## Изучение библиотеки NumPy.

Библиотека NumPy предназначена для работы с многомерными массивами чисел и часто используется при работе с другими библиотеками (в частности matplotlib).

### Установка

Нужно запустить Anaconda prompt (через пуск-Anaconda3), там ввести команды:
`conda install numpy pillow scikit-image matplotlib -user`.

Если у вас не установлена Anaconda, то нужно запустить `cmd` и ввести:
`pip install numpy pillow scikit-image matplotlib -user`

### Документация

- Оригинальный tutorial: https://docs.scipy.org/doc/numpy-dev/user/quickstart.html
- Оглавление документации: https://docs.scipy.org/doc/numpy/reference/index.html
- Полный список команд: https://docs.scipy.org/doc/numpy/genindex.html
- Перевод некоторых частей: https://pythonworld.ru/numpy/1.html

### Замечания по вводу-выводу

Каждая программа должна начинаться со строк с импортированием NumPy и настройкой вывода:

```
import numpy as np
np.set_printoptions(precision=4, threshold=np.nan, linewidth=np.nan, suppress=True)
```

Если на вход в задаче даётся хотя бы один массив, то получать все данные необходимо из файла `input.txt` при помощи команды вида:

```
ar = eval(open('input.txt').read())
N, K, ar = eval(open('input.txt').read())
ar1, ar2 = eval(open('input.txt').read())
```

При выводе массивов всегда используйте функцию `repr`.

### Создание массива

```
# Одномерный массив нулей
> > > np.zeros(5)
array([ 0., 0., 0., 0., 0.])
# Двумерный массив нулей
> > > np.zeros((2, 3))
array([[ 0., 0., 0.],
       [ 0., 0., 0.]])
# Трёхмерный массив единиц
> > > np.ones((2,3,4))
array([[[ 1., 1., 1., 1.],
       [ 1., 1., 1., 1.],
       [ 1., 1., 1., 1.]],

       [[ 1., 1., 1., 1.],
       [ 1., 1., 1., 1.],
       [ 1., 1., 1., 1.]]])
# Массив нулей с указанием типа
> > > np.zeros(5, dtype=np.int)
array([0, 0, 0, 0, 0])
# Массив на основе списка
> > > np.array([2,3,1,0])
array([2, 3, 1, 0])
# Массив на основе списка списков
```

```
> > > np.array([[1,2.0],[0,0],(1+1j,3.)])
array([[ 1.+0.j, 2.+0.j],
       [ 0.+0.j, 0.+0.j],
       [ 1.+1.j, 3.+0.j]])
# Арифметическая прогрессия
> > > np.arange(10)
array([0, 1, 2, 3, 4, 5, 6, 7, 8, 9])
# Арифметическая прогрессия с указанием типа
> > > np.arange(2, 10, dtype=np.float)
array([ 2., 3., 4., 5., 6., 7., 8., 9.])
# Арифметическая прогрессия с нецелой разностью
> > > np.arange(2, 3, 0.1)
array([ 2. , 2.1, 2.2, 2.3, 2.4, 2.5, 2.6, 2.7, 2.8, 2.9])
# Арифметическая прогрессия с заданным количеством членов
> > > np.linspace(1., 4., 6)
array([ 1. , 1.6, 2.2, 2.8, 3.4, 4. ])
```

A. *Массив нулей*

На вход даётся число $N$. Выведите массив нулей из $N$ строк и $2N$ столбцов.

Если правильный массив хранится в переменной `x`, то используйте для вывода команду

```
print(repr(x))
```

| Input | Output |
|---|---|
| 3 | array([[ 0., 0., 0., 0., 0., 0.],<br>[ 0., 0., 0., 0., 0., 0.],<br>[ 0., 0., 0., 0., 0., 0.]]) |

---

**Обращение по индексам**

```
> > > x = np.arange(12).reshape(3, 4)
array([[ 0, 1, 2, 3],
       [ 4, 5, 6, 7],
       [ 8, 9, 10, 11]])
# Координаты указываем через запятую
> > > x[1, 2]
6
```

---

B. *Числа в нулевой строке*

На вход даются числа $N$ и $M$. Выведите массив размера $N \times M$, в котором в первой строчке (строка с нулевым индексом) расположены числа от 0 до $M - 1$, а остальные числа равны 0. Тип элементов массива должен быть `np.int8`.

