Лабораторная работа №2

Начало работы с данными в pandas и numpy.

Оглавление

[Теоретические сведения 2](#_Toc145592165)

[Библиотеки NumPy, Pandas 2](#_Toc145592166)

[Основные объекты и типы данных NumPy 2](#_Toc145592167)

[Типы данных поддерживаемые NumPy 4](#_Toc145592168)

[Параметр dtype 5](#_Toc145592169)

[Функции генерации массива 6](#_Toc145592170)

[Индексы и срезы 9](#_Toc145592171)

[Ход работы 11](#_Toc145592172)

[NumPy 11](#_Toc145592173)

[Pandas 14](#_Toc145592174)

[Задания 21](#_Toc145592175)

[1. Использование команд библиотеки NumPy 21](#_Toc145592176)

[2. Использование команд библиотеки pandas 22](#_Toc145592177)

[3. Результаты 22](#_Toc145592178)

**Цель лабораторной работы**: изучить основные принципы работы с библиотеками Python NumPy и Pandas, освоить типы данных и методы работы с массивами данных, а также научиться импортировать и экспортировать данные из и в форматы .csv и .xlsx.

**Задачи лабораторной работы включают:**

1. Ознакомление с библиотеками NumPy и Pandas, их установка и импорт.

2. Создание массивов данных с использованием NumPy, ознакомление с основными типами данных (ndarray) и операциями над ними (индексация, срезы, математические операции).

3. Знакомство с библиотекой Pandas, создание и работа с DataFrame для хранения и обработки табличных данных.

4. Импорт данных из файлов .csv и .xlsx в объекты Pandas DataFrame.

5. Выполнение базовых операций с данными в DataFrame, таких как фильтрация, сортировка и группировка.

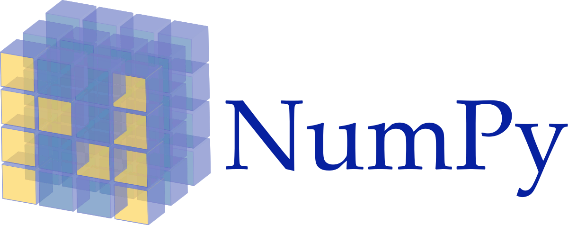
6. Экспорт данных из DataFrame в файлы .csv и .xlsx для сохранения результатов анализа.

# Теоретические сведения

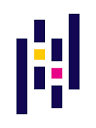
## Библиотеки NumPy, Pandas

**Основными библиотеками Python для работы с данными являются NumPy, pandas.**

**NumPy** — библиотека с открытым исходным кодом для Python. Возможности:

* поддержка многомерных массивов (включая матрицы);
* поддержка высокоуровневых математических функций, предназначенных для работы с многомерными массивами.

Документация: <http://www.numpy.org/>

 **pandas** — программная библиотека на языке Python для обработки и анализа данных. Работа pandas с данными строится поверх библиотеки NumPy, являющейся инструментом более низкого уровня. Предоставляет специальные структуры данных и операции для манипулирования числовыми таблицами и временны́ми рядами.

Документация: <https://pandas.pydata.org/>

## Основные объекты и типы данных NumPy

Основной элемент **библиотеки NumPy** — объект **ndarray** (что значит N-размерный массив). Этот объект является многомерным однородным массивом с заранее заданным количеством элементов. Однородный — потому что практически все объекты в нем одного размера или типа. На самом деле, тип данных определен другим объектом NumPy, который называется **dtype** (тип-данных). Каждый **ndarray** ассоциирован только с одним типом **dtype**.

Количество размерностей и объектов массива определяются его размерностью (**shape**), кортежем N-положительных целых чисел. Они указывают размер каждой размерности. Размерности определяются как оси, а количество осей — как ранг.

Еще одна странность массивов NumPy в том, что их размер фиксирован, а это значит, что после создания объекта его уже нельзя поменять. Это поведение отличается от такового у **списков Python**, которые могут увеличиваться и уменьшаться в размерах.

***Разница между массивами, списками и кортежами:***

\* Размер фиксирован, изменяемый: массив (**array**)

\* Размер фиксирован, неизменяемый: кортеж (**tuple**)

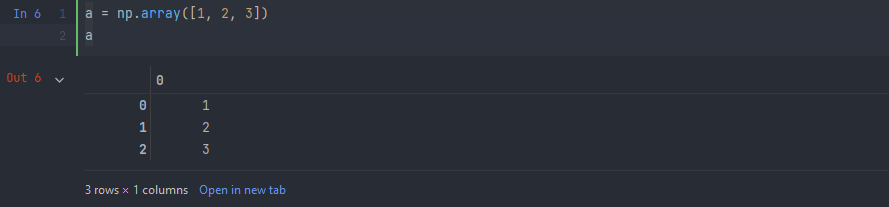
\* Размер не фиксирован, изменяемый: список (**list**)

Простейший способ определить новый объект **ndarray** — использовать функцию **array**(), передав в качестве аргумента Python-список элементов.

