Лекция 5.1. Кортеж, список, словарь, множество. Библиотеки pandas и numpy. Массивы. Работа с одномерными и двумерными массивами

**План лекции**

[**Введение** 2](#_Toc167890352)

[**1.** **Комплексные типы данных или коллекции** 4](#_Toc167890353)

[**List.** 4](#_Toc167890354)

[**Tuple.** 8](#_Toc167890355)

[**Dictionary.** 11](#_Toc167890356)

[**Set и frozenset.** 13](#_Toc167890357)

[**2.** **Библиотеки numpy и pandas. Массивы** 15](#_Toc167890358)

[**NumPy** 15](#_Toc167890359)

[**Pandas** 33](#_Toc167890360)

[**3.** **Визуализация данных в Python** 46](#_Toc167890361)

[**Заключение** 48](#_Toc167890362)

[**Список литературы** 48](#_Toc167890363)

**Структуры данных. Библиотеки для работы со структурами данных**

**Цель:** формирование знаний в отношении комплексных типов данных в языке Python, их принципов работы и методов, основных библиотек для работы с комплексными типами данных.

**План лекции:**Введение.1. Комплексные типы данных или коллекции.  
2. Библиотеки numpy и pandas. Массивы  
3. Визуализация данных в Python.Заключение.Список литературы.

# **Введение**

Python — один из наиболее популярных языков программирования благодаря своей простоте, читаемости и широкому спектру применения. Одним из важнейших аспектов Python является его мощная система комплексных типов данных, также известных как коллекции. Эти структуры данных, такие как списки, кортежи, множества и словари, позволяют разработчикам эффективно хранить и обрабатывать большие объемы информации. Коллекции предоставляют гибкие и интуитивно понятные инструменты для управления данными, что делает Python идеальным выбором для задач различной сложности — от простых сценариев до сложных аналитических приложений.

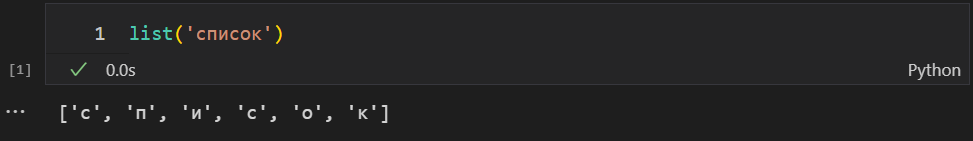
Однако, для выполнения более сложных вычислительных задач и работы с большими наборами данных, стандартных коллекций Python может быть недостаточно. В таких случаях на помощь приходят специализированные библиотеки, такие как NumPy и Pandas. NumPy предоставляет поддержку многомерных массивов и матриц, а также функции для выполнения математических операций с высокой производительностью. Pandas, в свою очередь, предлагает мощные инструменты для анализа данных и работы с табличными структурами, что делает его незаменимым в области данных и науки о данных. В этой статье мы рассмотрим основы работы с коллекциями в Python и погрузимся в возможности, которые предоставляют библиотеки NumPy и Pandas.

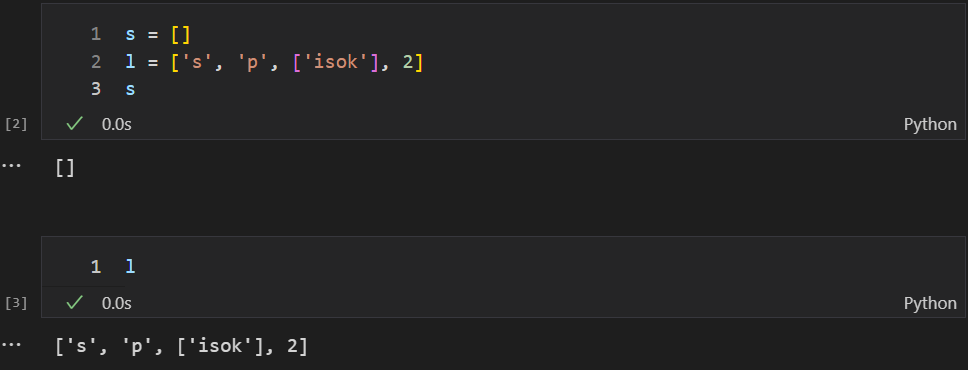
# **Комплексные типы данных или коллекции**

В других языках программирования, например С++, комплексным типом данных называется массив. В языке Python существует несколько комплексных типов данных, каждый из которых имеет свои отличия и особенности: set, list, tuple, dictionary, frozenset.

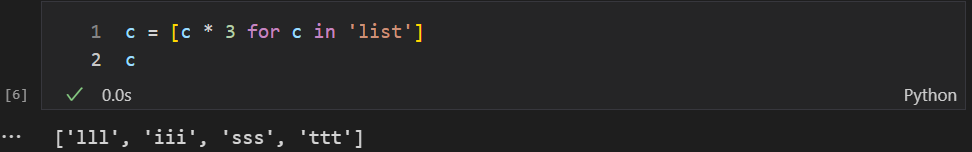
## **List.**

Списки в Python - упорядоченные изменяемые коллекции объектов произвольных типов (почти как массив, но типы могут отличаться).

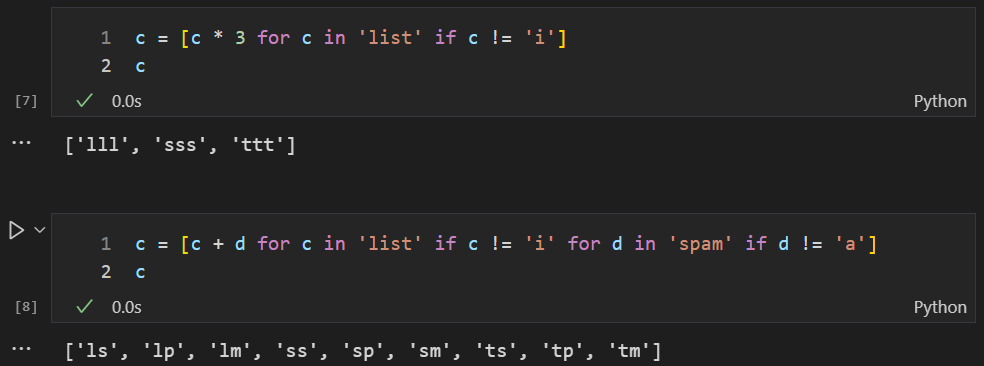
Чтобы использовать списки, их нужно создать. Создать список можно несколькими способами. Например, можно обработать любой итерируемый объект (например, строку) встроенной функцией list:

Список можно создать и при помощи литерала:

Как видно из примера, список может содержать любое количество любых объектов (в том числе и вложенные списки), или не содержать ничего.

И еще один способ создать список - это генераторы списков. Генератор списков - способ построить новый список, применяя выражение к каждому элементу последовательности. Генераторы списков очень похожи на цикл for.

Возможна и более сложная конструкция генератора списков:

Но в сложных случаях лучше пользоваться обычным циклом for для генерации списков.

**Встроенные функции, выполняющие преобразование типов**

**Функции и методы списков**

Создать создали, теперь нужно со списком что-то делать. Для списков доступны основные встроенные функции, а также методы списков.

*bool(x)* - преобразование к типу bool, использующая стандартную процедуру [проверки истинности](https://pythonworld.ru/osnovy/instrukciya-if-elif-else-proverka-istinnosti-trexmestnoe-vyrazhenie-ifelse.html). Если х является ложным или опущен, возвращает значение False, в противном случае она возвращает True.

*bytearray([источник [, кодировка [ошибки]]])* - преобразование к [bytearray](https://pythonworld.ru/tipy-dannyx-v-python/bajty-bytes-i-bytearray.html#bytearray). Bytearray - изменяемая последовательность целых чисел в диапазоне 0≤X<256. Вызванная без аргументов, возвращает пустой массив байт.

*bytes([источник [, кодировка [ошибки]]])* - возвращает объект типа [bytes](https://pythonworld.ru/tipy-dannyx-v-python/bajty-bytes-i-bytearray.html), который является неизменяемой последовательностью целых чисел в диапазоне 0≤X<256. Аргументы конструктора интерпретируются как для bytearray().

*complex([real[, imag]])* - преобразование к [комплексному числу](https://pythonworld.ru/tipy-dannyx-v-python/chisla-int-float-complex.html#complex).

*dict([object])* - преобразование к [словарю](https://pythonworld.ru/tipy-dannyx-v-python/slovari-dict-funkcii-i-metody-slovarej.html).

*float([X])* - преобразование к [числу с плавающей точкой](https://pythonworld.ru/tipy-dannyx-v-python/chisla-int-float-complex.html#float). Если аргумент не указан, возвращается 0.0.

*frozenset([последовательность])* - возвращает [неизменяемое множество](https://pythonworld.ru/tipy-dannyx-v-python/mnozhestva-set-i-frozenset.html#frozenset).

*int([object], [основание системы счисления])* - преобразование к [целому числу](https://pythonworld.ru/tipy-dannyx-v-python/chisla-int-float-complex.html#int).

*list([object])* - создает [список](https://pythonworld.ru/tipy-dannyx-v-python/spiski-list-funkcii-i-metody-spiskov.html).

*memoryview([object])* - создает объект memoryview.

*object()* - возвращает безликий объект, являющийся базовым для всех объектов.

*range([start=0], stop, [step=1])* - арифметическая прогрессия от start до stop с шагом step.

*set([object])* - создает [множество](https://pythonworld.ru/tipy-dannyx-v-python/mnozhestva-set-i-frozenset.html).

*slice([start=0], stop, [step=1])* - объект среза от start до stop с шагом step.

*str([object], [кодировка], [ошибки])* - строковое представление объекта. Использует метод \_\_str\_\_.

*tuple(obj)* - преобразование к [кортежу](https://pythonworld.ru/tipy-dannyx-v-python/kortezhi-tuple.html).

**Другие встроенные функции**

*abs(x)* - Возвращает абсолютную величину (модуль числа).

*all(последовательность)* - Возвращает True, если все элементы истинные (или, если последовательность пуста).

*any(последовательность)* - Возвращает True, если хотя бы один элемент - истина. Для пустой последовательности возвращает False.

*ascii(object)* - Как repr(), возвращает строку, содержащую представление объекта, но заменяет не-ASCII символы на экранированные последовательности.

*bin(x)* - Преобразование целого числа в двоичную строку.

*callable(x)* - Возвращает True для объекта, поддерживающего вызов (как функции).

*chr(x)* - Возвращает односимвольную строку, код символа которой равен x.

*classmethod(x)* - Представляет указанную функцию методом класса.

*compile(source, filename, mode, flags=0, dont\_inherit=False)* - Компиляция в программный код, который впоследствии может выполниться функцией eval или exec. Строка не должна содержать символов возврата каретки или нулевые байты.

*delattr(object, name)* - Удаляет атрибут с именем 'name'.

*dir([object])* - Список имен объекта, а если объект не указан, список имен в текущей локальной области видимости.

*divmod(a, b)* - Возвращает частное и остаток от деления a на b.

enumerate(iterable, start=0) - Возвращает итератор, при каждом проходе предоставляющем кортеж из номера и соответствующего члена последовательности.

*eval(expression, globals=None, locals=None)* - Выполняет строку программного кода.

*exec(object[, globals[, locals]])* - Выполняет программный код на Python.

*filter(function, iterable)* - Возвращает итератор из тех элементов, для которых function возвращает истину.

*format(value[,format\_spec])* - Форматирование (обычно [форматирование строки](https://pythonworld.ru/osnovy/formatirovanie-strok-metod-format.html)).

*getattr(object, name ,[default])* - извлекает атрибут объекта или default.

*globals()* - Словарь глобальных имен.

*hasattr(object, name)* - Имеет ли объект атрибут с именем 'name'.

*hash(x)* - Возвращает хеш указанного объекта.

*help([object])* - Вызов встроенной справочной системы.

*hex(х)* - Преобразование целого числа в шестнадцатеричную строку.

*id(object)* - Возвращает "адрес" объекта. Это целое число, которое гарантированно будет уникальным и постоянным для данного объекта в течение срока его существования.

*input([prompt])* - Возвращает введенную пользователем строку. Prompt - подсказка пользователю.

*isinstance(object, ClassInfo)* - Истина, если объект является экземпляром ClassInfo или его подклассом. Если объект не является объектом данного типа, функция всегда возвращает ложь.

*issubclass(класс, ClassInfo)* - Истина, если класс является подклассом ClassInfo. Класс считается подклассом себя.

*iter(x)* - Возвращает объект итератора.

*len(x)* - Возвращает число элементов в указанном объекте.

*locals()* - Словарь локальных имен.

*map(function, iterator)* - Итератор, получившийся после применения к каждому элементу последовательности функции function.

*max(iter, [args ...] \* [, key])* - Максимальный элемент последовательности.

*min(iter, [args ...] \* [, key])* - Минимальный элемент последовательности.

*next(x)* - Возвращает следующий элемент итератора.

*oct(х)* - Преобразование целого числа в восьмеричную строку.

*open(file, mode='r', buffering=None, encoding=None, errors=None, newline=None, closefd=True)* - Открывает файл и возвращает соответствующий поток.

*ord(с)* - Код символа.

*pow(x, y[, r])* - ( x \*\* y ) % r.

*reversed(object)* - Итератор из развернутого объекта.

*repr(obj)* - Представление объекта.

*print([object, ...], \*, sep=" ", end='\n', file=sys.stdout)* - Печать.

property(fget=None, fset=None, fdel=None, doc=None)

*round(X [, N])* - Округление до N знаков после запятой.

*setattr(объект, имя, значение)* - Устанавливает атрибут объекта.

*sorted(iterable[, key][, reverse])* - Отсортированный список.

*staticmethod(function)* - Статический метод для функции.

*sum(iter, start=0)* - Сумма членов последовательности.

*super([тип [, объект или тип]])* - Доступ к родительскому классу.

*type(object)* - Возвращает тип объекта.

*type(name, bases, dict)* - Возвращает новый экземпляр класса name.

*vars([object])* - Словарь из атрибутов объекта. По умолчанию - словарь локальных имен.

*zip(\*iters)* - Итератор, возвращающий кортежи, состоящие из соответствующих элементов аргументов-последовательностей.

**Таблица "методы списков"**

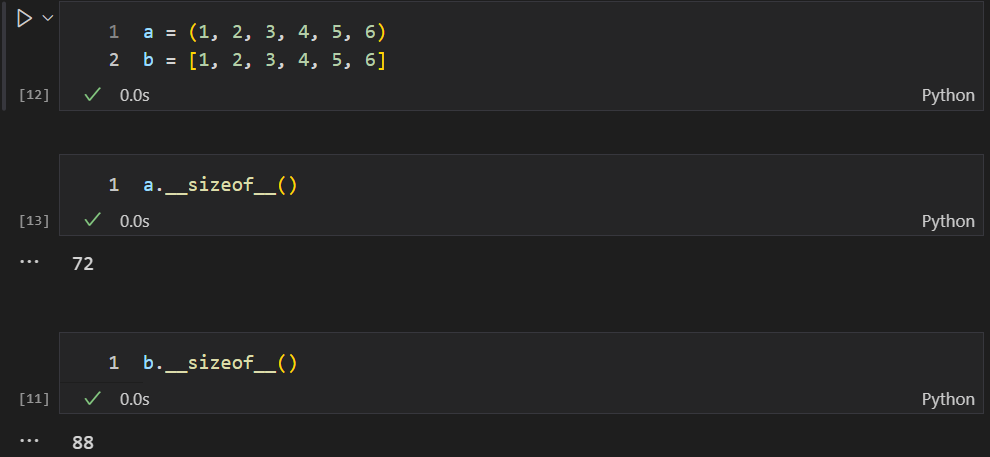
|  |  |
| --- | --- |
| Метод | Что делает |
| list.append(x) | Добавляет элемент в конец списка |
| list.extend(L) | Расширяет список list, добавляя в конец все элементы списка L |
| list.insert(i, x) | Вставляет на i-ый элемент значение x |
| list.remove(x) | Удаляет первый элемент в списке, имеющий значение x. ValueError, если такого элемента не существует |
| list.pop([i]) | Удаляет i-ый элемент и возвращает его. Если индекс не указан, удаляется последний элемент |
| list.index(x, [start [, end]]) | Возвращает положение первого элемента со значением x (при этом поиск ведется от start до end) |
| list.count(x) | Возвращает количество элементов со значением x |
| list.sort([key=функция]) | Сортирует список на основе функции |
| list.reverse() | Разворачивает список |
| list.copy() | Поверхностная копия списка |
| list.clear() | Очищает список |

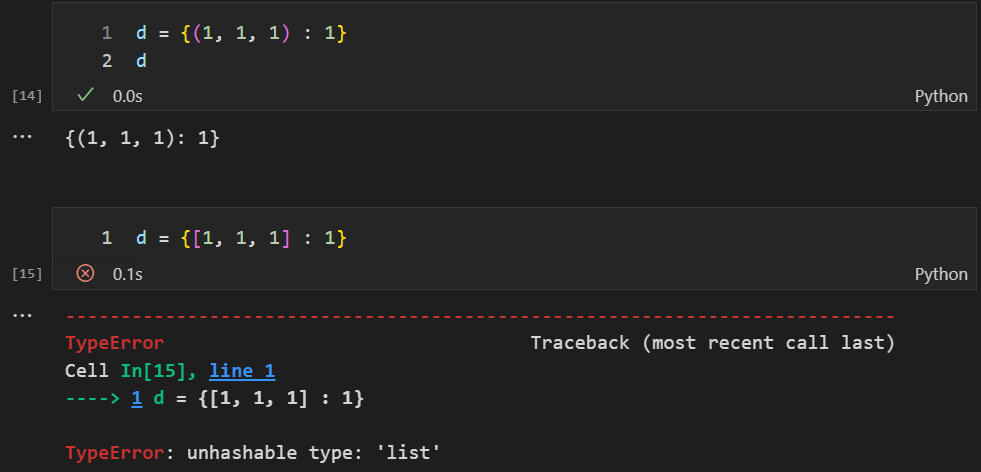
## **Tuple.**

Кортеж, по сути - неизменяемый [список](https://pythonworld.ru/tipy-dannyx-v-python/spiski-list-funkcii-i-metody-spiskov.html).