| Input | Output |
|---|---|
| 3 5 | array([[0, 1, 2, 3, 4],<br>[0, 0, 0, 0, 0],<br>[0, 0, 0, 0, 0]], dtype=int8) |

C. *Числа на диагонали*

На вход даётся число $N$. Выведите массив размера $N \times N$, в котором по диагонали расположены числа от 0 до $N - 1$. Тип элементов массива должен быть `np.int64`.

| Input | Output |
|---|---|
| 4 | array([[0, 0, 0, 0],<br>[0, 1, 0, 0],<br>[0, 0, 2, 0],<br>[0, 0, 0, 3]], dtype=int64) |

---

**Срезы**

```
В решениях этих задач необходимо обойтись без циклов.
> > > x = np.arange(12).reshape(3, 4)
array([[ 0, 1, 2, 3],
       [ 4, 5, 6, 7],
       [ 8, 9, 10, 11]])
# Любая координата может быть срезом (:, a:b, ::-1, a:b:c)
# Строка с индексом 1
> > > x[1, :]
array([4, 5, 6, 7])
# Столбец и индексом 2
> > > x[:, 2]
array([ 2, 6, 10])
# Строка в обратном порядке
> > > x[0, ::-1]
array([3, 2, 1, 0])
```

---

D. *Сбитый прицел*

На вход даются числа $N, R, C$. Выведите массив размера $N \times N$, в котором в строке $R$ и столбце $C$ стоят 1, а остальные числа равны 0.

Тип элементов массива должен быть таким, чтобы во-первых, числа выводились как целые, без точек. А во-вторых, чтобы при выводе при помощи `repr` конкретный тип не указывался. В документации можно увидеть список возможных типов.

| Input | Output |
|---|---|
| 5 1 3 | array([[0, 0, 0, 1, 0],<br>[1, 1, 1, 1, 1],<br>[0, 0, 0, 1, 0],<br>[0, 0, 0, 1, 0],<br>[0, 0, 0, 1, 0]]) |

E. *Почётные единицы*

На вход даётся число $N$. Выведите массив размера $N \times N$, в котором в строках с чётными индексами стоят 1, а в остальных — нули.

| Input | Output |
|---|---|
| 5 | array([[1, 1, 1, 1, 1],<br>[0, 0, 0, 0, 0],<br>[1, 1, 1, 1, 1],<br>[0, 0, 0, 0, 0],<br>[1, 1, 1, 1, 1]]) |

F. *Решето*

На вход даются числа $N, M, R, C$. Выведите массив размера $N \times M$, в котором в каждой $R$-ой строчке и в каждом $C$-ом столбце стоят нули, а остальные элементы равны 1.

| Input | Output |
|---|---|
| 5 11 3 4 | array([[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0],<br>[0, 1, 1, 1, 0, 1, 1, 1, 0, 1, 1],<br>[0, 1, 1, 1, 0, 1, 1, 1, 0, 1, 1],<br>[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0],<br>[0, 1, 1, 1, 0, 1, 1, 1, 0, 1, 1]]) |

G. *Шахматные единицы*

На вход даётся число $N$. Выведите массив размера $N \times N$, имеющий вид шахматной доски. В верхнем левом углу должна стоять единица.

| Input | Output |
|---|---|
| 5 | array([[1, 0, 1, 0, 1],<br>[0, 1, 0, 1, 0],<br>[1, 0, 1, 0, 1],<br>[0, 1, 0, 1, 0],<br>[1, 0, 1, 0, 1]]) |

**Использование списка индексов в срезах**

```
В решениях этих задач необходимо обойтись без циклов.
> > > x = np.arange(12).reshape(3, 4)
array([[ 0, 1, 2, 3],
       [ 4, 5, 6, 7],
       [ 8, 9, 10, 11]])
# Вместо конкретного индекса можно передать набор индексов
> > > x[:, [1, 3]]
array([[ 1, 3],
       [ 5, 7],
       [ 9, 11]])
# Все передаваемые наборы должны быть одинаковой длины
> > > x[[0, 1], [1, 3]]
array([1, 7])
> > > x[np.arange(3), np.arange(3)]
array([ 0, 5, 10])
```