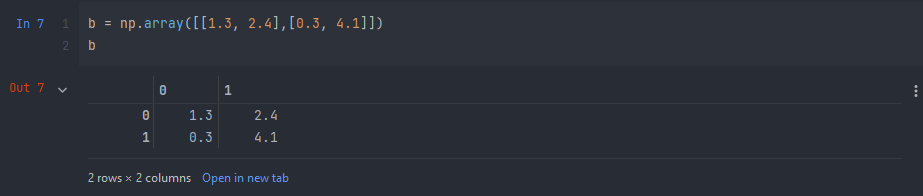
Примечание: В **IDE PyCharm** необходимо выполнить команду **File – Settings – Project**\_(название вашего проекта) – **Project Interpretation**. Нажать **«+»**, найти в поиске нужные библиотеки и нажать кнопку **Package Install. Таким образом, установились нужные нам библиотеки NumPy, pandas.**



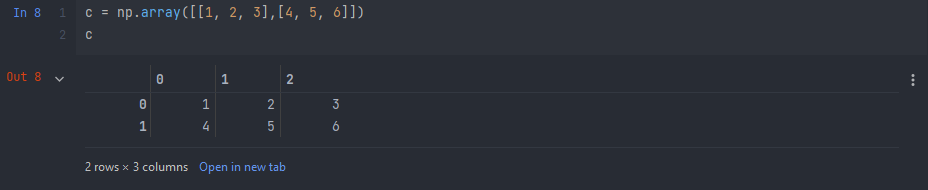
**Пример 1**



**Пример 2**

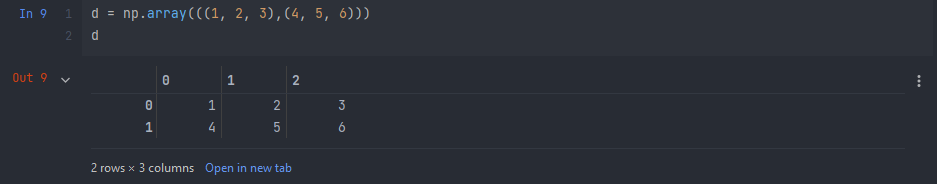


**Пример 3**



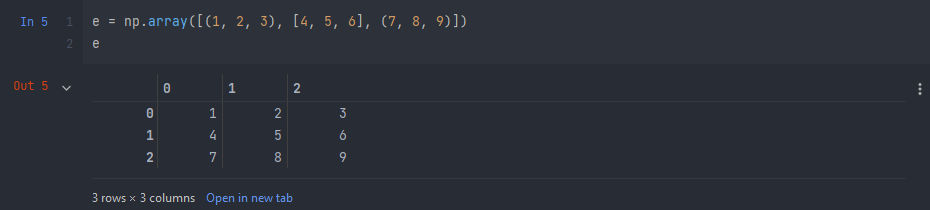
Функция **array()** также может принимать кортежи и последовательности кортежей.

**Пример 4**



Она также может принимать последовательности кортежей и взаимосвязанных списков.