Зачем нужны кортежи, если есть списки?

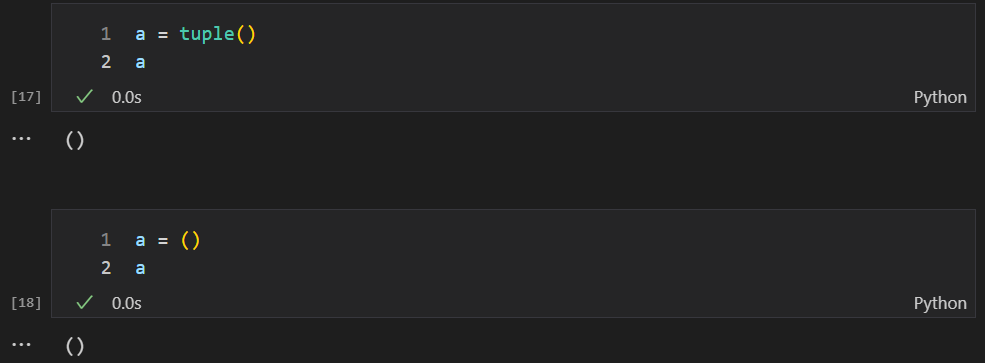
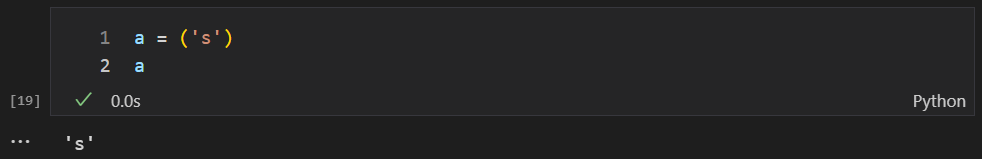
* Защита от дурака. То есть кортеж защищен от изменений, как намеренных (что плохо), так и случайных (что хорошо).
* Меньший размер. Дабы не быть голословным:

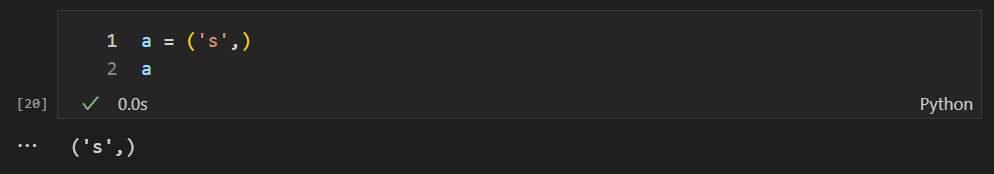
Возможность использовать кортежи в качестве ключей [словаря](https://pythonworld.ru/tipy-dannyx-v-python/slovari-dict-funkcii-i-metody-slovarej.html):

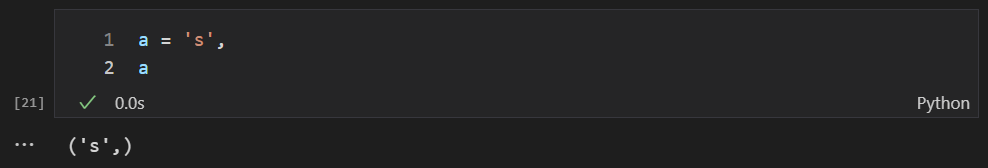
**Как работать с кортежами?**

С преимуществами кортежей разобрались, теперь встает вопрос - а как с ними работать. Примерно так же, как и со списками.

Создаем пустой кортеж:

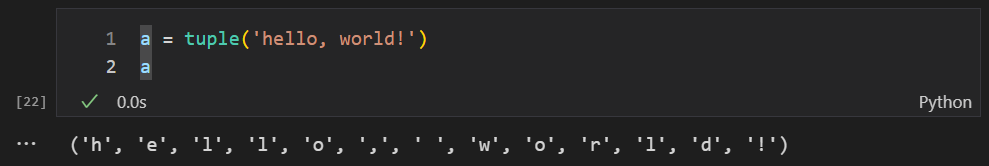
Создаем кортеж из одного элемента:

Стоп. Получилась строка. Но как же так? Мы же кортеж хотели! Как же нам кортеж получить?

Ура! Заработало! Все дело - в запятой. Сами по себе скобки ничего не значат, точнее, значат то, что внутри них находится одна инструкция, которая может быть отделена пробелами, переносом строк и прочим мусором. Кстати, кортеж можно создать и так:

Но все же не увлекайтесь, и ставьте скобки, тем более, что бывают случаи, когда скобки необходимы.

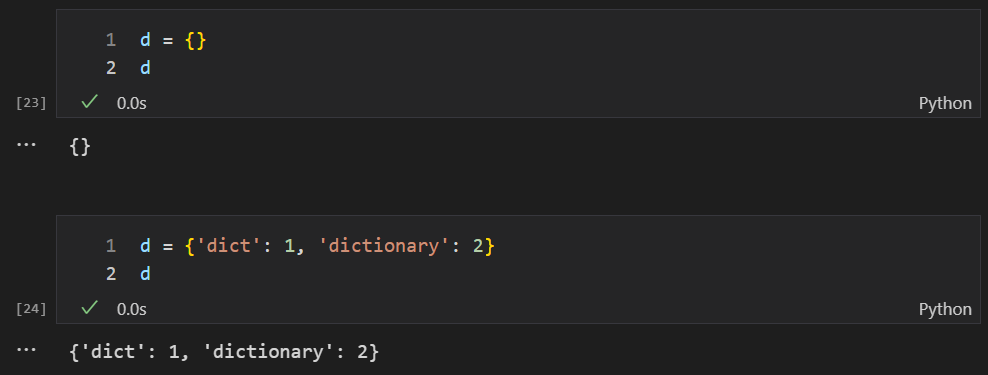
Ну и создать кортеж из итерируемого объекта можно с помощью все той же пресловутой функции tuple():

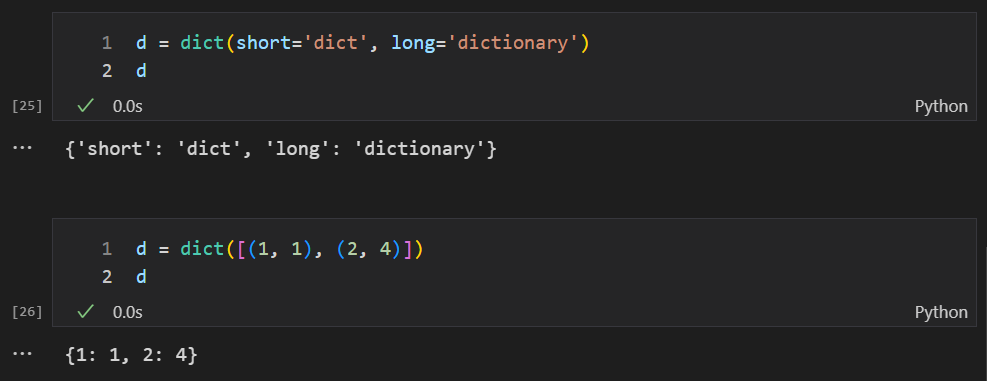
**Операции с кортежами**

Аналогичны операциям со списками.

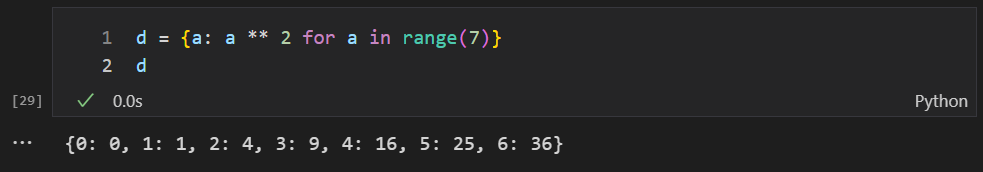
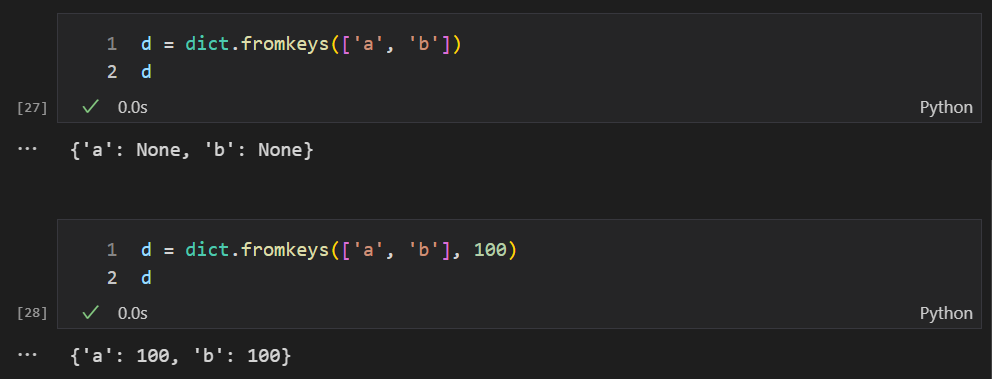
## **Dictionary.**

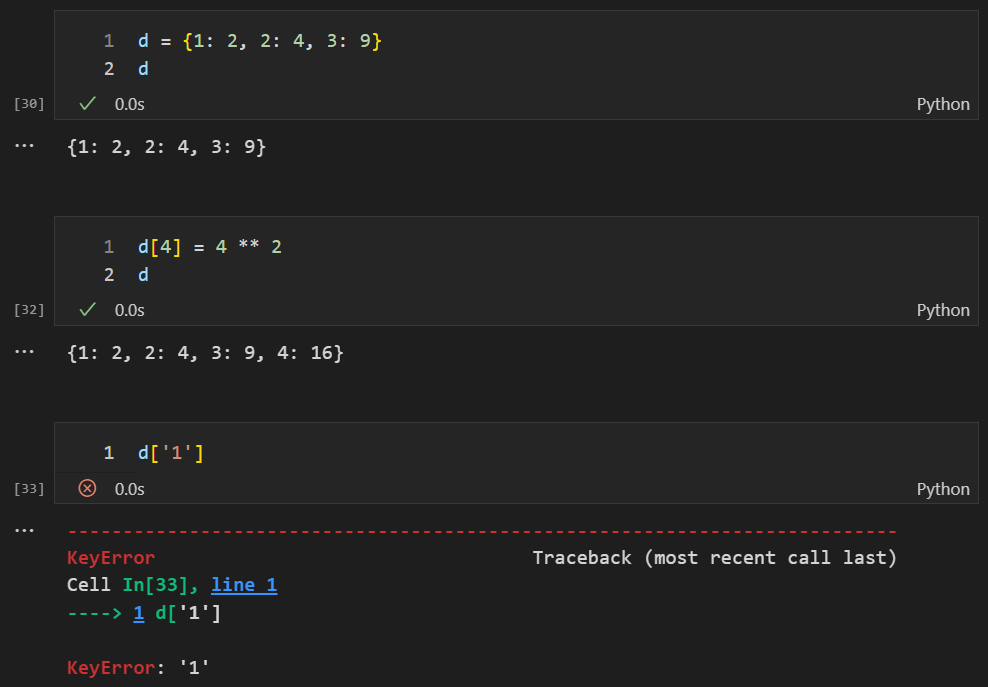
Словари в Python - неупорядоченные коллекции произвольных объектов с доступом по ключу. Их иногда ещё называют ассоциативными массивами или хеш-таблицами.

Чтобы работать со словарём, его нужно создать. Сделать это можно несколькими способами. Во-первых, с помощью литерала:

Во-вторых, с помощью функции dict:

В-третьих, с помощью метода fromkeys:

В-четвертых, с помощью генераторов словарей, которые очень похожи на [генераторы списков](https://pythonworld.ru/tipy-dannyx-v-python/spiski-list-funkcii-i-metody-spiskov.html).

Теперь попробуем добавить записей в словарь и извлечь значения ключей:

Как видно из примера, присвоение по новому ключу расширяет словарь, присвоение по существующему ключу перезаписывает его, а попытка извлечения несуществующего ключа порождает исключение. Для избежания исключения есть специальный метод (см. ниже), или можно [перехватывать исключение](https://pythonworld.ru/tipy-dannyx-v-python/isklyucheniya-v-python-konstrukciya-try-except-dlya-obrabotki-isklyuchenij.html).

Что же можно еще делать со словарями? Да то же самое, что и с другими объектами: [встроенные функции](https://pythonworld.ru/osnovy/vstroennye-funkcii.html), [ключевые слова](https://pythonworld.ru/osnovy/klyuchevye-slova-modul-keyword.html) (например, [циклы for и while](https://pythonworld.ru/osnovy/cikly-for-i-while-operatory-break-i-continue-volshebnoe-slovo-else.html)), а также специальные методы словарей.

**Методы словарей**

*dict.clear()* - очищает словарь.

*dict.copy()* - возвращает копию словаря.

*classmethod dict.fromkeys(seq[, value])* - создает словарь с ключами из seq и значением value (по умолчанию None).

*dict.get(key[, default])* - возвращает значение ключа, но если его нет, не бросает исключение, а возвращает default (по умолчанию None).

*dict.items()* - возвращает пары (ключ, значение).

*dict.keys()* - возвращает ключи в словаре.

*dict.pop(key[, default])* - удаляет ключ и возвращает значение. Если ключа нет, возвращает default (по умолчанию бросает исключение).

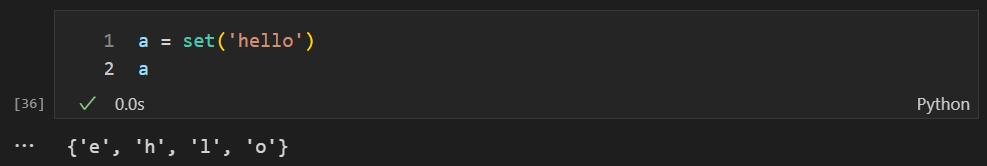
*dict.popitem()* - удаляет и возвращает пару (ключ, значение). Если словарь пуст, бросает исключение KeyError. Помните, что словари неупорядочены.

*dict.setdefault(key[, default])* - возвращает значение ключа, но если его нет, не бросает исключение, а создает ключ со значением default (по умолчанию None).

*dict.update([other])* - обновляет словарь, добавляя пары (ключ, значение) из other. Существующие ключи перезаписываются. Возвращает None (не новый словарь!).

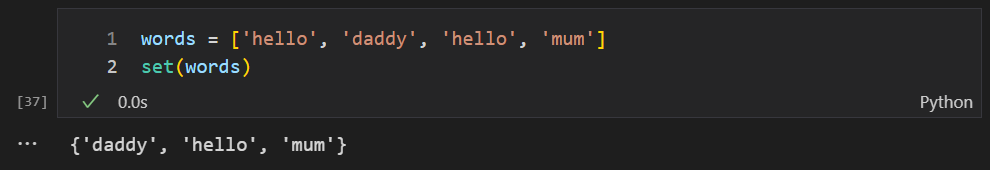
*dict.values()* - возвращает значения в словаре.

## **Set и frozenset.**

Множество в python (set) – "контейнер", содержащий не повторяющиеся элементы в случайном порядке.

Как видно из примера, множества имеет тот же литерал, что и словарь, но пустое множество с помощью литерала создать нельзя.

Множества удобно использовать для удаления повторяющихся элементов:

С множествами можно выполнять множество операций: находить объединение, пересечение...

* len(s) - число элементов в множестве (размер множества).
* x in s - принадлежит ли x множеству s.
* set.isdisjoint(other) - истина, если set и other не имеют общих элементов.
* set == other - все элементы set принадлежат other, все элементы other принадлежат set.
* set.issubset(other) или set <= other - все элементы set принадлежат other.
* set.issuperset(other) или set >= other - аналогично.
* set.union(other, ...) или set | other | ... - объединение нескольких множеств.
* set.intersection(other, ...) или set & other & ... - пересечение.
* set.difference(other, ...) или set - other - ... - множество из всех элементов set, не принадлежащие ни одному из other.
* set.symmetric\_difference(other); set ^ other - множество из элементов, встречающихся в одном множестве, но не встречающиеся в обоих.
* set.copy() - копия множества.

И операции, непосредственно изменяющие множество:

* set.update(other, ...); set |= other | ... - объединение.
* set.intersection\_update(other, ...); set &= other & ... - пересечение.
* set.difference\_update(other, ...); set -= other | ... - вычитание.
* set.symmetric\_difference\_update(other); set ^= other - множество из элементов, встречающихся в одном множестве, но не встречающиеся в обоих.
* set.add(elem) - добавляет элемент в множество.
* set.remove(elem) - удаляет элемент из множества. KeyError, если такого элемента не существует.
* set.discard(elem) - удаляет элемент, если он находится в множестве.
* set.pop() - удаляет первый элемент из множества. Так как множества не упорядочены, нельзя точно сказать, какой элемент будет первым.
* set.clear() - очистка множества.

# **Библиотеки numpy и pandas. Массивы**

## **NumPy**

NumPy — это библиотека языка Python, добавляющая поддержку больших многомерных массивов и матриц, вместе с большой библиотекой высокоуровневых (и очень быстрых) математических функций для операций с этими массивами.

**Начинаем работу**

Основным объектом NumPy является однородный многомерный массив (в numpy называется numpy.ndarray). Это многомерный массив элементов (обычно чисел), одного типа.

Наиболее важные атрибуты объектов ndarray:

*ndarray.ndim* - число измерений (чаще их называют "оси") массива.

*ndarray.shape* - размеры массива, его форма. Это кортеж натуральных чисел, показывающий длину массива по каждой оси. Для матрицы из n строк и m столбов, shape будет (n,m). Число элементов кортежа shape равно ndim.

