

Библиотека за изчисления NumPy. Основни понятия. NumPy масиви. Изчисления с NumPy масиви

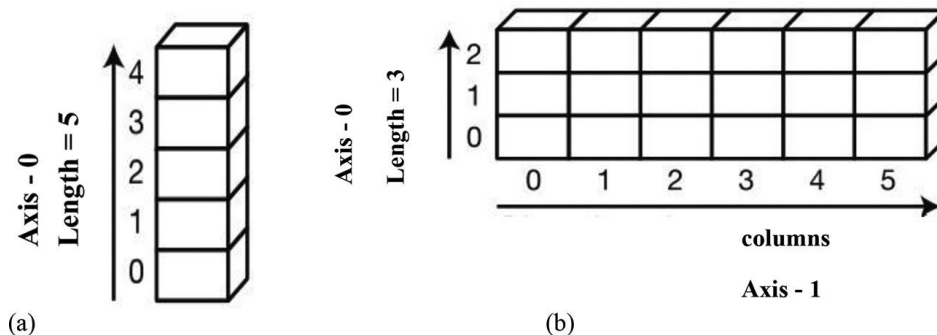
NumPy е основният пакет за научни изчисления с Python. NumPy означава „Numerical Python“. Той поддържа:

- N-мерни обекти на масив
- Broadcasting функции
- Инструменти за интегриране на C / C ++ и Fortran код
- Полезна линейна алгебра, преобразуване на Фурие и възможности за произволни числа.

Освен очевидните си научни приложения, NumPy може да се използва и като многомерен контейнер за съхраняване на общи данни. Могат да бъдат дефинирани и произволни типове данни. Това позволява на NumPy да се интегрира безпроблемно и бързо с голямо разнообразие от бази данни. Основният обект на NumPy е хомогенният многомерен масив. Класът на масива на NumPy се нарича **ndarray**. Известен е и с псевдонима **array**.

В NumPy масивите отделните елементи от данни се наричат елементи. Всички елементи на масива трябва да бъдат от един и същи тип. Масивите могат да бъдат съставени от произволен брой измерения. В NumPy размерите се наричат оси. Всяко измерение на масив има дължина, която е общият брой елементи в тази посока. Размерът на масива е общият брой елементи, съдържащи се във всички измерения. Размерът на NumPy масивите е фиксиран; веднъж създаден, той не може да бъде променен.

Например, ФИГУРА 1 показва осите (или размерите) и дължините на два примерни масива; 1 (а) е едномерен масив, а 1 (б) е двуизмерен масив. Едномерният масив има една ос, обозначена с Оста-0. Тази ос има пет елемента в себе си, затова казваме, че има дължина пет. Двуизмерения масив се състои от редове и колони.



Фиг. 1 NumPy масиви

Всички редове са обозначени с ос-0, а всички колони са означени с ос-1. В двумерния масив Axis-0 има три елемента, така че дължината му е три, а Axis-1 има шест елемента, така че дължината му е шест. За всяка ос индексите варират от 0 до дължина - 1. Индексите на масива се базират на 0.

За да използвате NumPy във вашата програма, трябва да импортирате NumPy:

```
import numpy as np
```

numpy обикновено се преименува на np.

1. Създаване на масиви NumPy с помощта на функцията array

Създаването на NumPy масив може да стане от обикновен списък на Python или кортеж чрез използване на функцията **np.array ()**.

Типът на получения масив се извежда от типа на елементите.

