2.2 Массивы NumPy

Суммирование значений в массиве (и другие агрегатные функции)

```
import numpy as np

rng = np.random.default_rng(1)
s = rng.random(50)
print(s)

# ⇒ [0.51182162 0.9504637  0.14415961 0.94864945 0.31183145 0.42332645
#    0.82770259 0.40919914 0.54959369 0.02755911 0.75351311 0.53814331
#    0.32973172 0.7884287  0.30319483 0.45349789 0.1340417  0.40311299
#    0.20345524 0.26231334 0.75036467 0.28040876 0.48519097 0.9807372
#    0.96165719 0.72478994 0.54122686 0.2768912  0.16065201 0.96992541
#    0.51606859 0.11586561 0.62348976 0.77668311 0.6130033  0.9172977
#    0.03959288 0.52858926 0.45933588 0.06234958 0.64132817 0.85263284
#    0.59294102 0.26009745 0.83988152 0.50949588 0.51088888 0.75303021
#    0.14792204 0.81962672]

print(sum(s))
print(np.sum(s))
# ⇒ 25.98570425803768

a = np.array([[1, 2, 3, 4, 5], [6, 7, 8, 9, 10]])
print(np.sum(a))
# ⇒ 55
print(np.sum(a, axis=0)) # сумма по столбцам
# ⇒ [ 7  9 11 13 15]
print(np.sum(a, axis=1)) # сумма по строкам
# ⇒ [15 40]

print(a.min())
# ⇒ 1
print(a.min(0))
# ⇒ [1 2 3 4 5]
print(a.min(1))
# ⇒ [1 6]

# NaN - Not a number
# Более безопасный поиск, когда может присутствовать NaN
print(np.nanmin(a))
# ⇒ 1
print(np.nanmin(a, axis=0))
# ⇒ [1 2 3 4 5]
```

```
print(np.nanmin(a, axis=1))
# ⇒ [1 6]
```

Транслирование (broadcasting)

Набор правил, которые позволяют осуществлять бинарные операции с массивами разных форм и размеров.

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

print(a + b)
# ⇒ [5 6 7]

print(a + 5) # '5' транслируется в [5, 5, 5], т.е. она подстраивается под размер массива a
# ⇒ [5 6 7]

a = np.array([[0, 1, 2], [3, 4, 6]])
print(a + 5)
# ⇒ [[ 5  6  7]
#    [ 8  9 11]]

a = np.array([0, 1, 2])
b = np.array([[0], [1], [2]])
print(a + b)
# ⇒ [[0 1 2]
#    [1 2 3]
#    [2 3 4]]
```

Правила

1. Если формы массивов не совпадают в каком-то измерении, то если у массива форма равна 1, то он растягивается до соответствия второго массива.
2. Если размерности массивов отличаются, то форма массива с меньшей размерностью дополняется 1 с левой стороны.
3. Если в каком-либо измерении размеры отличаются и ни один из них не равен 1, то генерируется ошибка.

```
import numpy as np

a = np.array([1, 2, 3], [4, 5, 6])
b = np.array([5])

print(a.ndim, a.shape)
# ⇒ 2 (3,)
print(b.ndim, b.shape)
# ⇒ 1 (1,)
```

```
# a         (2, 3)
# b (1,) → (1, 1) → (2, 3)
```

Пример 1

```
import numpy as np

a = np.ones((2,3))
b = np.arange(3)

print(a)
# ⇒ [[1. 1. 1.]
#    [1. 1. 1.]]
print(b)
# ⇒ [0 1 2]

print(a.ndim, a.shape)
# ⇒ 2 (2, 3)
print(b.ndim, b.shape)
# ⇒ 1 (3,)

# (2, 3)   (2, 3)    (2, 3)
# (3, ) → (1, 3) → (2, 3)

c = a + b
print(c, c.shape)
# ⇒ [[1. 2. 3.]
#    [1. 2. 3.]] (2, 3)
```

Пример 2

```
import numpy as np

a = np.arange(3).reshape((3, 1))
b = np.arange(3)

print(a)
# ⇒ [[0]
#    [1]
#    [2]]
print(b)
# ⇒ [0 1 2]

# (3, 1)   (3, 1) → (3, 3)
# (3, ) → (1, 3) → (3, 3)

c = a + b

# [ 0 0 0 ]   [ 0 1 2 ]
# [ 1 1 1 ] + [ 0 1 2 ]
```

```
# [ 2 2 2 ]   [ 0 1 2 ]

print(c, c.shape)
# ⇒ [[0 1 2]
#    [1 2 3]
#    [2 3 4]] (3, 3)
```

Пример 3, когда не работает

```
import numpy as np

a = np.ones((3, 2))
b = np.arange(3)
print(a)
# ⇒ [[1. 1.]
#    [1. 1.]
#    [1. 1.]]
print(b)
# ⇒ [0 1 2]

# 2 (3, 2)   (3, 2)
# 1 (3, ) → (1, 3) → (3, 3)

c = a + b
# ValueError: operands could not be broadcast together with shapes (3,2) (3,)
```

Задание для самостоятельной работы

1. Что необходимо изменить в примере 3, чтобы он заработал без ошибок ?

```
import numpy as np
# Поиск среднего значения
x = np.array([[1, 2, 3, 4, 5, 6, 7, 8, 9],
                    [9, 8, 7, 6, 5, 4, 3, 2, 1]])

xmean0 = x.mean(0)
print(xmean0)
# ⇒ [5. 5. 5. 5. 5. 5. 5. 5. 5.]

