ЛАБОРАТОРНАЯ РАБОТА №1: РАБОТА С МАССИВАМИ В NUMPY

Основы. NumPy — это специальная библиотека для работы с многомерными массивами. Обычно это матрицы элементов (чаще всего чисел) одного типа, где используются целочисленные кортежи для индексации. Например, координаты точки в трехмерном пространстве [1, 2, 1] имеют только одну ось. Эта ось имеет три элемента, поэтому её длина равна 3. В приведенном ниже примере массив имеет две оси. Первая ось имеет размерность 2, а вторая ось имеет размерность 3.

Массивы в NumPy являются экземплярами класса ndarray. Их также называют псевдо-массивами. Следует помнить, что массивы в NumPy не относятся к классу array.array, который поддерживает только одномерные массивы и имеет меньше функциональности. Основные атрибуты ndarray показаны в таблице №1.

Таблица 1 – Основные атрибуты ndarray

Атрибут	Описание
ndarray.ndim	Количество осей (измерений) массива.
ndarray.shape	Кортеж, описывающий размеры массива по каждой оси.
ndarray.size	Общее количество элементов в массиве.
ndarray.dtype	Тип данных элементов массива.
ndarray.itemsize	Размер (в байтах) одного элемента массива.
ndarray.data	Буфер данных, содержащий элементы массива.

Пример кода на Python, который создаёт матрицу размером 3 на 5, содержащую числа от 1 до 15 с использованием библиотеки NumPy, показан на рисунке 1.

```
import numpy as np
a = np.arange(15).reshape(3, 5)
```

Рисунок 1 – Пример создания матрицы.

Задача 1. Используя приведённый пример на рисунке 1, создайте матрицу *a*, найдите её размерность, тип данных, размер элемента в матрице, размер матрицы и количество элементов. Создайте массив *b*, содержащий одну строку [6, 7, 8].

Создание массива. Существует несколько способов создания массивов. Например, вы можете создать массив, используя стандартные типы Python, такие как список или кортеж, с помощью функции `array`. Функция `array` преобразует последовательность последовательностей в двумерный массив и так далее. Вы можете указать тип данных в массиве при его создании (Рисунок 2).

```
c = np.array([[1, 2], [3, 4]], dtype = complex)
print(c)
```

Рисунок 2 – Создание массивов.

Чаще всего элементы массива неизвестны, но часто известен их тип. Поэтому в NumPy существует ряд функций для создания массивов с заранее заданными значениями по умолчанию. Для создания матрицы, содержащей нулевые значения, используется функция `zeros` (Рисунок 3). Аналогично, с помощью функции `ones` можно создать матрицу, содержащую единицы.

```
a = np.zeros((3, 4))
b = np.zeros((3, 4), dtype = np.int16)
```

Рисунок 3– Массив заполненный нулевыми значенями.

Для создания последовательности чисел в NumPy есть функция `range`, которая возвращает массив, а не список (Рисунок 4).

```
c = np.arange(10, 30, 5)
print(c)
d = np.arange(0, 2, 0.3)
print(d)
```

Рисунок 4 – Создание последовательности чисел.

На практике, при работе с числами с плавающей точкой, чаще используется функция `linspace`, которая принимает количество нужных элементов в качестве аргументов вместо шага (Рисунок 5).

```
from numpy import pi
e = np.linspace(0, 2, 9)
x = np.linspace(0, 2*pi, 100)
f = np.sin(x)
print(e, x, f)
```

Рисунок 5 – Использование функции `linspace`.

При выводе массива NumPy использует следующий шаблон:

1) Значения печатаются слева направо и сверху вниз.

2) Для многомерных массивов каждый срез печатается отдельно.

Для настройки метода вывода используется функция `set_printoptions` библиотеки NumPy.

Задача 2. Напишите код, приведённый в разделе "Создание массива", в отдельном файле "myArray.py". Выведите все массивы на экран, сравните вывод двумерных и трёхмерных массивов. Независимо изучите работу функций `array`, `zeros`, `zeros_like`, `ones`, `ones_like`, `empty`, `empty_like`, `arange`, `linspace`, `numpy.random.rand`, `numpy.random.randn`, `fromfunction`, `fromfile`.

Основные операции. Арифметические операции над массивами выполняются поэлементно. Создаётся новый массив и заполняется вычисленными значениями (Рисунок 6). По умолчанию оператор `*` выполняет поэлементное умножение элементов матриц. Для выполнения умножения матриц в соответствии с правилами линейной алгебры используется функция `dot`.

```
a = np.array([20, 30, 40, 50])
b = np.arange(4)
c = a - b
d = b**2
f = 10*np.sin(a)
k = a<35
```

Рисунок 6 – Арифметические операции над массивами.

