Системы обработки мультимедиаданных

Матричные операции и обработка изображений в Python

д.т.н., доцент Вашкевич М. И.

vashkevich@bsuir.by

Белорусский государственный университет информатики и радиоэлектроники

Кафедра электронных вычислительных средств

План лекции

1. Инструментарий
2. Jupyter Notebook
3. Базовые типы Python
4. Массивы NumPy
5. Полезные функции работы с массивами
6. Базовые операции с изображениями

Инструментарий

Visual Studio Code

NumPy

SciPy

python

matplotlib

jupyter

Блокноты Jupyter

➢ Блокнот `Jupyter` – мощное средство, используемое на этапе разработки алгоритмов и моделей на `Python`.

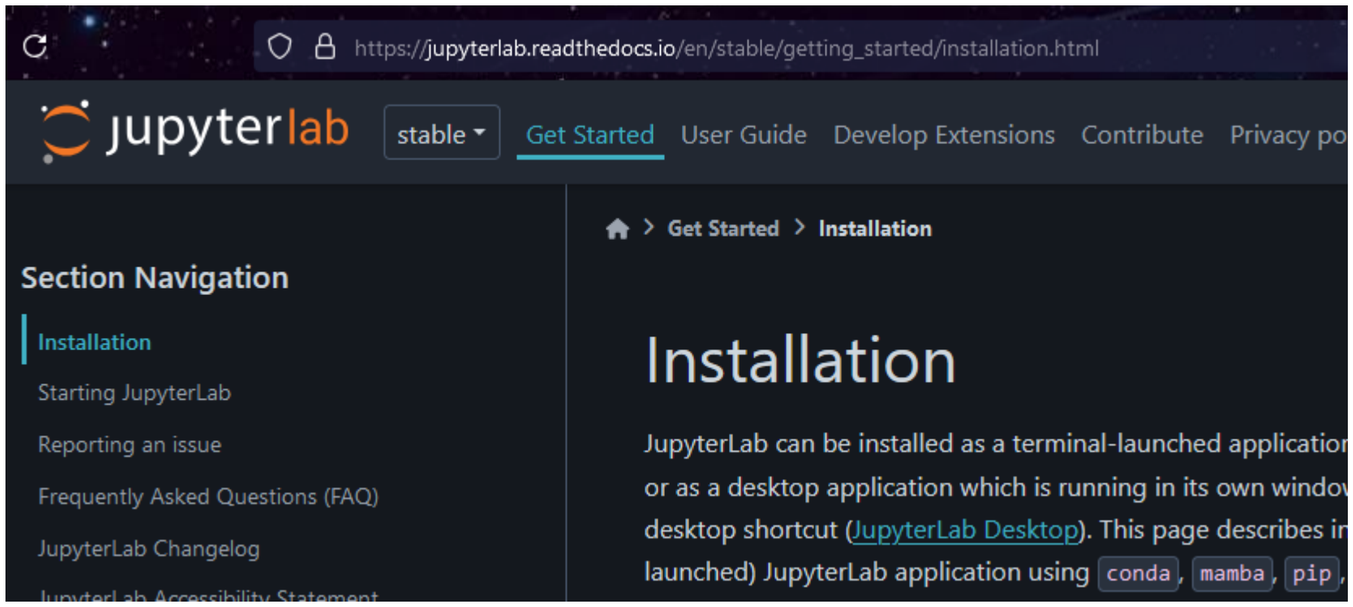
➢ Блокнот `Jupyter` выглядит, как страница в браузере, из которой можно запускать код `Python` в диалоговом режиме. Этот код обрабатывается *ядром* (`IPython kernel`). Ядро выполняет код, а также обеспечивает коммуникацию между пользовательским интерфейсом и вычислительным «движком» (*computational engine*).

➢ Блокнот хранит состояния ядра (все переменные, массивы) в памяти до завершения или перезапуска его работы.

