# **Лабораторная работа №24 (2 часа)**

**Тема работы: «Разработка, отладка и испытание программ, работающих**

**со списками больших объемов, и программ, взаимодействующих**

**с отдельными элементами массива»**

**1 Цель работы**

Научить создавать и обрабатывать списки больших объемов данных с помощью библиотеки NumPy.

**2 Задание**

Выполните следующие действия над массивами с помощью модуля NumPy:

* найти самую длинную группу подряд идущих положительных элементов массива;
* создать копию массива;
* объединить два массива в один;
* разбить массив на два массива.

**3 Оснащение работы**

Задание по варианту, ЭВМ, среда разработки **Python 3.7, IDLE**.

**4 Основные теоретические сведения**

NumPy – это библиотека языка Python, добавляющая поддержку больших многомерных массивов и матриц, вместе с большой библиотекой высокоуровневых (и очень быстрых) математических функций для операций с этими массивами.

**Индексы, срезы, итерации**

Одномерные массивы осуществляют операции индексирования, срезов и итераций очень схожим образом с обычными списками и другими последовательностями Python (разве что удалять с помощью срезов нельзя).

>>>

>>> a = np.arange(10) \*\* 3

>>> a

array([ 0, 1, 8, 27, 64, 125, 216, 343, 512, 729])

>>> a[1]

1

>>> a[3:7]

array([ 27, 64, 125, 216])

>>> a[3:7] = 8

>>> a

array([ 0, 1, 8, 8, 8, 8, 8, 343, 512, 729])

>>> a[::-1]

array([729, 512, 343, 8, 8, 8, 8, 8, 1, 0])

>>> del a[4:6]

Traceback (most recent call last):

File "<input>", line 1, in <module>

ValueError: cannot delete array elements

>>> for i in a:

... print(i \*\* (1/3))

...

0.0

1.0

2.0

2.0

2.0

2.0

2.0

7.0

8.0

9.0

У многомерных массивов на каждую ось приходится один индекс. Индексы передаются в виде последовательности чисел, разделенных запятыми (то бишь, кортежами):

>>>

>>> b = np.array([[ 0, 1, 2, 3],

... [10, 11, 12, 13],

... [20, 21, 22, 23],

... [30, 31, 32, 33],

... [40, 41, 42, 43]])

...

>>> b[2,3] # Вторая строка, третий столбец

23

>>> b[(2,3)]

23

>>> b[2][3] # Можно и так

23

>>> b[:,2] # Третий столбец

array([ 2, 12, 22, 32, 42])

>>> b[:2] # Первые две строки

array([[ 0, 1, 2, 3],

[10, 11, 12, 13]])

>>> b[1:3, : : ] # Вторая и третья строки

array([[10, 11, 12, 13],

[20, 21, 22, 23]])

Когда индексов меньше, чем осей, отсутствующие индексы предполагаются дополненными с помощью срезов:

>>>

>>> b[-1] # Последняя строка. Эквивалентно b[-1,:]

array([40, 41, 42, 43])

b[i] можно читать как b[i, <столько символов ':', сколько нужно>]. В NumPy это также может быть записано с помощью точек, как b[i, ...].

Например, если x имеет ранг 5 (то есть у него 5 осей), тогда

x[1, 2, ...] эквивалентно x[1, 2, :, :, :],

x[... , 3] то же самое, что x[:, :, :, :, 3] и

x[4, ... , 5, :] это x[4, :, :, 5, :].

>>>

>>> a = np.array(([[0, 1, 2], [10, 12, 13]], [[100, 101, 102], [110, 112, 113]]))

>>> a.shape

(2, 2, 3)

>>> a[1, ...] # то же, что a[1, : , :] или a[1]

array([[100, 101, 102],

[110, 112, 113]])

>>> c[... ,2] # то же, что a[: , : ,2]

array([[ 2, 13],

[102, 113]])

Итерирование многомерных массивов начинается с первой оси:

>>>

>>> for row in a:

... print(row)

...

