

Министерство науки и высшего образования Российской Федерации
Федеральное государственное автономное образовательное учреждение
высшего образования
«СЕВЕРО-КАВКАЗСКИЙ ФЕДЕРАЛЬНЫЙ УНИВЕРСИТЕТ»

Институт цифрового развития
Кафедра инфокоммуникаций

«Основы работы с библиотекой NumPy»

ОТЧЕТ
по лабораторной работе №2
дисциплины
«Технологии распознавания образов»

Выполнил:
Мизин Глеб Егорович
2 курс, группа ПИЖ-б-о-21-1,
011.03.04 «Программная инженерия»,
направленность (профиль) «Разработка
и сопровождение программного
обеспечения», очная форма обучения

(подпись)

Проверил:

(подпись)

Отчет защищен с оценкой _____ Дата защиты _____

Ставрополь, 2022 г.

Проработка примеров из методических указаний:

```
In [1]: import numpy as np
        # Теперь создадим матрицу, с которой будем работать.
        m = np.matrix('1 2 3 4; 5 6 7 8; 9 1 5 7')
        print(m)
```

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

```
In [2]: m[1, 0]
```

```
Out[2]: 5
```

```
In [3]: m[1, :]
```

```
Out[3]: matrix([[5, 6, 7, 8]])
```

```
In [4]: m[:, 2]
```

```
Out[4]: matrix([[3],
               [7],
               [5]])
```

```
In [5]: m[1, 2:]
```

```
Out[5]: matrix([[7, 8]])
```

```
In [6]: m[0:2, 1]
```

```
Out[6]: matrix([[2],
               [6]])
```

```
In [7]: cols = [0, 1, 3]
        m[:, cols]
```

```
Out[7]: matrix([[1, 2, 4],
               [5, 6, 8],
               [9, 1, 7]])
```

```
In [8]: # Для начала создадим матрицу, которая нам понадобится в работе.
        m = np.matrix('1 2 3 4; 5 6 7 8; 9 1 5 7')
        print(m)
```

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

```
In [9]: type(m)
```

```
Out[9]: numpy.matrix
```

Рисунок 1 – Проработка примеров

```
In [10]: # Matrix можно превратить в ndarray следующим образом:
        m = np.array(m)
        type(m)
```

```
Out[10]: numpy.ndarray
```

```
In [11]: #Для определения размерности массива Numpy используйте атрибут shape.
        m.shape
```

```
Out[11]: (3, 4)
```

```
In [12]: m[0:2, 1:3]
```

```
Out[12]: array([[2, 3],
               [6, 7]])
```

```
In [13]: m.max()
```

```
Out[13]: 9
```

```
In [14]: np.max(m)
```

```
Out[14]: 9
```

```
In [15]: m.max()
```

```
Out[15]: 9
```

```
In [16]: m.max(axis=1)
```

```
Out[16]: array([4, 8, 9])
```

```
In [17]: m.max(axis=0)
```

```
Out[17]: array([9, 6, 7, 8])
```

```
In [18]: m.mean()
```

```
Out[18]: 4.833333333333333
```

```
In [19]: m.mean(axis=1)
```

```
Out[19]: array([2.5, 6.5, 5.5])
```

```
In [20]: m.sum()
```

```
Out[20]: 58
```

```
In [21]: m.sum(axis=0)
```

```
Out[21]: array([15,  9, 15, 19])
```

Рисунок 2 – Проработка примеров

```

In [22]: nums = np.array([1, 2, 3, 4, 5, 6, 7, 8, 9, 10])
        letters = np.array(['a', 'b', 'c', 'd', 'a', 'e', 'b'])

In [23]: b = 5 > 7
        print(b)
        False

In [24]: less_than_5 = nums < 5
        less_than_5

Out[24]: array([ True,  True,  True,  True, False, False, False, False, False,
               False])

In [25]: pos_a = letters == 'a'
        pos_a

Out[25]: array([ True, False, False, False,  True, False, False])

In [26]: less_than_5 = nums < 5
        less_than_5
        nums[less_than_5]

Out[26]: array([1, 2, 3, 4])

In [27]: m = np.matrix('1 2 3 4; 5 6 7 8; 9 1 5 7')
        print(m)

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

In [28]: mod_m = np.logical_and(m>=3, m<=7)
        mod_m

Out[28]: matrix([[False, False,  True,  True],
                [ True,  True,  True, False],
                [False, False,  True,  True]])

In [29]: m[mod_m]

Out[29]: matrix([[3, 4, 5, 6, 7, 5, 7]])

In [30]: nums = np.array([1, 2, 3, 4, 5, 6, 7, 8, 9, 10])
        nums[nums < 5]

Out[30]: array([1, 2, 3, 4])

In [31]: nums[nums < 5] = 10
        print(nums)

        [10 10 10 10  5  6  7  8  9 10]