Операции такие как `+=` и `*=` не создают новый массив, а модифицируют исходный массив. При использовании этих операторов данные в массивах должны быть одного типа. Если вычисления выполняются с массивами разных типов данных, тип данных в результирующем массиве будет более точным из двух. Некоторые унарные операции, такие как вычисление суммы (`sum`), накопленной суммы (`cumsum`), минимума (`min`) или максимума (`max`) элементов в массиве, реализованы как методы объектов. По умолчанию эти функции работают со всем массивом, но их можно использовать для каждой отдельной оси (Рисунок 7).

```
b = np.arange(12).reshape(3, 4)
print(b)
s1 = b.sum(axis = 0)
print(s1)
s2 = b.sum(axis = 1)
print(s2)
```

Рисунок 7 – Некоторые унарные операции реализованы как методы объектов.

NumPy реализует математические функции, такие как `sin`, `cos` и `exp`. В NumPy они называются универсальными функциями. Эти функции выполняют операции над элементами массива поэлементно, возвращая другой массив.

Задача 3. Реализуйте код, приведённый в примерах в разделе «Основные операции». Для матрицы `*b*` найдите максимальный элемент всей матрицы, по строкам и по столбцам.

Индексы. Пример работы с массивами с использованием индексов и срезов (Рисунок 8).

```
a = np.arange(10)**3
print(a[2])
print(a[2:5])
a[:6:2] = -1000
print(a)
print(a[::-1])
for i in a:
  print(i**(1/3))
```

Рисунок 8 – Работа с массивами с использованием индексов.

Многомерные массивы могут иметь только один индекс для каждой оси. Эти индексы указываются в виде кортежа, разделённого запятыми (Рисунок 9).

```
b = np.arange(20, dtype = float).reshape(5, 4)
print(b[2, 3])
print(b[0:5, 1])
print(b[:, 1])
print(b[1:3, :])
```

Рисунок 9 – Пример индексации многомерного массива.

Если количество используемых индексов меньше размерности массива, недостающие индексы дополняются срезами (Рисунок 10).

```
print(b[-1]) # equals b[-1, :]
```

Рисунок 10 – Количество используемых индексов меньше размерности массива.

Выражение внутри скобок `b[i]` трактуется так, как если бы после `i` были перечислены столько срезов, сколько необходимо для описания оставшихся осей (Рисунок 11). Вместо `:` в NumPy можно также использовать троеточие `...` — `b[i, ...]`. Троеточие указывает на то, что следует добавить столько индексов, сколько нужно, чтобы описать все оси массива. Например, если массив `x` имеет пять осей, то:

1) `x[1,2,...]` эквивалентно `x[1,2,:,:,:]`
2) `x[...,3]` эквивалентно `x[:,:,:,:,:,3]`
3) `x[4,...,5,:]` эквивалентно `x[4,:,:,:,:,5,:]`.

```
c = np.array([[[ 0, 1, 2],[10,12, 13]], [[100, 101, 102], [110, 112, 113]]])
print(c)
print(c[1, ...]) # equals c[1, :, :] or c[1]
print(c[..., 2]) # equals c[:, :, 2]
```

Рисунок 11 – Триплеты.

Итерации начинаются с первого индекса. Однако, если требуется выполнять операции над каждым элементом, необходимо использовать атрибут `flat` (Рисунок 12).

```
for element in b.flat:
  print(element)
```

Рисунок 12 – Атрибут `flat`.

Задача 4: Напишите код, приведённый в примерах.

Преобразование формы матрицы. Массив имеет форму, определяемую количеством элементов на каждой оси (Рисунок 13).

```
a = np.floor(10*np.random.random((3, 4)))
print(a)
print(a.shape)
```

Рисунок 13 – Форма массива.

Существует несколько команд для изменения формы массива. Три команды, приведенные ниже, возвращают измененный массив, но не изменяют оригинальный массив (Рисунок 14).

```
print(a.ravel())
print(a.reshape(6, 2))
print(a.T)
print(a.T.shape)
print(a.shape)
```

Рисунок 14 – Команды для изменения формы массива.

Функция reshape возвращает другой массив с измененной формой, метод ndarray.resize изменяет сам массив. Если в качестве индекса указано значение -1, размерность вычисляется автоматически. Эти три команды возвращают измененный массив, но не изменяют оригинальный массив (Рисунок 15).

```
a = np.floor(10*np.random.random((3, 4)))
print(a)
a.resize((2, 6))
print(a)
a.reshape(3, -1)
print(a)
b = a.reshape(3, -1)
print(b)
```

Рисунок 15 – Функции изменения размера (resize) и изменения формы (reshape).

Задание 5. Реализуйте код, приведенный в примерах. Изучите работу функций ndarray.shape, reshape, resize, ravel.

Объединение и разбиение массива. Несколько массивов могут быть объединены вдоль одного из измерений (Рисунок 16).

```
a = np.floor(10*np.random.random((2, 2)))
b = np.floor(10*np.random.random((2, 2)))
v = np.vstack((a, b))
h = np.hstack((a, b))
print(v)
print(h)
```

Рисунок 16 – Объединенные вместе несколько массивов.