---

H. *Разлиновка*

На вход даётся число $N$. В следующей строке записано несколько целых неотрицательных чисел. Выведите массив размера $N \times N$, в котором в строках с перечисленными выше индексами стоят 1.

| Input | Output |
|---|---|
| 10<br>3 4 7 9 | array([[0, 0, 0, 0, 0, 0, 0, 0, 0, 0],<br>[0, 0, 0, 0, 0, 0, 0, 0, 0, 0],<br>[0, 0, 0, 0, 0, 0, 0, 0, 0, 0],<br>[1, 1, 1, 1, 1, 1, 1, 1, 1, 1],<br>[1, 1, 1, 1, 1, 1, 1, 1, 1, 1],<br>[0, 0, 0, 0, 0, 0, 0, 0, 0, 0],<br>[0, 0, 0, 0, 0, 0, 0, 0, 0, 0],<br>[1, 1, 1, 1, 1, 1, 1, 1, 1, 1],<br>[0, 0, 0, 0, 0, 0, 0, 0, 0, 0],<br>[1, 1, 1, 1, 1, 1, 1, 1, 1, 1]]) |

---

**Векторные операции (внимание — непросто!)**

```
В решениях этих задач необходимо обойтись без циклов.
> > > x = np.arange(12).reshape(2, 6)
# Можно ко всем элементам массива прибавлять числа и другие массивы
»> x + 10
array([[10, 11, 12, 13, 14, 15],
       [16, 17, 18, 19, 20, 21]])
# Можно все элементы массива умножать на числа или поэлементно на другие массивы
»> x * 3
array([[ 0, 3, 6, 9, 12, 15],
       [18, 21, 24, 27, 30, 33]])
# Если формы массивов не одинаковые, то происходит "размножение"
# вдоль координат с размерностью 1
# Например к массиву формы (2, 6) прибавляем массив формы (6) = (1, 6).
# Вдоль координаты 1 (то есть построчно) происходит размножение
> > > x + np.array([0, 10, 100, -10, -100, 0])
array([[ 0, 11, 102, -7, -96, 5],
 [ 6, 17, 108, -1, -90, 11]])
# Например к массиву формы (2, 6) прибавляем массив формы (2, 1)
# Вдоль координаты 1 (то есть по столбцам) происходит размножение
> > > x + np.array([[0], [-100]])
array([[ 0, 1, 2, 3, 4, 5],
       [-94, -93, -92, -91, -90, -89]])
```

```
> > > x + np.array([0, -100]).reshape((2, 1))
array([[ 0,  1,  2,  3,  4,  5],
       [-94, -93, -92, -91, -90, -89]])
# Это самое размножение называется broadcasting, тема достаточно сложная.
# Если вы ничего не поняли, то можно для начала запомнить, что к массиву формы
# (m, n) можно построчно прибавить массив формы (1, n)
# или постолбцово прибавить массив формы (m, 1)
# Можно ко всем элементам массива применять различные функции, в том числе логические
> > > np.sqrt(np.abs(x))
array([[ 0.  ,  1.  ,  1.4142,  1.7321,  2.  ,  2.2361],
       [ 2.4495,  2.6458,  2.8284,  3.  ,  3.1623,  3.3166]])
 > > > x > 3
array([[False, False, False, False,  True,  True],
       [ True,  True,  True,  True,  True,  True]], dtype=bool)
> > > (x > 3) & (x < 8)
array([[False, False, False, False,  True,  True],
       [ True,  True, False, False, False, False]], dtype=bool)
```

---

I. *Скалярное произведение*

На вход даются два массива векторов $v_i$ и $w_i$. Вычислите их скалярные произведения $(v_i, w_i)$.

| Input | Output |
|---|---|
| (np.array([[ 9, -3],<br>[ 2, -10],<br>[ 7, 8]]),<br>np.array([[-5, 2],<br>[ 7, -3],<br>[ 2, 1]])) | array([-51, 44, 22]) |

J. *Нормировка*

Дан массив действительных чисел. Сделайте его нормировку $X \to aX + b$ так, чтобы все числа лежали на отрезке $[0, 1]$ и значения 0 и 1 принимались. Гарантируется, что в массиве есть хотя бы два различных числа.

| Input | Output |
|---|---|
| np.array([[ 0., -8.],<br>[ 9., -4.],<br>[ 1., 4.]]) | array([[ 0.4706, 0. ],<br>[ 1. , 0.2353],<br>[ 0.5294, 0.7059]]) |

K. *Основное тригонометрическое тождество*

Дан массив действительных чисел $x_i$ длины $N$.

