Линейная алгебра на Python

Игорь Рязанцев

Лекция 04

2021г.

Что мы изучили

- Переменные
- Кортежи
- Списки (вставка, удаление, сортировка)
- Циклы **for**
- Оператор **if**
- Функции
- Классы (наследование)
- Импорт модулей
- Файлы
- Ввод-вывод

Переменные

```
# Объявление переменной

var = None

var = "world"
print("Hello, {0}!".format(var))

var = 10
print("{0}".format(var))

var = 3.14
print("{0}".format(var))
```

Переменные

```
# Приведение к другому типу переменной
# (int), (float), (str)
var = "3.14"

number = float(var)
pi_2 = 2 * number
print("{0}".format(pi_2))

var = 3.14

s_number = str(var)
print("{0}".format(s_number))
```

Переменные

```
# Область действия переменной (глобальные и локальные переменные)

# Верно
var = None
if var == None:
    var = 10
print("{0}".format(var))

# Неверно
if var == None:
    var = 10
print("{0}".format(var))
```

Кортежи tuple()

```
var = ("world", 2021, )
print("Hello, {0} {1}!".format(var[0], var[1]))
```

Списки list()

```
days = ['Monday', 'Tuesday', 'Wednesday',
        'Thursday', 'Friday', 'Saturday',
        'Sunday', ]

print("Today is {0}".format(days[0]))
```

Списки list()

 # append() – добавить элемент в конец

```
digits = [1, 2, 3, 4,]
digits.append(5)
for value in digits:
    print(value)
```

 # insert(позиция, элемент) – добавить элемент в указанную позицию
 # remove(элемент) – удаление элемента по значению
 # pop(позиция) – удаление элемента по индексу[1]
 # sort() – сортировка элементов списка
 # sort(reverse=True) – сортировка в обратном порядке

[1] Значение индекса в списке начинается с 0

Цикл for

```
days = ['Monday', 'Tuesday', 'Wednesday',
        'Thursday', 'Friday', 'Saturday',
        'Sunday', ]

for day in days:
    print("Today␣is", end='␣')
    print("{0}".format(day))

# Функция range() упрощает построение числовых последовательностей.

for value in range(1,5):
    print(value)
```

Оператор if

В **if** центральное место занимает выражение, результатом которого является логическая истина **(True)** или логическая ложь **(False)**; это выражение называется *условием*.

 # Общая форма записи:

```
if condition_1:
    action
elif condition_2:
    action
else:
    action
```

Оператор if

== или **is**	равно
>	больше
>=	больше или равно
<=	меньше или равно
<	меньше

```
power = 5
if power == 5:
    print('power = 5')
elif power < 5:
    print('power < 5')
else:
    print('power > 5')
```

Функции

Функция – именованный блок кода, предназначенный для решения одной конкретной задачи.

```
def greet_user(username, ages=21):
    print('{0} is {1}!'.format(username, ages))

greet_user(username='Helen')
```

Результат:
Helen is 21!

Функции. Аннотация типов.

```
# Использование аннотаций в функциях

def greet(username: str, ages: int = 21) -> None:
    print('{0} is {1}!'.format(username, ages))

greet(username='Helen')
```

Результат:
Helen is 21!

Классы

Класс определяет общее поведение для целой категории объектов. **Класс – это спецификация**.

```
class LED:
    def __init__(self, power, flux):
        self.power = power
        self.flux = flux
        self.switch_on = False
    ...
    def switch(self):
        ...

led = LED(60, 2000)
led.switch()
```

Наследование класса

```
class LED:
    def __init__(self, power, flux):
    ...
class LED_ext(LED):
    def __init__(self, power, flux, type_lid):
        super().__init__(power, flux)
        self.type_lid = type_lid

    def print_type_lid(self):
        print('type_lid={0}'.format(self.type_lid))

led = LED_ext(60, 2000, 1)
led.print_type_lid()
```

Импорт модулей

Вариант 1

```
import math

number = 5
print(math.factorial(number))
```

Результат:
120

Импорт модулей

Вариант 2

```
from math import factorial

number = 5
print(factorial(number))
```

Результат:
120

Импорт модулей

Вариант 3

```
from math import factorial as my_factorial

number = 5
print(my_factorial(number))
```

Результат:
120

Файлы (чтение)