```

Рисунок 3 – Проработка примеров

```

In [32]: m[m > 7] = 25
        print(m)

        [[ 1  2  3  4]
         [ 5  6  7 25]
         [25  1  5  7]]

In [33]: np.arange(10)

Out[33]: array([0, 1, 2, 3, 4, 5, 6, 7, 8, 9])

In [34]: np.arange(5, 12)

Out[34]: array([ 5,  6,  7,  8,  9, 10, 11])

In [35]: np.arange(1, 5, 0.5)

Out[35]: array([1. , 1.5, 2. , 2.5, 3. , 3.5, 4. , 4.5])

In [36]: a = [[1, 2], [3, 4]]
        np.matrix(a)

Out[36]: matrix([[1, 2],
                [3, 4]])

In [37]: b = np.array([[5, 6], [7, 8]])
        np.matrix(b)

Out[37]: matrix([[5, 6],
                [7, 8]])

In [38]: np.matrix('1 2; 3, 4')

Out[38]: matrix([[1, 2],
                [3, 4]])

In [39]: np.zeros((3, 4))

Out[39]: array([[0., 0., 0., 0.],
                [0., 0., 0., 0.],
                [0., 0., 0., 0.]])

In [40]: np.eye(3)

Out[40]: array([[1., 0., 0.],
                [0., 1., 0.],
                [0., 0., 1.]])

In [41]: A = np.array([[1, 2, 3], [4, 5, 6], [7, 8, 9]])
        A

Out[41]: array([[1, 2, 3],
                [4, 5, 6],
                [7, 8, 9]])

```

Рисунок 4 – Проработка примеров

```

In [42]: np.ravel(A)
Out[42]: array([1, 2, 3, 4, 5, 6, 7, 8, 9])

In [43]: np.ravel(A, order='C')
Out[43]: array([1, 2, 3, 4, 5, 6, 7, 8, 9])

In [44]: np.ravel(A, order='F')
Out[44]: array([1, 4, 7, 2, 5, 8, 3, 6, 9])

In [45]: a = np.array([0, 1, 2, 3, 4, 5, 6, 7, 8, 9])
np.where(a % 2 == 0, a * 10, a / 10)
Out[45]: array([ 0. ,  0.1, 20. ,  0.3, 40. ,  0.5, 60. ,  0.7, 80. ,  0.9])

In [46]: a = np.random.rand(10)
a
Out[46]: array([0.47612574, 0.16878137, 0.92061606, 0.65274831, 0.72557003,
0.94513283, 0.36165723, 0.85820188, 0.43819674, 0.54485466])

In [47]: np.where(a > 0.5, True, False)
Out[47]: array([False, False,  True,  True,  True,  True, False,  True, False,
 True])

In [48]: np.where(a > 0.5, 1, -1)
Out[48]: array([-1, -1,  1,  1,  1,  1, -1,  1, -1,  1])

In [49]: x = np.linspace(0, 1, 5)
x
Out[49]: array([0. , 0.25, 0.5 , 0.75, 1. ])

In [50]: y = np.linspace(0, 2, 5)
y
Out[50]: array([0. , 0.5, 1. , 1.5, 2. ])

In [51]: xg, yg = np.meshgrid(x, y)
xg
Out[51]: array([[0. , 0.25, 0.5 , 0.75, 1. ],
 [0. , 0.25, 0.5 , 0.75, 1. ],
 [0. , 0.25, 0.5 , 0.75, 1. ],
 [0. , 0.25, 0.5 , 0.75, 1. ],
 [0. , 0.25, 0.5 , 0.75, 1. ]])

