Домашнее задание 2 (5 баллов)

Все задания ниже имеют равный вес (5/10).

О задании

В этом домашнем задании вы попрактикуетесь в работе с библиотекой numpy, которая позволяет сравнительно легко и удобно выполнять разнообразные вычисления, избегая самостоятельной реализации поэлементной обработки.

Во всех задачах необходимо написать код решения внутри функции и убедиться, что она работает, с помощью assert (https://python-reference.readthedocs.io/en/latest/docs/statements/assert.html) на выражение с использованием этой функции для данных из условия.

При решении задач запрещается использовать циклы (`for` , `while`) и оператор `if` .

Везде, где встречаются массивы или матрицы, подразумевается, что это `numpy.array` .

numpy reference: https://numpy.org/doc/stable/reference/index.html (https://numpy.org/doc/stable/reference/index.html)

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

Задание 1

Напишите функцию, возвращающую округленную взвешенную сумму оценок по данным оценкам и весам. Можете посчитать свою оценку за курс :) В нашем случае вес экзамена равен 0.3, вес домашек - 0.4, вес контрольной - 0.2, вес самостоятельных - 0.1. Например, если за экзамен у вас 7, за домашки 10, за контрольную 8, а за самостоятельные 6, то вы получите отличную оценку 8!

In [2]:
```
def result_mark(weights: np.array, marks: np.array) -> int:
    return int(np.dot(weights, marks))
```

In [3]:
```
weights = np.array([0.3, 0.4, 0.2, 0.1])
marks = np.array([7, 10, 8, 6])

assert result_mark(weights, marks) == 8
```

In [4]:
```
weights = np.array([0.3, 0.4, 0.2, 0.1])
marks = np.array([7, 0, 8, 6])

assert result_mark(weights, marks) == 4
```

Задание 2

Напишите функцию, меняющую каждое третье (начиная с 0) значение массива целых чисел на заданное число. Например, если на вход поступает массив `array([3, 5, 1, 0, -3, 22, 213436])` и число `-111`, то на выходе должен получиться массив `array([-111, 5, 1, -111, -3, 22, -111])`.

In [5]:
```
def change_array(array: np.array, number: int) -> np.array:
    array[::3] = number
    return array
```

In [6]:
```
array = np.array([3, 5, 1, 0, -3, 22, 213436])
number = -111

assert np.allclose(change_array(array, number), np.array([-111, 5, 1, -111, -3, 2
```

In [7]:
```
array = np.array([3, 14, 15, 92, 6])
number = 8

assert np.allclose(change_array(array, number), np.array([8, 14, 15, 8, 6]))
```

Задание 3

Напишите функцию, выдающую индексы «близких» элементов заданных массивов, а именно тех пар элементов, чей модуль разницы не превосходит заданного значения. Например, если на вход поступают массив `array([1.5, 0.5, 2, -4.1, -3, 6, -1])`, массив `array([1.2, 0.5, 1, -4, 3, 0, -1.2])` и число `0.5`, то на выходе должен получиться массив `array([0, 1, 3, 6])` (**важно: не `tuple`, а одномерный массив типа `numpy.ndarray` (то есть `.ndim` от него равно 1)!**).

In [8]:
```
def find_close(array1: np.array, array2: np.array,
               precision: float) -> np.array:
    res = array1 - array2
    res = np.abs(res)
    res = res <= precision
    return np.arange(len(array1))[res]
```

In [9]:
```
array1 = np.array([1.5, 0.5, 2, -4.1, -3, 6, -1])
array2 = np.array([1.2, 0.5, 1, -4.0,  3, 0, -1.2])
precision = 0.5
res = find_close(array1, array2, precision)

assert res.ndim == 1
assert np.allclose(res, np.array([0, 1, 3, 6]))
```

In [10]:
```
array1 = np.array([3.1415, 2.7182, 1.6180, 6.6261])
array2 = np.array([6.6730, 1.3807, -1,     6.0222])
precision = 1.7
res = find_close(array1, array2, precision)

assert res.ndim == 1
assert np.allclose(res, np.array([1, 3]))
```

Задание 4

Напишите функцию, которая составляет блочную матрицу из четырех блоков, где каждый блок - это заданная матрица. Например, если на вход поступает матрица

$$\begin{pmatrix} 0 & 1 & 2 \\ 3 & 4 & 5 \end{pmatrix},$$

то ответом будет матрица

$$\begin{pmatrix} 0 & 1 & 2 & 0 & 1 & 2 \\ 3 & 4 & 5 & 3 & 4 & 5 \\ 0 & 1 & 2 & 0 & 1 & 2 \\ 3 & 4 & 5 & 3 & 4 & 5 \end{pmatrix}$$