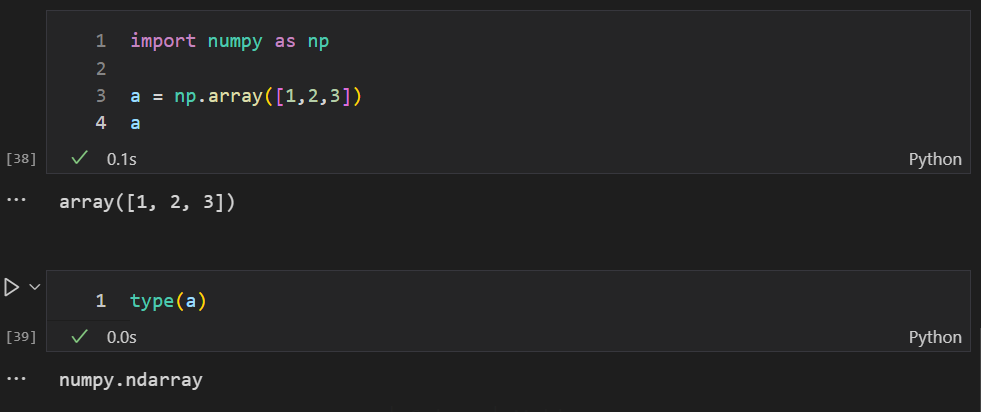
*ndarray.size* - количество элементов массива. Очевидно, равно произведению всех элементов атрибута shape.

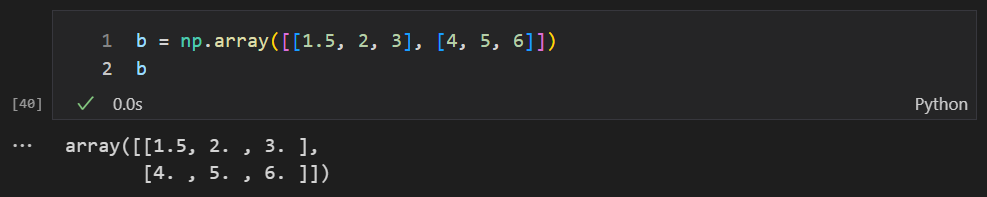
*ndarray.dtype* - объект, описывающий тип элементов массива. Можно определить dtype, используя стандартные типы данных Python. NumPy здесь предоставляет целый букет возможностей, как встроенных, например: bool\_, character, int8, int16, int32, int64, float8, float16, float32, float64, complex64, object\_, так и возможность определить собственные типы данных, в том числе и составные.

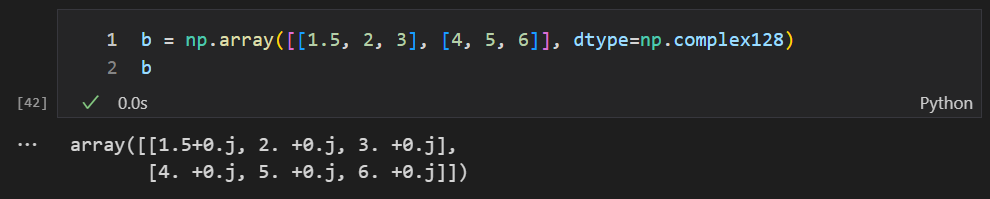
*ndarray.itemsize* - размер каждого элемента массива в байтах.

ndarray.data - буфер, содержащий фактические элементы массива. Обычно не нужно использовать этот атрибут, так как обращаться к элементам массива проще всего с помощью индексов.

**Создание массивов**

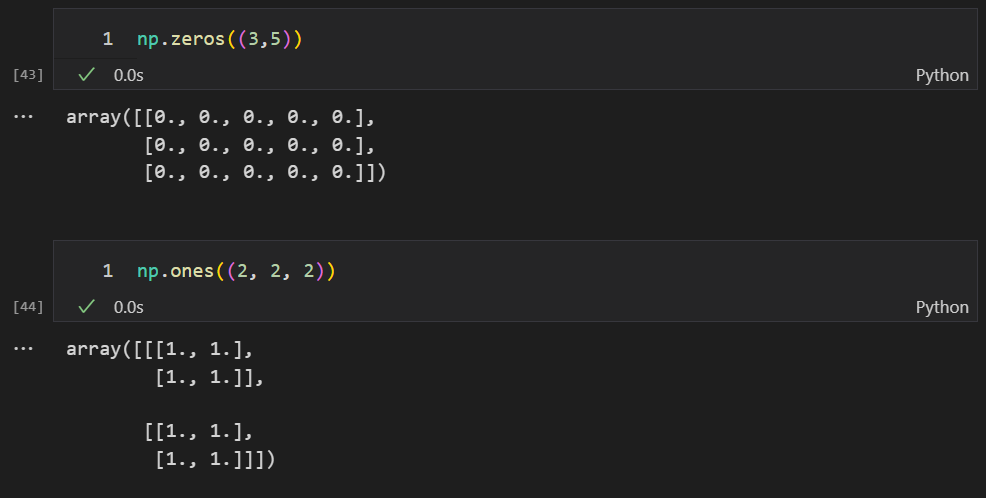
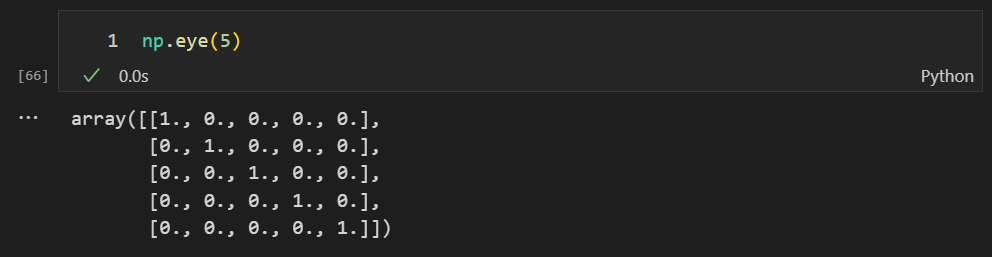
В NumPy существует много способов создать массив. Один из наиболее простых - создать массив из обычных списков или кортежей Python, используя функцию numpy.array() (запомните: array - функция, создающая объект типа ndarray):

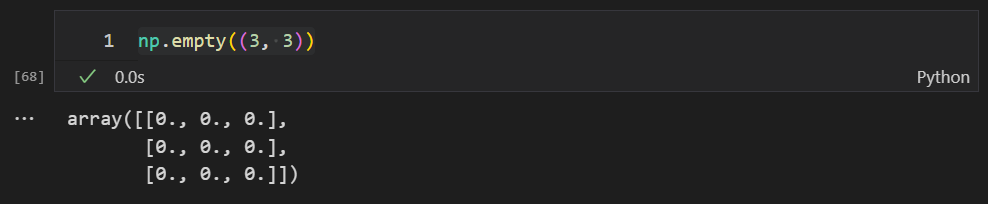
Функция array() трансформирует вложенные последовательности в многомерные массивы. Тип элементов массива зависит от типа элементов исходной последовательности (но можно и переопределить его в момент создания).

Можно также переопределить тип в момент создания:

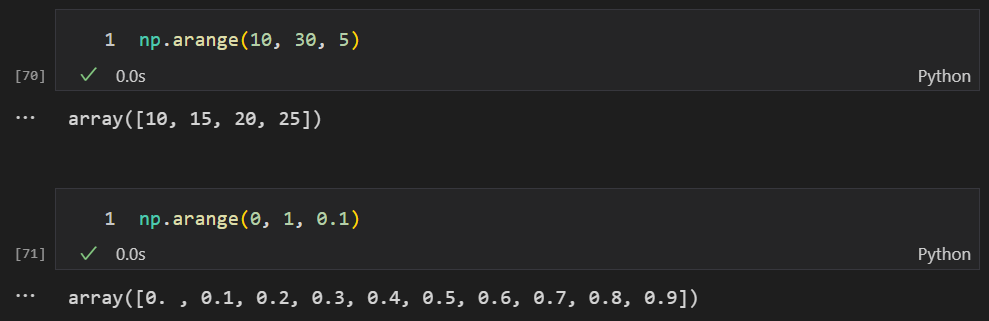
Функция array() не единственная функция для создания массивов. Обычно элементы массива вначале неизвестны, а массив, в котором они будут храниться, уже нужен. Поэтому имеется несколько функций для того, чтобы создавать массивы с каким-то исходным содержимым (по умолчанию тип создаваемого массива — float64).

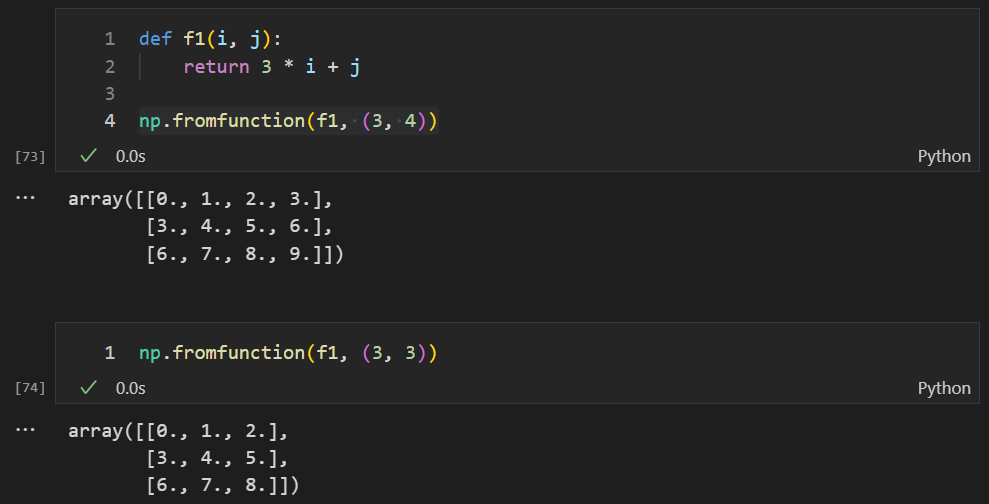
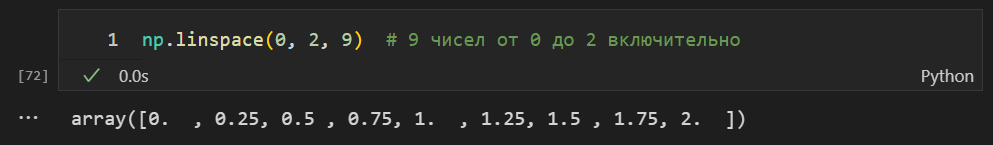
Функция zeros() создает массив из нулей, а функция ones() — массив из единиц. Обе функции принимают кортеж с размерами, и аргумент dtype:

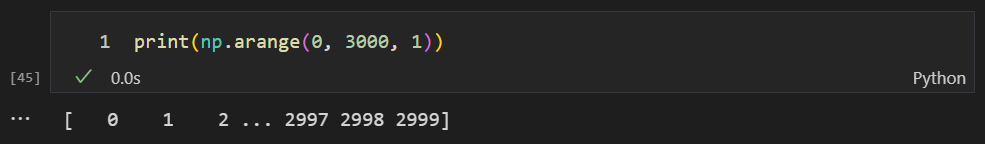
Функция eye() создаёт единичную матрицу (двумерный массив):

Функция empty() создает массив без его заполнения. Исходное содержимое случайно и зависит от состояния памяти на момент создания массива (то есть от того мусора, что в ней хранится):

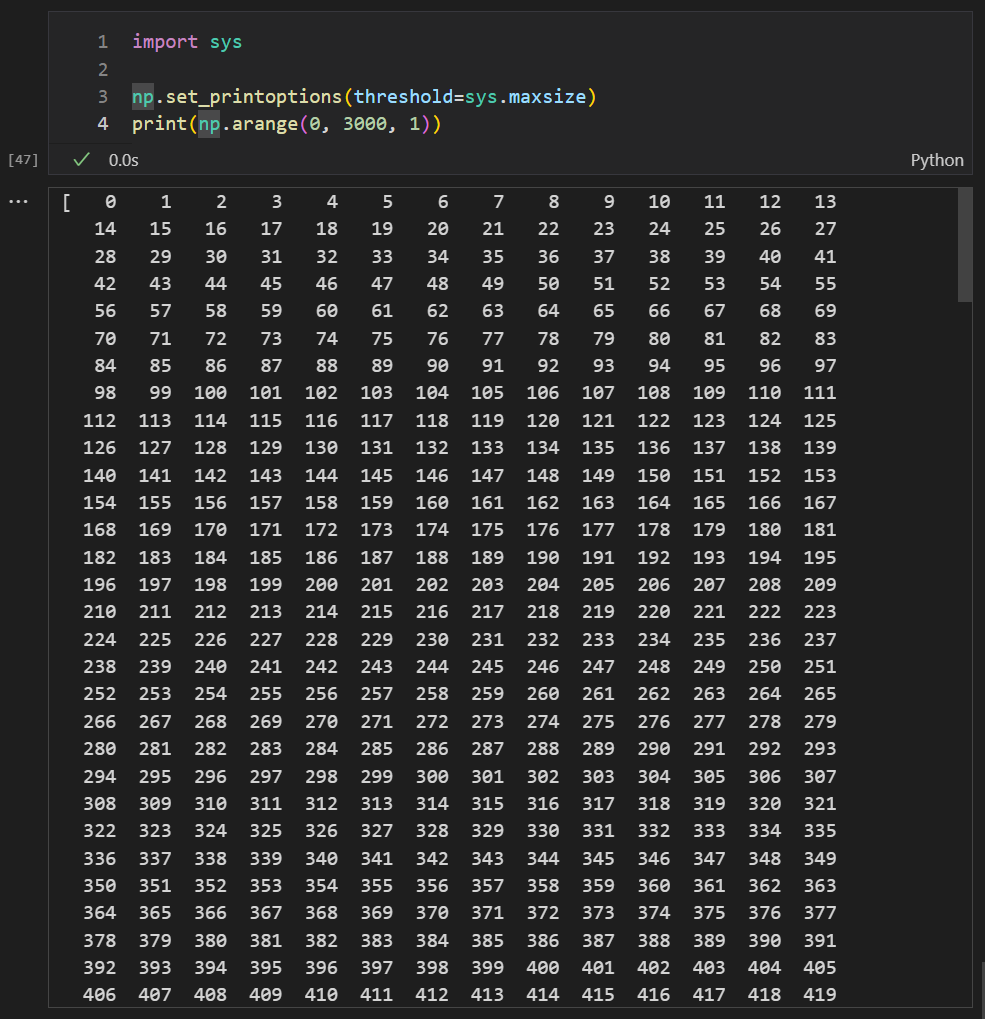
Для создания последовательностей чисел, в NumPy имеется функция arange(), аналогичная встроенной в Python range(), только вместо списков она возвращает массивы, и принимает не только целые значения:

Вообще, при использовании arange() с аргументами типа float, сложно быть уверенным в том, сколько элементов будет получено (из-за ограничения точности чисел с плавающей запятой). Поэтому, в таких случаях обычно лучше использовать функцию linspace(), которая вместо шага в качестве одного из аргументов принимает число, равное количеству нужных элементов:

**Печать массивов**

Если массив слишком большой, чтобы его печатать, NumPy автоматически скрывает центральную часть массива и выводит только его уголки.

Если вам действительно нужно увидеть весь массив, используйте функцию numpy.set\_printoptions:

np.set\_printoptions(threshold= sys.maxsize)

И вообще, с помощью этой функции можно настроить печать массивов "под себя". Функция numpy.set\_printoptions принимает несколько аргументов:

precision : количество отображаемых цифр после запятой (по умолчанию 8).

threshold : количество элементов в массиве, вызывающее обрезание элементов (по умолчанию 1000).

edgeitems : количество элементов в начале и в конце каждой размерности массива (по умолчанию 3).

linewidth : количество символов в строке, после которых осуществляется перенос (по умолчанию 75).

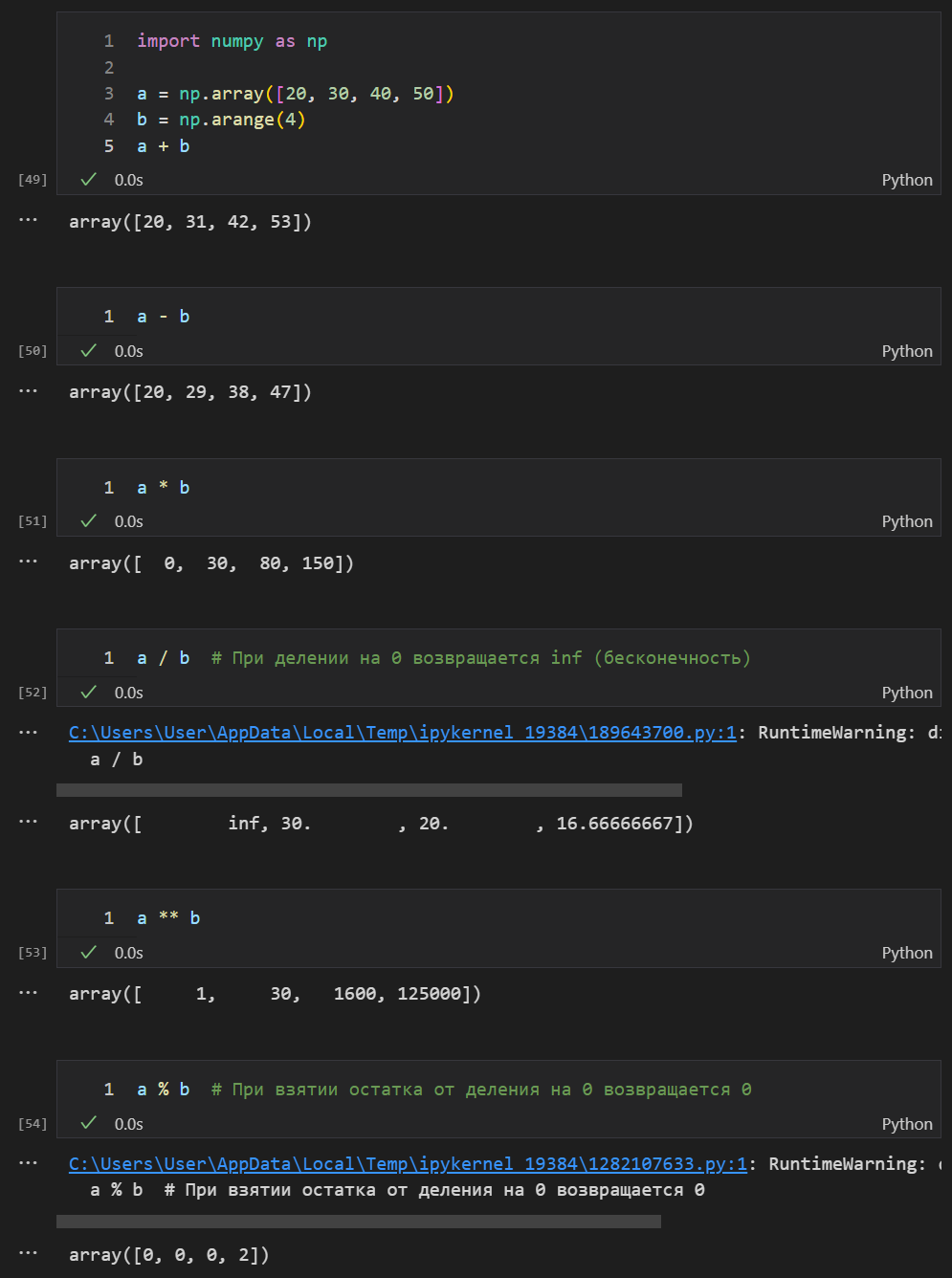
suppress : если True, не печатает маленькие значения в scientific notation (по умолчанию False).