```

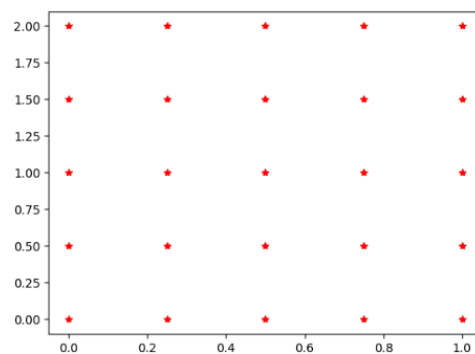
Рисунок 5 – Проработка примеров

```

In [52]: yg
Out[52]: array([[0. , 0. , 0. , 0. , 0. ],
 [0.5, 0.5, 0.5, 0.5, 0.5],
 [1. , 1. , 1. , 1. , 1. ],
 [1.5, 1.5, 1.5, 1.5, 1.5],
 [2. , 2. , 2. , 2. , 2. ]])

In [53]: import matplotlib.pyplot as plt
%matplotlib inline
plt.plot(xg, yg, color="r", marker="*", linestyle="none")
Out[53]: [<matplotlib.lines.Line2D at 0x226adf4e990>,
<matplotlib.lines.Line2D at 0x226ae127b10>,
<matplotlib.lines.Line2D at 0x226ed9e6590>,
<matplotlib.lines.Line2D at 0x226ae2c0b50>,
<matplotlib.lines.Line2D at 0x226ae2d8310>]

```



```

In [54]: np.random.permutation(7)
Out[54]: array([6, 0, 5, 3, 4, 1, 2])

In [55]: a = ['a', 'b', 'c', 'd', 'e']
np.random.permutation(a)
Out[55]: array(['d', 'c', 'a', 'b', 'e'], dtype='<U1')

In [56]: arr = np.linspace(0, 10, 5)
arr

```

Рисунок 6 – Проработка примеров

```
Out[56]: array([ 0. ,  2.5,  5. ,  7.5, 10. ])
```

```
In [57]: arr_mix = np.random.permutation(arr)
arr_mix
```

```
Out[57]: array([ 7.5, 10. ,  0. ,  2.5,  5. ])
```

```
In [58]: index_mix = np.random.permutation(len(arr_mix))
index_mix
```

```
Out[58]: array([0, 1, 3, 2, 4])
```

```
In [59]: arr[index_mix]
```

```
Out[59]: array([ 0. ,  2.5,  7.5,  5. , 10. ])
```

Рисунок 7 – Проработка примеров

Выполнение заданий из файла lab3.2:

Лабораторная работа 3.2. Знакомство с NumPy

Библиотека NumPy – быстрая библиотека для математики в Python, основная структура данных – массив `numpy.array`:



```
In [1]: # подключение модуля numpy под именем np
import numpy as np
```

```
In [2]: # основная структура данных - массив
a = np.array([1, 2, 3, 4, 5])
b = np.array([0.1, 0.2, 0.3, 0.4, 0.5])

print("a =", a)
print("b =", b)

a = [1 2 3 4 5]
b = [0.1 0.2 0.3 0.4 0.5]
```

Создайте массив с 5 любыми числами:

```
In [3]: my_list = np.array([1, 6.7, 8, -5, 0.1])
```

Арифметические операции, в отличие от операций над списками, применяются поэлементно:

Рисунок 8 – Проработка lab3.2

```
In [4]: list1 = [1, 2, 3]
        array1 = np.array([1, 2, 3])

        print("list1:", list1)
        print('\tlist1 * 3:', list1 * 3)
        print('\tlist1 + [1]:', list1 + [1])

        print('array1:', array1)
        print('\tarray1 * 3:', array1 * 3)
        print('\tarray1 + 1:', array1 + 1)

list1: [1, 2, 3]
list1 * 3: [1, 2, 3, 1, 2, 3, 1, 2, 3]
list1 + [1]: [1, 2, 3, 1]
array1: [1 2 3]
array1 * 3: [3 6 9]
array1 + 1: [2 3 4]
```