In [11]:
```
def block_matrix(block: np.array) -> np.array:
    n, m = block.shape
    matrix = np.ones((2*n, 2*m))
    matrix[:n, :m] = block
    matrix[n:, m:] = block
    matrix[:n, m:] = block
    matrix[n:, :m] = block
    return matrix
```

In [12]:
```
block = np.array([[1, 3, 3], [7, 0, 0]])

assert np.allclose(
    block_matrix(block),
    np.array([[1, 3, 3, 1, 3, 3],
              [7, 0, 0, 7, 0, 0],
              [1, 3, 3, 1, 3, 3],
              [7, 0, 0, 7, 0, 0]])
)
```

Задание 5

Напишите функцию, вычисляющую произведение всех ненулевых диагональных элементов на диагонали данной квадратной матрицы. Например, если на вход поступает матрица

$$\begin{pmatrix} 0 & 1 & 2 \\ 3 & 4 & 5 \\ 6 & 7 & 8 \end{pmatrix},$$

то ответом будет 32.

Элементы матрицы считать целочисленными.

In [13]:
```
def diag_prod(matrix: np.array) -> int:
    diag = np.diag(matrix)
    no_zeros_diag = diag != 0
    return diag[no_zeros_diag].prod()
```

In [14]:
```
matrix = np.array([[0, 1, 2, 3],
                   [4, 5, 6, 7],
                   [8, 9, 10, 11],
                   [12, 13, 14, 15]])

assert diag_prod(matrix) == 750
```

Задание 6

Для улучшения качества работы некоторых алгоритмов машинного обучения может быть полезно использовать [нормализацию данных (https://vk.cc/8xmfQk)](), чтобы привести признаки в выборке к одному масштабу — а именно, из каждого столбца вычесть среднее его значений и поделить на их стандартное отклонение. Напишите функцию, нормализующую входящую матрицу (по столбцам). Например, если на вход подается матрица

$$\begin{pmatrix} 1 & 4 & 4200 \\ 0 & 10 & 5000 \\ 1 & 2 & 1000 \end{pmatrix},$$

то результатом с точностью до сотых будет матрица

$$\begin{pmatrix} 0.71 & -0.39 & 0.46 \\ -1.41 & 1.37 & 0.93 \\ 0.71 & -0.98 & -1.39 \end{pmatrix}$$

Учтите, что в вашем матрице не должно получаться никаких nan. Подумайте, в каком случае они могут возникнуть и как обойти эту проблему.

Подсказка. Казалось бы, при чем тут деление на ноль.

In [15]:
```
def normalize(matrix: np.array) -> np.array:
    std = np.std(matrix, axis = 0) #ошибка деления на ноль возникает <=> есть сто
    std = np.where(std == 0, 1, std)
    return (matrix - matrix.mean(axis = 0)) / std
```

In [16]:
```
matrix = np.array([[1, 4, 4200], [0, 10, 5000], [1, 2, 1000]])

assert np.allclose(
    normalize(matrix),
    np.array([[ 0.7071, -0.39223,  0.46291],
              [-1.4142,  1.37281,  0.92582],
              [ 0.7071, -0.98058, -1.38873]])
)
```

In [17]:
```
matrix = np.array([[-7, 2, 42], [2, 10, 50], [5, 4, 10]])

assert np.allclose(
    normalize(matrix),
    np.array([[-1.37281, -0.98058,  0.46291],
              [ 0.39223,  1.37281,  0.92582],
              [ 0.98058, -0.39223, -1.38873]])
)
```

In [18]:
```
#Тест для нулевого столбца

matrix = np.array([[0, 4, 4200], [0, 10, 5000], [0, 2, 1000]])

assert np.allclose(
    normalize(matrix),
    np.array([[ 0, -0.39223,  0.46291],
              [0,  1.37281,  0.92582],
              [ 0, -0.98058, -1.38873]])
)
```

Задание 7

Напишите функцию, вычисляющую какую-нибудь первообразную данного полинома (в качестве константы возьмите 0). Например, если на вход поступает массив коэффициентов `array([4, 6, 0, 1])`, что соответствует полиному $4x^3 + 6x^2 + 1$, на выходе получается массив коэффициентов `array([1, 2, 0, 1, -2])`, соответствующий полиному $x^4 + 2x^3 + x - 2$.