# Центрирование
xcenter0 = x - xmean0
print(xcenter0)
# ⇒ [[-4. -3. -2. -1.  0.  1.  2.  3.  4.]
#    [ 4.  3.  2.  1.  0. -1. -2. -3. -4.]]

xmean1 = x.mean(1)
print(xmean1)
# ⇒ [5. 5.]

xmean1 = xmean1[:, np.newaxis] # повернули строку в столбец
```

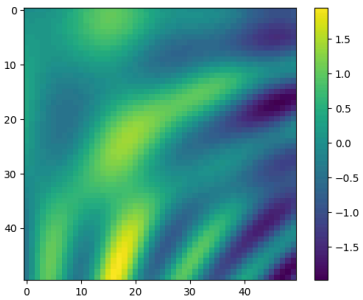
```
xcenter1 = x - xmean1
print(xcenter1)
# ⇒ [[-4. -3. -2. -1.  0.  1.  2.  3.  4.]
#    [ 4.  3.  2.  1.  0. -1. -2. -3. -4.]]
```

```
import numpy as np
import matplotlib.pyplot as plt

x = np.linspace(0, 5, 50)
y = np.linspace(0, 5, 50)[:, np.newaxis]
z = np.sin(x)**3 + np.cos(20 + y*x) * np.sin(y)
print(x)
print(y)
# ⇒ [0.         0.10204082 0.20408163 0.30612245 0.40816327 0.51020408
#    0.6122449  0.71428571 0.81632653 0.91836735 1.02040816 1.12244898
#    1.2244898  1.32653061 1.42857143 1.53061224 1.63265306 1.73469388
#    1.83673469 1.93877551 2.04081633 2.14285714 2.24489796 2.34693878
#    2.44897959 2.55102041 2.65306122 2.75510204 2.85714286 2.95918367
#    3.06122449 3.16326531 3.26530612 3.36734694 3.46938776 3.57142857
#    3.67346939 3.7755102  3.87755102 3.97959184 4.08163265 4.18367347
#    4.28571429 4.3877551  4.48979592 4.59183673 4.69387755 4.79591837
#    4.89795918 5.        ]

plt.imshow(z)
plt.colorbar()
plt.show()

print(z)
```



Сравнение

```
import numpy as np

x = np.array([1, 2, 3, 4, 5])
y = np.array([[1, 2, 3, 4, 5], [6, 7, 8, 9]])
print(x < 3)
print(np.less(x,3))
```

```
# ⇒ [ True  True False False False]

print(np.sum(x < 3)) # количество элементов
# ⇒ 2

print(np.sum(y < 4, axis=0)) # количество элементов по столбцам
# ⇒ [1 1 1 0 0]
print(np.sum(y < 4, axis=1)) # количество элементов по строкам
# ⇒ [3 0]
print(np.sum(y < 4, )) # количество элементов
# ⇒ 3
```

Задание для самостоятельной работы

2. Пример для y. Вычислить количество элементов (по обоим размерностям), значения которых больше 3 и меньше 9.

Маски - булевые массивы

```
import numpy as np
x = np.array([1, 2, 3, 4, 5])
y = print(x < 3)

print(x[x < 3])
# ⇒ [1 2]
```

Векторизация индекса

```
import numpy as np
x = np.array([0, 1, 2, 3, 4, 5, 6, 7, 8, 9])
index = [1, 5, 7]

print(x[index])
# ⇒ [1, 5, 7]

index = [[1, 5, 7], [2, 4, 8]]
print(x[index])
# ⇒ [[1 5 7]
#    [2 4 8]]
```

Форма результата отражает форму массива индексов, а не форму исходного массива.

```
import numpy as np

x = np.arange(12).reshape((3, 4))
print(x)
# ⇒ [[ 0  1  2  3]
#    [ 4  5  6  7]
#    [ 8  9 10 11]]
print(x[2])
```

```
# ⇒ [ 8  9 10 11]

print(x[2, [2, 0, 1]])
# ⇒ [10  8  9]

print(x[1:, [2, 0, 1]])
# ⇒ [[ 6  4  5]
#    [10  8  9]]
```

Замена элементов массива

```
import numpy as np

x = np.arage(10)
i = np.array([2, 1, 8, 4])

print(x)
# ⇒ [0 1 2 3 4 5 6 7 8 9]
x[i] = 999
print(x)
# ⇒ [0 999 999 3 999 5 6 7 999 9]
```

Сортировка массивов

```
import numpy as np

x = [3, 2, 3, 5, 2, 6, 7, 3, 6, 3, 2]
print(sorted(x))
print(np.sort(x))
# ⇒ [2, 2, 2, 3, 3, 3, 3, 5, 6, 6, 7]
```

Структурированные массивы

```
import numpy as np
data = np.zeros(4, dtype={'names': ('name', 'age'),
                          'formats': ('U10', 'i4')
})
print(data.dtype)
# ⇒ [('name', '<U10'), ('age', '<i4')]

name = ['name1', 'name2', 'name3', 'name4']
age = [10, 20, 30, 40]

data['name'] = name
data['age'] = age

print(data)
# ⇒ [('name1', 10) ('name2', 20) ('name3', 30) ('name4', 40)]
```

```
print(data['age'] > 20)
# ⇒ [False False  True  True]

print(data[data['age'] > 20]['name'])
# ⇒ ['name3' 'name4']
```

Массивы записи

```
import numpy as np

data = np.zeros(4, dtype={'names': ('name', 'age'),
                          'formats': ('U10', 'i4')
})

name = ['name1', 'name2', 'name3', 'name4']
age = [10, 20, 30, 40]

data['name'] = name
data['age'] = age

data_rec = data.view(np.recarray)
print(data_rec )
# ⇒ [('name1', 10) ('name2', 20) ('name3', 30) ('name4', 40)]
print(data_rec[0])
# ⇒ ('name1', 10)
```