

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»**

Выполнила:

Кувшин Ирина Анатольевна  
2 курс, группа ПИЖ-б-о-21-1,  
09.03.04 «Программная инженерия»,  
направленность (профиль) «Разработка  
и сопровождение программного  
обеспечения», очная форма обучения

---

(подпись)

Проверил:

---

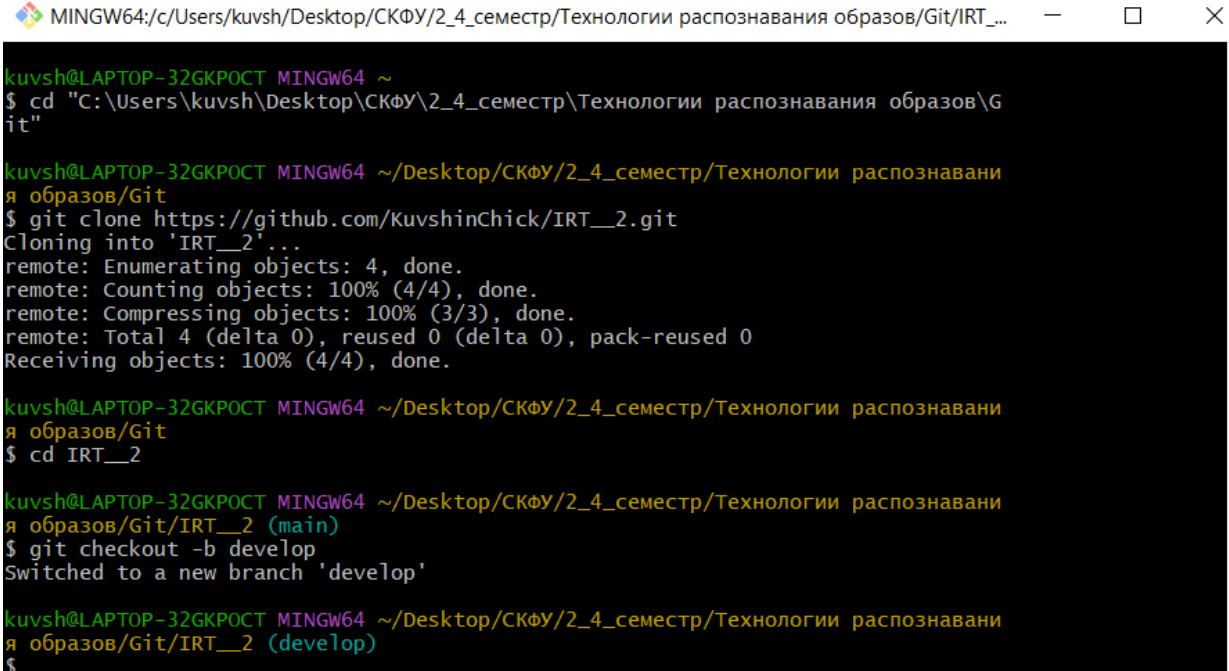
(подпись)

Отчет защищен с оценкой \_\_\_\_\_ Дата защиты \_\_\_\_\_

Ставрополь, 2023 г.

