2.1 Вводное в NumPy

Типы данных Pytnon

- **Динамическая типизация** в Python — это возможность языка автоматически определять тип данных переменной в момент присваивания значения, без необходимости явно указывать тип.

int, str, bool

```
import sys

x = 1
print(type(x))
# => <class 'int'>
print(sys.getsizeof(x))
# => 28

x = "hello"
print(type(x))
# => <class 'str'>

x = True
print(type(x))
# => <class 'bool'>
```

Плюсы	Минусы
Гибкость и простота	Ошибки типов в runtime
Быстрое прототипирование	Сложность в больших проектах
Меньше boilerplate-кода	Меньше поддержки со стороны IDE
	Сложность рефакторинга

list (список)

```
import sys

l1 = list([])
print(sys.getsizeof(l1))
# => 56

l2 = list([1, 2, 3])
print(sys.getsizeof(l2))
# => 88

l3 = list([1, "2", True)]
print(sys.getsizeof(l3))
# => 88
```

Чем плох **list** ?
Так как
list может хранить разные типы данных, то страдает производительность.
список (list) в Python не всегда подходит для работы с большими данными.

Array (массив)

```
import array

a1 = array.array('i', [1, 2, 3])
print(sys.getsizeof(a1))
# => 92
print(type(a1))
# => <class 'array.array'>
```

Array работает только с элементами одного типа данных.

Можно оптимизировать способ хранения данных.

Почему не устраивает array ? Он делает фокусировку на способе хранения элементов, а в NumPy позволяет не просто эффективно хранить данные, но и с этими данными производить какие-то операции.

Задания для самостоятельной работы

1. Какие еще существуют коды типов?
2. Напишите код, подобный приведенному выше, но с другим типом.

NumPy массивы

Одномерные массивы

```
import numpy as np

a = np.array([1, 2, 3, 4, 5])
print(type(a), a)
# => <class 'numpy.ndarray'> [1 2 3 4 5]

# Нельзая хранить разные типы данных, numpy приведет их к одному
# "Повыщающее" приведение типов
a = np.array([1.23, 2, 3, 4, 5])
print(type(a), a)
# => <class 'numpy.ndarray'> [1.23 2.   3.   4.   5.  ]

# Если хотим какого-определенного типа
a = np.array([1.23, 2, 3, 4, 5], dtype=int)
print(type(a), a)
# => <class 'numpy.ndarray'> [1 2 3 4 5]
```

Многомерные массивы

```
import numpy as np

a = np.array([range(i, i + 3) for i in [2, 4, 6]])
print(type(a))
# => <class 'numpy.ndarray'>
print(type(a), a
```

```
# => <class 'numpy.ndarray'> [[2 3 4]
#                             [4 5 6]
#                             [6 7 8]]
```

Массивы определенного типа

Гораздо эффективнее, чем любым другим создание любым другим образом

```
import numpy as np

a = np.zeros(10, dtype=int)
print(type(a), a)
# => <class 'numpy.ndarray'> [0 0 0 0 0 0 0 0 0 0]

a = np.ones((3, 5), dtype=float)
print(type(a), a)
# => <class 'numpy.ndarray'> [[1. 1. 1. 1. 1.]
#                                                  [1.
#                             [1. 1. 1. 1. 1.]]

a = np.full((4,5), 3.1415)
print(type(a), a)
# => <class 'numpy.ndarray'> [[3.1415 3.1415 3.1415 3.1415 3.141
#                             [3.1415 3.1415 3.1415 3.1415 3.141
#                             [3.1415 3.1415 3.1415 3.1415 3.141
#                             [3.1415 3.1415 3.1415 3.1415 3.141

a = np.arange(0, 20, 2)
print(type(a), a)
# => <class 'numpy.ndarray'> [ 0  2  4  6  8 10 12 14 16 18]

a = np.eye(4)
print(type(a), a)
# => <class 'numpy.ndarray'> [[1. 0. 0. 0.]
#                             [0. 1. 0. 0.]
```

```
#                                 [0. 0. 1. 0.]
#                                 [0. 0. 0. 1.]]
```