Пример:

```
1. >>> import numpy as np
2. >>> int_number_array = np.array([1,2,3,4])
3. >>> int_number_array
array([1, 2, 3, 4])
4. >>> type(int_number_array) <class 'numpy.ndarray'>
5. >>> int_number_array.dtype
dtype('int32')
6. >>> float_number_array = np.array([9.1, 8.1, 8.8, 3.0])
7. >>> float_number_array.dtype
dtype('float64')
8. >>> two_dimensional_array_list = np.array([[1,2,3],
[4,5,6]])
9. >>> two_dimensional_array_list
array([[1, 2, 3], [4, 5, 6]])
```

```

10. >>> two_dimensional_array_tuple = np.array(((1,2,3),
(4,5,6)))

11. >>> two_dimensional_array_tuple

array([[1, 2, 3],[4, 5, 6]])
12. >>> array_dtype = np.array([1,2,3,4], dtype = np.float64)
13. >>> array_dtype
array([1., 2., 3., 4.])
14. >>> array_dtype.dtype
dtype('float64')

```

Импортиране на библиотеката `numpy` ①. Предаване на списък с елементи на функция **`np.array ()`** и присвояване резултата на **`int_number_array`** обект ②. На изхода може да се види, че всички елементи са поставени в обект от клас `array` и е едномерен масив.

Обектът **`int_number_array`** принадлежи към клас **`numpy.ndarray`** ④. NumPy предоставя голям набор от числови типове данни, които могат да се използват за изграждане на масиви. NumPy се опитва да отгатне типа на данните при създаване на масив, но функциите, които изграждат масивите, обикновено включват и незадължителен аргумент, който изрично указва типа данни. Типът `int_number_array` е `dtype ('int32')` ⑤. Типът `float_number_array` ⑥ е `dtype ('float64')` ⑦. Функцията `np.array ()` приема като аргумент или единичен списък, или кортеж. Ако трябва да се посочат множество списъци или кортежи, тогава се предават като вложени списъци ⑧ – ⑨ или вложени кортежи (10-11). **`two_dimensional_array_list`** и **`two_dimensional_array_tuple`** са примери за двумерни масиви. Изрично може да се укаже типа данни на масива, като се присвои стойност на типа като `np.float64`, `np.int32` и други на атрибут `dtype` и се предаде като втори аргумент на функцията **`np.array ()`** (12-14).

2 Атрибути на масива

Съдържанието на `ndarray` може да бъде достъпно и модифицирано чрез индексирание или нарязване на масива и чрез методите и атрибутите на **`ndarray`**. По-важните атрибути на обекта **`ndarray`** са:

Атрибути на <code>ndarray</code>	Описание
---	----------

<code>ndarray.ndim</code>	Връща броя на осите или размерите в масива
<code>ndarray.shape</code>	Връща размерите на масива. За масив с n реда и m колони, формата ще бъде набор от цели числа (n, m) .
<code>ndarray.size</code>	Връща общия брой елементи на масива.
<code>ndarray.dtype</code>	Връща обект, описващ типа на елементите в масива. Може да създава или указва <code>dtype</code> , като използва стандартните типове на Python. Освен това NumPy предоставя свои собствени типове като <code>np.int32</code> , <code>np.int16</code> , <code>np.float64</code> и други.
<code>ndarray.itemsize</code>	Връща размера на всеки елемент от масива в байтове.
<code>ndarray.data</code>	Връща буфера, съдържащ действителните елементи на масива.

Пример:

```

1. >>> import numpy as np
2. >>> array_attributes = np.array([[10, 20, 30], [14, 12, 16]])
3. >>> array_attributes.ndim
2
4. >>> array_attributes.shape
(2, 3)
5. >>> array_attributes.size
6
6. >>> array_attributes.dtype
dtype('int32')
7. >>> array_attributes.itemsize
4
8. >>> array_attributes.data
<memory at 0x000001E61DB963A8>

Различни атрибути на ndarray (1-8)

```

3. Създаване на NumPy масиви с първоначално съдържание

Често елементите на масив първоначално са неизвестни, но размерът му е известен. NumPy предлага няколко функции за създаване на масиви с първоначално съдържание. Те свеждат до минимум необходимостта от нарастващи масиви, което е скъпа операция.