**Цель работы:** исследовать базовые возможности библиотеки NumPy языка программирования Python.

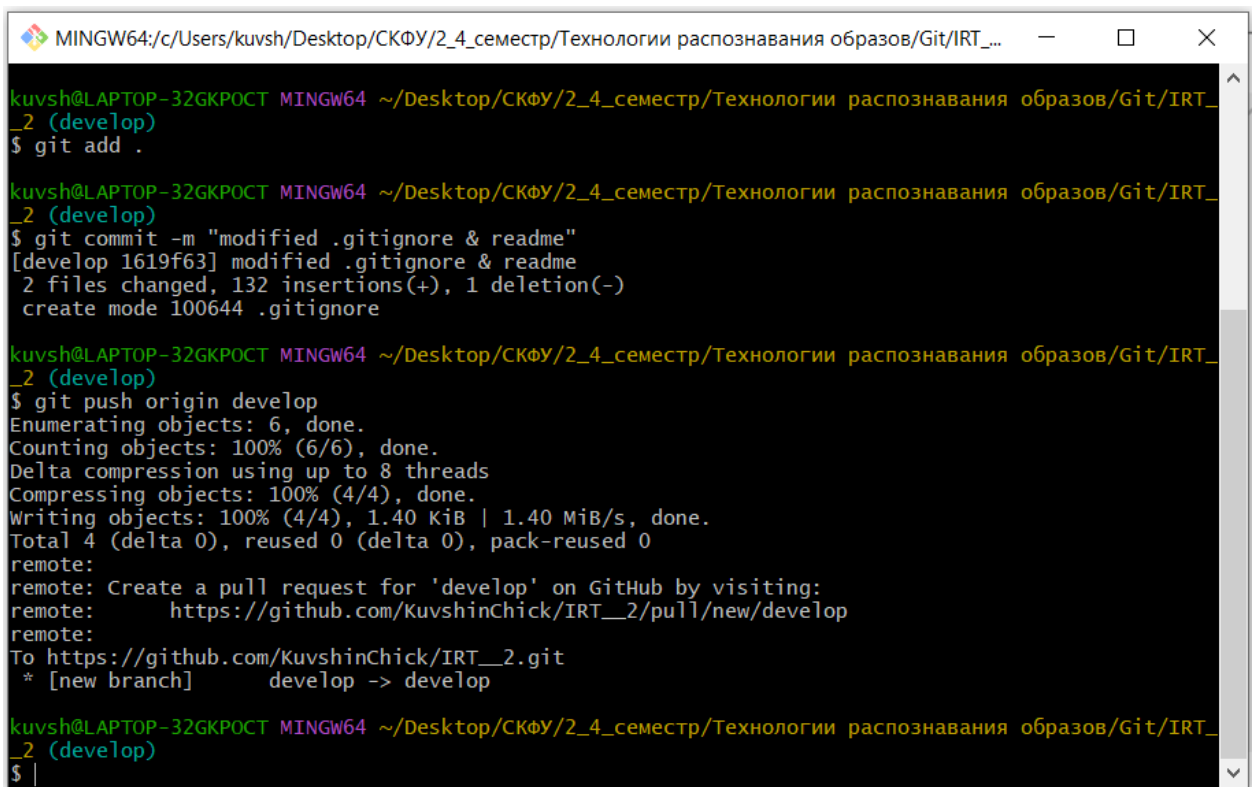
1. Изучить теоретический материал работы.
2. Создать общедоступный репозиторий на GitHub, в котором будет использована лицензия MIT и выбранный Вами язык программирования (выбор языка программирования будет доступен после установки флажка Add .gitignore).
3. Выполните клонирование созданного репозитория на рабочий компьютер.
4. Организуйте свой репозиторий в соответствии с моделью ветвления git-flow.



```
MINGW64:/c/Users/kuvsh/Desktop/СКФУ/2_4_семестр/Технологии распознавания образов/Git/IRT_...
kuvsh@LAPTOP-32GKPOCT MINGW64 ~
$ cd "C:\Users\kuvsh\Desktop\СКФУ\2_4_семестр\Технологии распознавания образов\Git"
kuvsh@LAPTOP-32GKPOCT MINGW64 ~/Desktop/СКФУ/2_4_семестр/Технологии распознавания образов/Git
$ git clone https://github.com/KuvshinChick/IRT__2.git
Cloning into 'IRT__2'...
remote: Enumerating objects: 4, done.
remote: Counting objects: 100% (4/4), done.
remote: Compressing objects: 100% (3/3), done.
remote: Total 4 (delta 0), reused 0 (delta 0), pack-reused 0
Receiving objects: 100% (4/4), done.
kuvsh@LAPTOP-32GKPOCT MINGW64 ~/Desktop/СКФУ/2_4_семестр/Технологии распознавания образов/Git
$ cd IRT__2
kuvsh@LAPTOP-32GKPOCT MINGW64 ~/Desktop/СКФУ/2_4_семестр/Технологии распознавания образов/Git/IRT__2 (main)
$ git checkout -b develop
Switched to a new branch 'develop'
kuvsh@LAPTOP-32GKPOCT MINGW64 ~/Desktop/СКФУ/2_4_семестр/Технологии распознавания образов/Git/IRT__2 (develop)
$
```