Задания для самостоятельной работы

1. Напишите код для создания массива с 5 значениями, располагающимися через равные интервалы в диапазоне от 0 до 1.
2. Напишите код для создания массива с 5 равномерно распределёнными случайными значениями в диапазоне от 0 до 1.
3. Напишите код для создания массива с 5 нормально распределёнными случайными значениями с мат. ожиданием = 0 и дисперсией = 1.
4. Напишите код для создания массива с 5 случайными целыми числами в [0, 10).

Массивы с рандомными значениями

```
import numpy as np

# Установка начального значения для генерации рандомных чисел
np.random.seed(1)

x1 = np.random.randint(10, size=3)
print(x1)
# => [9 8 6]

x2 = np.random.randint(10, size=10)
print(x2)
# => [5 7 3 4 0 8 4 5 7 4]

x3 = np.random.randint(10, size=(3, 2))
print(x3)
# => [[9 6]
#     [7 1]
#     [3 9]]
```

```
x4 = np.random.randint(10, size=(3, 2, 1))
print(x4)
# => [[[0 4]
#      [4 8]]
#     [[5 7]
#      [7 9]]
#     [[6 5]
#     [5 2]]]
```

Свойства массивов

```
import numpy as np

# Число размерностей | Размер каждой размерности | Общий размер
x1 = np.random.randint(10, size=3)
x2 = np.random.randint(10, size=10)
x3 = np.random.randint(10, size=(3, 2))
x4 = np.random.randint(10, size=(3, 2, 1))

print(x1.ndim, x1.shape, x1.size)
print(x2.ndim, x2.shape, x2.size)
print(x3.ndim, x3.shape, x3.size)
print(x4.ndim, x4.shape, x4.size)

# => x1 : 1 (3,) 3
#    x2 : 1 (10,) 10
#    x3 : 2 (3, 2) 6
#    x4 : 3 (3, 2, 1) 6
```

Получение доступа к элементам массива в одномерном массиве через индексы

```
import numpy as np

# Индекс (с 0)

a = np.array([1, 2, 3, 4, 5])
print(a[0])
# => 1

print(a[-2])
# => 4

a[1] = 20
print(a)
# => [1, 20, 3, 4, 5]
```

Получение доступа к элементам массива в многомерном массиве через индексы

```
import numpy as np

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

print(a[0,0])
# => 1

print(a[-1,-1])
# => 4

a[1,0] = 100
print(a)
```

```
# => [[1 2]
#           [100 4]]
```

Изменение типа данных массива

```
import numpy as np

# После создания массива нельзя изменить тип данных
a = np.array([1, 2, 3, 4])
b = np.array([1.0, 2, 3, 4])

print(a)
# => [1 2 3 4]
print(b)
# => [1. 2. 3. 4.]

a[0] = 10
print(a)
# => [10  2  3  4]

a[0] = 10.123
print(a)
# = > [10  2  3  4]
```

Срез массива

```
import numpy as np
# [start:finish:step]

a = np.array([1, 2, 3, 4, 5, 6])
print(a[0:3:1])
print(a[:3])
# => [1 2 3]
```

```
print(a[3:])
# => [4 5 6]

print(a[1:5])
print(a[1:-1])
# => [2 3 4 5]

print(a[1::2])
# => [2 4 6]

print(a[::1])
# => [1 2 3 4 5 6]

print(a{::-1])
# => [6 5 4 3 2 1]
```

Задания для самостоятельной работы

1. Написать код для создания срезов массива 3 на 4
    a. первые две строки и три столбца
    b. первые три строки и второй столбец
    c. все строки и столбцы в обратном порядке
    d. второй столбец
    e. третья строка
2. Продемонстрировать, как сделать срез-копию

