

3. МАТРИЦЫ И ВЕКТОРЫ

3.1. Массивы

[NumPy](#) — это библиотека языка Python, предназначенная для создания и обработки многомерных массивов числовых данных (векторов и матриц). Также NumPy включает большую библиотеку высокоуровневых (и очень быстрых) математических функций для операций с этими массивами. Подключение библиотеки:

```
import numpy as np
```

Для упрощения обращения к методам библиотеки модуль в программе будет именоваться **np**.

3.1.1. Понятие массива

Массив — это пронумерованная последовательность величин одинакового типа, обозначенная одним именем. Каждое из значений, составляющих массив, называется *компонентой* (или *элементом* массива). Для обращения к элементу массива используется его индекс (рисунок 24).



Рисунок 24. Структура массива

Для обращения к элементу по индексу 5 используется запись **a[5]**, изменить значение, хранящееся по этому индексу, можно с помощью оператора присваивания:

```
a[5] = 47
```

Создание массива

В NumPy существует много способов создать массив. Проще всего создать массив из списка. Для этого используется метод **array()**:

```
import numpy as np
a = np.array([-4, 5, 2])
```

При создании массива из списка можно переопределить тип его элементов, например, из списка, состоящего из строк, представляющих собой числа, можно создать массив из целых чисел.

```
a = np.array(['-4', '5', '2'], dtype=int)
```

Например, введем с клавиатуры строку, которая состоит из чисел, разделенных пробелом. А затем создадим массив из них:

```
line = input("Введите числа через пробел : ")
list_a = line.split()
a = np.array(list_a, dtype=int)
```

Вывод массива

Вывод массива осуществляется с помощью оператора **print()**:

```
a = np.array(['-4', '5', '2'], dtype=int)
print("a = ", a)
```

Результат:

```
a = [-4  5  2]
```

Задание для самопроверки

Отметьте **неверные** операторы и объясните, в чем ошибка. В начале программы модуль `numpy` импортирован следующим образом:

```
import numpy as np
```

При анализе операторов вывода считайте, что в переменной **a** хранится массив чисел.

- `print a`
- `a = array([1, 2, 3], dtype=int)`
- `a = input(np.array())`
- `a = np.array([1, 2, 3], dType=int)`
- `a = np.array(1, 2, 3)`
- `print("a = %3d " % a)`

3.1.2. Методы массивов

Модуль **numpy** предоставляет множество методов работы с массивами. Для обращения к ним после имени массива ставится точка, затем записывается метод и круглые скобки. В таблице 17 приведены некоторые методы, примененные к массиву с именем **a**.

Таблица 17 – Методы массивов

Метод	Описание
<code>a.sum()</code>	Сумма элементов массива <code>a</code>
<code>a.prod()</code>	Произведение элементов массива <code>a</code>
<code>a.mean()</code>	Среднее значение элементов массива <code>a</code>
<code>a.max()</code>	Максимальное значение из элементов массива <code>a</code>
<code>a.min()</code>	Минимальное значение из элементов массива <code>a</code>

Например, в следующей программе методы массивов применяются для вычисления суммы, произведения элементов массива, поиска среднего, максимального и минимального элемента.

```
import numpy as np
```

```
a = np.array([-4, 5, 2])
```

```
print("a = ", a)
print("Сумма = ", a.sum())
print("Произведение = ", a.prod())
print("Среднее = ", a.mean())
print("Максимум = ", a.max())
print("Минимум = ", a.min())
```

Результат:

```
a = [-4  5  2]
Сумма = 3
Произведение = -40
Среднее = 1.0
Максимум = 5
Минимум = -4
```

Задание для самопроверки

Вставьте в программу пропущенные фрагменты так, чтобы в результате выполнения программы получился следующий результат:

```
[200.    6.   -5.    1.]
```

Код программы:

```
import numpy as np
```

```
a = np.array([-5, 5, -1, 1, 2, 4])
```

```

b_0 = a.____
b_1 = a.____
b_2 = a.____
b_3 = a.____

b = np.array([b_0, b_1, b_2, b_3])
print(b)

```

3.1.3. Операции с массивами

Математические операции: сложение (+), умножение (*), вычитание (-), деление(/), остаток от деления (%), целочисленное деление (//) – применяются непосредственно к массивам. При их использовании создается новый массив, который заполняется результатами действия математической операции с соответствующими элементами массива. Массивы при этом должны быть одинакового размера.