Рисунок 2.1 – Клонирование репозитория и создание ветки develop

5. Дополните файл .gitignore необходимыми правилами для выбранного языка программирования, интерактивной оболочки Jupyter notebook и интегрированной среды разработки.



```
MINGW64:/c/Users/kuvsh/Desktop/СКФУ/2_4_семестр/Технологии распознавания образов/Git/IRT_...
kuvsh@LAPTOP-32GKPOCT MINGW64 ~/Desktop/СКФУ/2_4_семестр/Технологии распознавания образов/Git/IRT_
_2 (develop)
$ git add .

kuvsh@LAPTOP-32GKPOCT MINGW64 ~/Desktop/СКФУ/2_4_семестр/Технологии распознавания образов/Git/IRT_
_2 (develop)
$ git commit -m "modified .gitignore & readme"
[develop 1619f63] modified .gitignore & readme
2 files changed, 132 insertions(+), 1 deletion(-)
create mode 100644 .gitignore

kuvsh@LAPTOP-32GKPOCT MINGW64 ~/Desktop/СКФУ/2_4_семестр/Технологии распознавания образов/Git/IRT_
_2 (develop)
$ git push origin develop
Enumerating objects: 6, done.
Counting objects: 100% (6/6), done.
Delta compression using up to 8 threads
Compressing objects: 100% (4/4), done.
Writing objects: 100% (4/4), 1.40 KiB | 1.40 MiB/s, done.
Total 4 (delta 0), reused 0 (delta 0), pack-reused 0
remote:
remote: Create a pull request for 'develop' on GitHub by visiting:
remote:   https://github.com/KuvshinChick/IRT__2/pull/new/develop
remote:
To https://github.com/KuvshinChick/IRT__2.git
 * [new branch]      develop -> develop

kuvsh@LAPTOP-32GKPOCT MINGW64 ~/Desktop/СКФУ/2_4_семестр/Технологии распознавания образов/Git/IRT_
_2 (develop)
$
```

Рисунок 2.2 – Обновление .gitignore и readme

6. Проработать примеры лабораторной работы.

## Проработка примеров

### Доступ к частям многомерного массива

```
In [4]: 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 [6]: m[1, 0]
```

```
Out[6]: 5
```

```
In [7]: m[1, :]
```

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

```
In [8]: m[:, 2]
```

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

```
In [9]: m[1, 2:]
```

```
Out[9]: matrix([[7, 8]])
```

```
In [10]: m[0:2, 1]
```

```
Out[10]: matrix([[2],
                [6]])
```

```
In [12]: cols = [0, 1, 3]
m[:, cols]
```

```
Out[12]: matrix([[1, 2, 4],
                [5, 6, 8],
                [9, 1, 7]])
```

### Расчет статистик по данным в массиве

```
In [13]: # 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 [14]: type(m)
```

```
Out[14]: numpy.matrix
```

```
In [16]: # Matrix можно превратить в ndarray следующим образом:
m = np.array(m)
type(m)
```

```
Out[16]: numpy.ndarray
```

```
In [17]: #Для определения размерности массива Numpy используйте атрибут shape.
m.shape
```

```
Out[17]: (3, 4)
```

```
In [11]: m[0:2, 1:3]
```

```
Out[11]: matrix([[2, 3],
                [6, 7]])
```

Для расчета той или иной статистики, соответствующую функцию можно вызвать как метод объекта, с которым вы работаете.

Функции (методы) для расчета статистик в Numpy:

Имя метода	Описание
<b>argmax</b>	Индексы элементов с максимальным значением (по осям)
<b>argmin</b>	Индексы элементов с минимальным значением (по осям)
<b>max</b>	Максимальные значения элементов (по осям)
<b>min</b>	Минимальные значения элементов (по осям)
<b>mean</b>	Средние значения элементов (по осям)
<b>prod</b>	Произведение всех элементов (по осям)

<b>sum</b>	Сумма всех элементов (по осям)
<b>var</b>	Дисперсия (по осям)

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

```
Out[18]: 9
```

```
In [19]: np.max(m)
```

```
Out[19]: 9
```

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

```
Out[20]: 9
```

```
In [21]: m.max(axis=1)
```

```
Out[21]: array([4, 8, 9])
```

```
In [22]: m.max(axis=0)
```

```
Out[22]: array([9, 6, 7, 8])
```

```
In [23]: m.mean()
```

```
Out[23]: 4.833333333333333
```

```
In [24]: m.mean(axis=1)
```

```
Out[24]: array([2.5, 6.5, 5.5])
```

```
In [25]: m.sum()
```

```
Out[25]: 58
```

```
In [26]: m.sum(axis=0)
```

```
Out[26]: array([15, 9, 15, 19])
```

### Использование boolean массива для доступа к ndarray

```
In [27]: nums = np.array([1, 2, 3, 4, 5, 6, 7, 8, 9, 10])
letters = np.array(['a', 'b', 'c', 'd', 'a', 'e', 'b'])
```

```
In [28]: b = 5 > 7
print(b)
```

```
False
```

```
In [33]: # В этом примере мы создали boolean массив, в котором на месте
# элементов из nums, которые меньше пяти стоит True,
# в остальных случаях - False.
less_than_5 = nums < 5
less_than_5
```

```
Out[33]: array([ True,  True,  True,  True, False, False, False, False, False,
        False])
```

```
In [35]: # Построим массив, в котором значение True будут иметь элементы,
# чей индекс совпадает с индексами, на которых стоит символ 'a'
# в массиве letters.
pos_a = letters == 'a'
pos_a
```

```
Out[35]: array([ True, False, False, False,  True, False, False])
```

```
In [36]: # Самым замечательным в использовании boolean массивов при работе
# с ndarray является то, что их можно применять для построения выборок.
less_than_5 = nums < 5
less_than_5
nums[less_than_5]
```