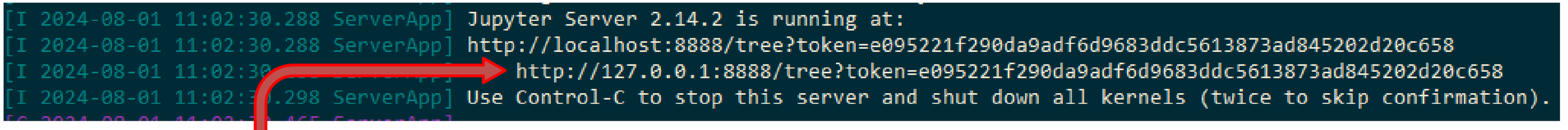
➢ Основная единица взаимодействия с блокнотом – *ячейка* (`cell`) – прямоугольное поле на странице, в которое можно вводить код.

Запуск блокнота Jupyter

1) Установка

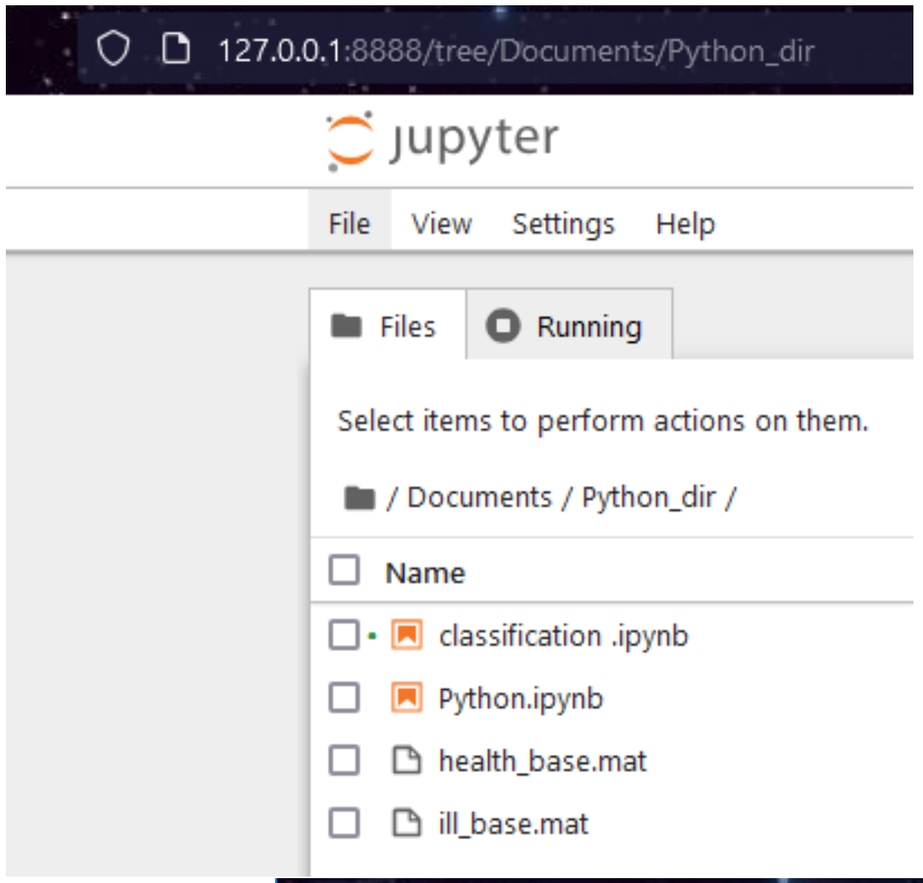


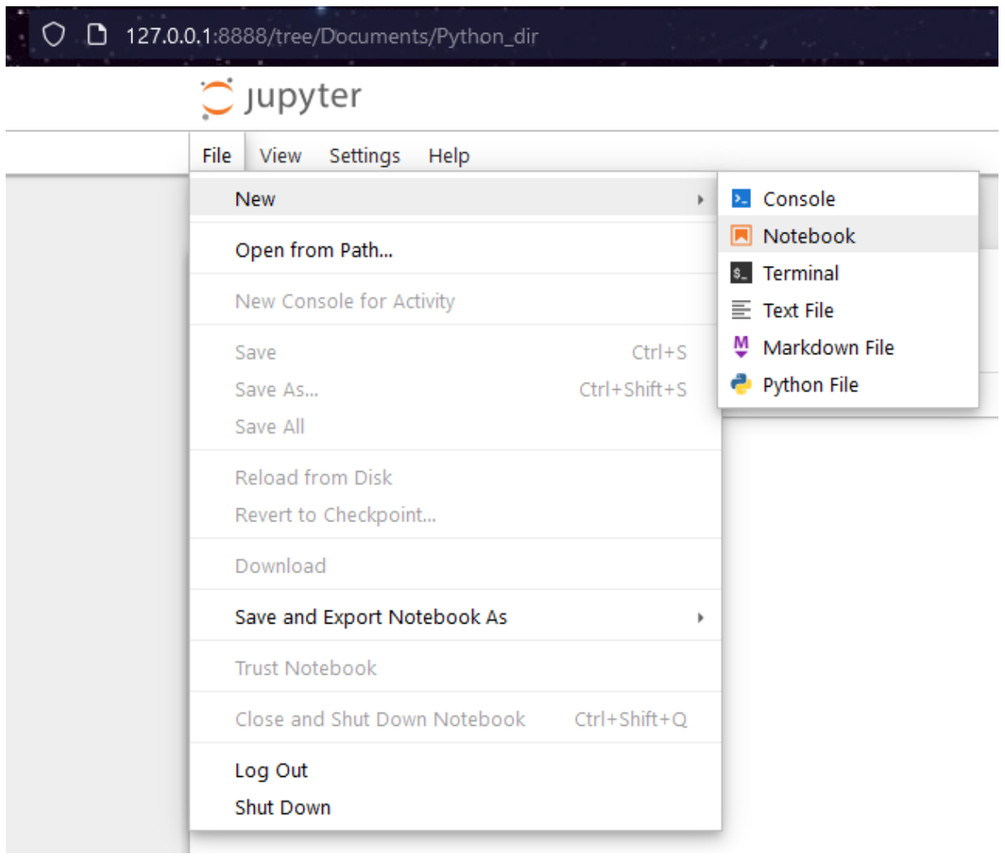
2) Запуск ядра из командной строки (cmd.exe) → `$ jupyter notebook`