**Пример 5**



## Типы данных поддерживаемые NumPy

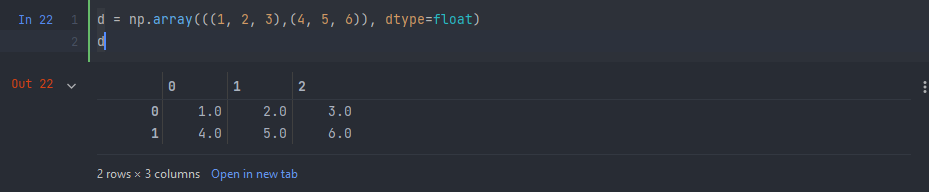
**Таблица 1 Типы данных NumPy**

|  |  |
| --- | --- |
| **Типы данных** | **Описание** |
| **bool** | Булевы значения (True или False) хранятся в виде байтов |
| **int** | Тип по умолчанию — целое число (то же, что long в C; обычно int64 или int32) |
| **intc** | Идентичный int в C (обычно int32 или int64) |
| **intp** | Целое число для использования в качестве индексов (то же, что и size\_t в C, обычно int32 или int64) |
| **int8** | Байт (от — 128 до 127) |
| **int16** | Целое число (от -32768 до 32767) |
| **Int32** | Целое число (от -2147483648 до 2147483647) |
| **Int64** | Целое число (от -9223372036854775808 до 9223372036854775807) |
| **uint8** | Целое число без знака (от 0 до 255) |
| **uint16** | Целое число без знака (от 0 до 65535) |
| **uint32** | Целое число без знака (от 0 до 4294967295) |
| **uint64** | Целое число без знака (от 0 до 18446744073709551615) |
| **float** | Обозначение float64 |
| **float16** | Число с плавающей точкой половинной точности; бит на знак, 5-битная экспонента, 10-битная мантисса |
| **float32** | Число с плавающей точкой единичной точности; бит на знак, 8-битная экспонента, 23-битная мантисса |
| **float64** | Число с плавающей точкой двойной точности; бит на знак, 11-битная экспонента, 52-битная мантисса |
| **complex** | Обозначение complex128 |
| **complex64** | Комплексное число, представленное двумя 32-битными float (с действительной и мнимой частями) |
| **complex128** | Комплексное число, представленное двумя 64-битными float (с действительной и мнимой частями) |

## Параметр dtype

Функция **array**() не принимает один аргумент. На примерах видно, что каждый объект **ndarray** ассоциирован с объектом **dtype**, определяющим **тип данных**, которые будут в массиве. По умолчанию функция **array**() можно ассоциировать самый подходящий тип в соответствии со значениями в последовательностях списков или кортежей. Их можно определить явно с помощью параметра **dtype** в качестве аргумента.

**Пример 6**

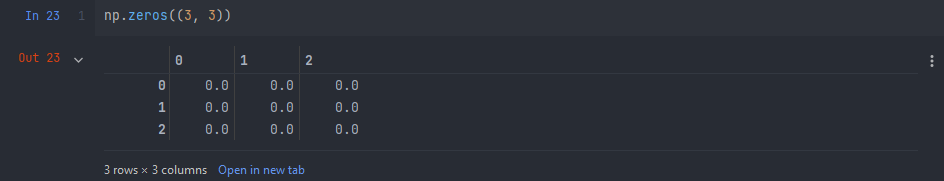


## Функции генерации массива

Библиотека **NumPy** предоставляет набор функций, которые генерируют ndarray с начальным содержимым. Они создаются с разным значениями в зависимости от функции. Это очень полезная особенность. С помощью всего одной строки кода можно сгенерировать большой объем данных.

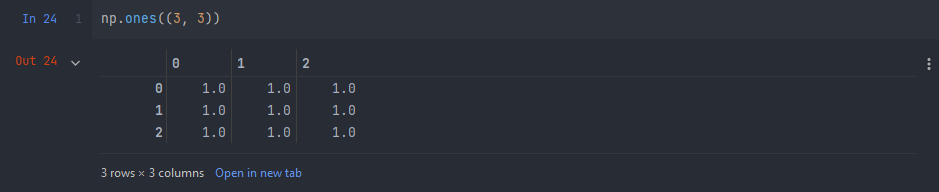
Функция **zeros**(), например, создает полный массив нулей с размерностями, определенными аргументом **shape**.

**Пример 7**



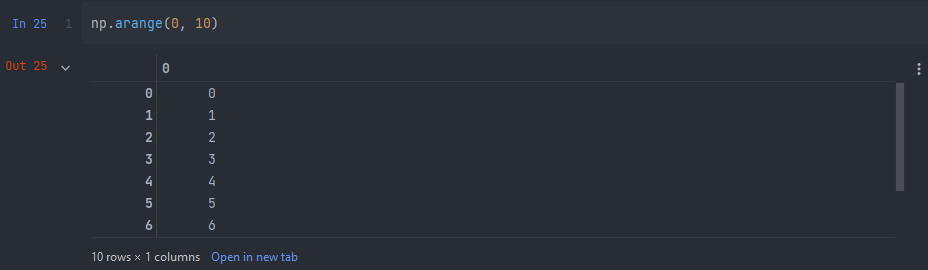
А функция **ones**() создает массив, состоящий из единиц.

**Пример 8**



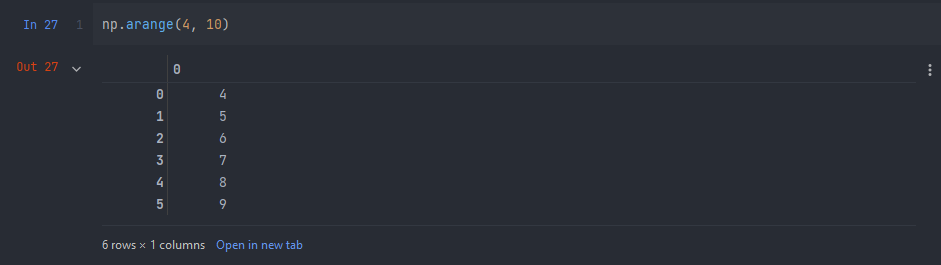
По умолчанию две функции создают массивы с типом данных **float64**. Полезная возможность — **arrange**(). Она генерирует массивы **NumPy** с числовыми последовательностями, которые соответствуют конкретным требованиям в зависимости от переданных аргументов.