В следующей программе математические операции применяются к массивам **a** и **b**.

```

import numpy as np

a = np.array([-4, 5, 2])
b = np.array([3, 2, 1])

c = a + b
print("a + b =", c)

c = a - b
print("a - b =", c)

c = a * b
print("a * b =", c)

c = a / b
print("a / b =", c)

c = a % b
print("a % b =", c)

c = a // b
print("a // b =", c)

```

Результат:

```

a + b = [-1  7  3]
a - b = [-7  3  1]
a * b = [-12 10  2]
a / b = [-1.33333333  2.5  2.]
a % b = [2 1 0]
a // b = [-2  2  2]

```

Задание для самопроверки

Какую операцию нужно вставить в выражение для формирования массива **c**, чтобы получился следующий результат:

```
[3 7 2]
```

Код программы:

```

import numpy as np

a = np.array([5, 5, 2])
b = np.array([2, 7, 1])

c = a.min()+ a * (b + b.min()) ____ b
print(c)

```

3.1.4. Функции массивов

В модуле **numpy** реализованы математические функции, выполняемые над массивами, в таблице приведены некоторые из них для массива с именем **a**. Перед вызовом функций в программе необходимо указать префикс «**np**» (имя модуля **numpy** в текущей программе) и точку.

Таблица 18 – Функции массивов

Функция	Описание
<code>round(a, k)</code>	Округляет значения элементов массива a до k знаков после запятой, результат – массив
<code>sin(a)</code>	Вычисляет синусы от каждого элемента массива a , результат – массив
<code>cos(a)</code>	Вычисляет косинусы от каждого элемента массива a , результат – массив
<code>tan(a)</code>	Вычисляет тангенсы от каждого элемента массива a , результат – массив
<code>arcsin(a)</code>	Вычисляет арксинусы от каждого элемента массива a , результат – массив
<code>arctan(a)</code>	Вычисляет арктангенсы от каждого элемента массива a , результат – массив
<code>degrees(a)</code>	Переводит каждый элемент массива a в градусы, результат – массив
<code>radians(a)</code>	Переводит каждый элемент массива a в радианы, результат – массив
<code>log(a)</code>	Вычисляет натуральный логарифм от каждого элемента массива a , результат – массив
<code>log10(a)</code>	Вычисляет десятичный логарифм от каждого элемента массива a , результат – массив
<code>exp(a)</code>	Вычисляет экспоненту от каждого элемента массива a , результат – массив
<code>sum(a)</code>	Вычисляет сумму элементов массива a , результат – число
<code>prod(a)</code>	Вычисляет произведение элементов массива a , результат – число
<code>mean(a)</code>	Вычисляет среднее значение элементов массива a , результат – число
<code>max(a)</code>	Вычисляет максимальное значение из элементов массива a , результат – число
<code>min(a)</code>	Вычисляет минимальное значение из элементов массива a , результат – число
<code>abs(a)</code>	Применяет функцию модуль для каждого элемента массива a , результат – массив
<code>pi</code>	Число пи , умноженное на все элементы массива.
<code>sqrt(a)</code>	Вычисляет корень от каждого элемента массива a , результат – массив

Пример. Решим задачу с вычислением численности населения по формуле Капицы, используя массивы из модуля **numpy**. Создадим массив с различными годами, а затем посчитаем численность населения Земли в эти года по формуле:

$$N(t) = \frac{C}{\tau} \cdot \arctg\left(\frac{T_1 - t}{\tau}\right),$$

где t – год, для которого вычисляется численность населения;

C – 172 миллиарда человек/лет;

T_1 – 2000 год;

τ – 45 лет.