Создайте массив из 5 чисел. Возведите каждый элемент массива в степень 3

```
In [5]: my_list = np.array([1, 6.7, 8, -5, 0.1])
        my_list ** 3

Out[5]: array([ 1.00000e+00,  3.00763e+02,  5.12000e+02, -1.25000e+02,
               1.00000e-03])
```

Если в операции участвуют 2 массива (по умолчанию – одинакового размера), операции считаются для соответствующих пар:

```
In [6]: print("a + b =", a + b)
        print("a * b =", a * b)

a + b = [1.1 2.2 3.3 4.4 5.5]
a * b = [0.1 0.4 0.9 1.6 2.5]
```

```
In [7]: # вот это разность
        print("a - b =", a - b)

        # вот это деление
        print("a / b =", a / b)

        # вот это целочисленное деление
        print("a // b =", a // b)

        # вот это квадрат
        print("a ** 2 =", a ** 2)

a - b = [0.9 1.8 2.7 3.6 4.5]
a / b = [10. 10. 10. 10. 10.]
a // b = [ 9.  9. 10.  9. 10.]
a ** 2 = [ 1  4  9 16 25]
```

Рисунок 9 – Проработка lab3.2

Создайте два массива одинаковой длины. Выведите массив, полученный делением одного массива на другой.

```
In [8]: my_list = np.array([1, 6.7, 8, -5, 0.1])
        new_list = np.array([2, 5, 1.4, 2.3, -2])
        div_list = my_list // new_list
        print(div_list)

[ 0.  1.  5. -3. -1.]
```

Л — логика

К элементам массива можно применять логические операции.

Возвращаемое значение – массив, содержащий результаты вычислений для каждого элемента (True -- "да" или False -- "нет").

```
In [9]: print("a =", a)
        print("\ta > 1: ", a > 1)
        print("\nb =", b)
        print("\tb < 0.5: ", b < 0.5)

        print("\nОдновременная проверка условий:")
        print("\t(a > 1) & (b < 0.5): ", (a > 1) & (b < 0.5))
        print("\tА вот это проверяет, что a > 1 ИЛИ b < 0.5: ", (a > 1) | (b < 0.5))

a = [1 2 3 4 5]
a > 1: [False True True True True]

b = [0.1 0.2 0.3 0.4 0.5]
b < 0.5: [ True True True True False]
```

Одновременная проверка условий:
(a > 1) & (b < 0.5): [False True True True False]
А вот это проверяет, что a > 1 ИЛИ b < 0.5: [True True True True True]

Создайте 2 массива из 5 элементов. Проверьте условие "Элементы первого массива меньше 6, элементы второго массива делятся на 3"

```
In [10]: my_list = np.array([1, 6.7, 8, -5, 0.1])
         new_list = np.array([2, 5, 1.4, 2.3, -2])
         print(my_list < 6)
         print(new_list % 3 == 0)

[ True False False  True  True]
[False False False False False]
```

Теперь проверьте условие "Элементы первого массива делятся на 2 или элементы второго массива больше 2"

```
In [11]: print((my_list % 2 == 0) & (new_list > 2))

[False False False False False]
```

Рисунок 10 – Проработка lab3.2

```
In [12]: print("a =", a)
print("a > 2:", a > 2)
# индексация - выбираем элементы из массива в тех позициях, где True
print("a[a > 2]:", a[a > 2])

a = [1 2 3 4 5]
a > 2: [False False  True  True  True]
a[a > 2]: [3 4 5]
```

Создайте массив с элементами от 1 до 20. Выведите все элементы, которые больше 5 и не делятся на 2

Подсказка: создать массив можно с помощью функции `np.arange()`, действие которой аналогично функции `range`, которую вы уже знаете.

```
In [13]: m = np.arange(1,21)
print(m[(m > 5) & (m % 2 != 0)])

[ 7  9 11 13 15 17 19]
```

А ещё NumPy умеет...