**Пример 9**



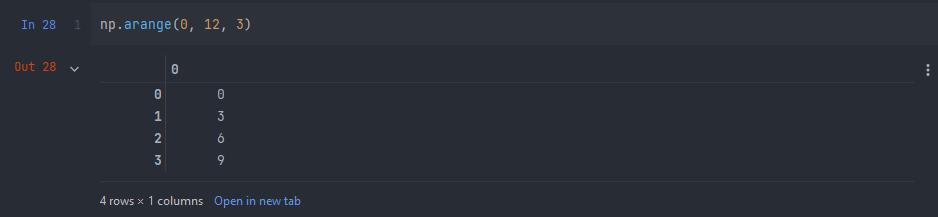
Если в начале нужен не ноль, то необходимо обозначить уже два аргумента: первый и последний.

**Пример 10**



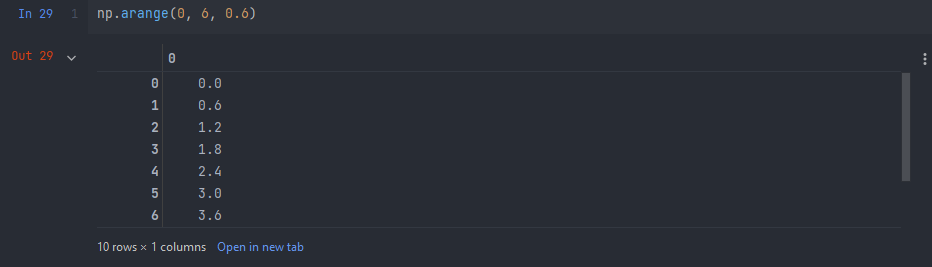
Также можно сгенерировать последовательность значений с точным интервалом между ними. Если определено и третье значение в **arrange**(), оно будет представлять собой промежуток между каждым элементом.

**Пример 11**



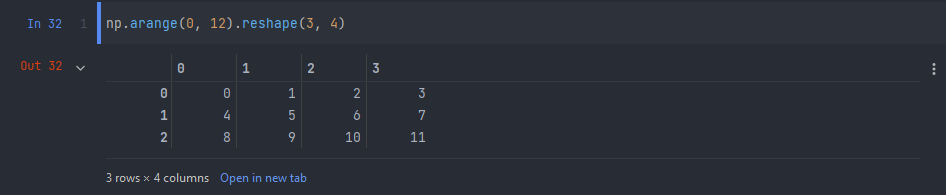
Оно может быть и числом с плавающей точкой.

**Пример 12**



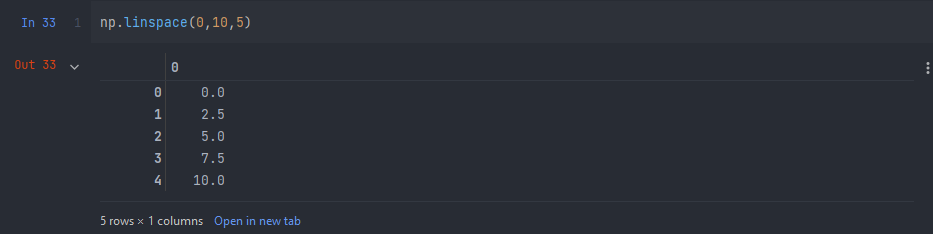
Для генерации двумерных массивов все еще можно использовать функцию **arrange**(), но вместе с **reshape**(). Она делит линейный массив на части способом, который указан в аргументе **shape**.

**Пример 13**



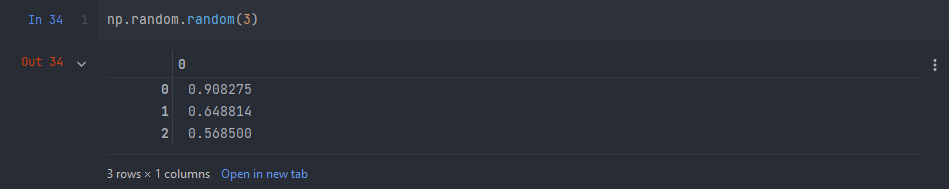
Похожая на **arrange**() функция — **linspace**(). Она также принимает в качестве первых двух аргументов первое и последнее значения последовательности, но третьим аргументом является не интервал, а количество элементов, на которое нужно разбить последовательность.

