - <https://yadi.sk/d/Tk-QnHWjtmEBog>

Из файла можно почерпнуть следующую информацию:

- Объекты типа `Symbol`
- Многочлены и рациональные функции.

- Элементарные функции.
- Структура выражений.
- Решение уравнений.
- Ряды.
- Производные.
- Интегралы.
- Суммирование рядов.
- Пределы.
- Дифференциальные уравнения.
- Линейная алгебра.
- Собственные значения и векторы.
- Жорданова нормальная форма.
- Графики

0.1.3 Работа с файлами в Python

- <https://pythonworld.ru/typy-dannyx-v-python/fajly-rabota-s-fajlami.html>
- <https://python-scripts.com/work-with-files-python>

0.2 Статистическая практика

0.2.1 Генерирование случайных величин

Нам понадобится генерировать случайные величины из различных распределений.

Для этого есть два наиболее простых варианта:

- Напрямую с помощью специальных функций. В R для этого пригодятся функции с буквой r: `rgnorm`, `runif`, `rgcauchy` и так далее. Более полный список вы можете получить с помощью команды `help.search("distribution package="stats")`. В Python можно либо использовать библиотеку `random`: `import random as rnd` загрузит ее в память под именем `rnd`, а внутри нее есть различные генераторы: `rnd.gauss`, `rnd.random`, `rnd.exprovariate`, либо использовать `scipy.stats`, где выбирая распределение можно применять функцию `rvs`, скажем `stats.norm.rvs(5,1,2)` создаст выборку из 5 величин $\mathcal{N}(1, 2)$.
- С помощью метода обратной функции. Если вы знаете функцию распределения F величины X , возьмете к ней обратную F^{-1} и сгенерируете равномерную $R \sim R[0, 1]$, то $F^{-1}(R)$ будет иметь функцию распределения F . Кстати, обратные к функциям распределения в R задаются буквой q: `qnorm`, `qunif`, а в Python все в том же `scipy.stats` можно использовать метод `ppf` вместо `rvs`.
- В Python За генерацию случайных величин отвечает библиотека `random`: `import random as rnd` Внутри нее есть различные генераторы: `rnd.gauss`, `rnd.random`, `rnd.exprovariate` и так далее. Отметим также `rnd.sample(array,number)`, выбирающее `number` случайных элементов из набора `sample`.

- Почитать про генерацию случайных чисел можно здесь <https://ps.readthedocs.io/ru/latest/random.html>

Для того, чтобы посмотреть что получилось, можно использовать, например, гистограммы. Простое описание нескольких базовых методов визуализации в Python можно найти по ссылке <https://clck.ru/JiqCW>.

Для построения графиков в Python можно использовать matplotlib.pyplot: `import matplotlib.pyplot as plt`
Гистограмма массива array с разбиением на columns колонок — `plt.hist(array,colums)`
Двумерный точечный график (scatterplot)x,y — `plt.plot(x,y)`

Функция Polygon

Можно прочитать подобно здесь https://matplotlib.org/3.1.1/api/_as_gen/matplotlib.patches.Polygon.html

Краткое описание

Для отображения многоугольника в Python используется процедура `obj = Polygon(Point(x1, y1), Point(x2, y2), ..., Point(xn, yn))` x1, y1, x2, y2, ..., xn, yn — координаты вершин многоугольника.

1) Пример программы на Python, которая отображает пятиугольник в графическом окне. `from graphics import *`
`win = GraphWin('Окно для графики', 400, 400)`
`obj = Polygon(Point(10, 10), Point(300, 50), Point(200, 300), Point(150, 150), Point(70, 70))`
`obj.draw(win)`
`win.getMouse()`
`win.close()`

2) Пример программы на Python, которая отображает фигуру с синей границей и заливкой в графическом окне.
`from graphics import *`
`win = GraphWin('Окно для графики', 310, 310)`
`obj = Polygon(Point(10, 10), Point(300, 50), Point(200, 300), Point(150, 150), Point(70, 70))`
`obj.setOutline('blue')`
`obj.setWidth(5)`
`obj.setFill('cyan')`
`obj.draw(win)`
`win.getMouse()`
`win.close()`

Дополнительно. Для заинтересованных в ознакомлении с языком R по ссылке https://r-analytics.blogspot.com/2011/11/r_06.html#.XUHN50gzaM8
- можно прочитать про гистограммы;
https://rpubs.com/AllaT/class5_old
в целом про визуализацию.

Пример визуализации и генерации случайных величин можно посмотреть здесь <https://habr.com/ru/post/331560/>

1. Сгенерировать тысячу величин а) экспоненциальных б) Коши в) нормальных каждым из методов
2. Построить для полученных распределений гистограммы

0.2.2 Графики

1. Построить точечный график (i, X_i) , где X_i – экспоненциальные с.в.
2. Построить график функции $y = 2x^2 - 3x^3 + 5x$
3. Построить два предыдущих графика на одном