# Что сегодня сделаем

Рекомендательный алгоритм сопутствующих товаров для пользователя

# Numpy и матричная алгебра

In [1]:

```
import numpy as np
```

In [2]:

```
x = np.array([1, 2, 3])
y = np.array([4, 5, 6])
```

In [ ]:

```
x.
```

In [4]:

```
# у переменной x тип, отличный от list
type(x)
```

Out[4]:

```
numpy.ndarray
```

In [3]:

```
# дополнительные методы
x.mean()
```

Out[3]:

```
2.0
```

## Поэлементные операции

In [5]:

```
5 *5
```

Out[5]:

```
25
```

In [6]:

```
5 + 5
```

Out[6]:

```
10
```

In [7]:

```
'5' + '5'
```

Out[7]:

```
'55'
```

In [8]:

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

Out[8]:

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

In [9]:

```
x, y
```

Out[9]:

```
(array([1, 2, 3]), array([4, 5, 6]))
```

In [10]:

```
x + y
```

Out[10]:

```
array([5, 7, 9])
```

In [11]:

```
x - y
```

Out[11]:

```
array([-3, -3, -3])
```

In [12]:

```
x * y
```

Out[12]:

```
array([ 4, 10, 18])
```

In [13]:

```
x / y
```

Out[13]:

```
array([0.25, 0.4 , 0.5 ])
```

In [15]:

```
x, y
```

Out[15]:

```
(array([1, 2, 3]), array([4, 5, 6]))
```

In [14]:

```
# элементы массива x возводятся в соответствующие степени элементов массива y
x ** y
```

Out[14]:

```
array([  1,  32, 729])
```

In [ ]:

```
10 = 3*3 + 1
```

In [16]:

```
# остаток от деления
# обратите внимание, что для удобства данного примера x и y идут в другом порядке

y % x
```

Out[16]:

```
array([0, 1, 0])
```

## Автоматическое наполнение массивов

In [17]:

```
# аналог range

np.arange(0, 10)
```

Out[17]:

```
array([0, 1, 2, 3, 4, 5, 6, 7, 8, 9])
```

In [23]:

```
list(range(10))
```

Out[23]:

```
[0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
```

In [21]:

```
for i in range(10**20):
    print(i)
    break
```

```
0
```

In [24]:

```
# создать массив из 10 чисел, заполненных единицами
np.full(10, 1)
```

Out[24]:

```
array([1, 1, 1, 1, 1, 1, 1, 1, 1, 1])
```

In [25]:

```
# создать матрицу 2x5 из нулей
np.full([2, 5], 0)
```

Out[25]:

```
array([[0, 0, 0, 0, 0],
       [0, 0, 0, 0, 0]])
```

In [26]:

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

Out[26]:

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

In [27]:

```
# объединить массивы в один
a = np.arange(10)
b = np.arange(10, 20)

# обратите внимание на двойные скобки
np.concatenate((a, b))
```

Out[27]:

```
array([ 0,  1,  2,  3,  4,  5,  6,  7,  8,  9, 10, 11, 12, 13, 14, 15, 16,
       17, 18, 19])
```

In [28]:

```
# перемешать элементы массива
import random

nums = np.arange(10)
random.shuffle(nums)

nums
```

Out[28]:

```
array([9, 6, 1, 4, 2, 8, 3, 0, 7, 5])
```

## Изменение размерности

In [29]:

```
x = np.arange( 0, 10 )
x
```

Out[29]:

```
array([0, 1, 2, 3, 4, 5, 6, 7, 8, 9])
```

In [30]:

```
# 10 - количество строк

x.shape
```

Out[30]:

```
(10,)
```

In [31]:

```
# первый аргумент - количество строк
# второй - столбцов

x.reshape(5, 2)
```

Out[31]:

```
array([[0, 1],
       [2, 3],
       [4, 5],
       [6, 7],
       [8, 9]])
```

In [32]:

```
x.reshape(3, 3)
```

```
---------------------------------------------------------------------------
ValueError                                Traceback (most recent call last)
<ipython-input-32-44cd19047743> in <module>
----> 1 x.reshape(3, 3)

ValueError: cannot reshape array of size 10 into shape (3,3)
```

In [33]:

```
# транспонирование матриц

np.array(
    [
        [1, 2],
        [3, 4],
        [5, 6]
    ]
).T
```