Изменение размера массива

```
import numpy as np

a = np.arange(1, 13)
print(a)
# => [1 2 3 4 5 6 7 8 9 10 11 12]
```

```
print(a.reshape(2, 6))
# => [[ 1  2  3  4  5  6]
#     [ 7  8  9 10 11 12]]

print(a.reshape(3, 4))
# => [[ 1  2  3  4]
#     [ 5  6  7  8]
#     [ 9 10 11 12]]
```

Задания для самостоятельной работы

1. Продемонстрировать использование newasix для получения вектора-столбца и вектора-строки
2. Разобраться как работает метод dstack, split, vsplit, hsplit, dsplit

Способы объединения массивов

```
import numpy as np

x = np.array([1, 2, 3])
y = np.array([4, 5])
z = np.array([6])

print(np.concatenate([x, y, z]))
# => [1 2 3 4 5 6]

x = np.array([1, 2, 3])
y = np.array([4, 5, 6])

# Вертикальное склеивание
r1 = np.vstack([x, y])
print(r1)
# => [[1 2 3]
#     [4 5 6]]
```

```
# Горизонтальное склеивание
r2 = np.hstack([x, y])
print(r2)
# => [1 2 3 4 5 6]

r3 = np.hstack([r1, r1])
print(r3)
# => [[1 2 3 1 2 3]
#     [4 5 6 4 5 6]]
```

Вычисления с массивами

Векторизованная операция

Это операции, которые независимо применяются к каждому элементу массива.

```
import numpy as np

x = np.arange(10)
print(x)
# => [0 1 2 3 4 5 6 7 8 9]

print(x * 2 + 1)
# => [ 1  3  5  7  9 11 13 15 17 19]
```

Универсальные функции

Это функции, которые выполняют поэлементные операции над данными массива

```
import numpy as np
```

```
print(np.multiply(x, 2))
# => [ 0  2  4  6  8 10 12 14 16 18]

print(np.add(np.multiply(x, 2), 1))
# => [ 1  3  5  7  9 11 13 15 17 19]
```

Задания для самостоятельной работы

1. Привести пример использования всех универсальных функций (-; - /; //; **; %)

Особенность независимых функций

```
import numpy as np

x = np.arange(5)
y = np.empty(5)
print(np.multiply(x, 10, out=y))
# => [ 0. 10. 20. 30. 40.]
print(y)
# => [ 0. 10. 20. 30. 40.]
```

Свертка массива к единственному элементу по каким-либо признакам

```
import numpy as np

x = np.arange(1, 5)
print(x)
# => [ 1 2 3 4 5]

print(np.add.reduce(x))
# => 10
```

```
print(np.add.accumulate(x))
# => [ 1  3  6 10]
```

Векторные произволения

```
import numpy as np

x = np.arange(1, 10)
print(np.add.outer(x, x))

# => [[ 2  3  4  5  6  7  8  9 10]
#     [ 3  4  5  6  7  8  9 10 11]
#     [ 4  5  6  7  8  9 10 11 12]
#     [ 5  6  7  8  9 10 11 12 13]
#     [ 6  7  8  9 10 11 12 13 14]
#     [ 7  8  9 10 11 12 13 14 15]
#     [ 8  9 10 11 12 13 14 15 16]
#     [ 9 10 11 12 13 14 15 16 17]
#     [10 11 12 13 14 15 16 17 18]]

print(np.multiply.outer(x, x))

# => [[ 1  2  3  4  5  6  7  8  9]
#     [ 2  4  6  8 10 12 14 16 18]
#     [ 3  6  9 12 15 18 21 24 27]
#     [ 4  8 12 16 20 24 28 32 36]
#     [ 5 10 15 20 25 30 35 40 45]
#     [ 6 12 18 24 30 36 42 48 54]
#     [ 7 14 21 28 35 42 49 56 63]
#     [ 8 16 24 32 40 48 56 64 72]
#    [ 9 18 27 36 45 54 63 72 81]]
```