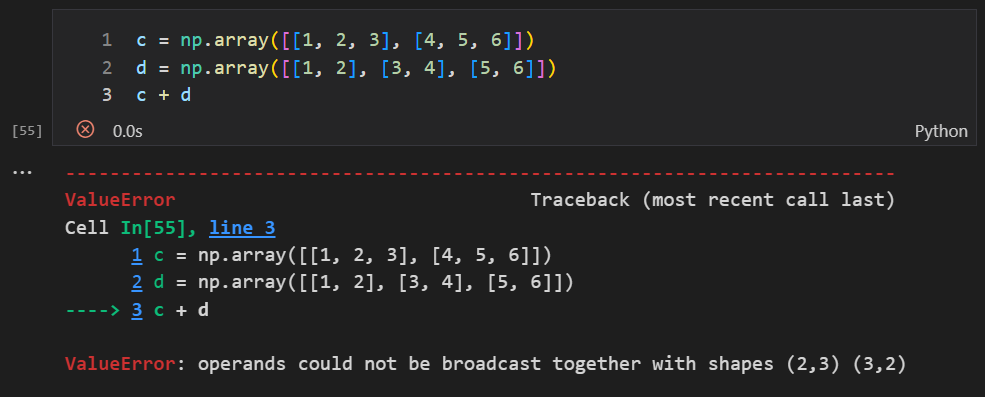
nanstr : строковое представление NaN (по умолчанию 'nan').

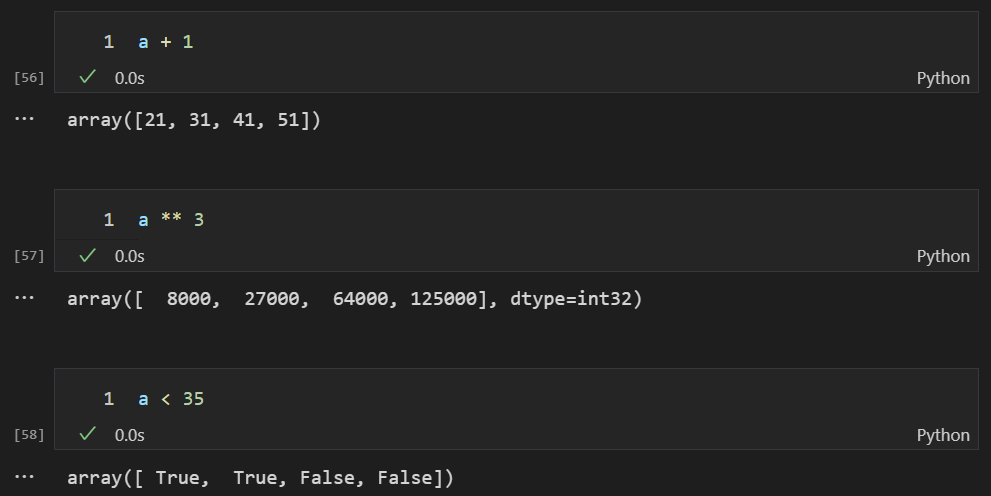
infstr : строковое представление inf (по умолчанию 'inf').

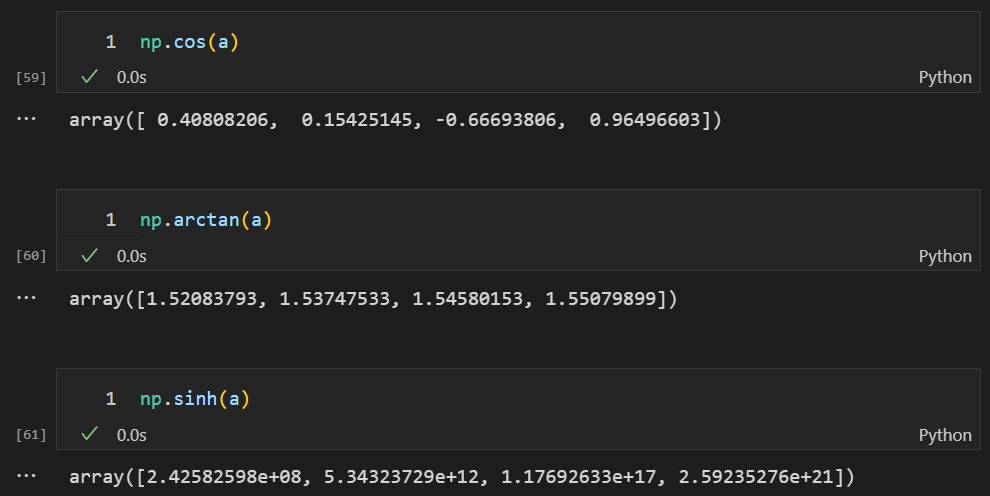
formatter : позволяет более тонко управлять печатью массивов.

**Базовые операции**

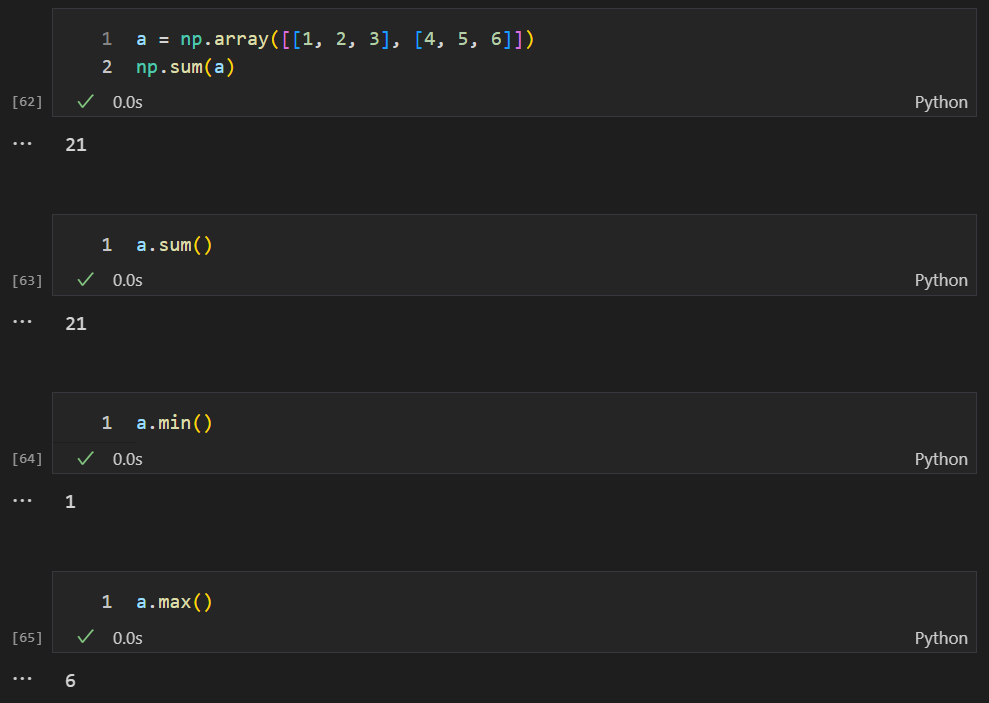
Математические операции над массивами выполняются поэлементно. Создается новый массив, который заполняется результатами действия оператора. Для этого, естественно, массивы должны быть одинаковых размеров.

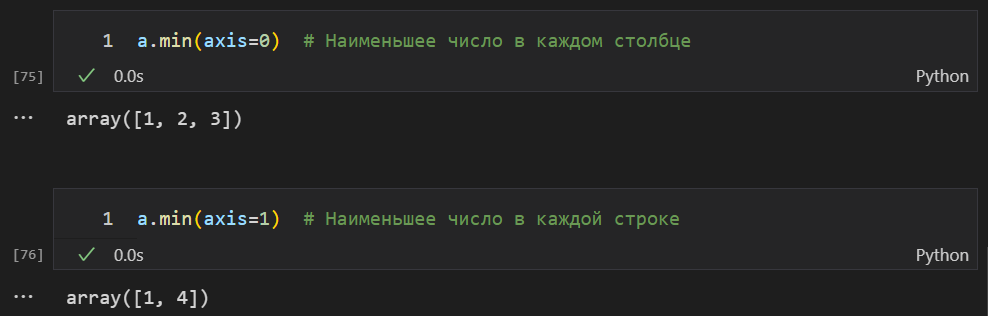
Также можно производить математические операции между массивом и числом. В этом случае к каждому элементу прибавляется (или что вы там делаете) это число.

NumPy также предоставляет множество математических операций для обработки массивов:

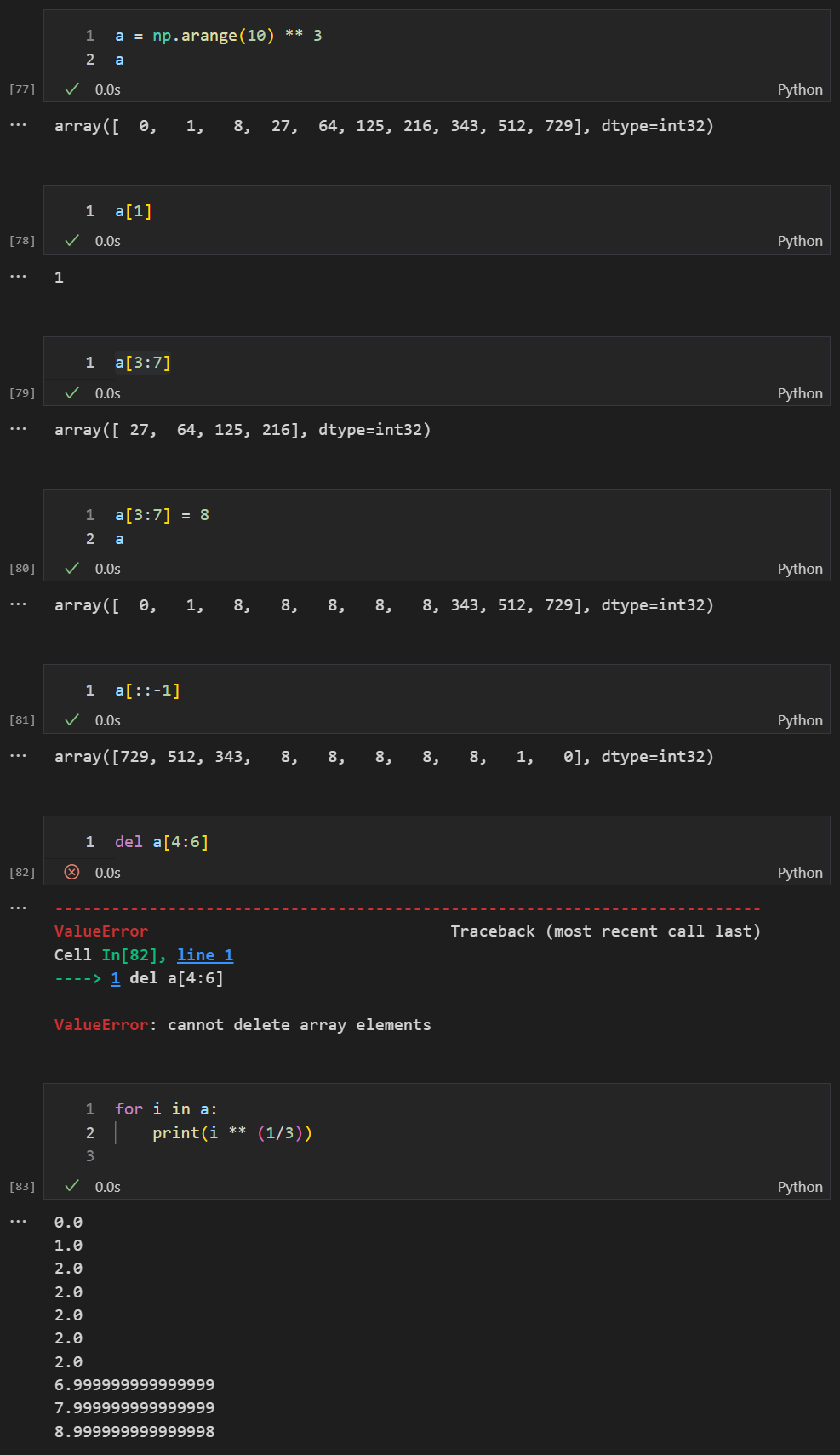
Полный список можно посмотреть [в](https://docs.scipy.org/doc/numpy/reference/routines.math.html) официальной документации по библиотеке.

Многие унарные операции, такие как, например, вычисление суммы всех элементов массива, представлены также и в виде методов класса ndarray.

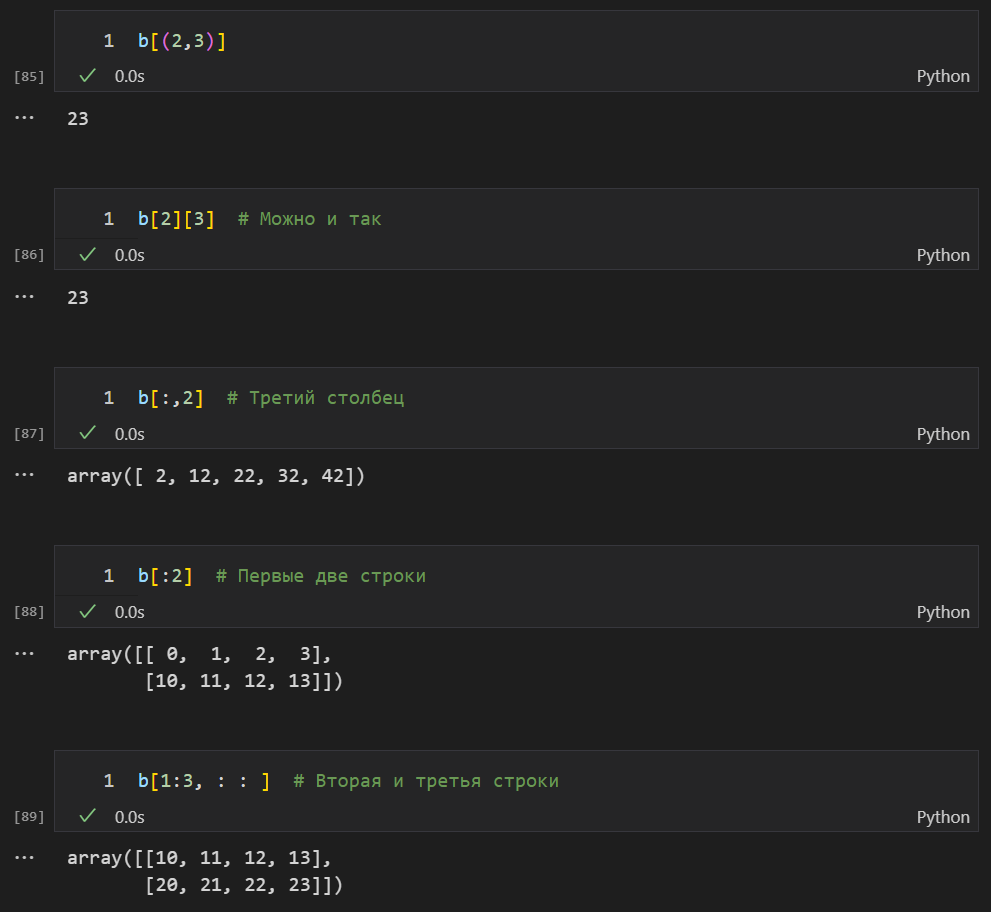
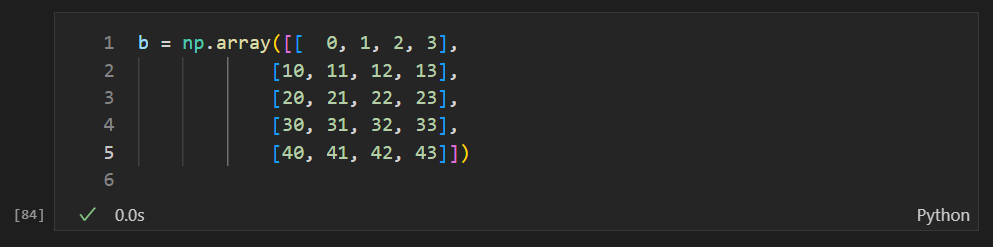
По умолчанию, эти операции применяются к массиву, как если бы он был списком чисел, независимо от его формы. Однако, указав параметр axis, можно применить операцию для указанной оси массива:

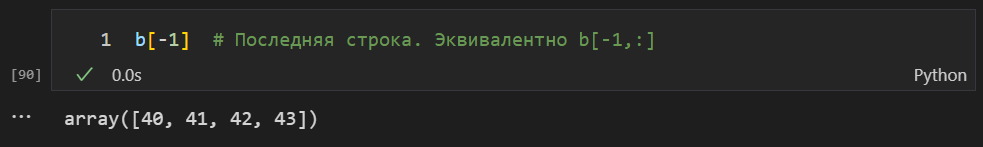
**Индексы, срезы, итерации**

Одномерные массивы осуществляют операции индексирования, срезов и итераций очень схожим образом с обычными списками и другими последовательностями Python (разве что удалять с помощью срезов нельзя).