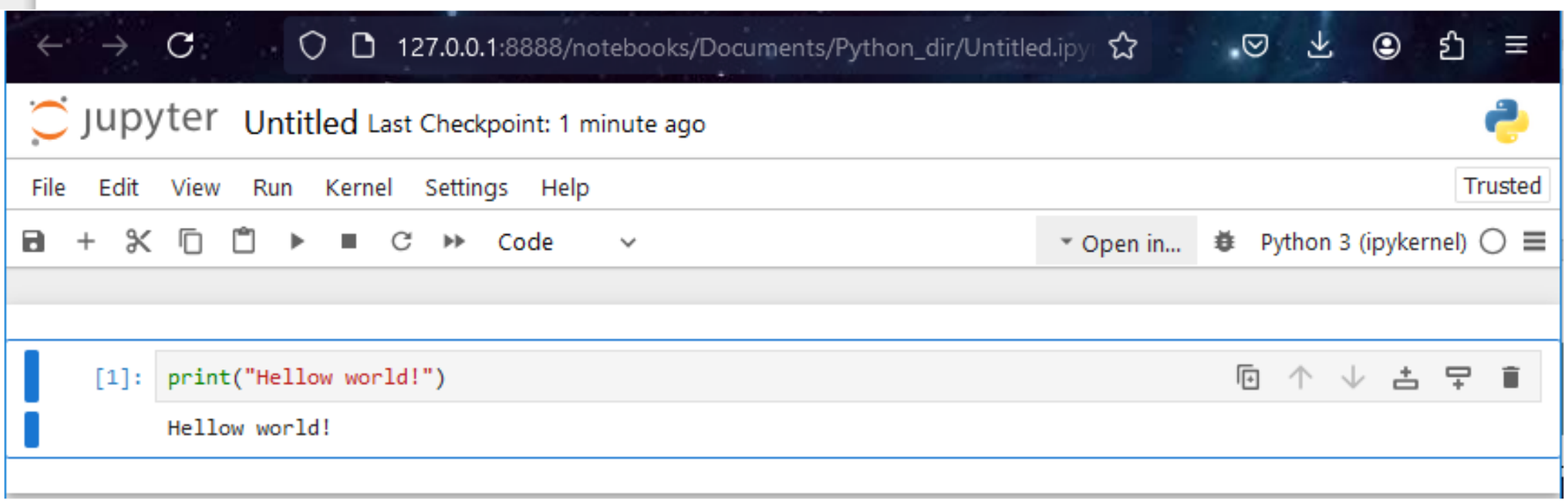


3) Скопировать адрес и вставить в браузер

Внешний вид блокнота Jupyter

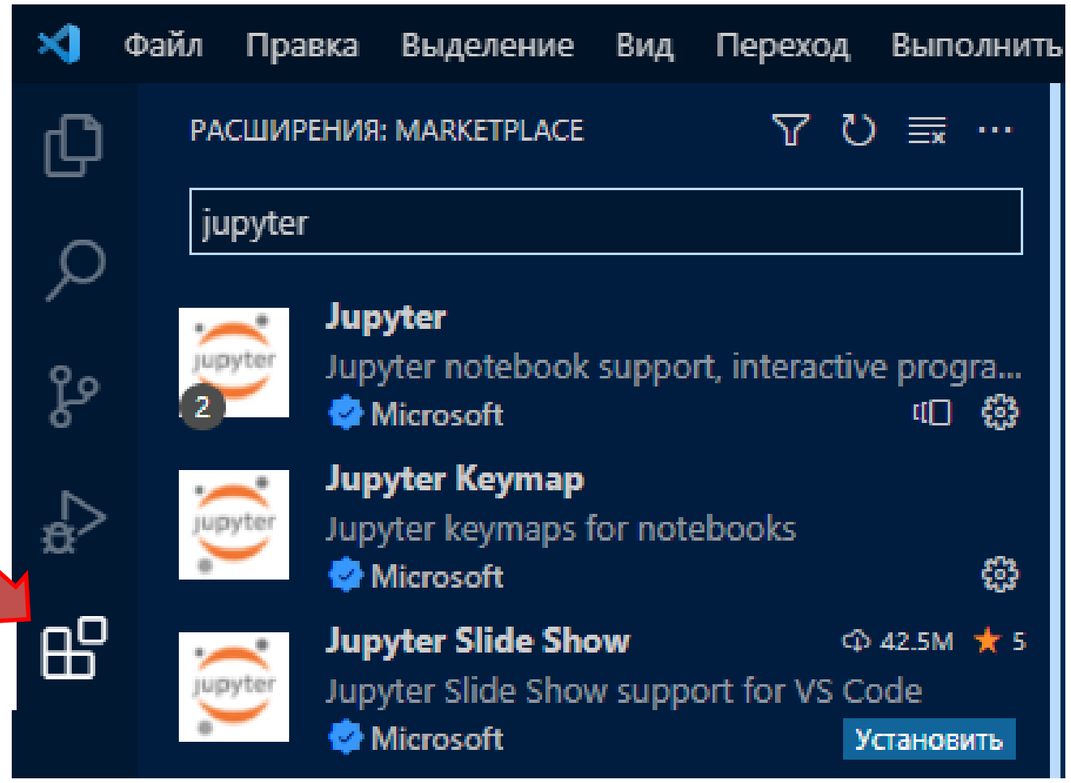




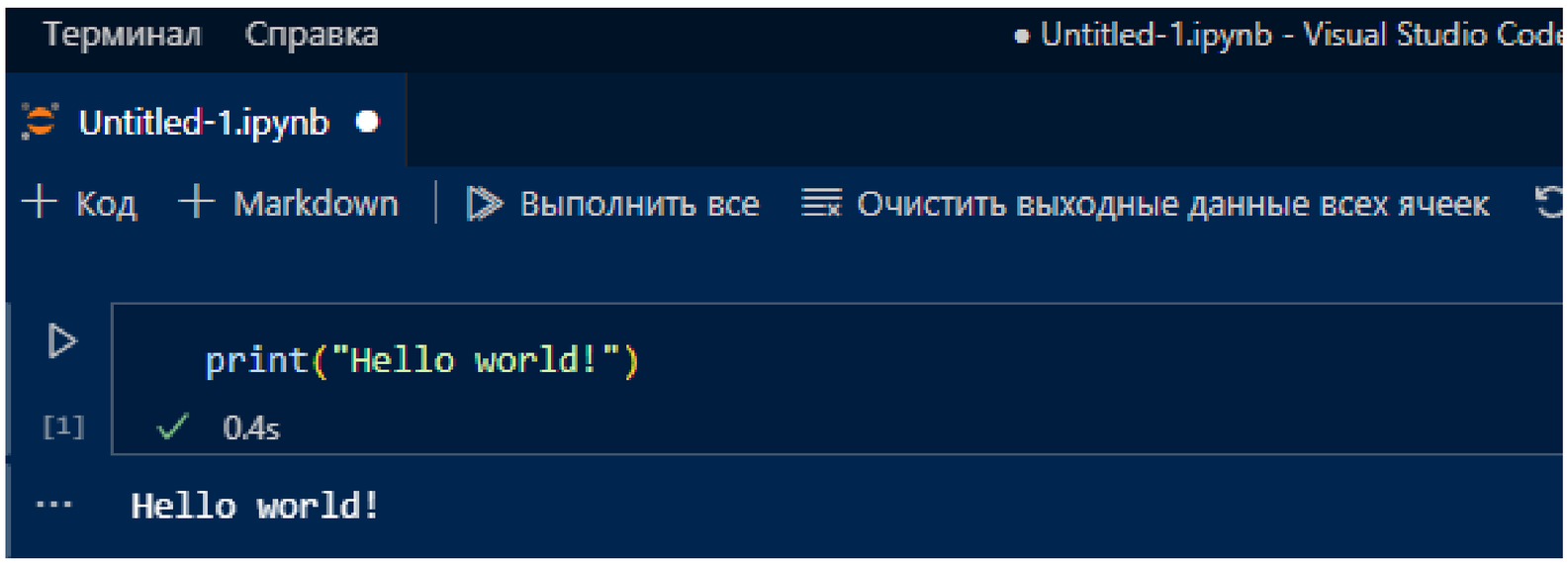


Блокноты Jupyter в VS Code

1) Установить расширение `Jupyter`



2) *Вид → Палитра команд*
→ Create: New Jupyter Notebook



Что необходимо знать?

- Структуры данных для хранения изображения
- Как читать/записывать цифровые изображения
- Как выполнять действия над цифровыми изображениями

Что необходимо знать?

- Структуры данных для хранения изображения
  - Массивы `NumPy`
- Как читать/записывать цифровые изображения
  - Библиотека `Matplotlib`
- Как выполнять действия над цифровыми изображениями
  - Библиотеки `NumPy, SciPy, CV2, scikit-image`

Почему Python?