Функция column_stack объединяет два одномерных массива в один двумерный массив. Эта функция эквивалентна hstack, но только для двумерных массивов. Функция newaxis добавляет еще одно измерение к массиву (Рисунок 17). В общем случае для массивов размерностью более двух функция hstack объединяет массивы вдоль второго измерения. Функция

vstack объединяет вдоль первого измерения. Функция concatenate позволяет указать в качестве аргумента ось, вдоль которой будет происходить объединение. С помощью функции hsplit можно разделить массив вдоль горизонтальной оси, указав количество частей, на которые нужно разделить массив, или указав номера столбцов, после которых должно быть выполнено разделение.

Задание 6. Реализуйте код, приведенный в примерах. Изучите работу функций: hstack, vstack, column_stack, concatenate, c_, r_.

```
from numpy import newaxis
d = np.column_stack((a, b))
print(d, 'column_stack')
a = np.array([4., 2.])
b = np.array([3., 8.])
d = np.column_stack((a, b))
print(d)
d = np.hstack((a, b))
print(a[:, newaxis], 'newaxis')
print(np.column_stack((a[:, newaxis], b[:, newaxis])))
print(np.hstack((a[:, newaxis], b[:, newaxis])))
```

Рисунок 17 – Объединение двух одномерных массивов.

Копирование массива. При копировании массивов иногда данные в массиве копируются в новый массив, а иногда нет. В NumPy существует три способа копирования массивов. Простое присваивание не копирует массив и данные в нем. Передача массива в функцию не создает копию массива. Для того чтобы скопировать данные, необходимо использовать функцию copy (Рисунок 18).

```
a = np.arange(12)
b = a
print(b is a)
print(b.shape)
print(a.shape)
def f(x):
  print(id(x))
print(id(a))
```

```
d = a.copy()
print(d is a)
print(d.base is a)
d[0] = 9999
print(a)
```

Рисунок 18 – Копирование массивов.

Задание 7. Реализуйте код, написанный в примере, и прокомментируйте строки без комментариев.

Индексирование через массивы. Массивы целых чисел и логических переменных могут использоваться для индексирования массивов. Если индексируемый массив используется для многомерных массивов, индексы применяются только к первому измерению. Массивы индексов можно создавать для нескольких измерений (Рисунок 19).

```
a = np.arange(12)**2
i = np.array([1, 1, 3, 8, 5])
print(a[i])
j = np.array([[3, 4], [9, 7]])
print(a[j])
```

Рисунок 19 – Индексирование массивов.

Если для индексирования используются логические значения, в этом случае мы явно указываем, какие массивы мы используем, а какие нет. Наиболее распространенный способ использования логических индексов – это создание массива такого же размера, как и оригинальный массив (Рисунок 20).

```
a = np.arange(12).reshape(3, 4)
b = a>4
print(b)
print(a[b])
```

Рисунок 20 – Логические значения для индексирования.

Задачи для закрепления теоретического материала

Задание 1. Обработка одномерных массивов. В одномерном массиве, состоящем из *n* вещественных элементов, выполните вычисления в соответствии с вариантом.

Вариант	Задание
1	1) сумма отрицательных элементов массива; 2) произведение элементов массива, расположенных между максимальным и минимальным элементами; 3) упорядочите элементы массива в порядке возрастания.
2	1) сумма положительных элементов массива; 2) произведение элементов массива, расположенных между элементами с максимальным и минимальным модулем; 3) упорядочите элементы массива в порядке убывания.
3	1) произведение элементов массива с четными номерами; 2) сумма элементов массива, расположенных между первым и последним нулевыми элементами; 3) преобразуйте массив так, чтобы сначала располагались все положительные элементы, а затем все отрицательные элементы (элементы, равные 0, следует считать положительными).
4	1) сумма элементов массива с нечетными номерами; 2) сумма элементов массива, расположенных между первым и последним отрицательными элементами; 3) сжать массив, удалив из него все элементы, модуль которых не превышает 1. Заполните свободные элементы в конце массива нулями.
5	1) максимальный элемент массива; 2) сумма элементов массива до последнего положительного элемента;

	3) сжать массив, удалив из него все элементы, модуль которых находится в интервале [a, b]. Заполните свободные элементы в конце массива нулями.

Задание 2. Двумерные массивы. Дана целочисленная прямоугольная матрица.

Вариант	Задание
1	количество строк, не содержащих ни одного нулевого элемента;
2	максимум чисел, встречающихся более одного раза в данной матрице;
3	определите количество столбцов, не содержащих нулевых элементов;
4	Характеристика строки целочисленной матрицы — это сумма ее положительных четных элементов. Переставьте строки данной матрицы и упорядочите их по возрастанию характеристик;
5	количество столбцов, содержащих хотя бы один нулевой элемент;