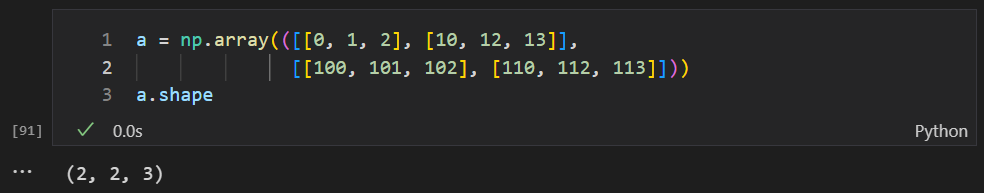


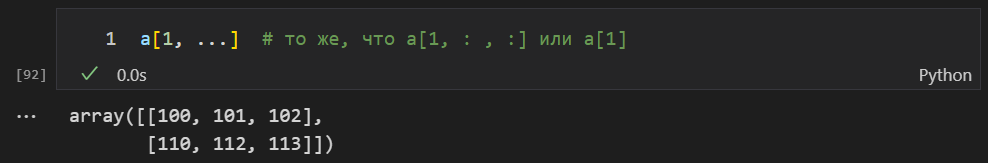
У многомерных массивов на каждую ось приходится один индекс. Индексы передаются в виде последовательности чисел, разделенных запятыми (то бишь, кортежами):

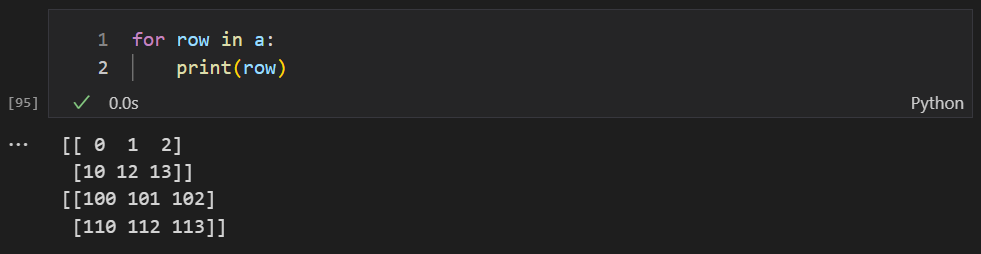
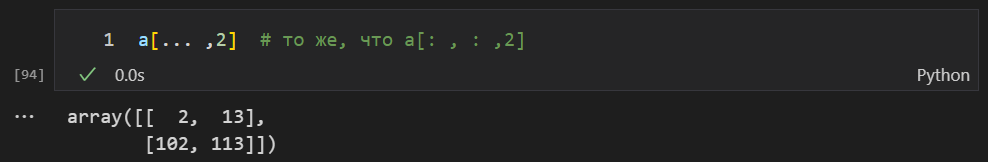
 Когда индексов меньше, чем осей, отсутствующие индексы предполагаются дополненными с помощью срезов:

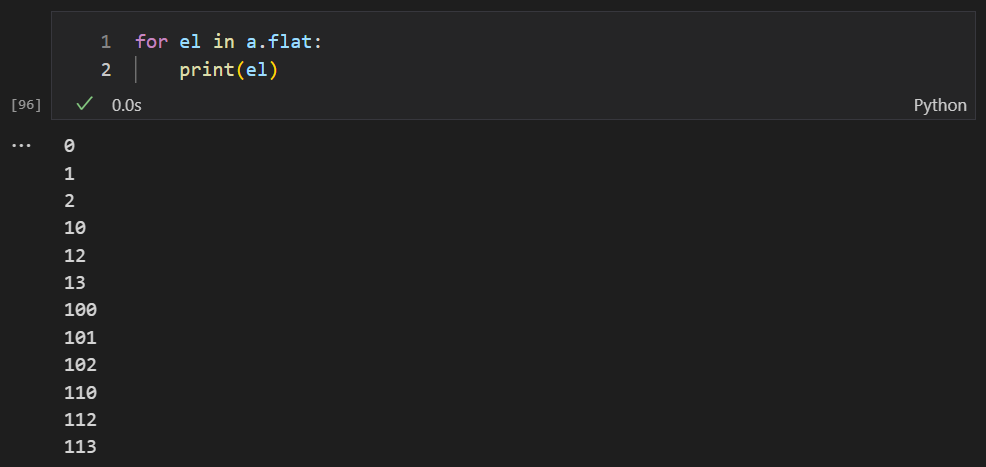
b[i] можно читать как b[i, <столько символов ':', сколько нужно>]. В NumPy это также может быть записано с помощью точек, как b[i, ...].

Например, если x имеет ранг 5 (то есть у него 5 осей), тогда

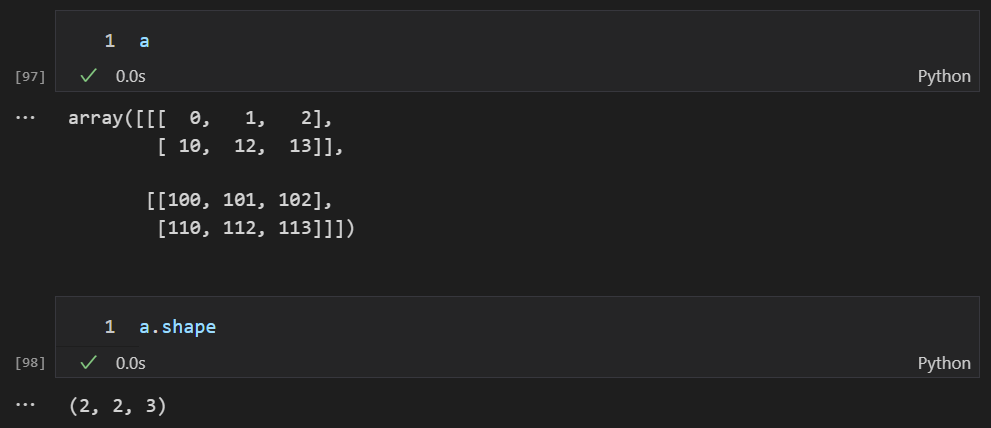
* x[1, 2, ...] эквивалентно x[1, 2, :, :, :],
* x[... , 3] то же самое, что x[:, :, :, :, 3] и
* x[4, ... , 5, :] это x[4, :, :, 5, :].



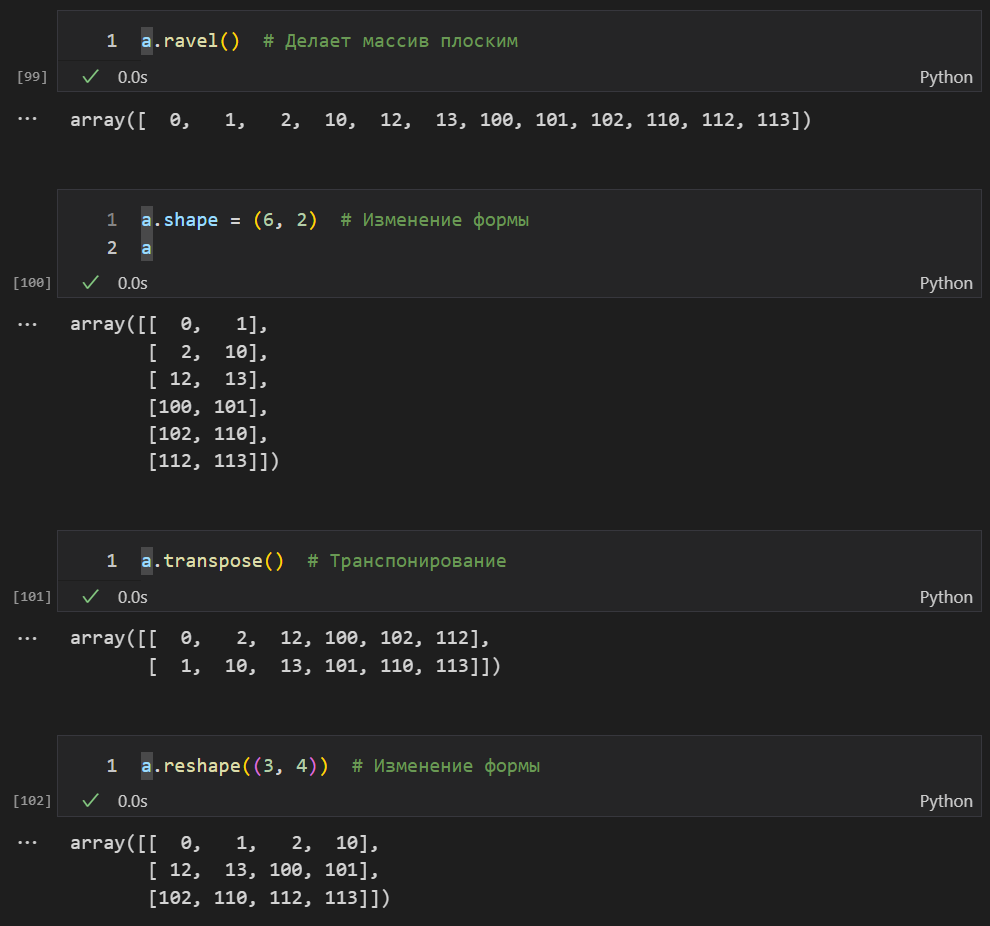
Итерирование многомерных массивов начинается с первой оси: 

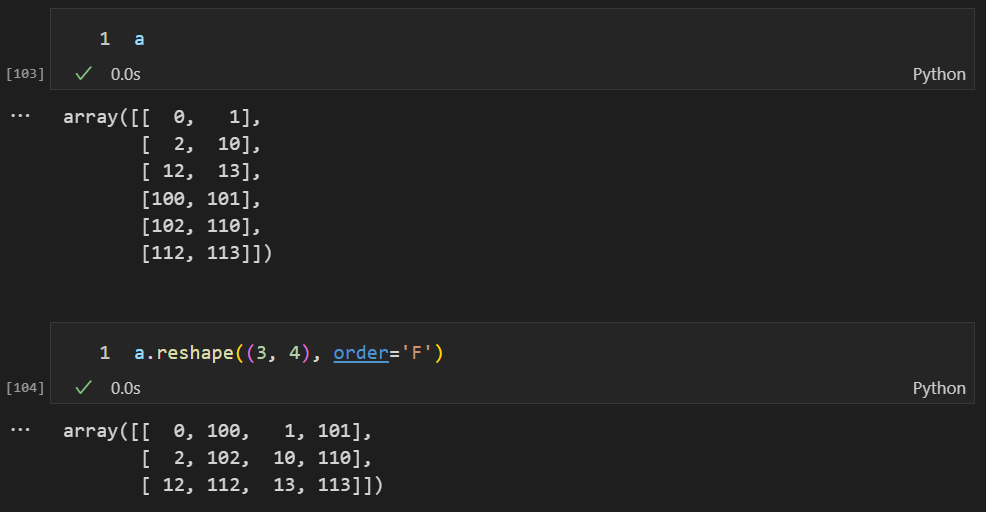
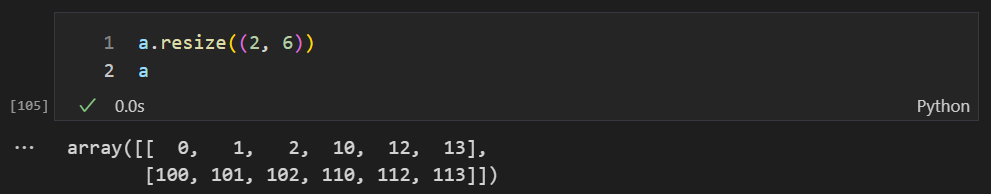
****Однако, если нужно перебрать поэлементно весь массив, как если бы он был одномерным, для этого можно использовать атрибут flat:

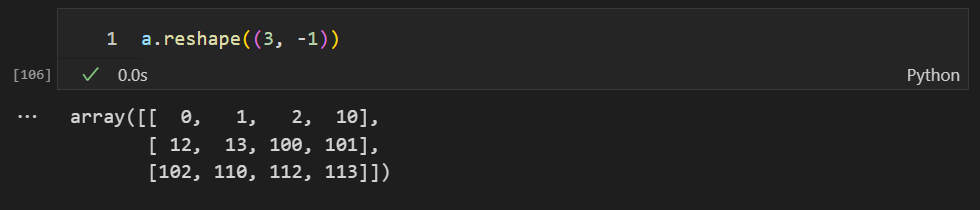
**Манипуляции с формой**

Как уже говорилось, у массива есть форма (shape), определяемая числом элементов вдоль каждой оси:

Форма массива может быть изменена с помощью различных команд:

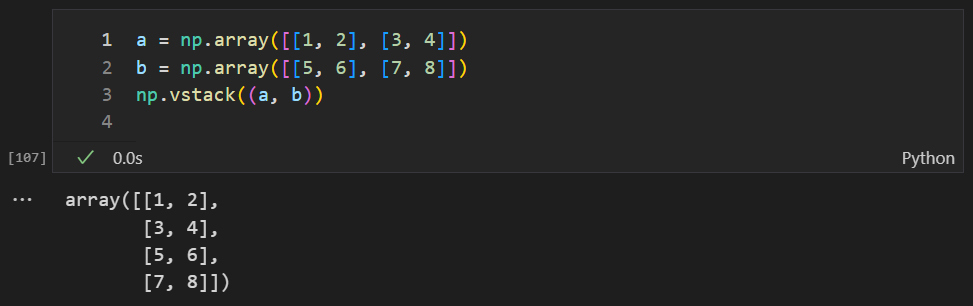
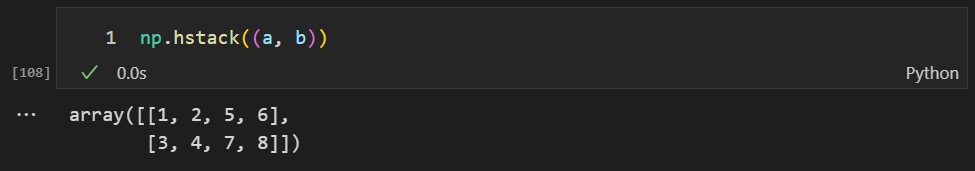
Порядок элементов в массиве в результате функции ravel() соответствует обычному "C-стилю", то есть, чем правее индекс, тем он "быстрее изменяется": за элементом a[0,0] следует a[0,1]. Если одна форма массива была изменена на другую, массив переформировывается также в "C-стиле". Функции ravel() и reshape() также могут работать (при использовании дополнительного аргумента) в FORTRAN-стиле, в котором быстрее изменяется более левый индекс.

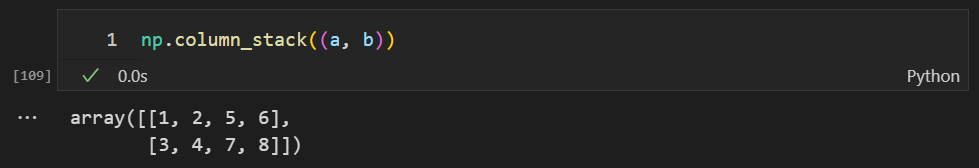
Метод reshape() возвращает ее аргумент с измененной формой, в то время как метод resize() изменяет сам массив:

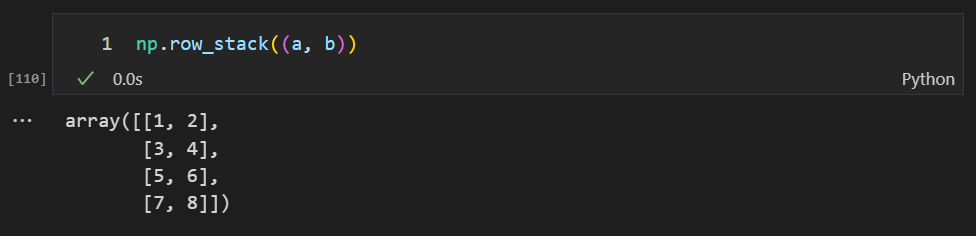
Если при операции такой перестройки один из аргументов задается как -1, то он автоматически рассчитывается в соответствии с остальными заданными:

**Объединение массивов**

Несколько массивов могут быть объединены вместе вдоль разных осей с помощью функций hstack и vstack.

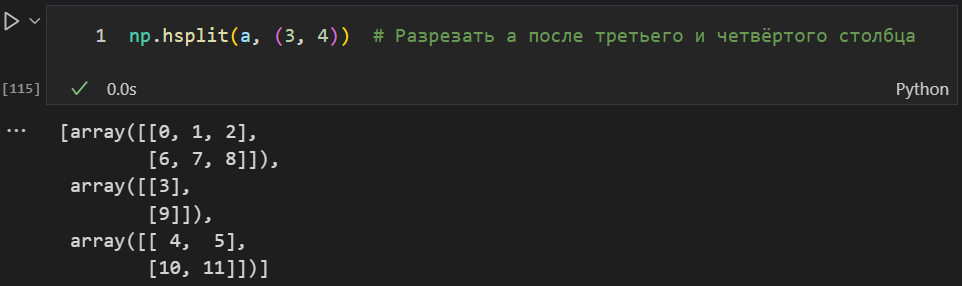
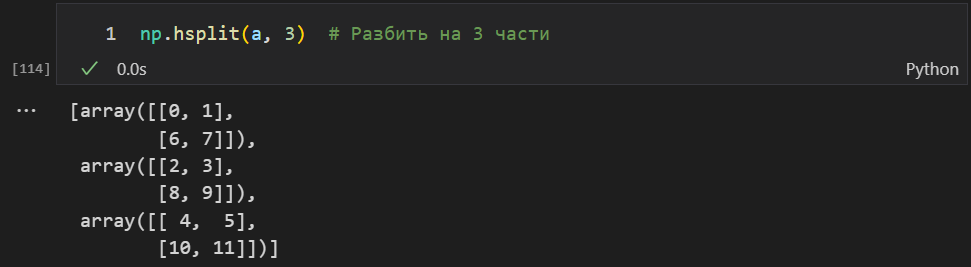
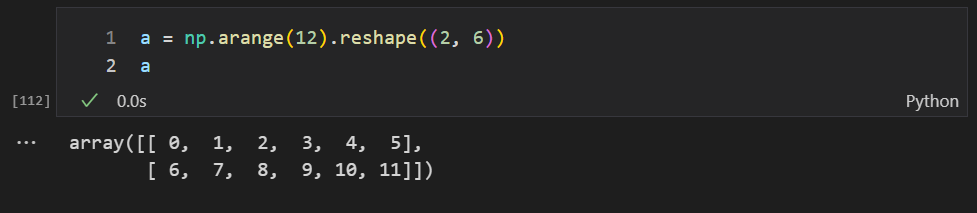
hstack() объединяет массивы по первым осям, vstack() — по последним:

Функция column\_stack() объединяет одномерные массивы в качестве столбцов двумерного массива:

Аналогично для строк имеется функция row\_stack().