Все операции NumPy оптимизированы для быстрых вычислений над целыми массивами чисел и в методах `np.array` реализовано множество функций, которые могут вам понадобиться:

```
In [14]: # теперь можно считать средний размер котиков в одну строку!
print("np.mean(a) =", np.mean(a))
# минимальный элемент
print("np.min(a) =", np.min(a))
# индекс минимального элемента
print("np.argmin(a) =", np.argmin(a))
# вывести значения массива без дубликатов
print("np.unique(['male', 'male', 'female', 'female', 'male']) =", np.unique(['male', 'male', 'female', 'female', 'male']))

# и ещё много всяких методов
# Google в помощь

np.mean(a) = 3.0
np.min(a) = 1
np.argmin(a) = 0
np.unique(['male', 'male', 'female', 'female', 'male']) = ['female' 'male']
```

Пора еще немного потренироваться с NumPy.

Рисунок 11 – Проработка lab3.2

Выполните операции, перечисленные ниже:

```
In [15]: print("Разность между a и b:", a - b)
          )
print("Квадраты элементов b:", b ** 2)
          )
print("Половины произведений элементов массивов a и b:", a * b)
          )

print()
print("Максимальный элемент b:", np.max(b))
          )
print("Сумма элементов массива b:", np.sum(b))
          )
print("Индекс максимального элемента b:", np.argmax(b))
          )

Разность между a и b: [0.9 1.8 2.7 3.6 4.5]
Квадраты элементов b: [0.01 0.04 0.09 0.16 0.25]
Половины произведений элементов массивов a и b: [0.1 0.4 0.9 1.6 2.5]

Максимальный элемент b: 0.5
Сумма элементов массива b: 1.5
Индекс максимального элемента b: 4
```

Задайте два массива: [5, 2, 3, 12, 4, 5] и ['f', 'o', 'o', 'b', 'a', 'r']

Выведите буквы из второго массива, индексы которых соответствуют индексам чисел из первого массива, которые больше 1, меньше 5 и делятся на 2

```
In [16]: fmas = np.array([5, 2, 3, 12, 4, 5])
smas = np.array(['f', 'o', 'o', 'b', 'a', 'r'])
smas[np.where ((fmas > 1) & (fmas < 5) & (fmas % 2 == 0))]

Out[16]: array(['o', 'a'], dtype='<U1')
```

Рисунок 12 – Проработка lab3.2

Выполнить задания из файла lab3.2hw:

Лабораторная работа 3.2. Домашнее задание

Задание №1

Создайте два массива: в первом должны быть четные числа от 2 до 12 включительно, а в другом числа 7, 11, 15, 18, 23, 29.

1. Сложите массивы и возведите элементы получившегося массива в квадрат.

```
In [1]: import numpy as np

a = np.arange(1,7) * 2
b = np.array([7, 11, 15, 18, 23, 29])
print((a + b) ** 2)

[ 81 225 441 676 1089 1681]
```

2. Выведите все элементы из первого массива, индексы которых соответствуют индексам тех элементов второго массива, которые больше 12 и дают остаток 3 при делении на 5.

```
In [2]: a[np.where((b > 12) & (b % 5 == 3))]

Out[2]: array([ 8, 10])
```

3. Проверьте условие "Элементы первого массива делятся на 4, элементы второго массива меньше 14". (Подсказка: в результате должен получиться массив с True и False)

```
In [3]: (a % 4 == 0) & (b < 14)

Out[3]: array([False,  True, False, False, False, False])
```

Рисунок 13 – выполнение заданий из lab3.2hw

Задание №2

- Найдите интересный для вас датасет. Например, можно выбрать датасет тут: <http://data.un.org/Explorer.aspx> (выбираете датасет, жмете на view data, потом download, выбирайте csv формат)
- Рассчитайте подходящие описательные статистики для признаков объектов в выбранном датасете
- Проанализируйте и прокомментируйте содержательно получившиеся результаты
- Все комментарии оформляйте строго в ячейках формата markdown

```
In [4]: import csv
import numpy as np
with open('world_prices.csv', 'r', newline='', encoding='utf-8') as csvfile:
    reader = csv.reader(csvfile, delimiter=',')
    glob = []
    rus = []
    next(reader)
    for i in reader:
        glob.append(float(i[9]))
        rus.append(float(i[7]))
    stat_glob = np.array(glob)
    stat_rus = np.array(rus)
    print(f"Среднее значение цен в мире {np.mean(stat_glob)}, среднее значение цен в России: {np.mean(stat_rus)}")
    print(f"Среднее отклонение цен в мире {np.std(stat_glob)}, среднее отклонение цен в России: {np.std(stat_rus)}")
    print(f"Медиана цен в мире {np.median(stat_glob)}, медиана цен в России: {np.median(stat_rus)}")
    print(f"Дисперсия цен в мире {np.var(stat_glob)}, дисперсия цен в России: {np.var(stat_rus)}")
```