In [19]:
```
def antiderivative(coefs: np.array) -> np.array:
    power = np.arange(len(coefs))
    power = power[::-1]
    power += 1
    ans = coefs / power
    return(np.hstack([ans, 0]))
```

In [20]:
```
coefs = np.array([4, 6, 0, 1])

assert np.allclose(
    antiderivative(coefs),
    np.array([1., 2., 0., 1., 0.])
)
```

In [21]:
```
coefs = np.array([1, 7, -12, 21, -6])

assert np.allclose(
    antiderivative(coefs),
    np.array([ 0.2, 1.75, -4., 10.5, -6., 0.])
)
```

Задание 8

Напишите функцию, делающую данную [треугольную матрицу (https://ru.wikipedia.org/wiki/%D0%A2%D1%80%D0%B5%D1%83%D0%B3%D0%BE%D0%BB%D](https://ru.wikipedia.org/wiki/%D0%A2%D1%80%D0%B5%D1%83%D0%B3%D0%BE%D0%BB%D) симметричной. Например, если на вход поступает матрица

$$\begin{pmatrix} 1 & 2 & 3 & 4 \\ 0 & 5 & 6 & 7 \\ 0 & 0 & 8 & 9 \\ 0 & 0 & 0 & 10 \end{pmatrix},$$

то на выходе должна быть матрица

$$\begin{pmatrix} 1 & 2 & 3 & 4 \\ 2 & 5 & 6 & 7 \\ 3 & 6 & 8 & 9 \\ 4 & 7 & 9 & 10 \end{pmatrix}.$$

In [22]:
```
def make_symmetric(matrix: np.array) -> np.array:
    diag = np.diag(np.diag(matrix))
    return matrix.T + matrix - diag
```

In [23]:
```
matrix = np.array([[1, 2, 3, 4], [0, 5, 6, 7], [0, 0, 8, 9], [0, 0, 0, 10]])

assert np.allclose(
    make_symmetric(matrix),
    np.array([[ 1,  2,  3,  4],
              [ 2,  5,  6,  7],
              [ 3,  6,  8,  9],
              [ 4,  7,  9, 10]])
)
```

In [24]:
```
matrix = np.array([[10, 21, 32, 49], [0, 53, 62, 78], [0, 0, 82, 92], [0, 0, 0, 1

assert np.allclose(
    make_symmetric(matrix),
    np.array([[10, 21, 32, 49],
              [21, 53, 62, 78],
              [32, 62, 82, 92],
              [49, 78, 92, 10]])
)
```

Задание 9

Напишите функцию, создающую прямоугольную матрицу из m одинаковых строк, заполненных последовательными натуральными числами от a до b включительно в возрастающем порядке. Например, если m = 5, a = 3, b = 10, то на выходе будет матрица

$$\begin{pmatrix} 3 & 4 & 5 & 6 & 7 & 8 & 9 & 10 \\ 3 & 4 & 5 & 6 & 7 & 8 & 9 & 10 \\ 3 & 4 & 5 & 6 & 7 & 8 & 9 & 10 \\ 3 & 4 & 5 & 6 & 7 & 8 & 9 & 10 \\ 3 & 4 & 5 & 6 & 7 & 8 & 9 & 10 \end{pmatrix}$$

In [25]:
```
def construct_matrix(m: int, a: int, b: int) -> np.array:
    string = np.arange(a, b + 1)
    matrix = np.ones((m, len(string)))
    return matrix * string
```

In [26]:
```
m = 5
a = 3
b = 10

assert np.allclose(
    construct_matrix(m, a, b),
    np.array([[ 3,  4,  5,  6,  7,  8,  9, 10],
              [ 3,  4,  5,  6,  7,  8,  9, 10],
              [ 3,  4,  5,  6,  7,  8,  9, 10],
              [ 3,  4,  5,  6,  7,  8,  9, 10],
              [ 3,  4,  5,  6,  7,  8,  9, 10]])
)
```

In [27]:
```
m = 3
a = 2
b = 6

assert np.allclose(
    construct_matrix(m, a, b),
    np.array([[2, 3, 4, 5, 6],
              [2, 3, 4, 5, 6],
              [2, 3, 4, 5, 6]])
)
```

Задание 10

Напишите функцию, вычисляющую [косинусную близость (https://en.wikipedia.org/wiki/Cosine_similarity)](https://en.wikipedia.org/wiki/Cosine_similarity) двух векторов. Например, если на вход поступают вектора `array([-2, 1, 0, -5, 4, 3, -3])` и `array([0, 2, -2, 10, 6, 0, 0])` , ответом будет -0.25.

In [28]:
```
def cosine_similarity(vec1: np.array, vec2: np.array) -> float:
    norm_1 = np.sqrt(np.sum(vec1 ** 2))
    norm_2 = np.sqrt(np.sum(vec2 ** 2))
    return np.dot(vec1, vec2) / (norm_1 * norm_2)
```

In [29]:
```
vec1 = np.array([-2, 1,  0, -5, 4, 3, -3])
vec2 = np.array([ 0, 2, -2, 10, 6, 0,  0])

assert np.allclose(cosine_similarity(vec1, vec2), -0.25)
```

In [30]:
```
vec1 = np.array([-4, 2,  9, -8, 9, 0, -2])
vec2 = np.array([ 3, 2, -4, -1, 3, 2,  2])

assert np.allclose(cosine_similarity(vec1, vec2), -0.119929)
```