**Разбиение массива**

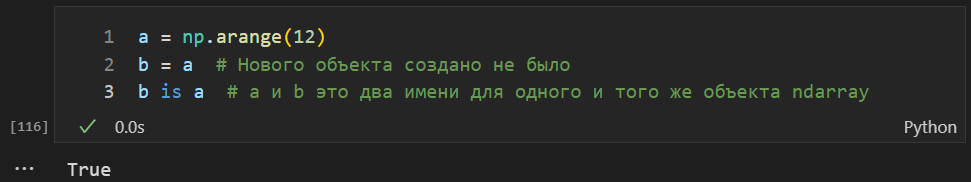
Используя hsplit() вы можете разбить массив вдоль горизонтальной оси, указав либо число возвращаемых массивов одинаковой формы, либо номера столбцов, после которых массив разрезается "ножницами":

Функция vsplit() разбивает массив вдоль вертикальной оси, а array\_split() позволяет указать оси, вдоль которых произойдет разбиение.

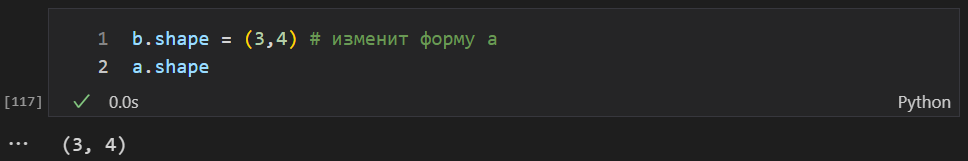
**Копии и представления**

При работе с массивами, их данные иногда необходимо копировать в другой массив, а иногда нет. Это часто является источником путаницы. Возможно 3 случая:

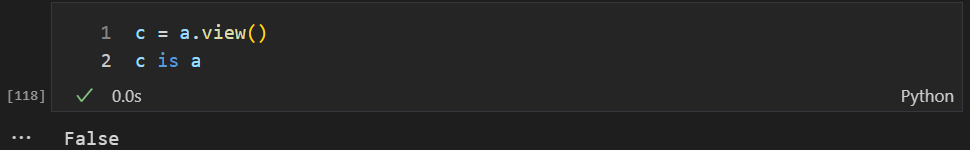
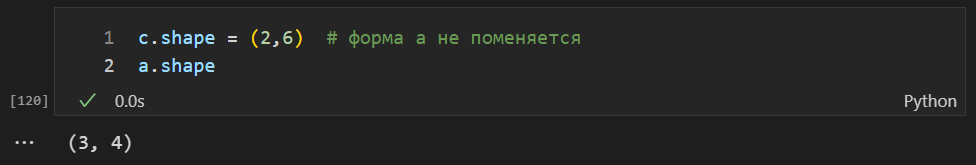
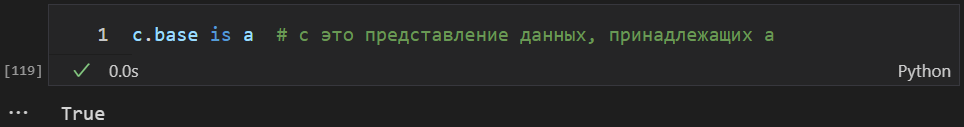
**Вообще никаких копий**

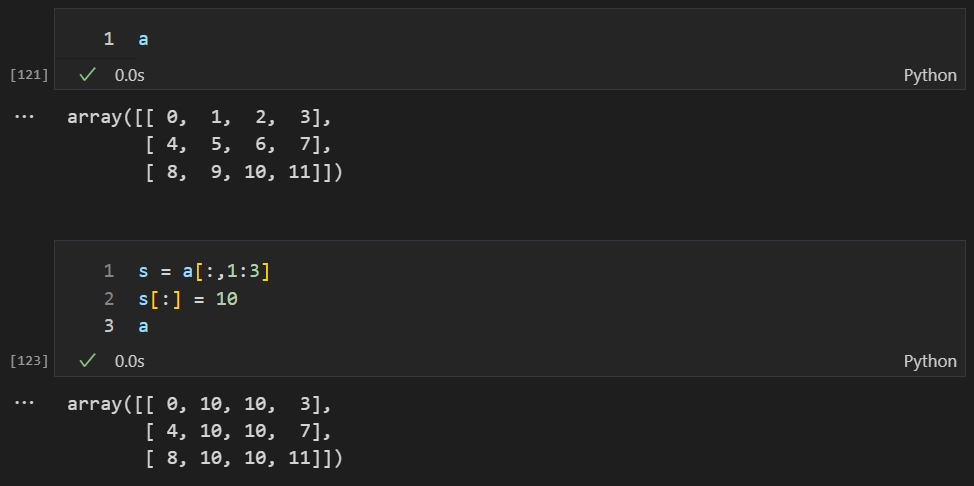
Простое присваивание не создает ни копии массива, ни копии его данных:

Python передает изменяемые объекты как ссылки, поэтому вызовы функций также не создают копий.

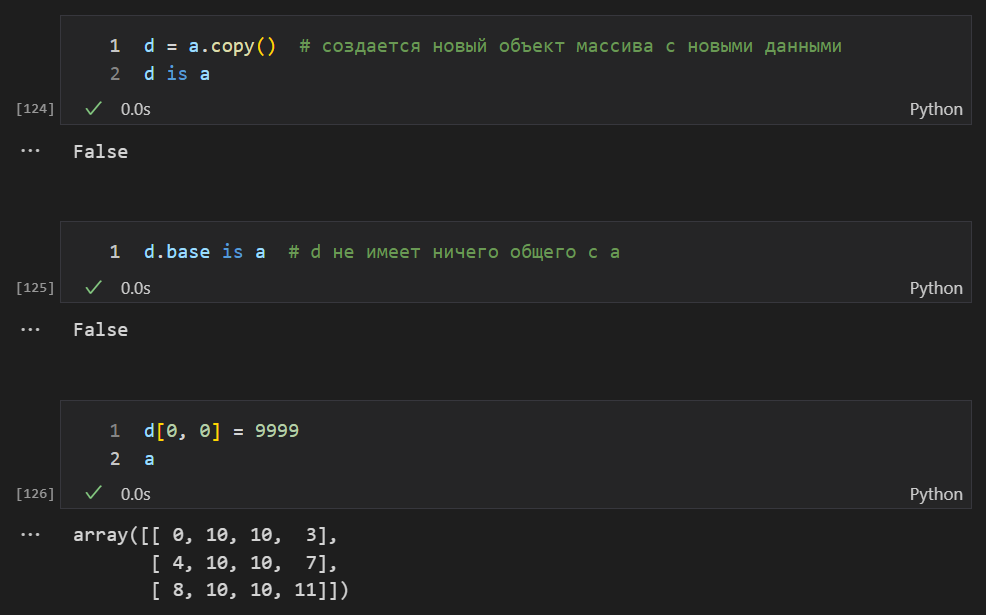
**Представление или поверхностная копия**

Разные объекты массивов могут использовать одни и те же данные. Метод view() создает новый объект массива, являющийся представлением тех же данных.

 >  >

Срез массива – это представление:

**Глубокая копия**

Метод copy() создаст настоящую копию массива и его данных:

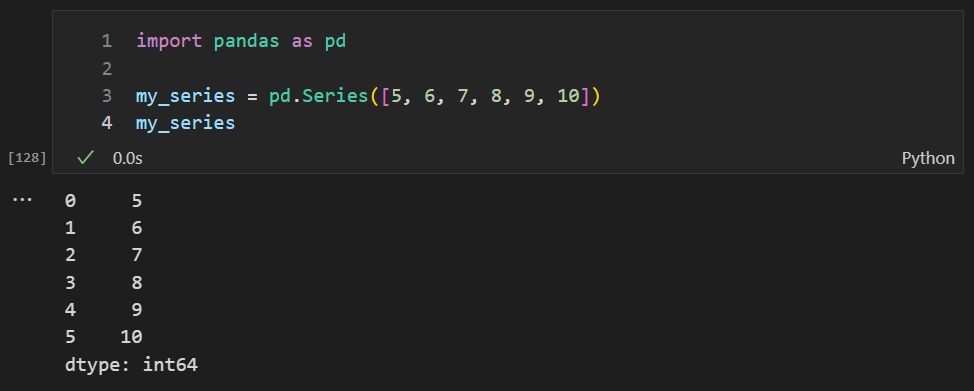
## **Pandas**

**Pandas** – это высокоуровневая [Python](https://khashtamov.com/category/python/) библиотека для анализа данных. Почему я её называю высокоуровневой, потому что построена она поверх более низкоуровневой библиотеки NumPy (написана на Си), что является большим плюсом в производительности. В экосистеме [Python](https://khashtamov.com/2016/06/why-python/), pandas является наиболее продвинутой и быстроразвивающейся библиотекой для обработки и анализа данных. В своей работе приходится пользоваться ею практически каждый день, поэтому я пишу эту краткую заметку для того, чтобы в будущем ссылаться к ней, если вдруг что-то забуду.

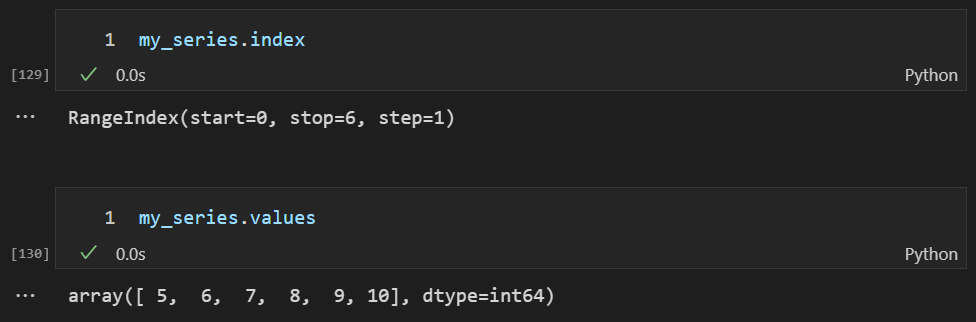
**DataFrame и Series**

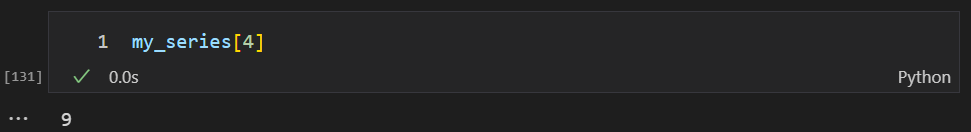
Чтобы эффективно работать с pandas, необходимо освоить самые главные структуры данных библиотеки: DataFrame и Series. Без понимания что они из себя представляют, невозможно в дальнейшем проводить качественный анализ.

**Series**

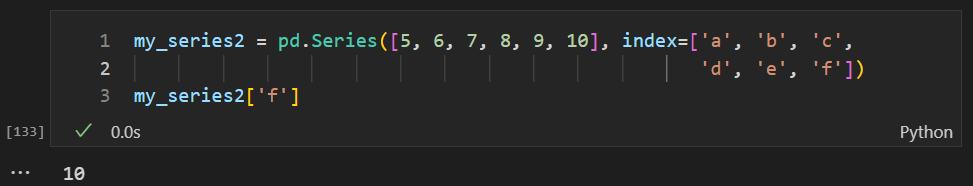
Структура/объект Series представляет из себя объект, похожий на одномерный массив (питоновский список, например), но отличительной его чертой является наличие ассоциированных меток, т.н. индексов, вдоль каждого элемента из списка. Такая особенность превращает его в ассоциативный массив или словарь в Python.

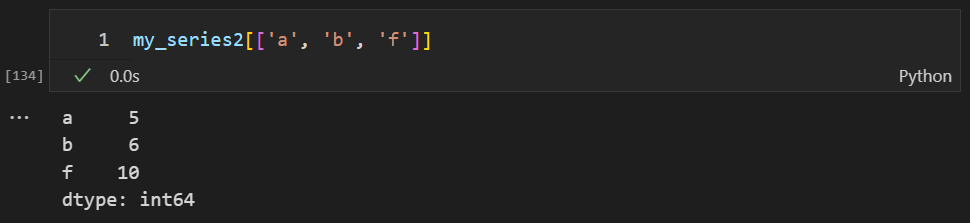
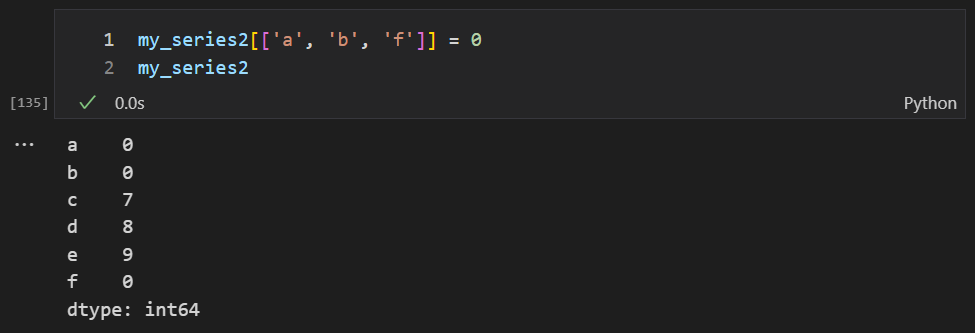
В строковом представлении объекта Series, индекс находится слева, а сам элемент справа. Если индекс явно не задан, то pandas автоматически создаёт RangeIndex от 0 до N-1, где N общее количество элементов. Также стоит обратить, что у Series есть тип хранимых элементов, в нашем случае это int64, т.к. мы передали целочисленные значения.

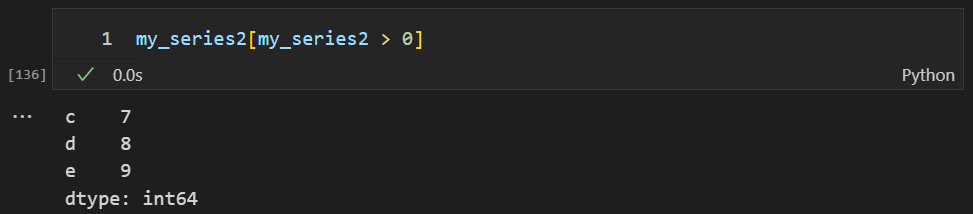
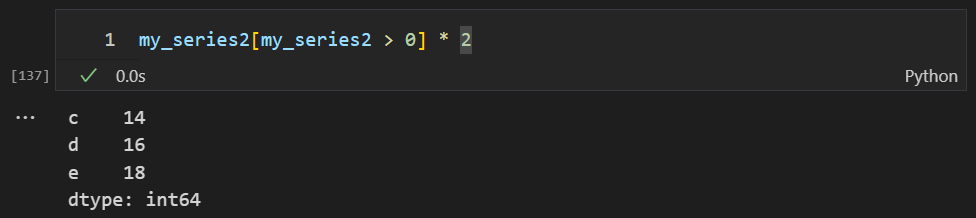
У объекта Series есть атрибуты через которые можно получить список элементов и индексы, это values и index соответственно.

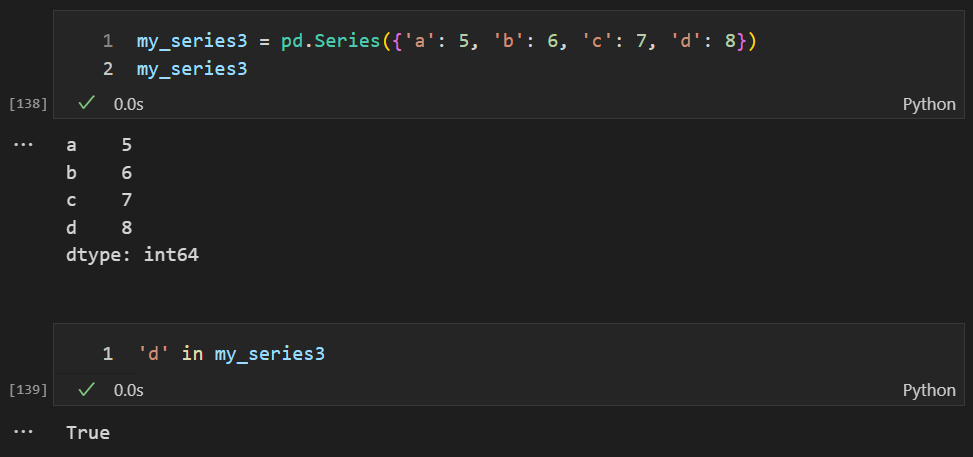
Доступ к элементам объекта Series возможны по их индексу (вспоминается аналогия со словарем и доступом по ключу).