Python – популярный язык программирования с открытым кодом, применяемый для написания автономным программ и сценарных приложений в широком разнообразии предметных областей

Особенности Python

– простота (язык высокого уровня)
– портируемость (платформонезависимость)
– интерпретируемый язык
– модульность
– читабельность кода (читаемость имеет значение – *Readability counts*)
– динамическая типизация
– поддержка ООП

Структура программы на Python

- **Программы** состоят из **модулей**
- **Модули** содержат **операторы (statements)**
- **Операторы** содержат **выражения**
- **Выражения** создают и обрабатывают объекты
- **Выражения** включают
  - назначение переменных, вызовы функций
  - управляющие конструкции, доступ к модулям
  - встроенные функции, встроенные объекты
  - вывод на экран

Встроенные объекты Python

- Числа: целые числа, числа с плавающей запятой
- Словари
- Строки
- Списки (lists)
- Кортежи (tuples)
- Файлы

Переменные

– ненужно объявлять заранее

– создается при первом присвоении значения

– заменяется их значениями при использовании в выражениях

– должно быть назначено перед использованием

Пример на Python

```
>>> a = 12
>>> type(a)
<class ‘int’>
>>> b = 12.5
>>> type(b)
<class ‘float’>
```

Строки в Python

Задание строк

Строки могут быть записаны с использованием одинарных или двойных кавычек

Пример на Python

```
>>> s1 = ‘Это строка Python’
>>> type(s1)
<class ‘str’>
>>> s2 = “и это тоже”
>>> type(s2)
<class ‘string’>
```

Операции со строками

Конкатенация (+)

Вычисление длины (len)

Повторение (*)

Индексирование/”вырезание” ([])

Пример на Python

```
s = 'Знание'
t = 'сила'
res = s + ' ' + t # 'Знание сила'
l = len(res)      # 11
u = t * 2         # 'ЗнаниеЗнание'
a = res[0]        # ‘З’
b = res[-1]       # ‘а’
c = res[5:]       # ‘е сила’
```

Словари (dict)

Словари также называют ассоциативными массивами, хэш-таблицами (*hashmaps*) и поисковыми таблицами (*lookup tables*).

Словарь – хранилище произвольного числа объектов, каждый из которых идентифицируется уникальным **ключом** словаря.

Тип данных `dict` встроен в ядро языка Python.

Создание словаря

```
>>> D1 = dict({'Вася':1520,
               'Петя':1700,
               'Коля':1350})
>>> type(D1)
<class ‘dict’>
>>> D2 = {'Вася':1520,
          'Петя':1700,
          'Коля':1350}
>>> type(D2)
<class ‘dict’>
```

Использование словаря

```
>>> D1[‘Вася’]
1520
>>> len(D1)
3
>>> D1.items()
dict_items([('Вася', 1520),
('Петя', 1700), ('Коля', 1350)])
>>> D1.values()
dict_values([1520, 1700, 1350])
```

Массивоподобные (arraylike) структуры данных

`list` – списки – изменяемые динамические массивы

`tuple` – кортежи – неизменяемые контейнеры

`array.array` – элементарные типизированные массивы

`str` – неизменяемые массивы символов Юникода

`bytes` – неизменяемые массивы одиночных байтов

`bytearray` – изменяемые массивы одиночных байтов

List: списки в Python

- Упорядоченное множество произвольных объектов
- Доступ осуществляется путем индексации
- Переменная длина (увеличивается автоматически)
- Гетерогенность (может содержать любой тип, поддающийся вложению)

Пример на Python (часть 1)

```
>>> s = ‘a’,‘b’,‘c’]
>>> t = [1, 2, 3]
>>> u = s + t
['a', 'b', 'c', 1, 2, 3]
>>> n = len(u)
6
>>> u[2]=0  # list можно менять
['a', 'b', 0, 1, 2, 3]
```