Выведите массивы: $\sin x_i, \cos x_i, \sin^2 x_i, \cos^2 x_i, \sin^2 x_i + \cos^2 x_i$.

| Input |
|---|
| np.array([ 5.6594, 14.8314, 11.1459, 5.9925, -9.2026]) |

| Output |
|---|
| array([-0.5841, 0.7686, -0.9887, -0.2866, -0.2204])<br>array([ 0.8117, -0.6398, 0.1497, 0.9581, -0.9754])<br>array([ 0.3412, 0.5907, 0.9776, 0.0821, 0.0486])<br>array([ 0.6588, 0.4093, 0.0224, 0.9179, 0.9514])<br>array([ 1., 1., 1., 1., 1.]) |

L. *Квадратные уравнения*

Даны три массива действительных чисел $a_i, b_i, c_i$ длины $N$.

Выведите два массива размера $N$ корней квадратных уравнений $a_i x^2 + b_i x + c_i$.

Гарантируется, что у каждого уравнения есть два корня. Сначала должны выводиться корни, в которых вычитается корень из дискриминанта.

| Input | Output |
|---|---|
| (np.array([-5, -1, 4, -2, 1]),<br>np.array([70, 8, -8, 24, -9]),<br>np.array([-225, 9, -192, -54, 14])) | array([ 9., 9., -6., 9., 2.])<br>array([ 5., -1., 8., 3., 7.]) |

### Использование массивов bool в срезах

```
В решениях этих задач необходимо обойтись без циклов.
> > > x = np.arange(12).reshape(2, 6)
# Вместо индекса можно передать массив bool-ов соответствующей длины: брать/не брать
> > > x[np.array([True, False]), :]
array([[0, 1, 2, 3, 4, 5]])
> > > x[:, np.array([True, True, False, False, False, True])]
array([[ 0, 1, 5],
       [ 6, 7, 11]])
# Соответствующие массивы bool-ов можно получать автоматически
> > > x > 3
array([[False, False, False, False, True, True],
       [ True, True, True, True, True, True]], dtype=bool)
> > > x[x > 3]
array([ 4, 5, 6, 7, 8, 9, 10, 11])
# Можно использовать логические связки &=and и |=or, но нужно не забывать про скобки
> > > (x > 3) & (x < 8)
array([[False, False, False, False, True, True],
       [ True, True, False, False, False, False]], dtype=bool)
> > > x[(x > 3) & (x < 8)]
array([4, 5, 6, 7])
```

M. *Больше нуля*

Дан массив действительных чисел. Выведите массив его положительных элементов.

| Input | Output |
|---|---|
| `np.array([-5, 1, 7, -8, -3, 9, -4, 6, -2, -9])` | `array([1, 7, 9, 6])` |

N. *Поменять знак*

Дан массив действительных чисел. Поменяйте знак у элементов, значения которых между 3 и 8.

| Input |
|---|
| `np.array([ -8, -6, -3, 8, -4, -10, -9, -7, -3, 6])` |

| Output |
|---|
| `array([ -8, -6, -3, -8, -4, -10, -9, -7, -3, -6])` |

### Максимумы и минимумы

```
> > > x = np.arange(12).reshape(2, 6)
# ... а также индексы соответствующих элементов
»> x = np.random.randint(0, 10, 10)
»> x
array([8, 2, 8, 3, 4, 8, 9, 9, 3, 1])
»> x.max(), x.min(), x.argmax(), x.argmin()
(9, 1, 6, 9)
# Если массив многомерный, то argmax и argmin возвращают
# не координату в массиве, а номер по порядку
> > > x = np.random.randint(0, 10, (2, 6))
> > > x
array([[1, 4, 9, 6, 6, 7],
       [9, 8, 9, 5, 2, 7]])
> > > x.argmax()
2
# Но его можно превратить в координату в массиве
```

```
> > > np.unravel_index(x.argmax(), x.shape)
(0, 2)
# Также найти все наборы координат по заданному правилу
> > > np.argwhere(x==x.max())
array([[0, 2],
       [1, 0],
       [1, 2]], dtype=int64)
```