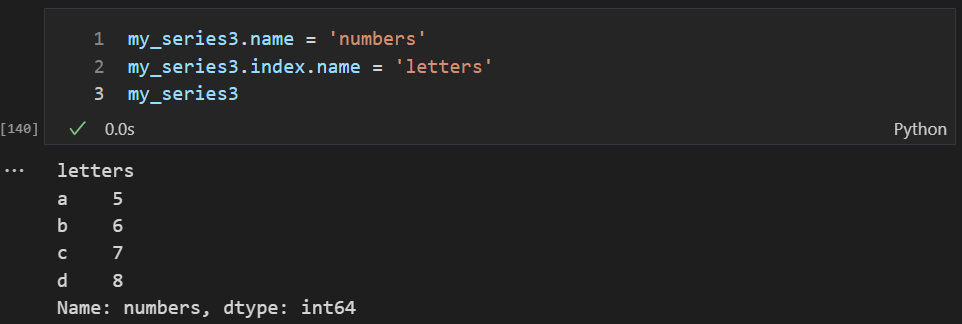
Индексы можно задавать явно:

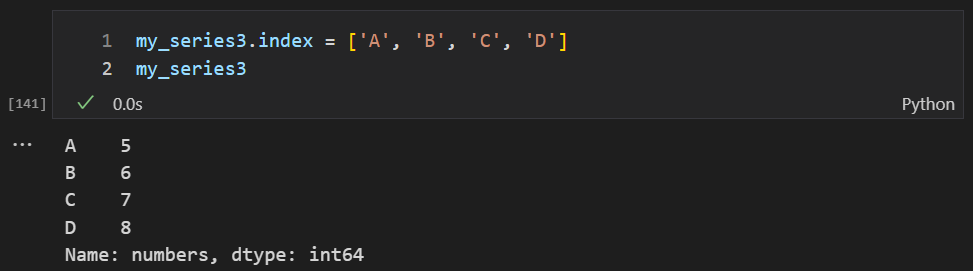
Делать выборку по нескольким индексам и осуществлять групповое присваивание:



Фильтровать Series как душе заблагорассудится, а также применять математические операции и многое другое:

Если Series напоминает нам словарь, где ключом является индекс, а значением сам элемент, то можно сделать так:

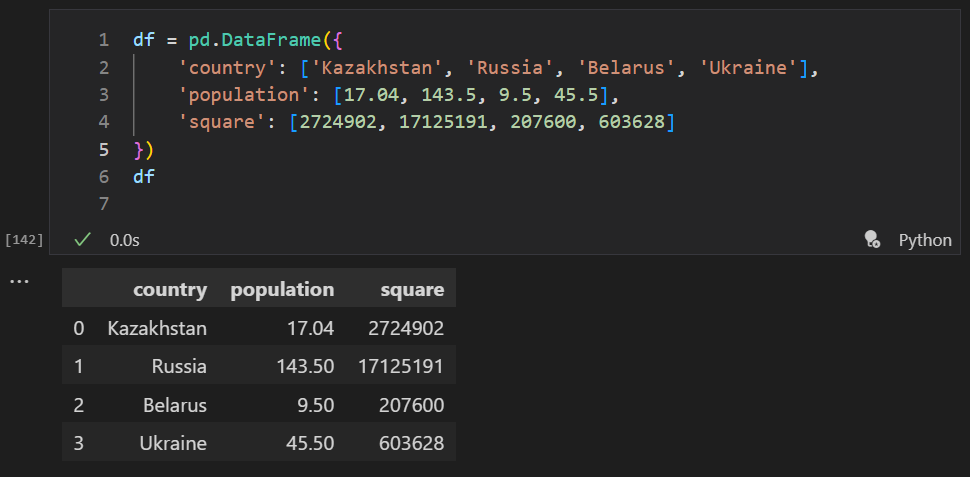
У объекта Series и его индекса есть атрибут name, задающий имя объекту и индексу соответственно.

Индекс можно поменять "на лету", присвоив список атрибуту index объекта Series

Имейте в виду, что список с индексами по длине должен совпадать с количеством элементов в Series.

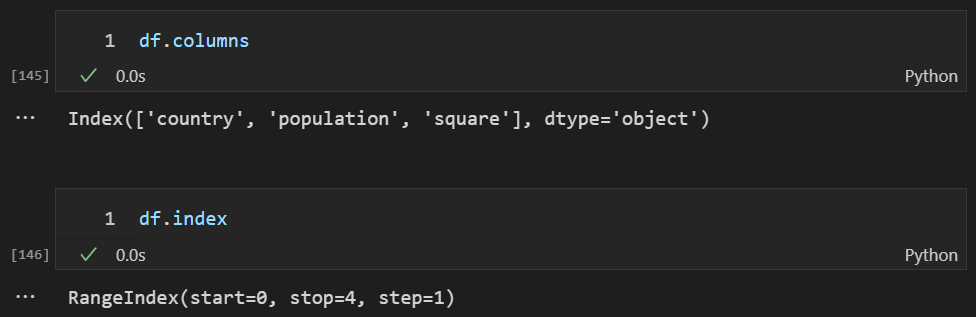
**DataFrame**

Объект DataFrame лучше всего представлять себе в виде обычной таблицы и это правильно, ведь DataFrame является табличной структурой данных. В любой таблице всегда присутствуют строки и столбцы. Столбцами в объекте DataFrame выступают объекты Series, строки которых являются их непосредственными элементами.

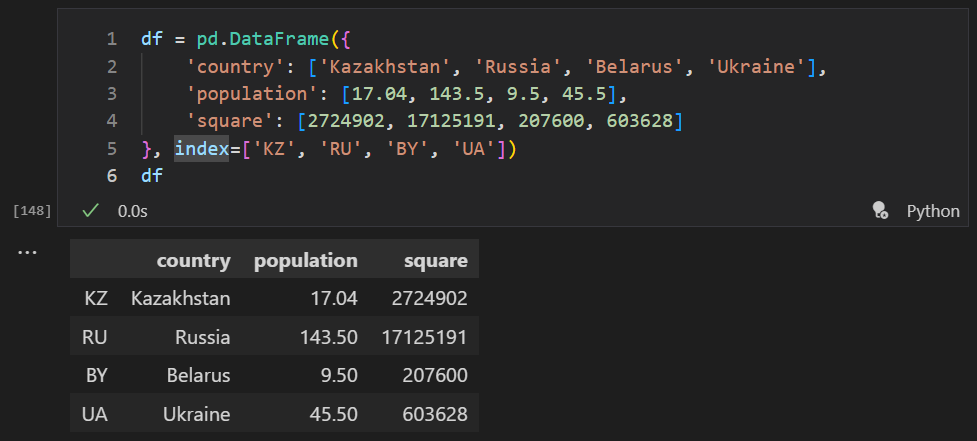
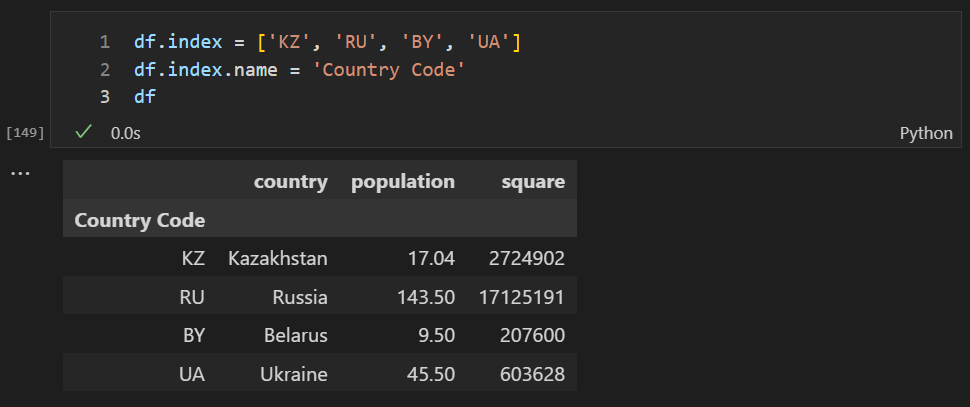
DataFrame проще всего сконструировать на примере питоновского словаря:

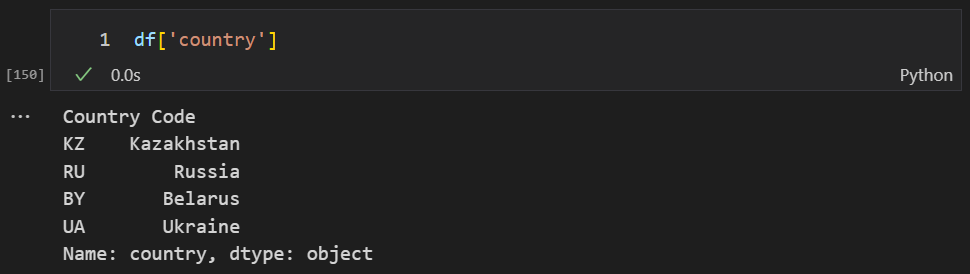
Чтобы убедиться, что столбец в DataFrame это Series, извлекаем любой:

Объект DataFrame имеет 2 индекса: по строкам и по столбцам. Если индекс по строкам явно не задан (например, колонка по которой нужно их строить), то pandas задаёт целочисленный индекс RangeIndex от 0 до N-1, где N это количество строк в таблице.

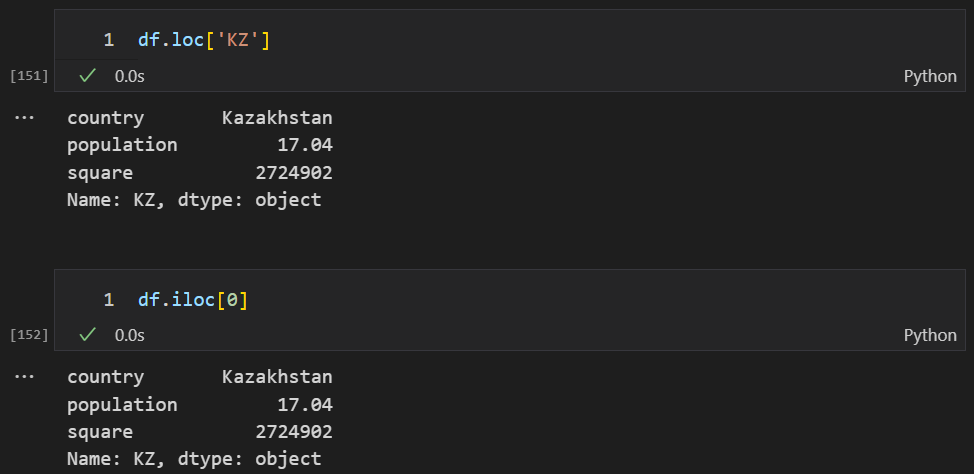
 В таблице у нас 4 элемента от 0 до 3.

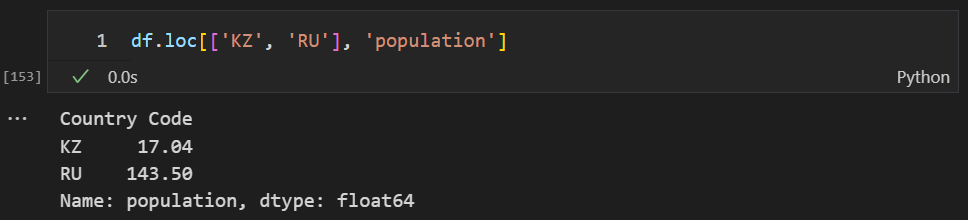
**Доступ по индексу в DataFrame**

Индекс по строкам можно задать разными способами, например, при формировании самого объекта DataFrame или "на лету":

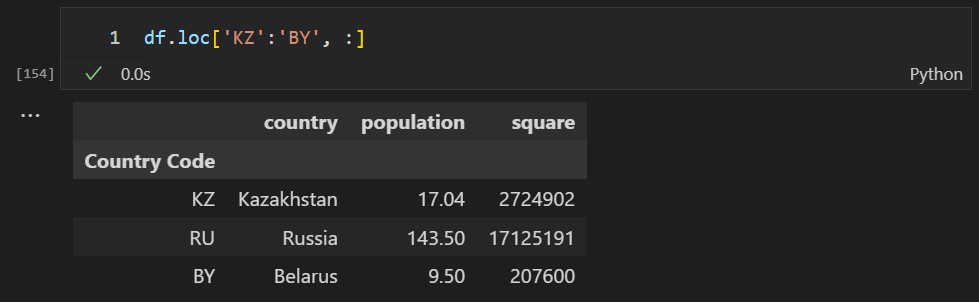
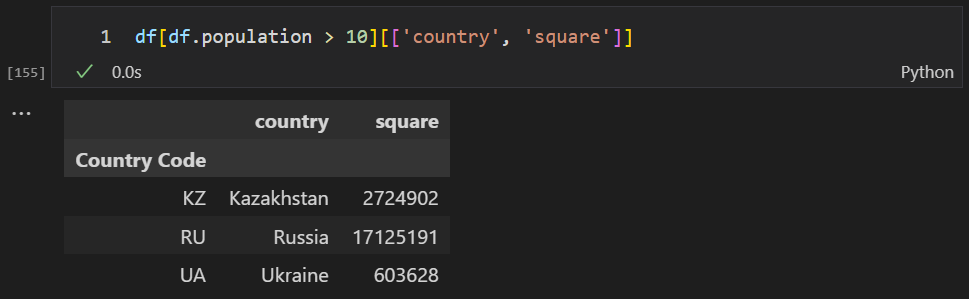
Как видно, индексу было задано имя - Country Code. Отмечу, что объекты Series из DataFrame будут иметь те же индексы, что и объект DataFrame:

Доступ к строкам по индексу возможен несколькими способами:

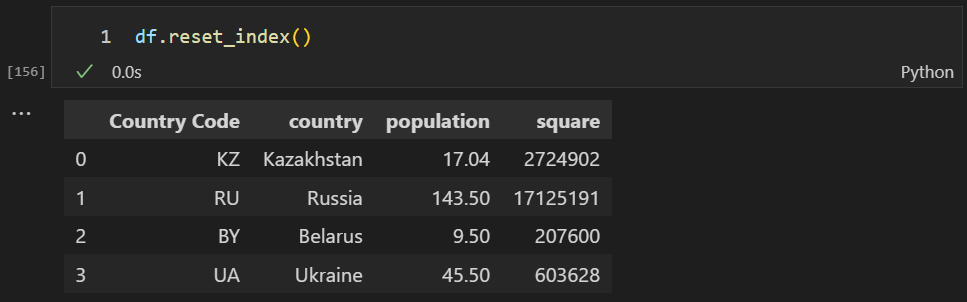
* .loc - используется для доступа по строковой метке
* .iloc - используется для доступа по числовому значению (начиная от 0)

Можно делать выборку по индексу и интересующим колонкам:

Как можно заметить, .loc в квадратных скобках принимает 2 аргумента: интересующий индекс, в том числе поддерживается слайсинг и колонки.

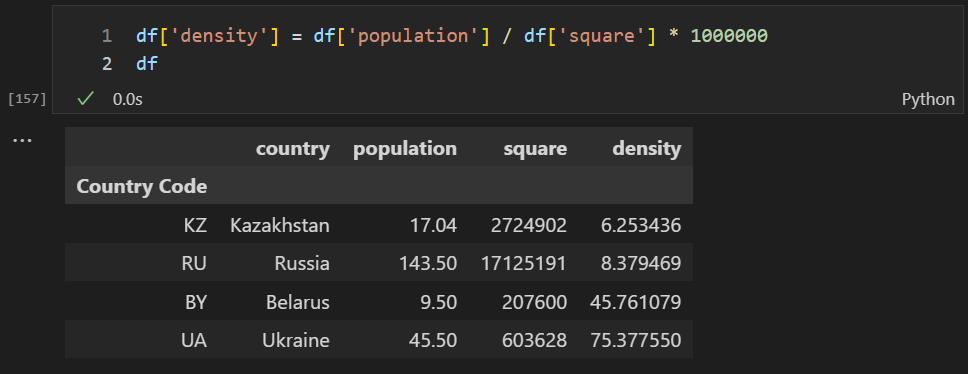
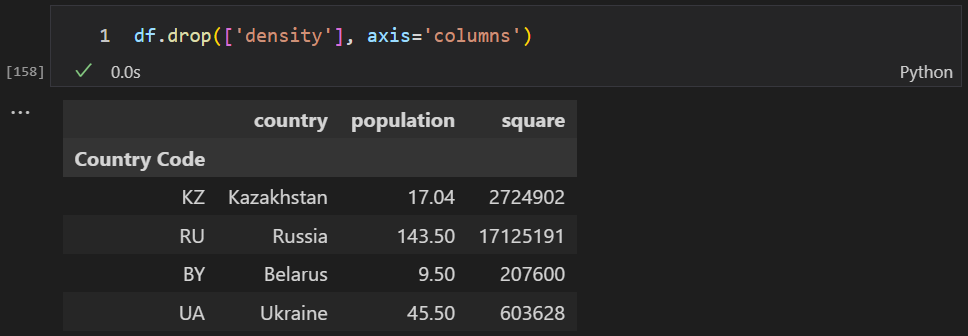
Фильтровать DataFrame с помощью т.н. булевых массивов:

Кстати, к столбцам можно обращаться, используя атрибут или нотацию словарей Python, т.е. df.population и df['population'] это одно и то же.

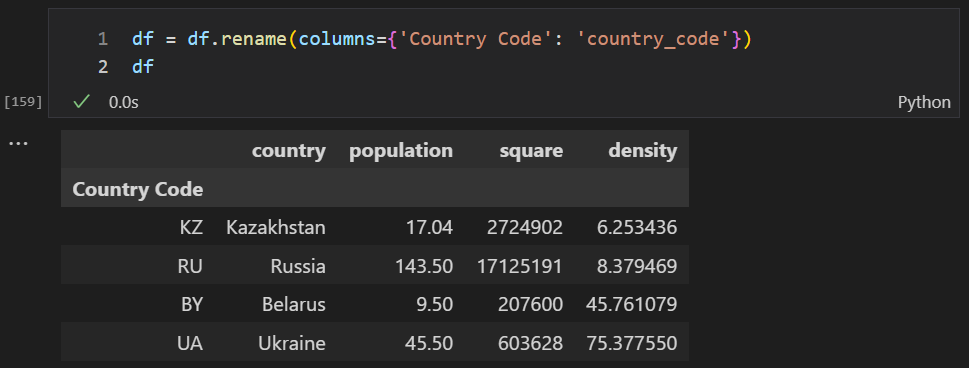
Сбросить индексы можно вот так:

Pandas при операциях над DataFrame, возвращает новый объект DataFrame.

Добавим новый столбец, в котором население (в миллионах) поделим на площадь страны, получив тем самым плотность:

Не нравится новый столбец? Не проблема, удалим его:

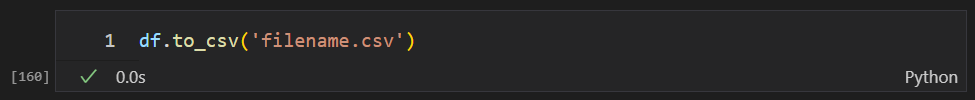
Особо ленивые могут просто написать del df['density'].