(часть 2)

```
>>> del u[1]
['a', 0, 1, 2, 3]
>>> u.append('Много')
['a', 0, 1, 2, 3, 'Много']
>>> u[0:3] = ['x','y','z']
['x', 'y', 'z', 2, 3, 'Много']
>>> u.extend(['r','s'])
['x', 'y', 'z', 2, 3, 'Много',
'r', 's']
```

Tuple: кортежи в Python

- Встроены в ядро Python (как и `list`)
- Кортежи не изменяются
- Все элементы в кортеже должны быть определены во время создания

Пример на Python (часть 1)

```
# Простой кортеж
>>> tup = (4,5,6)

# Вложенный кортеж
>>> nested_tup = (4,5,6), (7,8)
((4, 5, 6), (7, 8))
# Преобразование списка в кортеж
>>> set_01 = [7, 0, 1]
>>> tup_01 = tuple(set_01)
(7, 0, 1)
```

(часть 2)

```
# Кортеж нельзя менять!
>>> tup=tuple(['foo',[1,2], True])
>>> tup[2] = False
TypeError: 'tuple' object does
not support item assignment

# Можно менять изменяемый объект
кортежа
>>> tup[1].append(3)
('foo', [1, 2, 3], True)
```

Массивы NumPy

- ***Массив NumPy*** – упорядоченная коллекция однородных данных.
- Создание массива из нулей размера 2x3:

```
import numpy as np
A = np.zeros((2,3))
print(A)
```

```
[3]  ✓ 0.3s

...  [[0. 0. 0.]
      [0. 0. 0.]]
```

- `numpy.arange([start, ]stop, [step, ])` – возвращает вектор значений в пределах заданного интервала `[srart, stop]` с шагом `step`.

```
b = np.arange(10)
print(f'b = {b}')
```

```
[9]  ✓ 0.3s

...  b = [0 1 2 3 4 5 6 7 8 9]
```

Вырезка (slice) массивов NumPy

Списки

```
l1 = list(range(10))
l2 = l1[1:-1]
print('l1: %s\nl2: %s' % (l1, l2))
l1[4] = -1
print('l1_: %s\nl2_: %s' % (l1, l2))
```

Массивы

```
a1 = np.arange(10)
a2 = a1[1:-1]
print('a1: %s\na2: %s' % (a1, a2))
a1[4] = -1
print('a1_: %s\na2_: %s' % (a1, a2))
```

Вырезка (slice) массивов NumPy

Списки

```
l1 = list(range(10))
l2 = l1[1:-1]
print('l1: %s\nl2: %s' % (l1, l2))
l1[4] = -1
print('l1_: %s\nl2_: %s' % (l1, l2))
```

[14] ✓ 0.3s

```
l1: [0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
l2: [1, 2, 3, 4, 5, 6, 7, 8]
l1_: [0, 1, 2, 3, -1, 5, 6, 7, 8, 9]
l2_: [1, 2, 3, 4, 5, 6, 7, 8]
```

Массивы

```
a1 = np.arange(10)
a2 = a1[1:-1]
print('a1: %s\na2: %s' % (a1, a2))
a1[4] = -1
print('a1_: %s\na2_: %s' % (a1, a2))
```

[15] ✓ 0.3s

```
a1: [0 1 2 3 4 5 6 7 8 9]
a2: [1 2 3 4 5 6 7 8]
a1_: [ 0  1  2  3 -1  5  6  7  8  9]
a2_: [ 1  2  3 -1  5  6  7  8]
```

- Вырезка из массива ≠ созданию копии массива!
- Для создания копии необходимо использовать метод `.copy()`:

```
a2 = a1.copy()
```

Вырезка из списка = новый объект, а из массива нет

Списки

```
l1 = list(range(10))
l2 = l1[1:-1]
```

```
[0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
```

l1

```
[1, 2, 3, 4, 5, 6, 7, 8]
```

l2

Массивы

```
a1 = np.arange(10)
a2 = a1[1:-1]
```

```
[0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
```

a1

a2

Полезные функции при работе с массивами

```
# Исходные данные
A = np.zeros((2,3))
```