Среднее значение цен в мире 0.1684310699588477, среднее значение цен в России: 0.5174794238683128
Среднее отклонение цен в мире 0.36284116538512023, среднее отклонение цен в России: 0.9494874752338788
Медиана цен в мире 0.08, медиана цен в России: 0.29
Дисперсия цен в мире 0.1316537112980322, дисперсия цен в России: 0.9015264656260055

In []:

Рисунок 14 – выполнение заданий из lab3.2hw

Индивидуальное задание:

Индивидуальное задание

Вариант 12: Осуществить циклический сдвиг элементов прямоугольной матрицы на n элементов вправо или вниз (в зависимости от введенного режима), может быть больше количества элементов в строке или столбце.

```
In [1]: import numpy as np
```

Создадим матрицу случайных чисел

```
In [2]: M = np.random.randint (0, 20, (5, 5))  
print(M)
```

```
[[ 1  6 15  1 19]  
 [ 8  5  0  1 15]  
 [17 10 16  2 17]  
 [ 9 18 19 19  5]  
 [ 4 15  8  0 15]]
```

Для циклического сдвига вдоль указанной оси библиотеке numpy есть функция roll()

Данная функция принимает 3 параметра: a - numpy массив shift - определяет число смещений axis - определяет ось смещения

```
In [3]: rej_choose = int(input("Введите 0 для сдвига вниз или 1 для сдвига вправо"))  
num = int(input("Укажите число на которое необходимо сдвинуть матрицу"))  
if rej_choose == 0:  
    M = np.roll(M, num, axis = rej_choose) # Сдвиг вниз  
    print(M)  
elif rej_choose == 1:  
    M = np.roll(M, num, axis = rej_choose) # Сдвиг вправо  
    print(M)  
else:  
    print("Error")
```

```
Введите 0 для сдвига вниз или 1 для сдвига вправо0  
Укажите число на которое необходимо сдвинуть матрицу3  
[[17 10 16  2 17]  
 [ 9 18 19 19  5]  
 [ 4 15  8  0 15]  
 [ 1  6 15  1 19]  
 [ 8  5  0  1 15]]
```

Рисунок 15 – Индивидуальное задание

Контрольные вопросы

Контрольные вопросы

1. Каково назначение библиотеки NumPy?

numpy – это библиотека для языка программирования Python, которая предоставляет в распоряжение разработчика инструменты для эффективной работы с многомерными массивами и высокопроизводительные вычислительные алгоритмы.

2. Что такое массивы ndarray?

Основной элемент библиотеки NumPy — объект ndarray (что значит N-размерный массив). Этот объект является многомерным однородным массивом с заранее заданным количеством элементов.

3. Как осуществляется доступ к частям многомерного массива?

Извлечем элемент из нашей матрицы с координатами (1, 0), 1 – это номер строки, 0 – номер столбца.

1	2	3	4
5	6	7	8
9	1	5	7

```
>>> m[1, 0]  
5
```

Получим вторую строку матрицы.

1	2	3	4
5	6	7	8
9	1	5	7

```
>>> m[1, :]  
matrix([[5, 6, 7, 8]])
```

Извлечем третий столбец матрицы.

1	2	3	4
5	6	7	8
9	1	5	7

```
>>> m[:, 2]
matrix([[3],
        [7],
        [5]])
```

Иногда возникает задача взять не все элементы строки, а только часть: рассмотрим пример, когда нам из второй строки нужно извлечь все элементы, начиная с третьего.

1	2	3	4
5	6	7	8
9	1	5	7

```
>>> m[1, 2:]
matrix([[7, 8]])
```

Запись **2:** означает, что начиная с третьего столбца включительно (т.к. нумерация начинается с 0, то третий элемент имеет индекс 2) взять все оставшиеся в ряду элементы .

Часть столбца матрицы

Аналогично предыдущему примеру, можно извлечь только часть столбца матрицы.