---

O. *Заменить все максимумы*

Дан массив действительных чисел. Замените все значения, равные максимальному, на $-1$.

| Input |
|---|
| `np.array([4, 8, 7, 5, 1, 6, 1, 1, 8, 5])` |

| Output |
|---|
| `array([ 4, -1,  7,  5,  1,  6,  1,  1, -1,  5])` |

P. *Ближайшее*

Дан массив действительных чисел и некоторое число. Найдите ближайшее по модулю к этому числу значение в массиве и его индекс.

В решении нужно обойтись без циклов.

| Input | Output |
|---|---|
| `(np.array([ 0.91,  0.55,  0.87,  0.52,  0.34,  0.69,  0.74]), 0.5)` | `0.52`<br>`3` |

Q. *Построчный и постолбцовый максимум*

Дан прямоугольный массив действительных чисел.

Выведите массив максимальных значений в каждой строчке, а за ним массив максимальных значений в каждом столбце.

| Input | Output |
|---|---|
| `np.array([[80, 85, 61],`<br>`[57, 41, 22]])` | `array([85, 57])`<br>`array([80, 85, 61])` |

R. *Среднее, построчное и постолбцовое среднее*

Дан прямоугольный массив действительных чисел `X`.

Выведите три массива: массив X, у которого из каждого элемента вычли:

- среднее арифметическое всех чисел массива `X`
- среднее арифметическое всех чисел данной строчки массива `X`
- среднее арифметическое всех чисел данного столбца массива `X`

Выведите среднее арифметическое чисел массива, массив средний значений в каждой строчке, а за ним массив средний значений в каждом столбце.

| Input | Output |
|---|---|
| `np.array([[7, 3, 2],`<br>`[6, 8, 4]])` | `array([[ 2., -2., -3.],`<br>`[ 1.,  3., -1.]])`<br>`array([[ 3., -1., -2.],`<br>`[ 0.,  2., -2.]])`<br>`array([[ 0.5, -2.5, -1. ],`<br>`[-0.5,  2.5,  1. ]])` |

---

**Сортировка**

```
> > > a = np.array([[1,4],[3,1]])
array([[1, 4],
       [3, 1]])
> > > np.sort(a) # Сортировка вдоль последней оси (построчно)
array([[1, 4],
```

```
       [1, 3]])
> > > np.sort(a, axis=None) # Сортировка "выпрямленного"массива
array([1, 1, 3, 4])
> > > np.sort(a, axis=0) # Сортировка вдоль первой оси (по столбцам)
array([[1, 1],
       [3, 4]])
> > > x = np.array([3, 1, 2])
> > > np.argsort(x)
array([1, 2, 0], dtype=int64)
> > > x[np.argsort(x)]
array([1, 2, 3])
> > > x = np.array([[3, 1, 2], [0, 1, 2], [10, 11, 12]])
> > > x
array([[ 3, 1, 2],
       [ 0, 1, 2],
       [10, 11, 12]])
> > > x[:, np.argsort(x[0, :])]
array([[ 1, 2, 3],
       [ 1, 2, 0],
       [11, 12, 10]])
> > > x[np.argsort(x[:, 0]), :]
array([[ 0, 1, 2],
       [ 3, 1, 2],
       [10, 11, 12]])
```

---

S. *Сортировка*

Дан линейный массив действительных чисел `X`. Отсортируйте его.

| Input | Output |
|---|---|
| `np.array([1, 9, 2, 8, 3, 7])` | `array([1, 2, 3, 7, 8, 9])` |

T. *argsort*

Дан линейный массив действительных чисел $X$. Выведите массив индексов, обладающих следующим свойством: в позиции $i$ стоит индекс элемента массива $X$, который окажется $i$-ым, если массив отсортировать.

| Input | Output |
|---|---|
| `np.array([1, 9, 2, 8, 3, 7])` | `array([0, 2, 4, 5, 3, 1])` |

U. *argsort — 2*

Даны три линейных массива одинаковой длины. В первом массиве хранятся стоимости брусков, во втором — длины, а в третьем — массы. Отсортируйте массивы так, чтобы данные по брускам шли в порядке возрастания стоимости.