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

```
In [39]: 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 [38]: mod_m = np.logical_and(m>=3, m<=7)
mod_m
```

```
Out[38]: matrix([[False, False,  True,  True],
 [ True,  True,  True, False],
 [False, False,  True,  True]])
```

```
In [40]: m[mod_m]
```

```
Out[40]: matrix([[3, 4, 5, 6, 7, 5, 7]])
```

```
In [41]: nums = np.array([1, 2, 3, 4, 5, 6, 7, 8, 9, 10])
nums[nums < 5]
```

```
Out[41]: array([1, 2, 3, 4])
```

```
In [43]: nums[nums < 5] = 10
print(nums)
```

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

```
In [45]: m[m > 7] = 25
print(m)
```

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

### Дополнительные функции

```
In [46]: np.arange(10)
```

```
Out[46]: array([0, 1, 2, 3, 4, 5, 6, 7, 8, 9])
```

```
In [47]: np.arange(5, 12)
```

```
Out[47]: array([ 5,  6,  7,  8,  9, 10, 11])
```

```
In [48]: np.arange(1, 5, 0.5)
```

```
Out[48]: array([1. , 1.5, 2. , 2.5, 3. , 3.5, 4. , 4.5])
```

```
In [49]: a = [[1, 2], [3, 4]]
np.matrix(a)
```

```
Out[49]: matrix([[1, 2],
 [3, 4]])
```

```
In [50]: b = np.array([[5, 6], [7, 8]])
np.matrix(b)
```

```
Out[50]: matrix([[5, 6],
 [7, 8]])
```

```
In [51]: np.matrix('[1, 2; 3, 4]')
```

```
Out[51]: matrix([[1, 2],
 [3, 4]])
```

```
In [52]: np.zeros((3, 4))
```

```
Out[52]: array([[0., 0., 0., 0.],
 [0., 0., 0., 0.],
 [0., 0., 0., 0.]])
```

```
In [53]: np.eye(3)
```

```
Out[53]: array([[1., 0., 0.],
 [0., 1., 0.],
 [0., 0., 1.]])
```

```
In [54]: A = np.array([[1, 2, 3], [4, 5, 6], [7, 8, 9]])
A
```

```
Out[54]: array([[1, 2, 3],
               [4, 5, 6],
               [7, 8, 9]])
```

```
In [55]: # Функция np.ravel() используется для того, чтобы преобразовать
# матрицу в одномерный вектор.
np.ravel(A)
```

```
Out[55]: array([1, 2, 3, 4, 5, 6, 7, 8, 9])
```

```
In [56]: # 'C', что означает - массив будет собираться из строк исходной матрицы.
np.ravel(A, order='C')
```

```
Out[56]: array([1, 2, 3, 4, 5, 6, 7, 8, 9])
```

```
In [57]: # Если указать order='F',
# то в качестве элементов для сборки будут выступать столбцы матрицы.
np.ravel(A, order='F')
```

```
Out[57]: array([1, 4, 7, 2, 5, 8, 3, 6, 9])
```

```
In [58]: a = np.array([0, 1, 2, 3, 4, 5, 6, 7, 8, 9])
np.where(a % 2 == 0, a * 10, a / 10)
```

```
Out[58]: array([ 0. ,  0.1, 20. ,  0.3, 40. ,  0.5, 60. ,  0.7, 80. ,  0.9])
```

```
In [60]: a = np.random.rand(10)
a
```

```
Out[60]: array([0.00820486, 0.78118219, 0.27780098, 0.38368287, 0.40021572,
               0.30764839, 0.32263822, 0.84334978, 0.27491653, 0.17409483])
```

```
In [61]: np.where(a > 0.5, True, False)
```

```
Out[61]: array([False,  True, False, False, False, False, False,  True, False,
               False])
```

```
In [62]: np.where(a > 0.5, 1, -1)
```

```
Out[62]: array([-1,  1, -1, -1, -1, -1, -1,  1, -1, -1])
```

Функция meshgrid() позволяет получить матрицу координат из координатных векторов. Если, например, у нас есть два одномерных вектора координат, то передав их в качестве аргументов в meshgrid() мы получим две матрицы, в которой элементы будут составлять пары, заполняя все пространство, определяемое этими векторами. Проще посмотреть это на примере.

```
In [63]: x = np.linspace(0, 1, 5)
x
```

```
Out[63]: array([0. , 0.25, 0.5 , 0.75, 1.  ])
```

```
In [64]: y = np.linspace(0, 2, 5)
y
```

```
Out[64]: array([0. , 0.5, 1. , 1.5, 2.  ])
```

```
In [66]: xg, yg = np.meshgrid(x, y)
xg
```