 # Чтение файла

```
file = open('file.txt', 'r')
lines = file.readlines()
for line in lines:
    print(line, end='')
file.close()
```

Результат:
line 1
line 2
line 3

Файлы (запись)

```
# Запись в новый файл
# Присоединение данных к файлу (режим «a»)

file = open('new_file.txt', 'w')
file.write('line␣1')
file.close()
```

Результат (в файле):

```
line 1
```

Ввод данных с клавиатуры

```
# Ожидает ввод целого числа
a = int(input())

# Ожидает ввод вещественного числа
a = float(input())

# Ожидает ввод сроки
a = str(input())
```

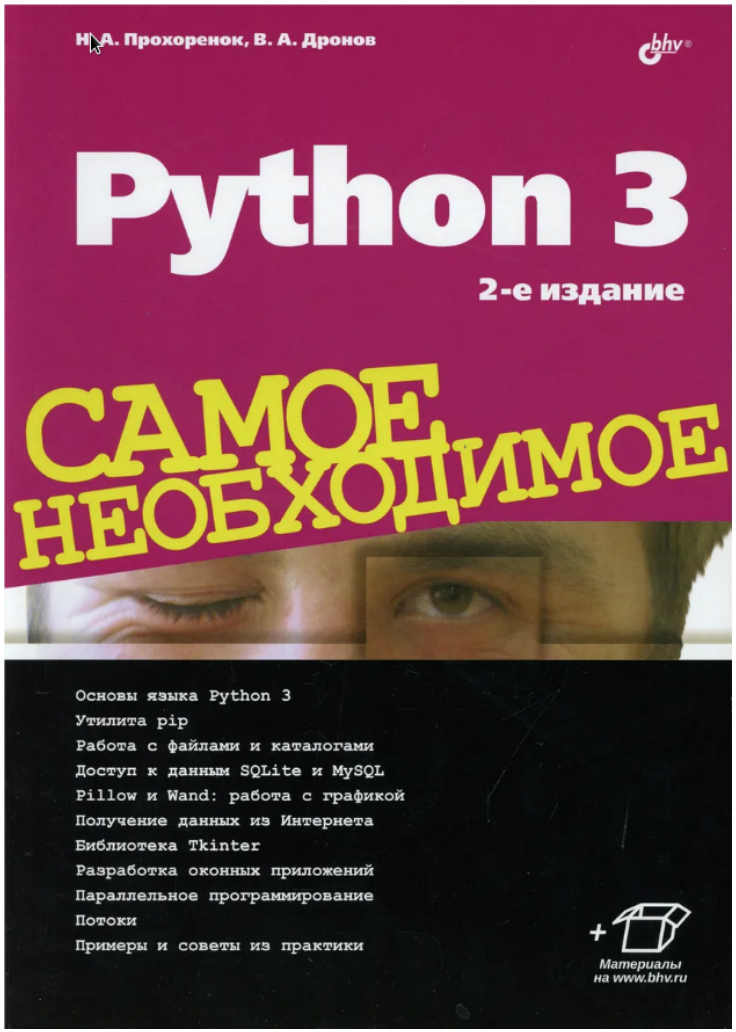
Что мы еще не изучили

- Переменные (правила именования, типы данных, удаление переменных)
- Специальные символы (табуляция, возврат каретки, перевод строки, и т.д.)
- Операторы (деление с округлением, остаток от деления, возведение в степень, двоичные операторы (И, ИЛИ))
- Структурное сопоставление шаблонов (match...case)
- Исключения
- Строки (операции над строками)
- Регулярные выражения

Что мы еще не изучили

- Работа со временем и датой
- Множества, словари
- Итераторы, контейнеры, перечисления
- Циклы (while, continue, break)
- Функции (декораторы, рекурсия, вложенные функции)
- Классы (деструктор, абстрактные методы, статические методы, свойства класса, множественное наследование и декораторы классов)
- Пакеты
- Файлы (работа с каталогами)

Литература



Н.А. Прохоренок
Python 3. Самое необходимое.
2-е изд., перераб.и доп [2021]