Функция	Описание
<code>np.zeros()</code>	Създаване на масив с 0.
<code>np.ones()</code>	Създаване на масив с 1.
<code>np.empty()</code>	Създаване на празен масив.
<code>np.full()</code>	Създава пълен масив
<code>np.eye()</code>	Създава матрица за идентичност
<code>np.random.random()</code>	Създава масив със случайни стойности
<code>np.arange()</code>	Синтаксисът за <code>arange()</code> : <code>np.arange([start,] stop, [step,] [dtype = None])</code> Връща равномерно разпределени стойности в рамките на даден интервал, където <code>start</code> (число по избор) е началото на интервала и стойността му по подразбиране е нула, <code>stop</code> (число) е края на интервала и <code>step</code> (число по избор) е разстоянието между стойностите и <code>dtype</code> е видът на изходния масив.
<code>np.linspace()</code>	Синтаксисът за <code>linspace</code> е: <code>numpy.linspace(start, stop, num = 50, dtype = None)</code> Връща равномерно разпределени числа през определен интервал, където <code>start</code> е началната стойност на последователността, <code>stop</code> е крайната стойност на последователността, а <code>num</code> (цяло число и по избор) е броят на генерираните проби. По подразбиране е 50. Трябва да е неотрицателно. <code>dtype</code> е типът на изходния масив и е незадължителен елемент.

Пример:

```

1. >>> import numpy as np
2. >>> np.zeros((2,3))
array([[0., 0., 0.], [0., 0., 0.]])
3. >>> np.ones((3,4))
array([[1., 1., 1., 1.], [1., 1., 1., 1.], [1., 1., 1., 1.]])
4. >>> np.empty((2,3))
array([[0., 0., 0.], [0., 0., 0.]])
5. >>> np.full((3,3),2)
array([[2, 2, 2], [2, 2, 2], [2, 2, 2]])
6. >>> np.eye(2,2)
array([[1., 0.], [0., 1.]])
7. >>> np.random.random((2,2))
array([[0.95022839, 0.23253555], [0.843828 , 0.57976282]])

```

```
8. >>> np.arange(10, 30, 5)
array([10, 15, 20, 25])
9. >>> np.arange(0, 2, 0.3)
array([0. , 0.3, 0.6, 0.9, 1.2, 1.5, 1.8])
10. >>> np.linspace(0, 2, 9)
array([0. , 0.25, 0.5 , 0.75, 1. , 1.25, 1.5 , 1.75, 2. ])
```

Различни функции на NumPy за създаване на масиви са показани в ①–⑩. Когато функцията **arange ()** се използва с аргументи с плаваща запетая, обикновено не е възможно да се предскаже броят на получените елементи, поради прецизността с крайна плаваща запетая ⑨. Поради тази причина обикновено е по-добре да се използва функцията **linspace ()**, на която може да се предаде аргумент, указващ броя на елементите, които да се генерират. Функцията **linspace ()** генерира девет елемента от данни между нула и две.

4. Цялостно индексирание, индексирание на масив, булево индексирание на масив, разделяне и итерация в масиви

Едномерните масиви могат да бъдат индексирани, нарязани и повторени, подобно на списъци и други последователности на Python.

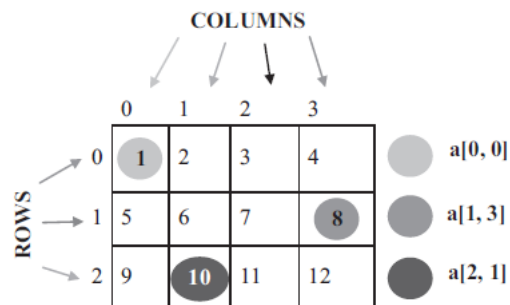
Пример:

```
1. >>> import numpy as np
2. >>> a = np.arange(5)
3. >>> a
array([0, 1, 2, 3, 4])
4. >>> a[2]
2
5. >>> a[2:4]
array([2, 3])
6. >>> a[:4:2] = -999
7. >>> a
array([-999, 1, -999, 3, 4])
8. >>> a[::-1]
array([ 4, 3, -999, 1, -999])
9. >>> for each_element in a:
10. ... print(each_element)
-999
1
-999
3
4
```

Индексирание, разделяне и итерация на едномерни масиви NumPy ①–⑩.