**Пример 14**



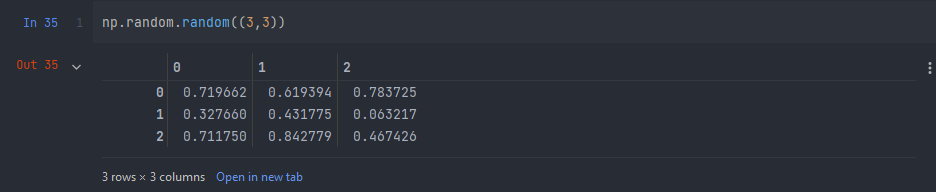
Еще один способ получения массива — заполнение его случайными значениями. Это можно сделать с помощью функции **random**() из модуля **numpy.random**. Эта функция генерирует массив с тем количеством элементов, которые указаны в качестве аргумента.

**Пример 15**



Полученные числа будут отличаться с каждым запуском. Для создания многомерного массива, нужно передать его размер в виде аргумента.

**Пример 16**



## Индексы и срезы

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).

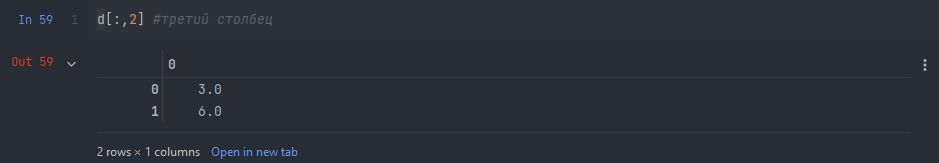
 Общий синтаксис срезов выглядит так:

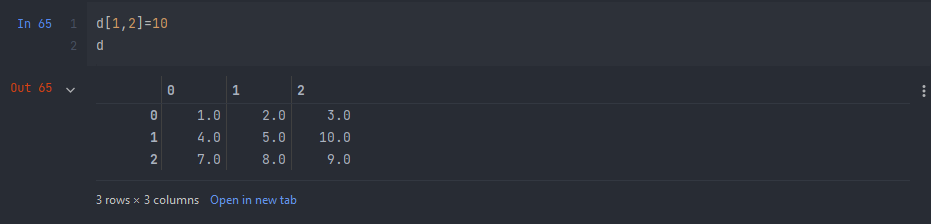
<имя массива>[start:stop:step]

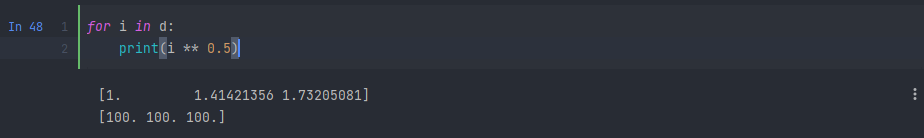
Дополнительная информация:

<https://proproprogs.ru/modules/numpy-indeksaciya-srezy-iterirovanie-massivov>

**Пример 17**

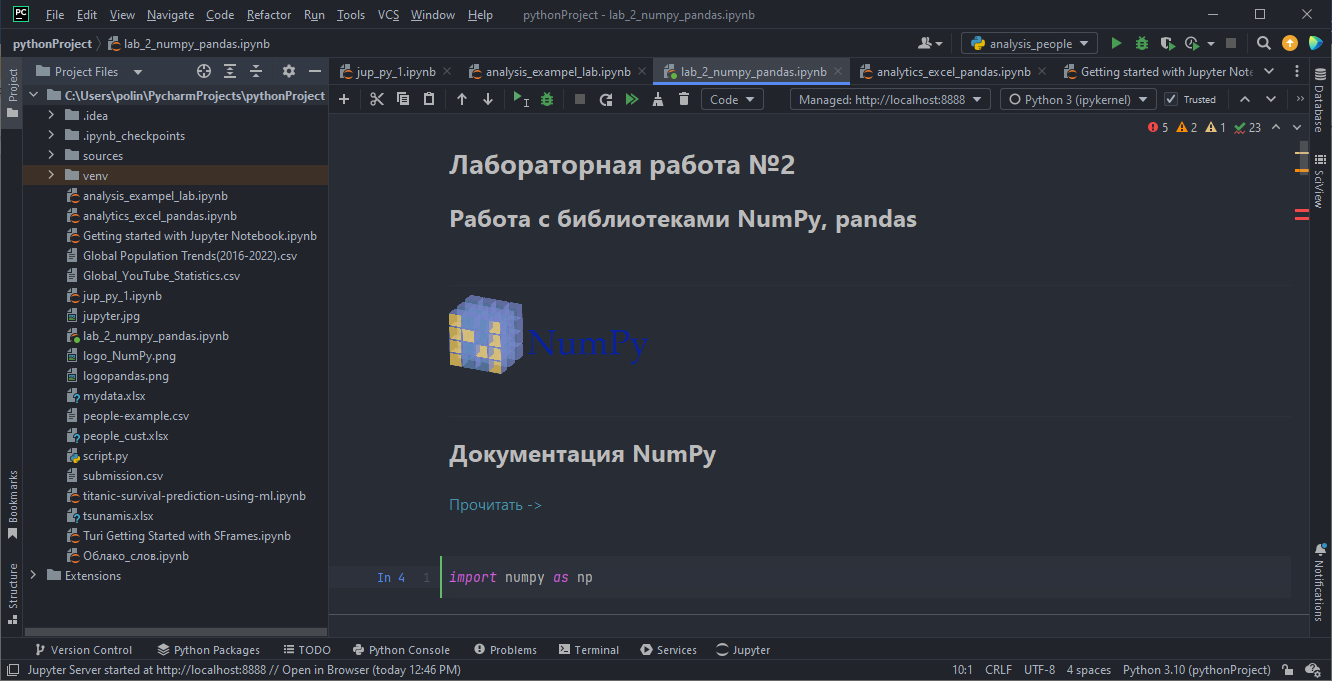


****



# Ход работы

Создадим новый **Jupyter** блокнот с подключением библиотек **numpy**, **pandas** (см. рис. 1)

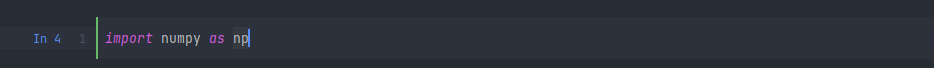


*Рис. 1 Создание нового файла в среде Jupyter Notebook*

В **IDE PyCharm** необходимо выполнить команду **File – Settings – Project**\_(название вашего проекта) – **Project Interpretation**. Нажать **«+»**, найти в поиске нужные библиотеки и нажать кнопку **Package Install. Таким образом, установились нужные нам библиотеки NumPy, pandas.**

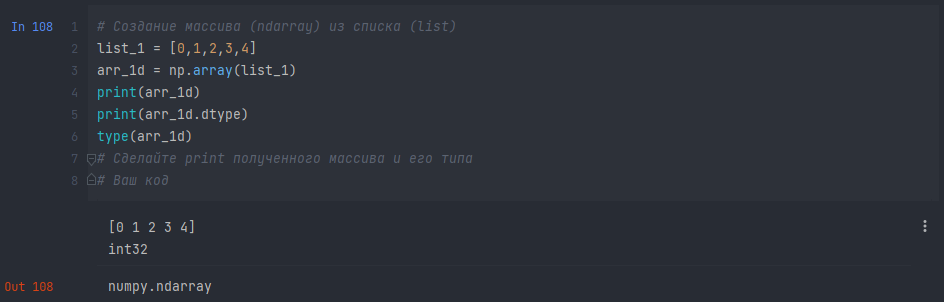
## NumPy

**Подключение библиотек осуществляется через команду import**



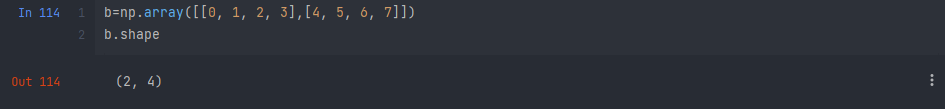
Рассмотрим возможности пакета и некоторые примеры пакета NumPy.

Создадим массив из списка (см. рис. 2)