Оглавление

Оглавление

Библиотека matplotlib

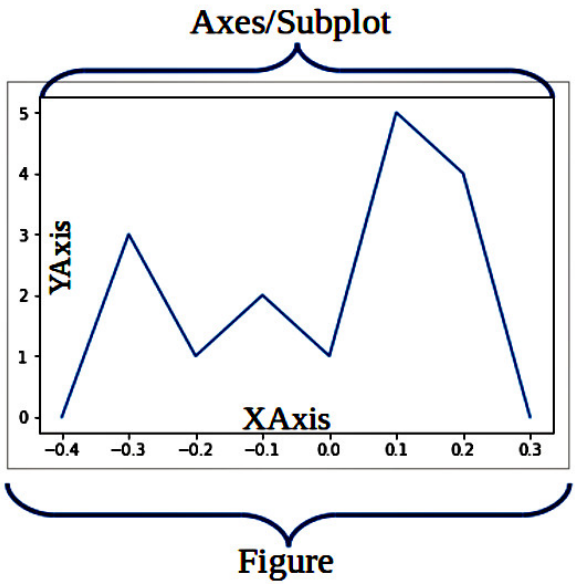
Открыть

Библиотека matplotlib

 # Установка библиотеки matplotlib

pip install matplotlib

Основные компоненты matplotlib

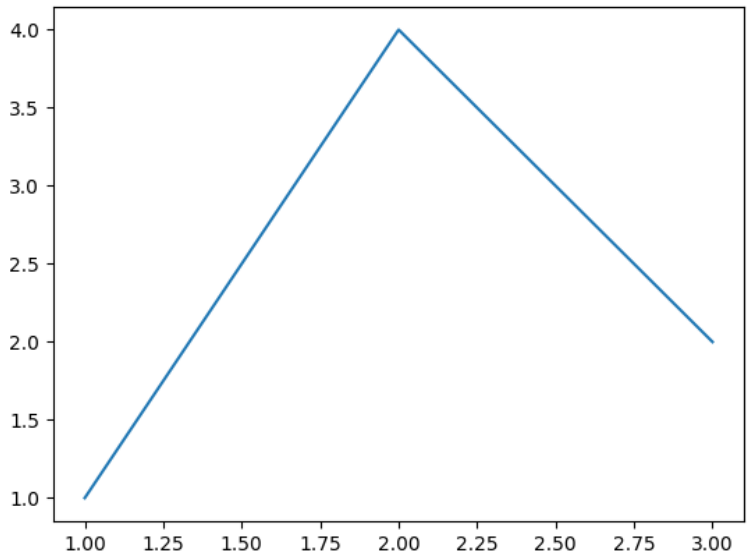


- **Figure** – контейнер самого верхнего уровня, которых может содержать несколько контейнеров *Axes*.
- **Axes** – область на которой отражаются графики, а так же все вспомогательные атрибуты (линии сетки, метки, указатели и т.д.).
- Каждая область **Axes** содержит **XAxis** и **YAxis**, которые в свою очередь содержат: деления, метки и прочие вспомогательные атрибуты.

Вывод графика

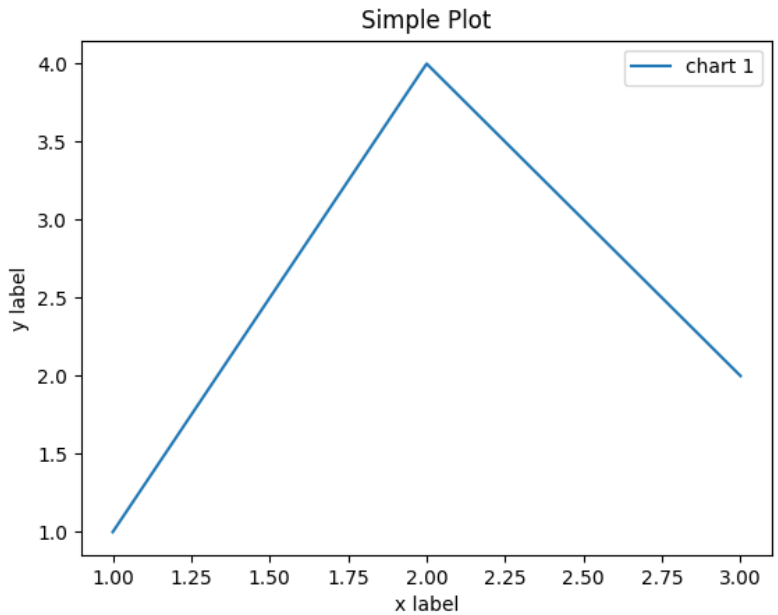
```
import matplotlib.pyplot as plt
import numpy as np

(fig, ax) = plt.subplots()
ax.plot([1, 2, 3, ], [1, 4, 2, ], label='chart 1')
plt.show()
```