| Input |
|---|
| `(np.array([1046, 1267, 1044, 1875, 1620, 1412]),`<br>` np.array([418, 477, 240, 290, 272, 211]),`<br>` np.array([ 3.5002,  1.3224,  1.2677,  3.9586,  1.0951,  1.144 ]))` |

| Output |
|---|
| `array([1044, 1046, 1267, 1412, 1620, 1875])`<br>`array([240, 418, 477, 211, 272, 290])`<br>`array([ 1.2677,  3.5002,  1.3224,  1.144 ,  1.0951,  3.9586])` |

V. *Порядок индексов*

Даны числа $W$ и $H$ — ширина и высота картинки в пикселях.

Создайте подходящий массив, в котором можно хранить трёхцветную картинку данной ширины и высоты. При этом данные в памяти должны храниться так, чтобы сначала шли три цвета одного пикселя, затем цвета следующего пикселя в этой же строчке или первый пиксель из

следующей строчки. Каждый элемент должен быть однобайтным беззнаковым целым числом (`np.uint8`).

| Input | Output |
|---|---|
| 3 3 | `array([[[0, 0, 0],`<br>`[0, 0, 0],`<br>`[0, 0, 0]],`<br>`[[0, 0, 0],`<br>`[0, 0, 0],`<br>`[0, 0, 0]],`<br>`[[0, 0, 0],`<br>`[0, 0, 0],`<br>`[0, 0, 0]]], dtype=uint8)` |

W. *Размер картинки*

В файле `input.png` сохранена картинка. Прочитайте её при помощи команды вида:

```
import matplotlib
matplotlib.use('Agg')
from skimage import io
img = io.imread('input.png')
```

Картинка может быть чёрно-белой или цветной. Выведите её размеры: сначала ширину в пикселях, а затем высоту.

Пример входного изображения 1

```
Ответ:
30
60
```

Пример входного изображения 2

```
Ответ:
30
60
```

X. *Зелёный слой*

Из файла `input.png` прочитайте цветную картинку. Извлеките из неё зелёный слой и в виде ч/б изображения, то есть двумерного массива, сохраните в файл output.png.

Пример входного изображения

Результат

Y. *Состыковка — 1*

Из файла `input.png` прочитайте ч/б картинку. В файл output.png запишите картинку, полученную из двух таких картинок состыковкой «одна над другой».

Пример входного изображения

Результат

Z. *Состыковка — 2*

Из файла `input.png` прочитайте ч/б картинку. В файл output.png запишите картинку, полученную из двух таких картинок состыковкой «плечо к плечу».

Пример входного изображения

Результат

ZA. *Удаление — 1*

Из файла `input.png` прочитайте ч/б картинку. В файл `output.png` запишите картинку, полученную из исходной удалением по центру вертикальной полосы шириной в 10 пикселей.

Гарантируется, что ширина картинки не меньше 12 пикселей.

Пример входного изображения

Результат

ZB. *Удаление — 2*

Из файла `input.png` прочитайте цветную картинку. В файл `output.png` запишите картинку, полученную из исходной удалением по центру горизонтальной полосы высотой в 20 пикселей, затем удалением вертикальной полосы шириной в 10 пикселей от $\frac{3}{4}W - 5$ до $\frac{3}{4}W + 4$ включительно ($W$ — ширина картинки).

Гарантируется, что высота картинки чётная и не меньше 22 пикселей, ширина не меньше 12 пикселей, а также то, что $W$ делится на 4.

Пример входного изображения

Результат

ZC. *Вставка — 1*

Из файла `input.png` прочитайте ч/б картинку. В файл `output.png` запишите картинку, полученную из исходной вставкой по центру белой вертикальной полосы шириной в 10 пикселей.

Пример входного изображения

Результат

ZD. *Вставка — 2*

Из файла `input.png` прочитайте цветную картинку. В файл `output.png` запишите картинку, полученную из исходной вставкой по центру зелёной горизонтальной полосы высотой в 10 пикселей, затем вставкой красной вертикальной полосы шириной в 17 пикселей так, чтобы пространство слева было в два раз шире пространства справа.

Пример входного изображения

Результат