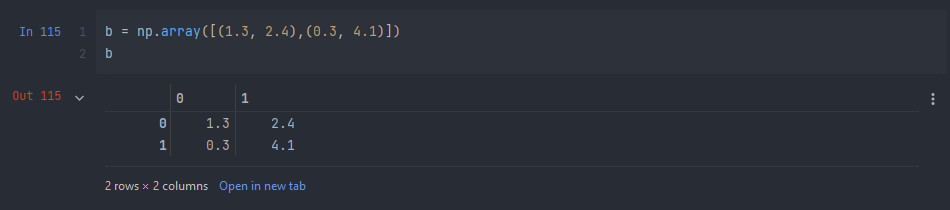
*Рис. 2 Код выполнения и результат*

Проверка размерности массива, см. рис. 3



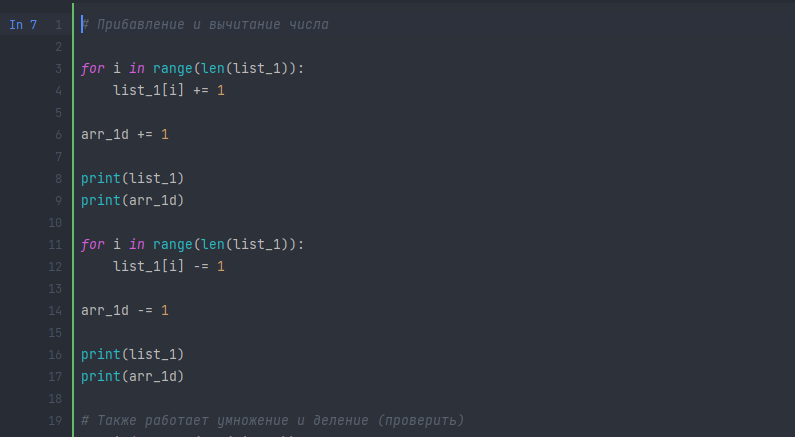
*Рис. 3 Вызов команды shape()*

Создание массива из последовательности кортежей показано на рис. 4



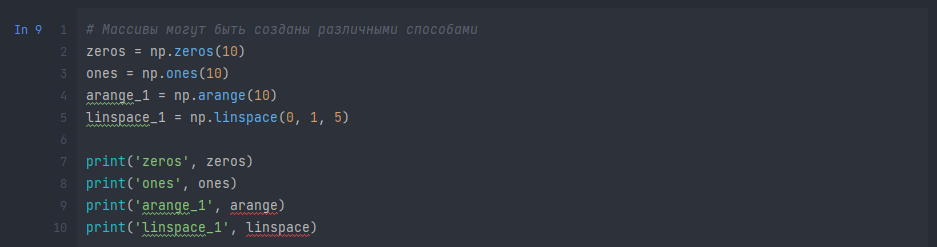
*Рис. 4 Создание массива типа float*

Арифметические действия с массивами показаны на рис. 5



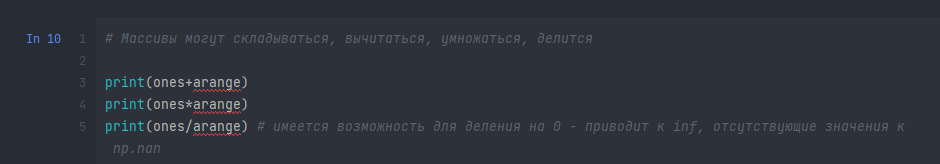
*Рис. 5 Выполнение операций сложения и вычитания*

Дополнительные возможности получения массивов с помощью функций **zeros**(), **ones**(), **arrange**(), **linspace**(), продемонстрированы на рис. 6



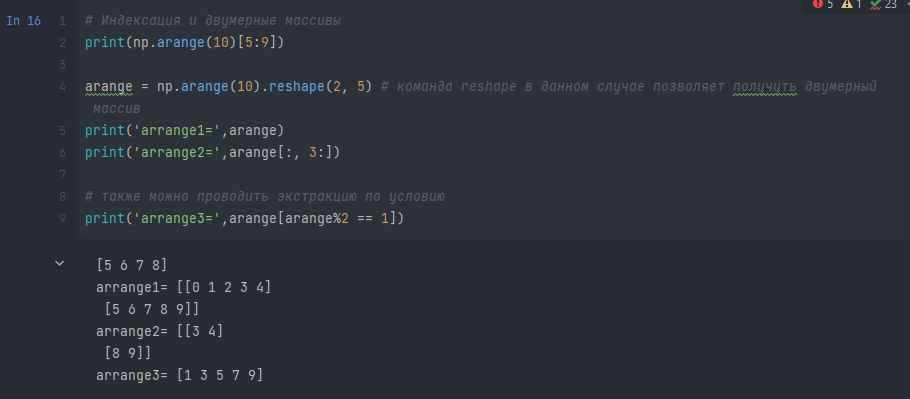
*Рис. 6 Дополнительные команды для создания массивов*

Действия над массивами показаны на рис. 7 ниже.