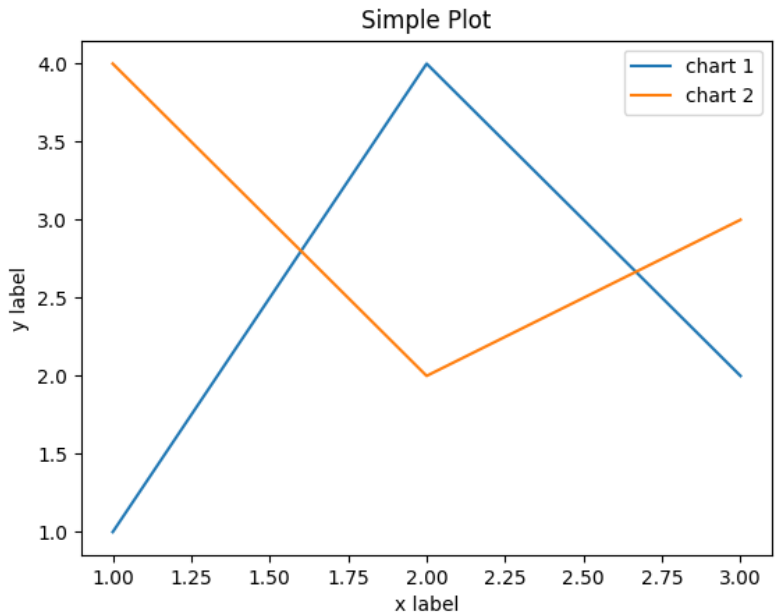
Заголовок, подписи, легенда

```
...
ax.set_xlabel('x label')
ax.set_ylabel('y label')
ax.set_title("Simple Plot")
ax.legend()
...
```



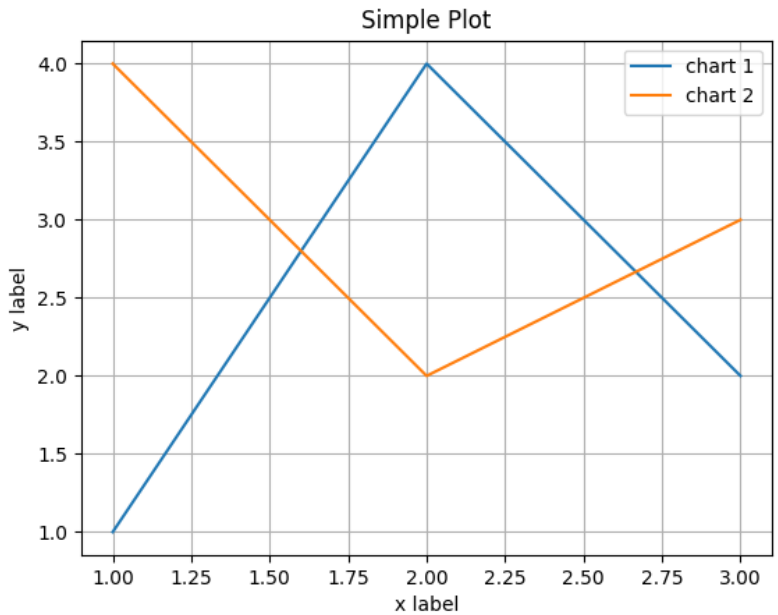
Два и более графиков

```
...
ax.plot([1, 2, 3, ], [1, 4, 2, ], label='chart 1')
ax.plot([1, 2, 3, ], [4, 2, 3, ], label='chart 2')
...
...
...
```



Сетка

```
...
(fig, ax) = plt.subplots()
ax.grid(True)
ax.plot([1, 2, 3, ], [1, 4, 2, ], label='chart 1')
ax.plot([1, 2, 3, ], [4, 2, 3, ], label='chart 2')
...
```



Оглавление

Библиотека numpy

Установка библиотек

pip install numpy

pip install scipy

NumPy – библиотека поддержки многомерных массивов (включая матрицы) и высокоуровневых математических функций, предназначенных для работы с многомерными массивами. NumPy можно рассматривать как свободную альтернативу MATLAB.