Переименовывать столбцы нужно через метод rename:

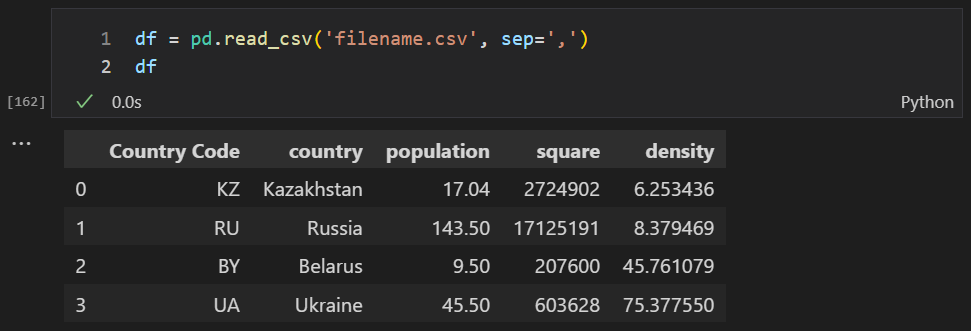
В этом примере перед тем как переименовать столбец Country Code, убедитесь, что с него сброшен индекс, иначе не будет никакого эффекта.

**Чтение и запись данных**

pandas поддерживает все самые популярные форматы хранения данных: csv, excel, sql, буфер обмена, html и многое другое:

Чаще всего приходится работать с csv-файлами. Например, чтобы сохранить наш DataFrame со странами, достаточно написать:

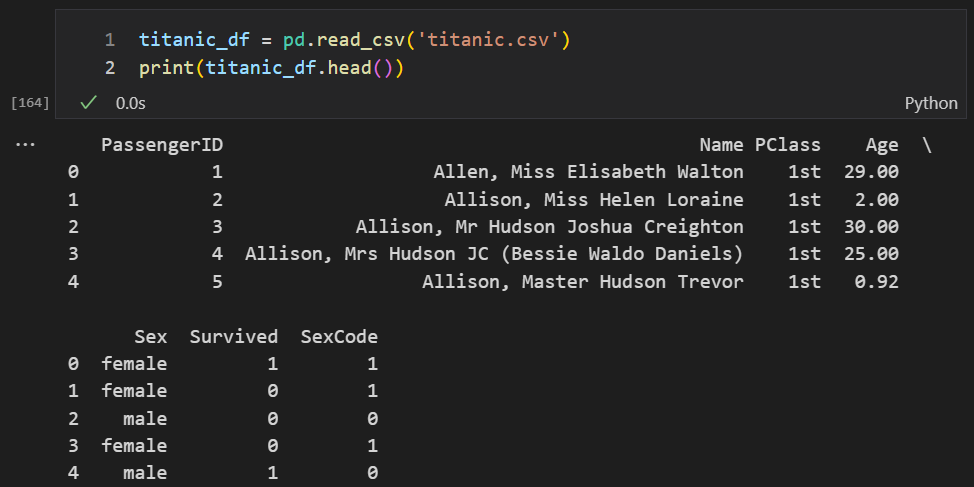
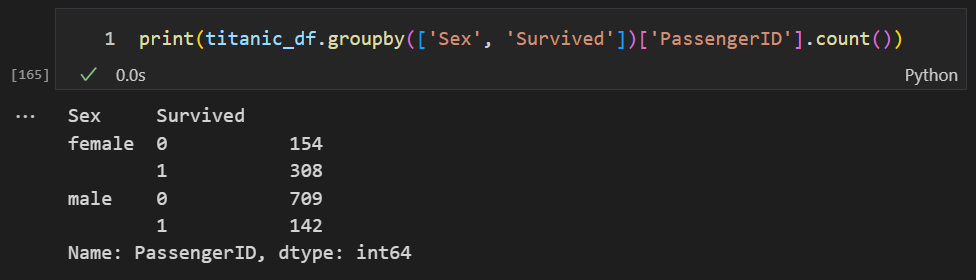
Функции to\_csv ещё передаются различные аргументы (например, символ разделителя между колонками) о которых подробнее можно узнать в официальной документации.

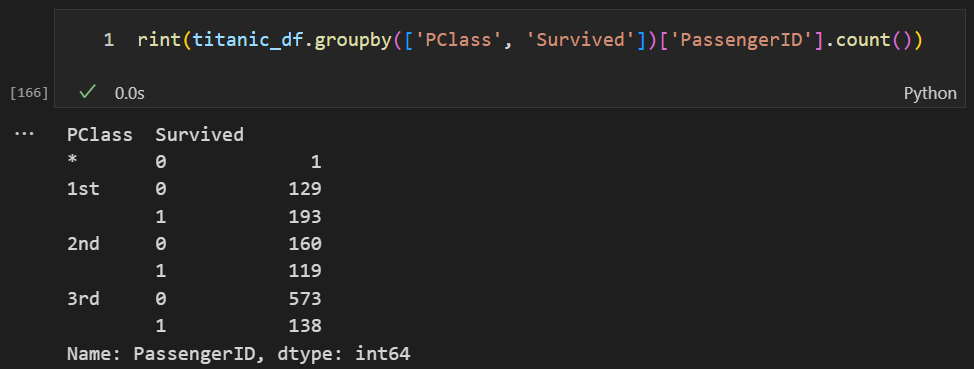
Считать данные из csv-файла и превратить в DataFrame можно функцией read\_csv.

Аргумент sep указывает разделитесь столбцов. Существует ещё масса способов сформировать DataFrame из различных источников, но наиболее часто используют CSV, Excel и SQL. Например, с помощью функции read\_sql, pandas может выполнить SQL запрос и на основе ответа от базы данных сформировать необходимый DataFrame. За более подробной информацией стоит обратиться к официальной документации.

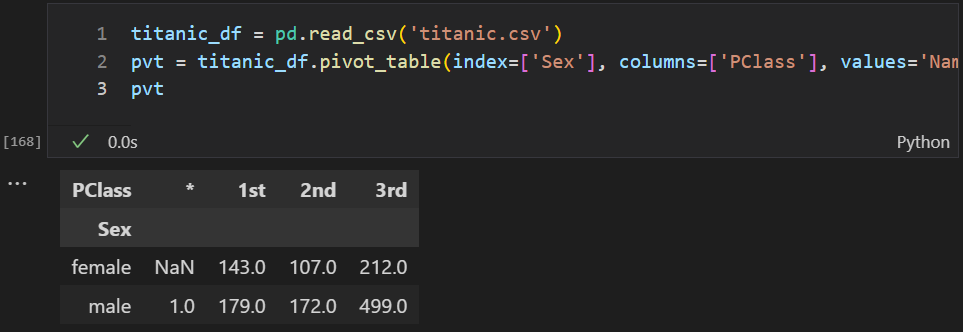
**Группировка и агрегирование в pandas**

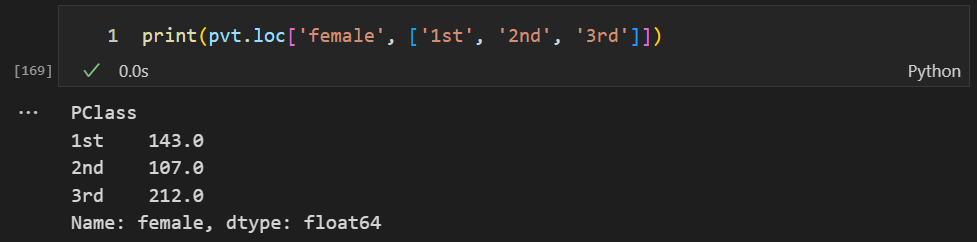
Группировка данных один из самых часто используемых методов при анализе данных. В pandas за группировку отвечает метод .groupby. Я долго думал какой пример будет наиболее наглядным, чтобы продемонстрировать группировку, решил взять стандартный набор данных (dataset), использующийся во всех курсах про анализ данных — данные о пассажирах Титаника.

Необходимо подсчитать, сколько женщин и мужчин выжило, а сколько нет. В этом нам поможет метод .groupby.

А теперь проанализируем в разрезе класса каюты:

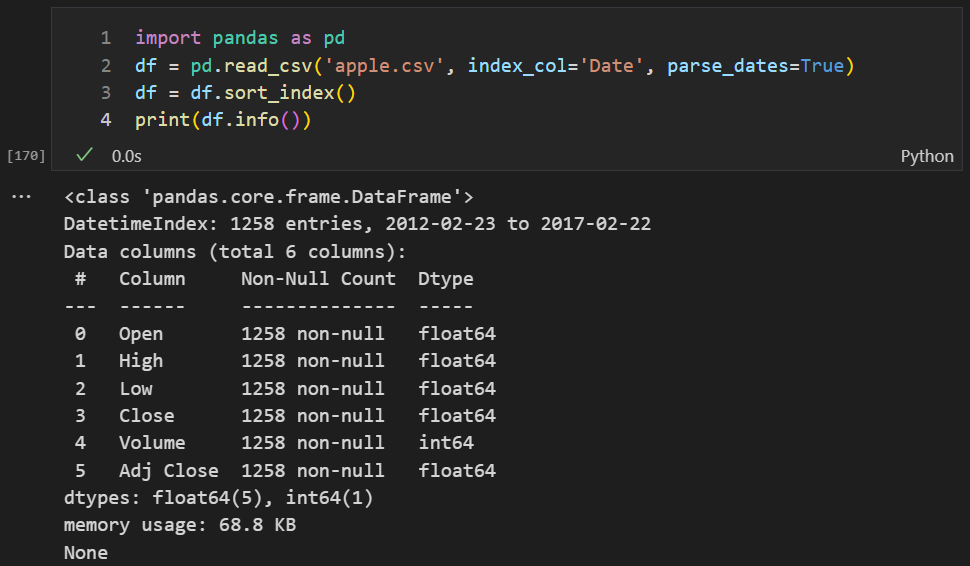
**Сводные таблицы в pandas**

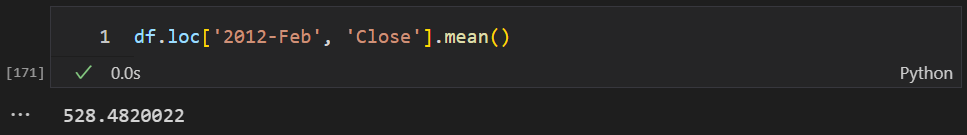
Термин "сводная таблица" хорошо известен тем, кто не по наслышке знаком с инструментом Microsoft Excel или любым иным, предназначенным для обработки и анализа данных. В pandas сводные таблицы строятся через метод .pivot\_table. За основу возьмём всё тот же пример с Титаником. Например, перед нами стоит задача посчитать сколько всего женщин и мужчин было в конкретном классе корабля:

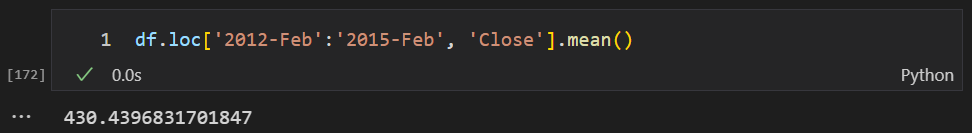
В качестве индекса теперь у нас будет пол человека, колонками станут значения из PClass, функцией агрегирования будет count (подсчёт количества записей) по колонке Name.

**Анализ временных рядов**

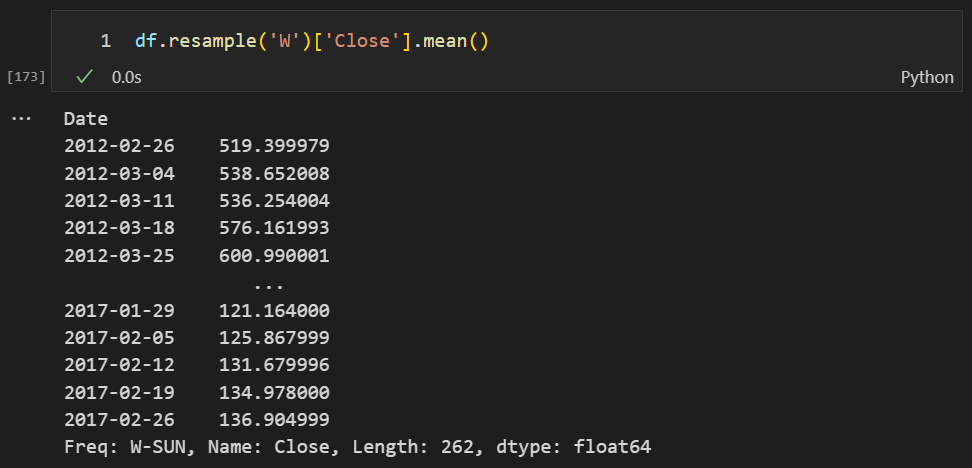
В pandas очень удобно анализировать временные ряды. В качестве показательного примера я буду использовать цену на акции корпорации Apple за 5 лет по дням. Файл с данными можно скачать [тут](https://yadi.sk/d/po_usmXT3ExwzV).

Здесь мы формируем DataFrame с DatetimeIndex по колонке Date и сортируем новый индекс в правильном порядке для работы с выборками. Если колонка имеет формат даты и времени отличный от ISO8601, то для правильного перевода строки в нужный тип, можно использовать метод [pandas.to\_datetime](http://pandas.pydata.org/pandas-docs/stable/generated/pandas.to_datetime.html).

Давайте теперь узнаем среднюю цену акции (mean) на закрытии (Close):

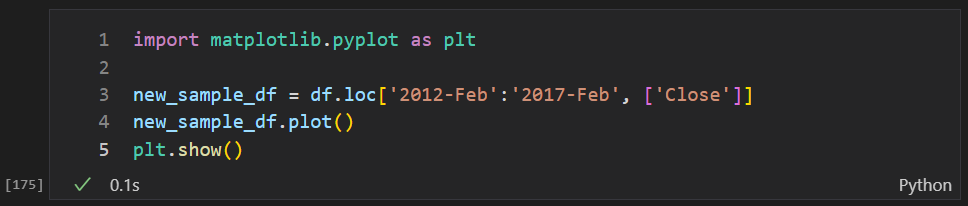
А если взять промежуток с февраля 2012 по февраль 2015 и посчитать среднее:

А что если нам нужно узнать среднюю цену закрытия по неделям?!

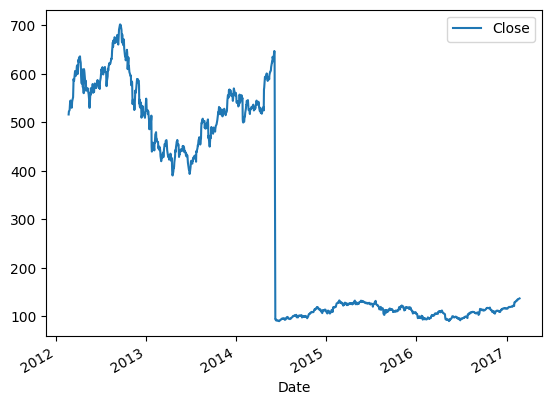
Resampling мощный инструмент при работе с временными рядами (time series), помогающий переформировать выборку так, как удобно вам. Метод resample первым аргументом принимает строку rule. Все доступные значения можно найти в [документации](http://pandas.pydata.org/pandas-docs/stable/timeseries.html#offset-aliases).

# **Визуализация данных в Python**

Для визуального анализа данных, pandas использует библиотеку ***matplotlib***. Продемонстрирую простейший способ визуализации в pandas на примере с акциями Apple.

Берём цену закрытия в промежутке между 2012 и 2017.

И видим вот такую картину:



По оси X, если не задано явно, всегда будет индекс. По оси Y в нашем случае цена закрытия. Если внимательно посмотреть, то в 2014 году цена на акцию резко упала, это событие было связано с тем, что Apple проводила сплит 7 к 1. Так мало кода и уже более-менее наглядный анализ.

# **Заключение**

Комплексные типы данных и специализированные библиотеки, такие как NumPy и Pandas, являются основополагающими инструментами для эффективного программирования на Python. Коллекции, включая списки, кортежи, множества и словари, предоставляют гибкие и мощные средства для организации и управления данными. Они позволяют разработчикам легко реализовывать алгоритмы и решать широкие спектры задач, от простых до сложных. Понимание и умение работать с этими структурами данных — важный шаг для любого программиста, стремящегося использовать Python в полной мере.

NumPy и Pandas значительно расширяют возможности Python, особенно в области научных и инженерных вычислений, анализа данных и машинного обучения. NumPy предлагает высокопроизводительные массивы и матрицы, а также широкий спектр математических функций, которые делают обработку числовых данных более эффективной. Pandas, с его удобными структурами данных, такими как DataFrame, и мощными инструментами для манипуляции и анализа данных, стал стандартом де-факто в сообществе аналитиков данных. Использование этих библиотек позволяет решать задачи, которые выходят далеко за пределы возможностей стандартных коллекций Python.

В конечном итоге, глубокое понимание и умение использовать как встроенные комплексные типы данных Python, так и продвинутые библиотеки, такие как NumPy и Pandas, открывают перед разработчиками и аналитиками данные широкие горизонты. Эти инструменты не только повышают эффективность и производительность кода, но и позволяют решать более сложные и масштабные задачи. Освоение этих технологий является важным шагом для любого специалиста, работающего с данными, и способствует более глубокому пониманию возможностей и потенциала языка программирования Python.

# **Список литературы**

1. https://khashtamov.com/ru/pandas-introduction/

2. https://ru.hexlet.io/courses/python-basics/lessons/tuples/theory\_unit

3. https://numpy.org/doc/stable/reference/generated/numpy.ufunc.html# numpy.ufunc

4. https://habr.com/ru/companies/ruvds/articles/500426/

5. https://proglib.io/p/django-pandas-i-chart-js-dlya-bystroy-paneli-instrumentov-2020-11-18

6. https://devpractice.ru/matplotlib-lesson-3-1-work-with-legend/