Out[33]:

```
array([[1, 3, 5],
       [2, 4, 6]])
```

In [34]:

```
# склеивание набора списков
[1, 2, 3] + [4, 5, 6]
```

Out[34]:

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

In [35]:

```
# склеивание массива из списков

x = np.array( [ [1, 2, 3], [4, 5, 6] ] )
x.ravel()
```

Out[35]:

```
array([1, 2, 3, 4, 5, 6])
```

In [36]:

```
# можно и так

x.reshape(6)
```

Out[36]:

```
array([1, 2, 3, 4, 5, 6])
```

In [37]:

```
# результат разный, если добавить 1 в качестве количества строк

x.reshape(1, 6)
```

Out[37]:

```
array([[1, 2, 3, 4, 5, 6]])
```

## Создание матриц

In [38]:

```
# нулевой вектор заданной размерности

np.zeros(10)
```

Out[38]:

```
array([0., 0., 0., 0., 0., 0., 0., 0., 0., 0.])
```

In [39]:

```
# единичная матрица

np.eye(5)
```

Out[39]:

```
array([[1., 0., 0., 0., 0.],
       [0., 1., 0., 0., 0.],
       [0., 0., 1., 0., 0.],
       [0., 0., 0., 1., 0.],
       [0., 0., 0., 0., 1.]])
```

In [41]:

```
# более общий случай диагональной матрицы

np.diag(np.arange(2, 50, 15), k=1)
```

Out[41]:

```
array([[ 0,  2,  0,  0,  0],
       [ 0,  0, 17,  0,  0],
       [ 0,  0,  0, 32,  0],
       [ 0,  0,  0,  0, 47],
       [ 0,  0,  0,  0,  0]])
```

In [42]:

```
# матрица со случайными значениями

np.random.random(10)
```

Out[42]:

```
array([0.4695114 , 0.5167444 , 0.620242  , 0.26422004, 0.4644014 ,
       0.50061411, 0.91162335, 0.18816725, 0.87896904, 0.74962345])
```

In [44]:

```
# более универсальный вариант создания матриц

?np.linspace
```

In [43]:

```
np.linspace(5, 25, 30)
```

Out[43]:

```
array([ 5.        ,  5.68965517,  6.37931034,  7.06896552,  7.75862069,
        8.44827586,  9.13793103,  9.82758621, 10.51724138, 11.20689655,
       11.89655172, 12.5862069 , 13.27586207, 13.96551724, 14.65517241,
       15.34482759, 16.03448276, 16.72413793, 17.4137931 , 18.10344828,
       18.79310345, 19.48275862, 20.17241379, 20.86206897, 21.55172414,
       22.24137931, 22.93103448, 23.62068966, 24.31034483, 25.        ])
```

## Более сложные распределения

https://docs.scipy.org/doc/scipy-0.14.0/reference/generated/scipy.signal.gaussian.html (https://docs.scipy.org/doc/scipy-0.14.0/reference/generated/scipy.signal.gaussian.html)

# Скалярное произведение векторов

$$\vec{a} \cdot \vec{b} = |\vec{a}| \ |\vec{b}| \ cos(\vec{a}, \vec{b})$$

Пусть

$$\vec{a} = (a_1, a_2, a_3)$$
$$\vec{b} = (b_1, b_2, b_3)$$

Тогда скалярное произведение векторов равно

$$\vec{a} \cdot \vec{b} = a_1 b_1 + a_2 b_2 + a_3 b_3$$

In [45]:

```
a = np.array( [4, 3] )
b = np.array( [2, 1] )
```

In [47]:

```
a, b
```

Out[47]:

```
(array([4, 3]), array([2, 1]))
```

In [48]:

```
4*2 + 3*1
```

Out[48]:

```
11
```

Пример расчета скалярного произведения векторов

In [46]:

```
np.dot( a, b )
```

Out[46]:

```
11
```

Можно посчитать и таким образом

In [ ]:

```
# первый шаг

for pair in zip( a, b ):
    print( pair )
```

In [ ]:

```
# второй шаг

[ pair[0] * pair[1] for pair in zip( a, b ) ]
```

In [49]:

```
# итоговый результат

sum( [ pair[0] * pair[1] for pair in zip( a, b ) ] )
```