SciPy – библиотека для выполнения научных и инженерных расчётов.

График функции y = sin(x)

```
import matplotlib.pyplot as plt
import math, numpy

x_ax = []
y_ax = []

for x in numpy.arange(0, 2 * math.pi, 0.1):
    x_ax .append(x)
    y_ax.append(math.sin(x))

(fig, ax) = plt.subplots()
ax.grid(True)
ax.plot(x_ax, y_ax, label='sin')
plt.show()
```

График функции y = sin(x)

```
import matplotlib.pyplot as plt
import numpy

(fig, ax) = plt.subplots()
x = numpy.linspace(0, 2 * numpy.pi, 100)
y = numpy.sin(x)
ax.plot(x, y, label='chart 1')
plt.show()
```

Матрицы

Матрицей в математике называют объект, записываемый в виде прямоугольной таблицы, элементами которой являются числа (могут быть как действительные, так и комплексные).

$$A_{3\times 3} = \begin{pmatrix} a_{11} & a_{12} & a_{13} \\ a_{21} & a_{22} & a_{23} \\ a_{31} & a_{32} & a_{33} \end{pmatrix}$$

- Матрица состоит из i-строк и j-столбцов;
- Каждый ее элемент имеет соответствующее позиционное обозначение: a_{ij} находится на i-ой строке и j-м столбце;
- Главная диагональ – элементы, у которых совпадают номера строк и столбцов.

Вектор

Вектором называется матрица, у которой есть только один столбец или одна строка.

Вектор-строка имеет следующую математическую запись

$$v = \begin{pmatrix} a_{11} & a_{12} \end{pmatrix}$$

в Python можно задать следующим образом

```
import numpy
Vh = numpy.matrix([1, 2])
print(Vh)
```

Результат:

```
[[1 2]]
```

Вектор

Вектор-столбец имеет следующую математическую запись

$$v = \begin{pmatrix} a_{11} \\ a_{21} \end{pmatrix}$$

в Python можно задать следующим образом

```
import numpy
Vv = numpy.matrix([[1], [2]])
print(Vv)
```

Результат:

```
[[1]
 [2]]
```

Квадратная матрица

Матрица называется **квадратной**, если количество строк и столбцов совпадают.

$$v = \begin{pmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 9 \end{pmatrix}$$

```
import numpy
A = numpy.matrix([[1, 2, 3], [4, 5, 6],
                  [7, 8, 9]])
print(A)
```

Результат:

```
[[1 2 3]
 [4 5 6]
 [7 8 9]]
```

Диагональная матрица

Диагональная матрица – матрица, у которой все элементы, кроме тех, что расположены на главной диагонали, равны нулю.

$$v = \begin{pmatrix} 1 & 0 & 0 \\ 0 & 5 & 0 \\ 0 & 0 & 9 \end{pmatrix}$$

```
import numpy
A = numpy.matrix([[1, 0, 0], [0, 5, 0],
                  [0, 0, 9]])
print(A)
```

Результат:

```
[[1 0 0]
 [0 5 0]
 [0 0 9]]
```

Единичная матрица

Единичной матрицей называют такую квадратную матрицу, у которой элементы главной диагонали равны единицы, а все остальные нулю.

$$v = \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix}$$

```
import numpy
A = numpy.matrix([[1, 0, 0], [0, 1, 0],
                  [0, 0, 1]])
#A = numpy.identity(3)
print(A)
```

Результат:

```
[[1 0 0]
 [0 1 0]
 [0 0 1]]
```

Сложение матриц A + B

$$A = \begin{pmatrix} a_{11} & a_{12} & a_{13} \\ a_{21} & a_{22} & a_{23} \\ a_{31} & a_{32} & a_{33} \end{pmatrix}$$

$$B = \begin{pmatrix} b_{11} & b_{12} & b_{13} \\ b_{21} & b_{22} & b_{23} \\ b_{31} & b_{32} & b_{33} \end{pmatrix}$$

Сложение матриц A + B

$$A + B = \begin{pmatrix} a_{11} + b_{11} & a_{12} + b_{12} & a_{11} + b_{13} \\ a_{21} + b_{21} & a_{22} + b_{22} & a_{23} + b_{23} \\ a_{31} + b_{31} & a_{32} + b_{32} & a_{33} + b_{33} \end{pmatrix}$$