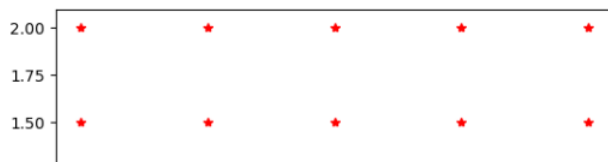
```
Out[66]: array([[0. , 0.25, 0.5 , 0.75, 1.  ],
               [0. , 0.25, 0.5 , 0.75, 1.  ],
               [0. , 0.25, 0.5 , 0.75, 1.  ],
               [0. , 0.25, 0.5 , 0.75, 1.  ],
               [0. , 0.25, 0.5 , 0.75, 1.  ]])
```

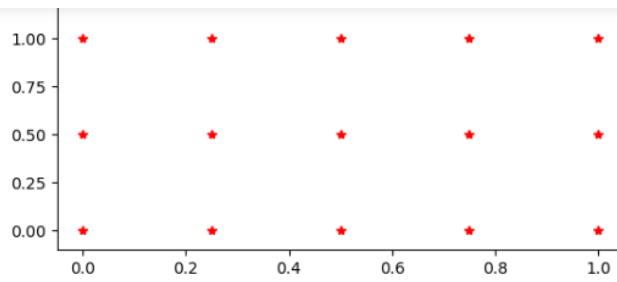
```
In [67]: yg
```

```
Out[67]: 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 [68]: import matplotlib.pyplot as plt
%matplotlib inline
plt.plot(xg, yg, color="r", marker="*", linestyle="none")
```

```
Out[68]: [<matplotlib.lines.Line2D at 0x12e06adcf40>,
<matplotlib.lines.Line2D at 0x12e06af0040>,
<matplotlib.lines.Line2D at 0x12e06af0130>,
<matplotlib.lines.Line2D at 0x12e06af0220>,
<matplotlib.lines.Line2D at 0x12e06af0310>]
```





```
In [69]: np.random.permutation(7)
```

```
Out[69]: array([4, 3, 2, 6, 0, 1, 5])
```

```
In [70]: a = ['a', 'b', 'c', 'd', 'e']  
np.random.permutation(a)
```

```
Out[70]: array(['a', 'b', 'd', 'c', 'e'], dtype='<U1')
```

```
In [71]: arr = np.linspace(0, 10, 5)  
arr
```

```
Out[71]: array([ 0. ,  2.5,  5. ,  7.5, 10. ])
```

```
In [72]: arr_mix = np.random.permutation(arr)  
arr_mix
```

```
Out[72]: array([10. ,  0. ,  5. ,  2.5,  7.5])
```

```
In [73]: index_mix = np.random.permutation(len(arr_mix))  
index_mix
```

```
Out[73]: array([0, 3, 4, 1, 2])
```

```
In [74]: arr[index_mix]
```

```
Out[74]: array([ 0. ,  7.5, 10. ,  2.5,  5. ])
```

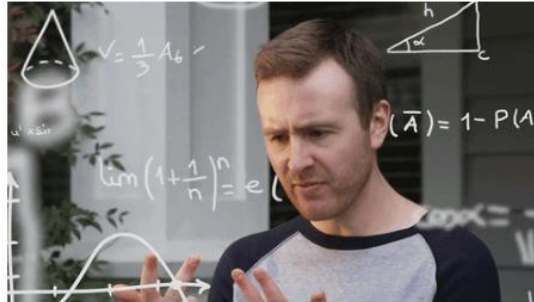
Рисунок 2.3 – Результат проработки примеров



## 7. Решить задания в ноутбуках, выданных преподавателем

### Лабораторная работа 3.2. Знакомство с NumPy

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



```
In [57]: # подключение модуля numpy под именем np
import numpy as np
```

```
In [58]: # 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 [59]: a = np.random.randint(20, size=5)
a
```

```
Out[59]: array([ 9,  1, 10,  9,  0])
```

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

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

```
In [60]: 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 [61]: a = np.random.randint(20, size=5)
print(a)
print(a**3)

[ 4  6 18 13 18]
[ 64 216 5832 2197 5832]
```

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

```
In [62]: print("a + b =", a + b)
print("a * b =", a * b)

a + b = [ 4.1  6.2 18.3 13.4 18.5]
a * b = [0.4 1.2 5.4 5.2 9. ]
```

```
In [63]: # 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)
```