Out[49]:

```
11
```

# Косинусное сходство между векторами

$$cos(\vec{a}, \vec{b}) = \frac{\vec{a} \cdot \vec{b}}{|\vec{a}| \ |\vec{b}|}$$

In [50]:

```
import matplotlib.pyplot as plt
```

In [52]:

```
a, b
```

Out[52]:

```
(array([4, 3]), array([2, 1]))
```

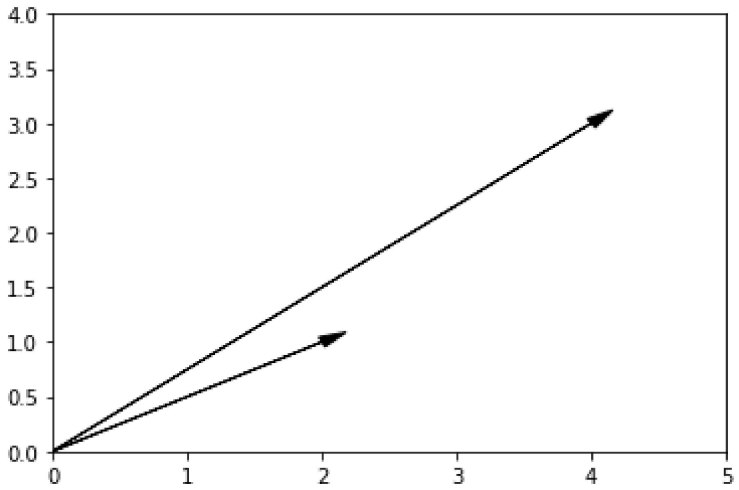
In [51]:

```
ax = plt.axes()

plt.xlim( [0, 5] )
plt.ylim( [0, 4] )

ax.arrow( 0, 0, a[0], a[1], head_width=0.1, head_length=0.2, fc='k', ec='k' )
ax.arrow( 0, 0, b[0], b[1], head_width=0.1, head_length=0.2, fc='k', ec='k' )

plt.show()
```




In [53]:

```
def cosine( a, b ):
    """
    Подсчет косинуса угла между векторами a, b по их координатам
    """

    # длины векторов
    aLength = np.linalg.norm( a )
    bLength = np.linalg.norm( b )

    return np.dot( a, b ) / ( aLength * bLength )
```

In [ ]:

```
# длины векторов можно было посчитать и так

aLength = np.sqrt( (a*a).sum() )
bLength = np.sqrt( (b*b).sum() )
```

In [54]:

```
cosine( a, b )
```

Out[54]:

```
0.9838699100999074
```

In [55]:

```
# угол между векторами в радианах

np.arccos( cosine( a, b ) )
```

Out[55]:

```
0.17985349979247847
```

In [56]:

```
# угол между векторами в градусах

np.arccos( cosine( a, b ) ) * 360 / 2 / np.pi
```

Out[56]:

```
10.304846468766044
```

## Задача 4 домашнего задания

Имеется матрица покупок в интернет-магазине. Столбец A - ID пользователя. Остальные столбцы - количество покупок категорий товаров этим пользователем:

In [57]:

```
from IPython.display import Image
Image("user_matrix.JPG")
```

Out[57]:

| User ID | Смартфон | Smart TV | Ноутбук | Шкаф | Холодильник | Посуда |
|---|---|---|---|---|---|---|
| 1 | 2 | 1 | 0 | 0 | 0 | 0 |
| 2 | 1 | 1 | 2 | 1 | 0 | 0 |
| 3 | 2 | 0 | 1 | 0 | 0 | 0 |
| 4 | 1 | 1 | 2 | 1 | 0 | 1 |
| 5 | 0 | 0 | 1 | 2 | 0 | 0 |
| 6 | 0 | 0 | 0 | 0 | 0 | 5 |
| 7 | 1 | 0 | 0 | 0 | 0 | 0 |
| 8 | 0 | 1 | 1 | 0 | 0 | 0 |
| 9 | 0 | 0 | 0 | 1 | 1 | 3 |
| 10 | 1 | 0 | 0 | 2 | 1 | 4 |

Матрица в виде numpy array

In [ ]:

```
users_stats = np.array(
    [
        [2, 1, 0, 0, 0, 0],
        [1, 1, 2, 1, 0, 0],
        [2, 0, 1, 0, 0, 0],
        [1, 1, 2, 1, 0, 1],
        [0, 0, 1, 2, 0, 0],
        [0, 0, 0, 0, 0, 5],
        [1, 0, 0, 0, 0, 0],
        [0, 1, 1, 0, 0, 0],
        [0, 0, 0, 1, 1, 3],
        [1, 0, 0, 2, 1, 4]
    ],
    np.int32
)
```