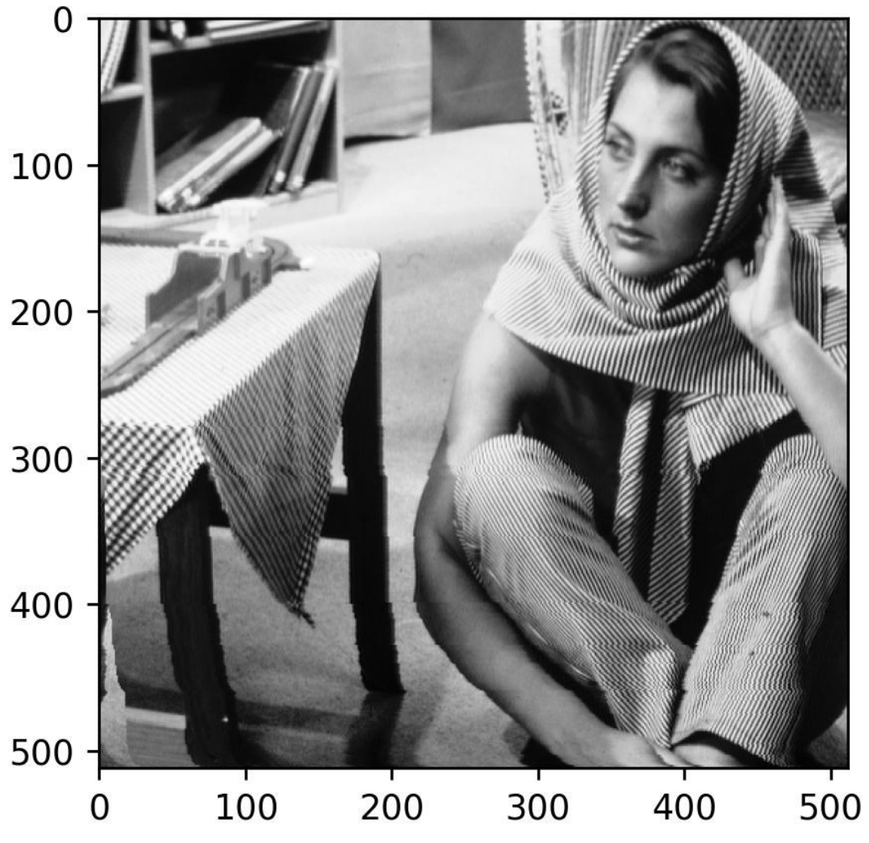
A

0	0	0
0	0	0

Функция	Результат	Примечания
`A.shape`	`(2,3)`	Картеж размерностей
`A.ndim`	`2`	Число размерностей
`A.T`	`2x3 массив`	Транспонирование
`np.append(A, 2)`	`1x7 вектор`	Вытягивание (*flatten*) матрицы по столбцам и добавление элемента.
`A = np.append(A,[[2,2,2]], axis=0)`	`3x3 массив`	Добавление значений вдоль измерения axis.
`np.concatenate( (A,A) )`	`4x3 массив`	Конкатенация массивов

Чтение изображение в Python

```
>>> import matplotlib.pyplot as plt
>>> my_img = plt.imread("img\\barbara.tif")
>>> print ('Shape = ', my_img.shape)
  Shape = (512, 512)
>>> print ('Type = ', my_img.dtype)
  Type = uint8
>>> fig = plt.figure()
>>> plt.imshow(my_img, cmap='gray')
```



Изображения в Python

Выведем первые 100 строк и 200 столбцов изображения.

```
>>> A = my_img[:100, :200]
>>> fig = plt.figure()
>>> plt.imshow(A, cmap='gray')
```