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

```
In [66]: a = np.random.randint(100, size=5)
print(a)
b = np.random.randint(20, size=5)
print(b)

print(a < 6)
print(b % 3 == 0)

[60 10 82 38 95]
[3 2 0 3 3]
[False False False False False]
[ True False  True  True  True]
```

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

```
In [67]: print((a % 2 == 0)|(b > 2))

[ True  True  True  True  True]
```

Зачем это нужно? Чтобы выбирать элементы массива, удовлетворяющие какому-нибудь условию.

```
In [68]: 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 = [60 10 82 38 95]
a > 2: [ True  True  True  True  True]
a[a > 2]: [60 10 82 38 95]
```

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 [69]: c = np.arange(1,21)
print(c)
print(c[(c > 5) & (c % 2 != 0)])

[ 1  2  3  4  5  6  7  8  9 10 11 12 13 14 15 16 17 18 19 20]
[ 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

```
# и ещё много всяких методов  
# Google в помощь
```

```
np.mean(a) = 57.0  
np.min(a) = 10  
np.argmax(a) = 1  
np.unique(['male', 'male', 'female', 'female', 'male']) = ['female' 'male']
```

Пора еще немного потренироваться с NumPy.

Выполните операции, перечисленные ниже:

```
In [71]: 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: [57  8 82 35 92]  
Квадраты элементов b: [9 4 0 9 9]  
Половины произведений элементов массивов a и b: [180 20  0 114 285]
```

```
Максимальный элемент b: 3  
Сумма элементов массива b: 11  
Индекс максимального элемента b: 0
```

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 [87]: num = np.array([5, 2, 3, 12, 4, 5])  
          let = np.array(['f', 'o', 'o', 'b', 'a', 'r'])  
          print(let[np.where((num > 1) & (num < 5) & (num % 2 == 0))])  
  
          ['o' 'a']
```

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

```
In [64]: a = np.random.randint(100, size=5)
print(a)
b = np.random.randint(20, size=5)
print(b)
print(a/b)

[14 62 96 60 33]
[11 15 13 11 7]
[1.27272727 4.13333333 7.38461538 5.45454545 4.71428571]
```

#### Л — логика

К элементам массива можно применять логические операции.

Возвращаемое значение — массив, содержащий результаты вычислений для каждого элемента (True — "да" или False — "нет"):

```
In [65]: 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("А вот это проверяет, что a > 1 ИЛИ b < 0.5: ", (a > 1) | (b < 0.5))

a = [14 62 96 60 33]
a > 1: [ True  True  True  True  True]

b = [11 15 13 11 7]
b < 0.5: [False False False False False]

Одновременная проверка условий:
(a > 1) & (b < 0.5): [False False False False 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 [66]: a = np.random.randint(100, size=5)
print(a)
b = np.random.randint(20, size=5)
print(b)

print(a < 6)
print(b % 3 == 0)
```

## Рисунок 2.4 — Результат выполнения заданий

### 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 [4]: import numpy as np

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

[ 2  4  6  8 10 12]
[ 7 11 15 18 23 29]
[ 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 [12]: print(a[np.logical_and(b > 12, b % 5 == 3)])
print((a % 4 == 0) & (b < 14))

[ 8 10]
[False  True False False False False]
```

## 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

<https://www.kaggle.com/datasets/themrityunjaypathak/tesla-stock-price-2005-2023> Цена акций Tesla [2005-2023]

```
In [18]: import csv
import numpy as np

# Открытие файла только для чтения
with open('Tesla_stock_Price.csv', 'r', newline='', encoding='utf-8') as csvfile:
    reader = csv.reader(csvfile, delimiter=',')
    # Списки
    Price = []
    High_price = []
    # Пропуск заголовка
    next(reader)
    # Заполнение списков
    for row in reader:
        if row[3] != '':
            Price.append(float(row[1]))
            High_price.append(float(row[3]))
    Price = np.array(Price)
    High_price = np.array(High_price)
    print(f"Средняя цена акций {np.mean(Price)}, Средняя цена акций из максимальных: {np.mean(High_price)}")
    print(f"Среднее отклонение цен {np.std(Price)}, среднее отклонение цен из максимальных: {np.std(High_price)}")
    print(f"Медиана цен {np.median(Price)}, медиана максимальных цен: {np.median(High_price)}")
    print(f"Дисперсия цен {np.var(Price)}, дисперсия максимальных цен: {np.var(High_price)}")

Средняя цена акций 59.68772498426684, Средняя цена акций из максимальных: 61.02593769666456
Среднее отклонение цен 95.57366675624947, среднее отклонение цен из максимальных: 97.80466634694456
Медиана цен 16.325, медиана максимальных цен: 16.56
Дисперсия цен 9134.325777234624, дисперсия максимальных цен: 9565.75275923715
```

## Рисунок 2.5 – Результат выполнения заданий

8. Создать ноутбук, в котором выполнить решение индивидуального задания. Ноутбук должен содержать условие индивидуального задания. При решении индивидуального задания не должны быть использованы условный оператор if, а также операторы циклов while и for, а только средства библиотеки NumPy. Привести в ноутбуке обоснование принятых решений. Номер варианта индивидуального задания необходимо уточнить у преподавателя.