На сайт заходит очередной посетитель, о покупках которого известно следующее:

In [ ]:

```
next_user_stats = np.array([0, 1, 2, 0, 0, 0])
```

Найдите самого похожего пользователя. Т. е. посчитайте косинусное сходство между этим пользователем и всеми пользователями из массива user_stats

In [ ]:

In [ ]:

In [ ]:

# Перемножение матриц

**Определение**

Пусть даны две матрицы a и b размером l x m и m x n соответственно. l - количество строк, n - количество столбцов.

\begin{equation*} \LARGE a = \begin{bmatrix} a_{11} & a_{12} \dots a_{1m} \\ a_{21} & a_{22} \dots a_{2m} \\ \vdots & \vdots \dots \vdots \\ a_{l1} & a_{l2} \dots a_{lm} \end{bmatrix} \end{equation*}

\begin{equation*} \LARGE b = \begin{bmatrix} b_{11} & b_{12} \dots b_{1n} \\ b_{21} & b_{22} \dots b_{2n} \\ \vdots & \vdots \dots \vdots \\ b_{m1} & b_{m2} \dots b_{mn} \end{bmatrix} \end{equation*}

Тогда произведением матриц a и b будет матрица с размерностью l x n:

\begin{equation*} \LARGE c = \begin{bmatrix} c_{11} & c_{12} \dots c_{1n} \\ c_{21} & c_{22} \dots c_{2n} \\ \vdots & \vdots \dots \vdots \\ c_{l1} & c_{l2} \dots c_{ln} \end{bmatrix} \end{equation*}

\begin{equation*} \LARGE c_{ij} = \sum_{k=1}^m a_{ik} b_{kj} \end{equation*}

$$\underset{3\times4\text{ matrix}}{\begin{bmatrix} \cdot & \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot & \cdot \\ 1 & 2 & 3 & 4 \end{bmatrix}} \underset{4\times5\text{ matrix}}{\begin{bmatrix} \cdot & \cdot & \cdot & a & \cdot \\ \cdot & \cdot & \cdot & b & \cdot \\ \cdot & \cdot & \cdot & c & \cdot \\ \cdot & \cdot & \cdot & d & \cdot \end{bmatrix}} = \underset{3\times5\text{ matrix}}{\begin{bmatrix} \cdot & \cdot & \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot & x_{3,4} & \cdot \end{bmatrix}}$$

In [58]:

```
a = np.array(
    [
        [1, 2],
        [3, 4]
    ]
)
```

In [59]:

```
b = np.array(
    [
        [5, 6],
        [7, 8]
    ]
)
```

In [60]:

```
c = np.dot( a, b )
c
```

Out[60]:

```
array([[19, 22],
       [43, 50]])
```

В numpy есть специальный тип matrix, который отличается от ndarray

In [61]:

```
a * b
```

Out[61]:

```
array([[ 5, 12],
       [21, 32]])
```

In [62]:

```
aMatrix = np.matrix( [ [1, 2], [3, 4] ] )
bMatrix = np.matrix( [ [5, 6], [7, 8] ] )
```

In [63]:

```
type(aMatrix)
```

Out[63]:

```
numpy.matrix
```

In [64]:

```
aMatrix * bMatrix
```

Out[64]:

```
matrix([[19, 22],
        [43, 50]])
```

In [65]:

```
type( aMatrix ), type( a )
```

Out[65]:

```
(numpy.matrix, numpy.ndarray)
```

In [ ]:

```
np.mat( a ) * np.mat( b )
```

# Линейные уравнения

Дана система линейных уравнений

\begin{equation*} \LARGE x + 3*y = 9 \\ \LARGE 2*x - 4*y = 8 \end{equation*}

In [66]:

```
# коэффициенты при переменных в левой части уравнения

a = np.array( [ [1, 3], [2, -4] ] )
```

In [67]:

```
# значения в правой части уравнения

b = np.array( [9, 8] )
```

In [68]:

```
# решение

from numpy import linalg
```

In [69]:

```
linalg.solve(a, b)
```

Out[69]:

```
array([6., 1.])
```

In [70]:

```
# проверка верности

np.allclose( np.dot(a, linalg.solve(a, b)), b )
```

Out[70]:

```
True
```