Сложение матриц A + B

```
import numpy
A = numpy.matrix([[1, 2, 3],
                  [4, 5, 6],
                  [7, 8, 9]])
B = numpy.matrix([[1, 2, 3],
                  [4, 5, 6],
                  [7, 8, 9]])
C = A + B
print(C)
```

Результат:

```
[[ 2  4  6]
 [ 8 10 12]
 [14 16 18]]
```

Вычитание матриц A - B

$$A = \begin{pmatrix} a_{11} & a_{12} & a_{13} \\ a_{21} & a_{22} & a_{23} \\ a_{31} & a_{32} & a_{33} \end{pmatrix}$$

$$B = \begin{pmatrix} b_{11} & b_{12} & b_{13} \\ b_{21} & b_{22} & b_{23} \\ b_{31} & b_{32} & b_{33} \end{pmatrix}$$

Вычитание матриц A - B

$$A - B = \begin{pmatrix} a_{11} - b_{11} & a_{12} - b_{12} & a_{11} - b_{13} \\ a_{21} - b_{21} & a_{22} - b_{22} & a_{23} - b_{23} \\ a_{31} - b_{31} & a_{32} - b_{32} & a_{33} - b_{33} \end{pmatrix}$$

Вычитание матриц A - B

```
import numpy
A = numpy.matrix([[1, 2, 3],
                  [4, 5, 6],
                  [7, 8, 9]])
B = numpy.matrix([[1, 2, 3],
                  [4, 5, 6],
                  [7, 8, 9]])
C = A - B
print(C)
```

Результат:

```
[[0 0 0]
 [0 0 0]
 [0 0 0]]
```

Скалярное произведение A × n

В скалярном произведении постоянное значение умножается на каждый элемент матрицы.

$$A = \begin{pmatrix} a_{11} & a_{12} & a_{13} \\ a_{21} & a_{22} & a_{23} \\ a_{31} & a_{32} & a_{33} \end{pmatrix}$$

Умножения матрицы A на число n

$$A \times n = \begin{pmatrix} a_{11} \times n & a_{12} \times n & a_{11} \times n \\ a_{21} \times n & a_{22} \times n & a_{23} \times n \\ a_{31} \times n & a_{32} \times n & a_{33} \times n \end{pmatrix}$$

Скалярное произведение A × n

Оператор $*$ используется для умножения скалярного значения на элементы входной матрицы.

```
import numpy
A = numpy.matrix([[1, 2, 3],
                  [4, 5, 6],
                  [7, 8, 9]])

C = A * 5
print(C)
```

Результат:

```
[[ 5 10 15]
 [20 25 30]
 [35 40 45]]
```

Скалярное деление A / n

В скалярном деление каждый элемент матрицы делиться на постоянное значение.

$$A = \begin{pmatrix} a_{11} & a_{12} & a_{13} \\ a_{21} & a_{22} & a_{23} \\ a_{31} & a_{32} & a_{33} \end{pmatrix}$$

Деление матрицы A на число n

$$A/n = \begin{pmatrix} a_{11}/n & a_{12}/n & a_{11}/n \\ a_{21}/n & a_{22}/n & a_{23}/n \\ a_{31}/n & a_{32}/n & a_{33}/n \end{pmatrix}$$

Скалярное деление A / n

Оператор ‘/’ делит каждый элемент матрицы на скалярное / постоянное значение.

```
import numpy
A = numpy.matrix([[1, 2, 3],
                  [4, 5, 6],
                  [7, 8, 9]])

C = A / 2
print(C)
```

Результат:

```
[[0.5  1.   1.5]
 [2.   2.5  3. ]
 [3.5  4.   4.5]]
```

Произведение матриц A × B

$$A = \begin{pmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \end{pmatrix}$$

$$B = \begin{pmatrix} b_{11} & b_{12} \\ b_{21} & b_{22} \end{pmatrix}$$

Произведение матриц A × B

$$A \times B = \begin{pmatrix} a_{11} \times b_{11} + a_{12} \times b_{21} & a_{11} \times b_{12} + a_{12} \times b_{22} \\ a_{21} \times b_{11} + a_{22} \times b_{21} & a_{21} \times b_{12} + a_{22} \times b_{22} \end{pmatrix}$$