8. Соседями элемента  $A_j$  в матрице назовем элементы  $A_k$  с  $i - 1 < k < i + 1$ ,  $j - 1 < 1 < j + 1$ ,  $(k, 1)/(i, j)$ . Операция сглаживания матрицы дает новую матрицу того же размера, каждый элемент которой получается как среднее арифметическое имеющихся соседей соответствующего элемента исходной матрицы. Построить результат сглаживания заданной вещественной матрицы размером 7 на 7. В сглаженной матрице найти сумму модулей элементов, расположенных ниже главной диагонали.

8. Соседями элемента  $A_{ij}$  в матрице назовем элементы  $A_{kl}$   $ci - 1 \leq k \leq i + 1, j - 1 \leq l \leq j + 1, (k, l) \neq (i, j)$ . Операция сглаживания матрицы дает новую матрицу того же размера, каждый элемент которой получается как среднее арифметическое имеющихся соседей соответствующего элемента исходной матрицы. Построить результат сглаживания заданной вещественной матрицы размером 7 на 7. В сглаженной матрице найти сумму модулей элементов, расположенных ниже главной диагонали.

1. Создадим исходную матрицу 7\*7 и нулевую матрицу для результата сглаживания

```
In 382 1 # подключение модуля numpy под именем np
2 import numpy as np
3 main_matrix = np.random.randint(0,5,(7,7))
4 print("Сгенерированная матрица: ")
5 print(main_matrix)
6
7 task_matrix = np.zeros((7,7))
```

Сгенерированная матрица:

```
[[4 3 0 0 2 4 1]
 [1 4 4 2 1 1 2]
 [1 4 1 0 3 4 4]
 [4 4 1 2 0 1 4]
 [4 3 3 0 2 4 2]
 [1 2 3 3 1 0 3]
 [0 4 3 0 1 0 3]]
```

2. Для прохождения массива будем использовать `numpy.ndenumerate`. Для поиска соседей используются срезы матрицы. Для правильной работы нахождения среднего соседей, необходимо с помощью конструкций `if` ввести проверку на первые строку и столбец (если этого не делать, срез возвращает неправильный результат, так как выходит за границу основного массива). Затем с помощью функции `sum` суммируем элементы новой матрицы(среза), вычтем `value` (так как оно не относится к соседям, а является главным элементом) и поделим на число элементов массива(среза) - 1 (исключая главный элемент). Таким образом, найдя все средние арифметические значения, построим сглаженную матрицу.

```
In 381 1 for (i,j), value in np.ndenumerate(main_matrix):
2     # Берем "срез соседей" матрицы, текущий элемент будет исключен при расчетах
3     # Нужна проверка на крайние нулевые столбец и строку для правильного среза
4     if i == 0 and j == 0:
5         m = main_matrix[0:i+2, 0:j+2]
6         mean = (np.sum(m)-value)/(m.size - 1)
```

```
8 elif i == 0:
9     m = main_matrix[0:i+2, j-1:j+2]
10    mean = (np.sum(m)-value)/(m.size - 1)
11
12 elif j == 0:
13     m = main_matrix[i-1:i+2, 0:j+2]
14     mean = (np.sum(m)-value)/(m.size - 1)
15
16 else:
17     m = main_matrix[i-1:i+2, j-1:j+2]
18     mean = (np.sum(m)-value)/(m.size - 1)
19     task_matrix[i,j] = round(mean,2)
20
21 print("Сглаженная матрица: ")
22 print(task_matrix)
23
```

Сглаженная матрица:

```
[[2.33 2.   1.6 1.8  1.6 1.6 1.33]
 [1.4  1.75 1.25 1.38 1.5  1.25 1.6 ]
 [2.   1.88 2.38 2.25 2.38 1.75 1.2 ]
 [1.8  2.   2.25 2.12 2.   1.88 1.6 ]
 [3.   2.88 2.38 2.38 2.88 2.25 2.8 ]
 [3.67 2.8  2.4  2.   2.2  2.8  2.  ]]
```