За многомерни масиви може да се посочи индекс или отрязък на ос.

Пример:



```
1. >>> import numpy as np
2. >>> a = np.array([[1,2,3,4], [5,6,7,8], [9,10,11,12]])

3. >>> a[1, 3]
8
4. >>> a[:2, 1:3]
array([[2, 3],
       [6, 7]])
5. >>> lower_axes = a[1, :]
6. >>> lower_axes
array([5, 6, 7, 8])
7. >>> lower_axes.ndim
1
8. >>> same_axes = a[1:2, :]
9. >>> same_axes
array([[5, 6, 7, 8]])
10. >>> same_axes.ndim
2
11. >>> a[:, 1]
array([ 2, 6, 10])
12. >>> a[:, 1:2]
array([[ 2],
       [ 6],
       [10]])
13. >>> for row in a:
14. ... print(row)
[1 2 3 4]
[5 6 7 8]
[ 9 10 11 12]
15. >>> for each_element in a.flat:
16. ... print(each_element)
1
2
3
4
5
```

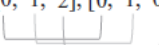
```
6
7
8
9
10
11
12
```

За извеждане на елементите налични в ред 1 и колона 3 се използва целочислено индексирание ③. Може да се смесва целочислено индексирание с индексирание на срезове. В ④ се извежда масива, състоящ се от ред 0 и ред 1 и колона 1 и колона 2 с форма (2, 2). Показване на елементите, присъстващи във всички колони на ред 1 ⑤ – ⑥, ⑧ – ⑨. Смесването на целочислено индексирание с рязове дава масив от оси ⑦. Използването на срезове във всички оси дава масив от същите оси като оригиналния масив ⑩. Показване на всички елементи, присъстващи във всички редове на колона 1 (11-12). Итерацията върху многомерни масиви се извършва по отношение на първата ос (13-14). За извършване на операция върху всеки елемент в масива, може да се използва **flat** атрибут, който е итератор за всички елементи на масива (15-16).

```
1. >>> import numpy as np
2. >>> a = np.array([[1, 2], [3, 4], [5, 6]])
3. >>> a
array([[1, 2], [3, 4], [5, 6]])
4. >>> a[[0, 1, 2], [0, 1, 0]]
array([1, 4, 5])
```

Индексирането на целочислен масив позволява конструирането на произволни масиви, използвайки данните от друг масив ②. Кодов ред ④ се използва за конструиране на масив от друг масив ③.

`a[[0, 1, 2], [0, 1, 0]]`



Елементите, намерени в (0, 0), (1, 1) и (2, 0), се изваждат и показват.

```
1. >>> import numpy as np
2. >>> a = np.array([[11, 12], [13, 14], [15, 16]])
3. >>> a
array([[11, 12], [13, 14], [15, 16]])
```



```
4. >>> a[a > 13]
array([14, 15, 16])
```

Булевото индексване на масив позволява избор на елементите на масив, които отговарят на някакво условие. С булево индексване на масив изрично се избират кои елементи в масива да присъстват и кои не. При булево индексване на масив на `[a > 13]` ④ елементи, елементи по-големи от 13 а, се показват като едномерни масиви.

5. Основни аритметични операции на масиви NumPy

Основните математически функции изпълняват елементарно действие върху масиви и са достъпни като функции в модула NumPy.