[[ 0 1 2]

[10 12 13]]

[[100 101 102]

[110 112 113]]

Однако, если нужно перебрать поэлементно весь массив, как если бы он был одномерным, для этого можно использовать атрибут flat:

>>>

>>> for el in a.flat:

... print(el)

...

0

1

2

10

12

13

100

101

102

110

112

113

**Манипуляции с формой**

Как уже говорилось, у массива есть форма (shape), определяемая числом элементов вдоль каждой оси:

>>>

>>> a

array([[[ 0, 1, 2],

[ 10, 12, 13]],

[[100, 101, 102],

[110, 112, 113]]])

>>> a.shape

(2, 2, 3)

Форма массива может быть изменена с помощью различных команд:

>>>

>>> a.ravel() # Делает массив плоским

array([ 0, 1, 2, 10, 12, 13, 100, 101, 102, 110, 112, 113])

>>> a.shape = (6, 2) # Изменение формы

>>> a

array([[ 0, 1],

[ 2, 10],

[ 12, 13],

[100, 101],

[102, 110],

[112, 113]])

>>> a.transpose() # Транспонирование

array([[ 0, 2, 12, 100, 102, 112],

[ 1, 10, 13, 101, 110, 113]])

>>> a.reshape((3, 4)) # Изменение формы

array([[ 0, 1, 2, 10],

[ 12, 13, 100, 101],

[102, 110, 112, 113]])

Порядок элементов в массиве в результате функции ravel() соответствует обычному «C-стилю», то есть, чем правее индекс, тем он «быстрее изменяется»: за элементом a[0,0] следует a[0,1]. Если одна форма массива была изменена на другую, массив переформировывается также в «C-стиле». Функции ravel() и reshape() также могут работать (при использовании дополнительного аргумента) в FORTRAN-стиле, в котором быстрее изменяется более левый индекс.

>>>

>>> a

array([[ 0, 1],

[ 2, 10],

[ 12, 13],

[100, 101],

[102, 110],

[112, 113]])

>>> a.reshape((3, 4), order='F')

array([[ 0, 100, 1, 101],

[ 2, 102, 10, 110],

[ 12, 112, 13, 113]])

Метод reshape() возвращает ее аргумент с измененной формой, в то время как метод resize() изменяет сам массив:

>>>

>>> a.resize((2, 6))

>>> a

array([[ 0, 1, 2, 10, 12, 13],

[100, 101, 102, 110, 112, 113]])

Если при операции такой перестройки один из аргументов задается как -1, то он автоматически рассчитывается в соответствии с остальными заданными:

>>>

>>> a.reshape((3, -1))

array([[ 0, 1, 2, 10],

[ 12, 13, 100, 101],

[102, 110, 112, 113]])

**Объединение массивов**

Несколько массивов могут быть объединены вместе вдоль разных осей с помощью функций hstack и vstack.

hstack() объединяет массивы по первым осям, vstack() — по последним:

>>>

>>> a = np.array([[1, 2], [3, 4]])

>>> b = np.array([[5, 6], [7, 8]])

>>> np.vstack((a, b))

array([[1, 2],

[3, 4],

[5, 6],

[7, 8]])

>>> np.hstack((a, b))

array([[1, 2, 5, 6],

[3, 4, 7, 8]])

Функция column\_stack() объединяет одномерные массивы в качестве столбцов двумерного массива:

>>>

>>> np.column\_stack((a, b))

array([[1, 2, 5, 6],

[3, 4, 7, 8]])

Аналогично для строк имеется функция row\_stack().