Код программы:

```
import numpy as np

def compute_population(t):
    c = 172
    t_1 = 2000
    tau = 45
    y = c / tau * (np.pi / 2 - np.arctan((t_1-t)/tau))
    return y

t = np.array([1000, 1500, 1800, 1850, 1900, 1950, 1990, 2019, 2030])
```

```
print("Годы:", t)
print("Численность:", np.round(compute_population(t), 3))
```

Результат:

```
Годы: [1000 1500 1800 1850 1900 1950 1990 2019 2030]
Численность: [0.172 0.343 0.846 1.114 1.616 2.801 5.168 7.531 8.251]
```

Обратите внимание, что в программе не используются циклы, которые осуществляют поэлементный перебор, не используются конструкторы списков. Все операции выполняются целиком над массивами. Поэтому перед всеми функциями (**pi**, **arctan**, **round**) стоит префикс **np**.

Задание для самопроверки

Дан фрагмент программы:

```
import numpy as np
a = np.array([2, 5, -2, 0, 8])
b = np.array([5, -1, -3, 6, 2])
```

Выберите те операторы, в результате которых получается массив.

- `np.sum(a + b)`
- `np.sum(a) + np.mean(a)`
- `a + np.sum(b)`
- `np.pi - a`
- `np.round(a, 2)`
- `a + b`

3.1.5. Функция *where* и логические функции

Для того чтобы найти индексы элементов в массиве, отвечающие некоторому условию, используется метод **where()**, в качестве аргумента которого задается логическое выражение.

Это выражение может включать:

- знаки отношения – равно(==), не равно(!=), больше(>), больше или равно(>=), меньше(<), меньше или равно(<=);
- имена массивов;
- константы;
- выражения.

Результат выполнения функции – массив модуля **numpy**, чтобы получить список индексов в привычном виде, необходимо обратиться к результату по индексу 0 ([0]).

Рассмотрим, как работает функция **where()**, в качестве исходных данных будем использовать массивы:

```
import numpy as np
a = np.array([5, 5, 2, 6, -1])
b = np.array([2, 7, 2, 3, -1])
```

- если в логическом выражении используются массивы, то они должны быть одинаковой длины, при этом каждый элемент одного массива сравнивается с элементом с тем же индексом второго массива, если логическое выражение выполняется, то индекс этого элемента включается в итоговый список:

```
print(np.where(a == b)[0]) → [2 4]
print(np.where(a > b)[0]) → [0 3]
```

- если в логическом выражении используются константа, то каждый элемент массива сравнивается с этой константой, если логическое выражение выполняется, то индекс этого элемента включается в итоговый список:

```
print(np.where(a < 3)[0]) → [2 4]
print(np.where(b > 10)[0]) → []
```

- если в логическом выражении используются выражения, то их результатом может быть либо число, либо массив, но тогда его длина должна соответствовать длине других массивов, используемых в выражении:

```
print(np.where(a >= b.mean())[0]) → [0 1 3]
print(np.where(a + b >= 8)[0]) → [1 3]
```

Также в логическом выражении можно использовать логические операции, которые реализуются с помощью функций:

ИЛИ – для реализации этой операции используется функция `np.logical_or()`;

И – `np.logical_and()`;

НЕ – `np.logical_not()`.