Пример:

```
1. >>> import numpy as np
2. >>> a = np.array( [20, 30, 40, 50] )
3. >>> b = np.arange(4)
4. >>> b
array([0, 1, 2, 3])
5. >>> a + b
array([20, 31, 42, 53])
6. >>> np.add(a, b)
array([20, 31, 42, 53])
7. >>> a - b
array([20, 29, 38, 47])
8. >>> np.subtract(a, b)
array([20, 29, 38, 47])
9. >>> A = np.array( [[1, 1], [6, 1]] )
10. >>> B = np.array( [[2, 8], [3, 4]] )
11. >>> A * B
array([[2, 8],[18, 4]])
12. >>> np.multiply(A, B)
array([[ 2,  8],[18,  4]])
13. >>> A / B
array([[0.5 , 0.125],[2. , 0.25 ]])
14. >>> np.divide(A, B)
array([[0.5 , 0.125],
       [2. , 0.25 ]])
15. >>> np.dot(A, B)
array([[ 5, 12],[15, 52]])
16. >>> B**2
array([[ 4, 64],[ 9, 16]], dtype=int32)
```

Извършват се елементарни операции по сумиране, изваждане, умножаване и разделяне, което води до масив (5-16).

6. Математически функции в NumPy

В NumPy се поддържат различни математически функции.

Пример:

```
1. >>> import numpy as np
2. >>> a = np.array( [20, 30, 40, 50] )
3. >>> np.sin(a)
array([ 0.91294525, -0.98803162, 0.74511316, -0.26237485])
4. >>> np.cos(a)
array([ 0.40808206, 0.15425145, -0.66693806, 0.96496603])
5. >>> np.tan(a)
array([ 2.23716094, -6.4053312 , -1.11721493, -0.27190061])
6. >>> a = np.array([-1.7, -1.5, -0.2, 0.2, 1.5, 1.7, 2.0])
7. >>> np.floor(a)
array([-2., -2., -1., 0., 1., 1., 2.])
8. >>> np.ceil(a)
array([-1., -1., -0., 1., 2., 2., 2.])
9. >>> np.sqrt([1,4,9])
array([ 1., 2., 3.])
10. >>> np.maximum([2, 3, 4], [1, 5, 2])
array([2, 5, 4])
11. >>> np.minimum([2, 3, 4], [1, 5, 2])
array([1, 3, 2])
12. >>> np.sum([0.5, 1.5])
2.0

13. >>> np.sum([[0, 1], [0, 5]], axis=0)
array([0, 6])
14. >>> np.sum([[0, 1], [0, 5]], axis=1)
array([1, 5])
```

Поддържат се тригонометрични операции като **sin ()**, **cos ()** и **tan ()** ③ - ⑤. Функцията **floor ()**, имаща синтаксис като floor (x), връща най-голямата целочислена стойност, по-малка или равна на x, по елемент ⑦. Функцията **ceil ()** със синтаксис като ceil (x), връща най-малката целочислена стойност, по-голяма или равна на x. Функцията **sqrt ()** връща квадратен корен на елементите на масив ⑨. Функцията **maximum ()** сравнява два масива и връща нов масив, съдържащ максимума от масива. Функцията **minimum ()** сравнява два масива и връща нов масив, съдържащ минимума от масива. Функцията **sum ()** връща сумата от елементи на масив по дадена ос. Всички горепосочени функции връщат нов масив.

7. Промяна на формата на масив

Формата на масив може да се променя.

Пример:

```
1. import numpy as np
2. >>> a = np.floor(10*np.random.random((3,4)))
3. >>> a
array([[2., 3., 2., 5.],
       [3., 3., 8., 7.],
       [8., 6., 5., 0.]])
4. >>> a.shape
(3, 4)
5. >>> a.ravel()
array([2., 3., 2., 5., 3., 3., 8., 7., 8., 6., 5., 0.])
6. >>> a.reshape(6,2)
array([[2., 3.],
       [2., 5.],
       [3., 3.],
       [8., 7.],
       [8., 6.],
       [5., 0.]])
```

Масивът има форма, зададена от броя на елементите по всяка ос

④. Формата на масив може да се променя с функциите **ravel ()** ⑤ и **reshape ()** ⑥. Както **ravel ()**, така и **reshape ()** връщат модифициран масив, но не променят оригиналния масив. Функцията **ravel ()** връща сплескан масив, като едномерен масив, съдържащ елементите на входа. Функцията **reshape ()** придава нова форма на масив, без да променя данните му.

8. Подреждане и разделяне на масиви

Може да се подредите няколко масива заедно или да се раздели масив на няколко масива.