>>>

>>> np.row\_stack((a, b))

array([[1, 2],

[3, 4],

[5, 6],

[7, 8]])

**Разбиение массива**

Используя hsplit() вы можете разбить массив вдоль горизонтальной оси, указав либо число возвращаемых массивов одинаковой формы, либо номера столбцов, после которых массив разрезается «ножницами»:

>>>

>>> a = np.arange(12).reshape((2, 6))

>>> a

array([[ 0, 1, 2, 3, 4, 5],

[ 6, 7, 8, 9, 10, 11]])

>>> np.hsplit(a, 3) # Разбить на 3 части

[array([[0, 1], [6, 7]]),

array([[2, 3], [8, 9]]),

array([[ 4, 5], [10, 11]])]

>>> np.hsplit(a, (3, 4)) # Разрезать a после третьего и четвёртого столбца

[array([[0, 1, 2], [6, 7, 8]]),

array([[3], [9]]),

array([[ 4, 5], [10, 11]])]

Функция vsplit() разбивает массив вдоль вертикальной оси, а array\_split() позволяет указать оси, вдоль которых произойдет разбиение.

**Копии и представления**

При работе с массивами, их данные иногда необходимо копировать в другой массив, а иногда нет. Это часто является источником путаницы. Возможно 3 случая:

*Вообще никаких копий*

Простое присваивание не создает ни копии массива, ни копии его данных:

>>>

>>> a = np.arange(12)

>>> b = a # Нового объекта создано не было

>>> b is a # a и b это два имени для одного и того же объекта ndarray

True

>>> b.shape = (3,4) # изменит форму a

>>> a.shape

(3, 4)

Python передает изменяемые объекты как ссылки, поэтому вызовы функций также не создают копий.

*Представление или поверхностная копия*

Разные объекты массивов могут использовать одни и те же данные. Метод view() создает новый объект массива, являющийся представлением тех же данных.

>>>

>>> c = a.view()

>>> c is a

False

>>> c.base is a # c это представление данных, принадлежащих a

True

>>> c.flags.owndata

False

>>>

>>> c.shape = (2,6) # форма а не поменяется

>>> a.shape

(3, 4)

>>> c[0,4] = 1234 # данные а изменятся

>>> a

array([[ 0, 1, 2, 3],

[1234, 5, 6, 7],

[ 8, 9, 10, 11]])

Срез массива это представление:

>>>

>>> s = a[:,1:3]

>>> s[:] = 10

>>> a

array([[ 0, 10, 10, 3],

[1234, 10, 10, 7],

[ 8, 10, 10, 11]])

*Глубокая копия*

Метод copy() создаст настоящую копию массива и его данных:

>>>

>>> d = a.copy() # создается новый объект массива с новыми данными

>>> d is a

False

>>> d.base is a # d не имеет ничего общего с а

False

>>> d[0, 0] = 9999

>>> a

array([[ 0, 10, 10, 3],

[1234, 10, 10, 7],

[ 8, 10, 10, 11]])

**5 Порядок выполнения работы**

1. Выделить ключевые моменты задачи.

2. Построить алгоритм решения задачи.

3. Запрограммировать полученный алгоритм.

4. Провести тестирование полученной программы.

**6 Форма отчета о работе**

*Лабораторная работа № \_\_\_*

*Номер учебной группы \_\_\_\_\_\_\_\_\_\_\_\_*

*Фамилия, инициалы учащегося: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_*

*Дата выполнения работы: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_*

*Тема работы: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_*

*Цель работы: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_*

*Оснащение работы: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_*

*Результат выполнения работы: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_*

*\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_*

**7 Контрольные вопросы и задания**

1. Назовите назначение модуля NymPy.
2. Что такое индексы, срезы и итерации?
3. Перечислите функции с помощью которых можно управлять формой массива.
4. Опишите способы разбиения массива.
5. Опишите способы копирования массива.

**8 Рекомендуемая литература**

**Плас, Дж. В.** Python для сложных задач. Наука о данных и машинное обучение / Дж.В. Плас. – СПб: Питер, 2018.

**Прохоренок, Н.А.** Python 3. Самое необходимое / Н.А Прохоренок, В.А. Дронов – СПб.: БВХ-Петербург, 2016.