Произведение матриц A × B

```
import numpy
A = numpy.matrix([[1,  2],
                  [4,  5]])

B = numpy.matrix([[1,  2],
                  [4,  5]])

C = numpy.dot(A, B)
print(C)
```

Результат:

```
[[ 9  12]
 [24  33]]
```

Транспонирование матрицы

Транспонирование матрицы включает в себя переворачивание матрицы по соответствующим диагоналям, т. е. меняются местами строки и столбцы входной матрицы.

$$A = \begin{pmatrix} a_{11} & a_{12} & a_{13} \\ a_{21} & a_{22} & a_{23} \end{pmatrix}$$

После выполнения операции транспонирования Matrix.T

$$A = A.T = \begin{pmatrix} a_{11} & a_{21} \\ a_{12} & a_{22} \\ a_{13} & a_{23} \end{pmatrix}$$

Транспонирование матрицы

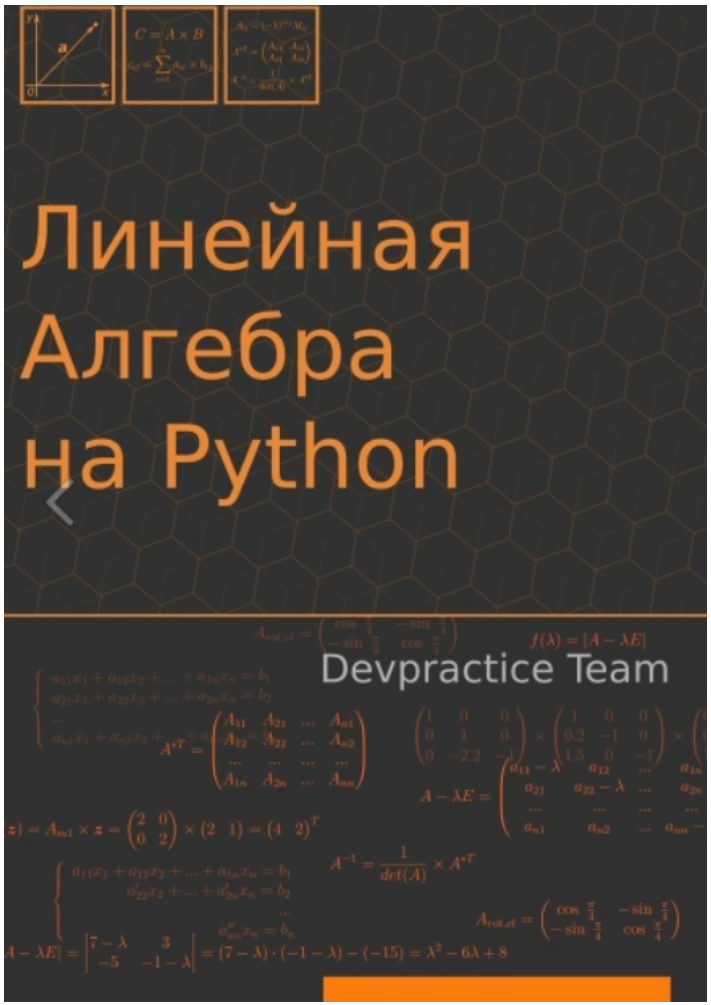
```
import numpy
A = numpy.matrix([[1, 2, 3],
                  [4, 5, 6]])
print(A, end='\n\n')
C = A.T    # C = A.transpose()
print(C)
```

Результат:

```
[[1 2 3]
 [4 5 6]]

[[1 4]
 [2 5]
 [3 6]]
```

Литература



Devpractice Team.
Линейная алгебра на Python [2019]

Литература

[1] Презентация [Лекции 01-04]

https://github.com/IRyazantsev/mpei_python_mini-course_2021/tree/main/bin

[2] Python 3. Самое необходимое | Дронов В.А., Прохоренок Н.А.

[3] Изучаем Python. Том 1, 2 | Лутц Марк

[4] Python 3 и PyQt 5. Разработка приложений | Прохоренок Н.А., Дронов В.А.

[5] Django 3.0. Практика создания веб-сайтов на Python | Дронов В. А.

[6] Разработка веб-приложений с использованием Flask | Гринберг Мигель

Вопросы