Пример:

```
1. >>> import numpy as np
2. >>> a = np.array([[3, 1], [8, 7]])
3. >>> b = np.array([[2, 4], [4, 8]])
4. >>> np.vstack((a, b))
array([[3, 1],
       [8, 7],
       [2, 4],
       [4, 8]])
```

```

[2, 4],
[4, 8]])
5. >>> np.hstack((a, b))
array([[3, 1, 2, 4],
       [8, 7, 4, 8]])
6. >>> a = np.floor(10*np.random.random((2, 12)))
7. >>> a
array([[8., 3., 6., 3., 5., 5., 5., 8., 9., 7., 6., 8.],
       [8., 3., 1., 2., 9., 0., 5., 5., 0., 3., 3., 8.]])
8. >>> np.hsplit(a, 3)
[array([[8., 3., 6., 3.],
       [8., 3., 1., 2.]]) , array([[5., 5., 5., 8.],
       [9., 0., 5., 5.]]) , array([[9., 7., 6., 8.],
       [0., 3., 3., 8.]])]
9. >>> np.hsplit(a, (3, 4))
[array([[8., 3., 6.],
       [8., 3., 1.]]) , array([[3.],
       [2.]]) , array([[5., 5., 5., 8., 9., 7., 6., 8.],
       [9., 0., 5., 5., 0., 3., 3., 8.]])]
10. >>> np.vsplit(a, 2)
[array([[2., 9., 4., 4., 3., 0., 2., 9., 1., 2., 0., 1.]]) ,
array([[3., 4., 8., 2., 5., 8., 5., 5., 7., 7.,
       7., 8.]])]

```

Няколко масива могат да бъдат подредени заедно по различни размери, като се използват функциите **vstack ()** и **hstack ()**. С функциите **vstack ()** ④ и **hstack ()** ⑤ се подреждат масивите а и b по ред и колона. Броят на колоните при подреждане с **vstack ()** и броят на редовете при подреждане с **hstack ()** трябва да бъдат еднакви. Функция **hsplit ()** се използва, за да се раздели масив по хоризонталната му ос. Може да се посочи броя на еднакво оформените масиви, които да се върнат, или да се посочат колоните, след които трябва да се извърши разделянето. В ⑧ масива а се разделя на три подмасива. В ⑨ масив а се разделя след третата и четвъртата колона. Функция **vsplit ()** се използва, за да се раздели масив по вертикалната ос (10).

Задачи за изпълнение:

Задача 1: Напишете програма с NumPy, за тестване на даден масив за елементи за ограниченост (без безкрайност или число).

Задача 2: Напишете програма с NumPy, за да проверите дали нито един от елементите на даден масив не е нула.

Задача 3: Напишете програма с NumPy, за да създадете масив със стойности 1, 7, 13, 105 и да определите размера на паметта, заета от масива.

Задача 4: Напишете програма с NumPy, за да създадете масив от всички четни числа от 30 до 70.

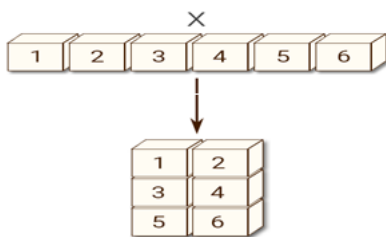
Задача 5: Напишете програма с NumPy, за да създадете матрица за идентичност 3x3.

Задача 6: Напишете програма с NumPy, за създаване на нулев вектор с размер 10 и актуализирайте шестата стойност на 11.

Задача 7: Напишете програма с NumPy, за намеране на общи стойности между два масива.

Задача 8: Напишете програма с NumPy, за получаване на уникалните елементи в масив.

Задача 9: Напишете програма с NumPy, за създаване на нова форма на масив, без да се променят данните му.



Задача 10: Напишете програма с NumPy, за създаване на 2-D масив, чийто диагонал е равен на [4, 5, 6, 8] и е запълнен с 0.