В скобках через запятую перечисляются логические выражения, которые должны быть соединены соответствующей операцией. Например, логическое выражение **a > 0 и a <= 6** в методе `where()` записывается так:

```
np.logical_and(a > 0, a <= 6) результат [0 1 2 3]
```

Допускается использование вложенных функций, описывающих логические операции. Например, логическое выражение **(a > 0 и a < 6) или (b < 0 или b > 7)** в методе `where()` записывается так:

```
np.logical_or(
    np.logical_and(a > 0, a < 6),
    np.logical_or(b < 0, b > 7)
) результат [0 1 2 4]
```

Задание для самопроверки

Дан фрагмент программы:

```
import numpy as np
a = np.array([2, 5, 2, 0, 0])
b = np.array([5, -1, -1, 6, 2])
c = # выражение
print(c)
```

Что будет напечатано в консоли, если массив `c` будет сформирован с помощью следующих выражений:

- `np.where(a == b)[0]`
- `np.where(np.logical_not(b <= 0))[0]`
- `np.where(np.logical_or(a >= 0, b == 6))[0]`
- `np.where(a == 0)[0]`
- `np.where(np.logical_and(a > 1, b < 0))[0]`

3.1.6. Срезы и индексы

Для обращения к элементам или группе элементов массива используются индексы и срезы.

Индекс – это номер элемента в массиве, нумерация начинается с 0. Для обращения к элементу с заданным индексом используются квадратные скобки: `a[3]` – элемент массива `a` с индексом 3.

Срез – это часть массива, которую можно выделить по индексам:

массив `[нижняя_граница : верхняя_граница : шаг]`

Для выделения среза в квадратных скобках указывается нижняя граница среза, верхняя граница, которая в срез не включается, и шаг, с которым выделяются индексы. Части описания среза могут быть опущены.

Например, рассмотрим массив объявленный следующим образом:

```
import numpy as np
a = np.array([5, 5, 1, 6, 15, 3, 8, -1])
```

тогда:

`a[2:6]` – в срез включаются элементы с индексами 2, 3, 4, 5 – `[1 6 15 3]`

`a[2:]` – элементы с индексами, начиная со 2 до последнего – `[1 6 15 3 8 -1]`

`a[:6]` – элементы с индексами, начиная со 0 до 5 включительно – `[5 5 1 6 15 3]`

`a[1:6:2]` – элементы с индексами 1, 3, 5 – `[5 6 3]`

`a[1::2]` – элементы с индексами 1, 3, 5, 7 – `[5 6 3 -1]`

`a[:6:2]` – элементы с индексами 0, 2, 4 – `[5 1 15]`

`a[::2]` – элементы с индексами 0, 2, 4, 6 – `[5, 1, 15, 8]`

Срез можно рассматривать как новый массив. Оператор:

`b = a[1:3]`

выделяет элементы из массива **a**, начиная с элемента с индексом 1 и заканчивая элементом с индексом 2 (получается результат `[5 2]`), полученный массив записывается в переменную **b**.

Срезы позволяют с помощью оператора присваивания изменять значения нескольких элементов массива одновременно. Оператор:

`a[1:4] = 0`

присваивает значение 0 элементам массива **a**, начиная с индекса 1 по индекс 3, результат – `[5 0 0 0 -1]`.

Задание для самопроверки

Напишите программу для решения задачи:

Дан массив, в который занесены расходы человека на проезд по каждому месяцу в течение года. Сравнить, в какой период – зимний или летний – тратится больше денег на проезд. Вывести номера месяцев (начиная с 1), в которые расходы были наибольшими.

Тест 1

Расходы: 1200 1300 900 1450 1300 1000 900 1000 1450 1450 1300 1400

Зимой на проезд потрачено больше денег, сумма: 3900 руб.

Самая большая сумма: 1450 руб., потрачена в следующих месяцах: [4 9 10]

Тест 2

Расходы: 1000 900 750 1050 1000 900 1200 1300 1050 1050 1300 1200

Зимой на проезд потрачено больше денег, сумма: 3400 руб.

Самая большая сумма: 1300 руб., потрачена в следующих месяцах: [8 11]

Тест 3

Расходы: 1200 1300 900 1450 1300 1000 1500 1000 1450 1450 1300 1000

Зимой и летом на проезд потрачены одинаковые деньги, сумма: 3500 руб.

Самая большая сумма: 1450 руб., потрачена в следующих месяцах: [7]