1	2	3	4
5	6	7	8
9	1	5	7

```
>>> m[0:2, 1]
matrix([[2],
        [6]])
```

Непрерывная часть матрицы

Извлечем из заданной матрицы матрицу, располагающуюся так как показано на рисунке ниже.

1	2	3	4
5	6	7	8
9	1	5	7

```
\>>> m[0:2, 1:3]
matrix([[2, 3],
        [6, 7]])
```

4. Как осуществляется расчет статистик по данным?

Функции (методы) для расчета статистик в *Numpy*

Ниже, в таблице, приведены методы объекта *ndarray* (или *matrix*), которые, как мы помним из раздела выше, могут быть также вызваны как функции библиотеки *Numpy*, для расчета статистик по данным массива.

Имя метода	Описание
argmax	Индексы элементов с максимальным значением (по осям)
argmin	Индексы элементов с минимальным значением (по осям)
max	Максимальные значения элементов (по осям)
min	Минимальные значения элементов (по осям)
mean	Средние значения элементов (по осям)
prod	Произведение всех элементов (по осям)
std	Стандартное отклонение (по осям)
sum	Сумма всех элементов (по осям)
var	Дисперсия (по осям)

Вычислим некоторые из представленных выше статистик.

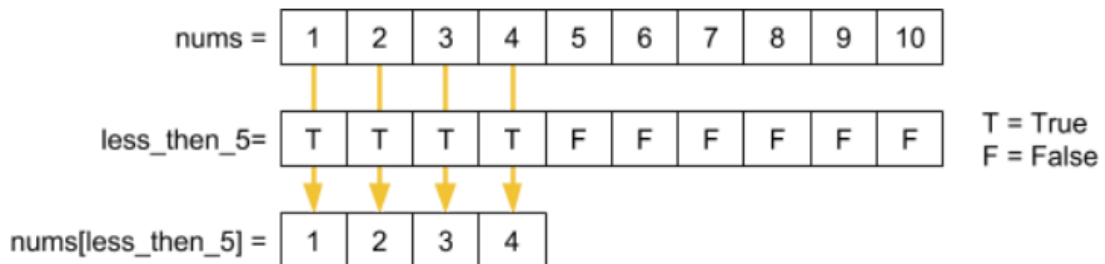
```
>>> m.mean()
4.833333333333333
>>> m.mean(axis=1)
matrix([[2.5],
        [6.5],
        [5.5]])
>>> m.sum()
58
>>> m.sum(axis=0)
matrix([[15,  9, 15, 19]])
```

5. Как выполняется выборка данных из массивов ndarray?

Самым замечательным в использовании *boolean* массивов при работе с *ndarray* является то, что их можно применять для построения выборок. Вернемся к рассмотренным выше примерам.

```
>>> less_than_5 = nums < 5
>>> less_than_5
array([ True,  True,  True,  True, False, False, False, False, False,
        False])
```

Если мы переменную *less_than_5* передадим в качестве списка индексов для *nums*, то получим массив, в котором будут содержаться элементы из *nums* с индексами равными индексам *True* позиций массива *less_than_5*, графически это будет выглядеть так.



```
>>> nums[less_than_5]
array([1, 2, 3, 4])
```

Данный подход будет работать с массивами большей размерности. Возьмем уже знакомую нам по предыдущим урокам матрицу.

```
>>> m = np.matrix('1 2 3 4; 5 6 7 8; 9 1 5 7')
>>> print(m)
[[1 2 3 4]
 [5 6 7 8]
 [9 1 5 7]]
```

Построим логическую матрицу со следующим условием: $m \geq 3$ and $m \leq 7$, в *Numpy* нельзя напрямую записать такое условие, поэтому воспользуемся функцией *logical_and()*, ее и многие другие полезные функции вы сможете найти на странице [Logic functions](#).

```
>>> mod_m = np.logical_and(m>=3, m<=7)
>>> mod_m
matrix([[False, False,  True,  True],
        [ True,  True,  True, False],
        [False, False,  True,  True]])
>>> m[mod_m]
matrix([[3, 4, 5, 6, 7, 5, 7]])
```

В результате мы получили матрицу с одной строкой, элементами которой являются все отмеченные как *True* элементы из исходной матрицы.