3. С помощью функции `np.tril` вернем копию массива с обнуленными элементами ниже главной диагонали и найдем их сумму.

```
In 378 1 task_matrix_null = np.tril(task_matrix, -1)
2 sum_elements = np.sum(task_matrix_null)
3 print("Сглаженная матрица с обнуленными значениями выше главной диагонали: ")
4 print(task_matrix_null)
5 print("Сумма элементов: ", sum_elements)
```

Сглаженная матрица с обнуленными значениями выше главной диагонали:

```
[[0.  0.  0.  0.  0.  0.  0. ]
 [1.4  0.  0.  0.  0.  0.  0. ]
 [2.   1.88 0.  0.  0.  0.  0. ]
 [1.8  2.   2.25 0.  0.  0.  0. ]
 [3.   2.88 2.38 2.38 0.  0.  0. ]
 [3.4  2.75 2.62 2.12 2.5  0.  0. ]
 [3.67 2.8  2.4  2.   2.2  2.8  0.  ]]
```

Сумма элементов: 51.23

Рисунок 2.6 – Результат выполнения задания

9. Зафиксируйте сделанные изменения в репозитории.

10. Создать ноутбук, в котором выполнить решение вычислительной задачи (например, задачи из области физики, экономики, математики, статистики и т. д.), условие которой предварительно необходимо согласовать с преподавателем.



## Индивидуальное задание - Вычислительная задача

Условие задачи:

В баллоне находится кислород при давлении 4 МПа и температуре 42° С. Определить плотность кислорода в баллоне.

Дано:

$$p = 4 \text{ МПа}$$

$$t = 42^\circ \text{ C}$$

$$\rho = ?$$

Решение задачи: Запишем уравнение Клапейрона-Менделеева:

$$pV = \frac{m}{M} \times RT$$

Поделим обе части уравнения на объем:

$$p = \frac{m}{MV} \times RT$$

Отношение массы газа  $m$  к объему газа  $V$  в правой части есть плотность газа  $\rho$ , поэтому:

$$p = \frac{\rho}{M} \times RT$$

Выразим искомую плотность  $\rho$ :

$$\rho = \frac{pM}{RT}$$

Молярная масса кислорода  $M$  равна 0,032 кг/моль, универсальная газовая постоянная  $R$  – 8,31 Дж/(моль·К). Переведем температуру газа в шкалу абсолютных температур:

$$42^\circ \text{ C} = 315 \text{ K}$$

```
In [4]: # Вычисление
p = 4 |
t = 315
rho = 4*10**6*0.032/8.31/315
print(rho, "кг/м^3")

48.89882146199835 кг/м^3
```

Рисунок 2.7 – Результат выполнения задачи

11. Зафиксируйте сделанные изменения в репозитории.
12. Выполните слияние ветки для разработки с веткой main (master).
13. Отправьте сделанные изменения на сервер GitHub.

## 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
<b>argmax</b>	Индексы элементов с максимальным значением (по осям)
<b>argmin</b>	Индексы элементов с минимальным значением (по осям)
<b>max</b>	Максимальные значения элементов (по осям)
<b>min</b>	Минимальные значения элементов (по осям)
<b>mean</b>	Средние значения элементов (по осям)
<b>prod</b>	Произведение всех элементов (по осям)
<b>std</b>	Стандартное отклонение (по осям)
<b>sum</b>	Сумма всех элементов (по осям)
<b>var</b>	Дисперсия (по осям)

Вычислим некоторые из представленных выше статистик.

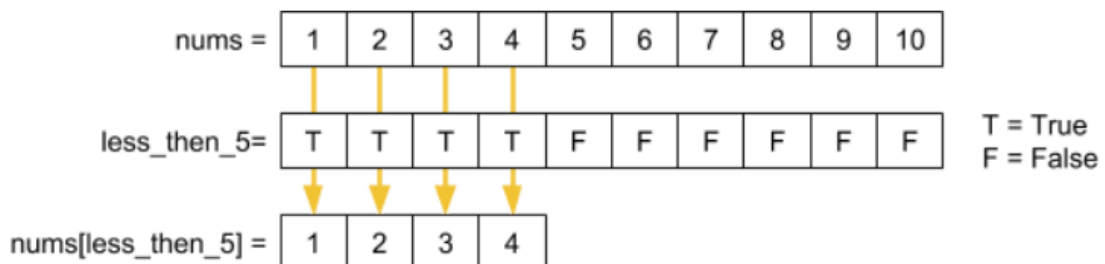
```
>>> m.mean()
4.833333333333333
>>> m.mean(axis=1)
matrix([[2.5],
        [6.5],
        [5.5]])
>>> m.sum()
58
>>> m.sum(axis=0)
matrix([[15,  9, 15, 19]])
```

## 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.