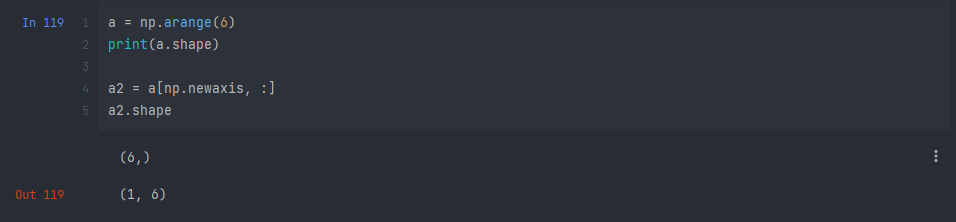


*Рис. 7 Действия над массивами*

Ниже приведены примеры индексации элементов массива



*Рис. 8 Индексация и двумерные массивы*



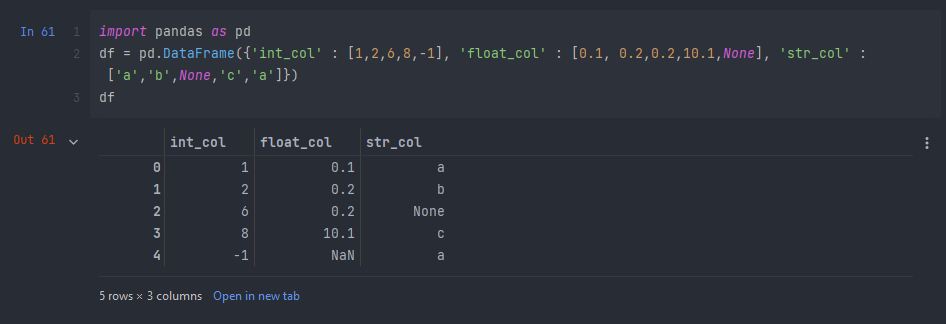
*Рис. 9 Применение метода np.newaxis*

Использование **np.newaxis** увеличит размеры вашего массива на одно измерение при однократном использовании. Это означает, что 1D-массив станет 2D-массивом, 2D-массив станет 3D-массивом и так далее.

## Pandas

Подключение библиотеки **pandas**. Основные объекты, с которыми работают команды pandas **DataFrame.**

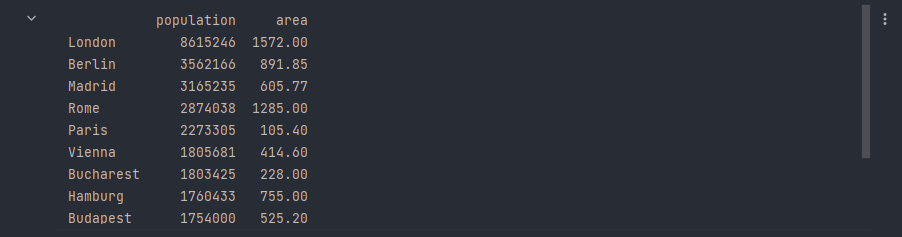
Создание DataFrame продемонстрировано на рис. 10, 10а



*Рис. 10 Создание DataFrame с разными типами данных*

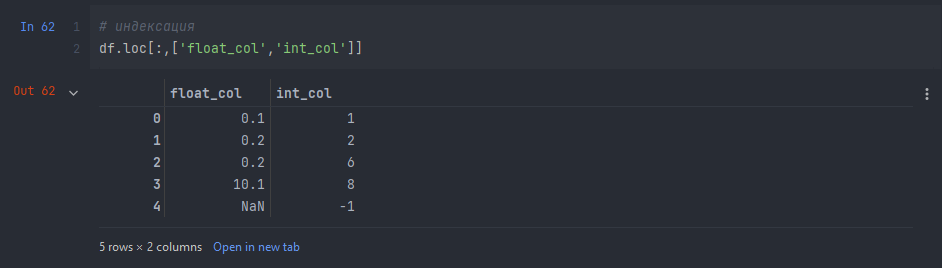
**

*Результат*

**

*Рис. 10а Создание фрейма*

Индексация элементов df может происходить так как показано на рис. 11

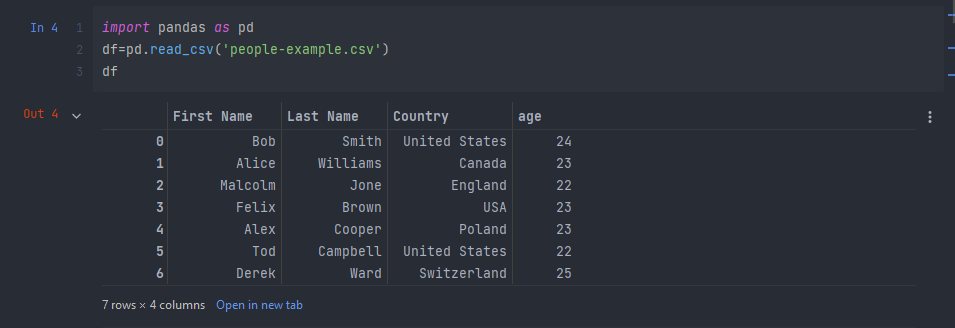


*Рис. 11 Индексация DataFrame*

Pandas позволяет создавать DataFrame импортируя данные из внешних файлов, например, **.csv, .xlsx.**

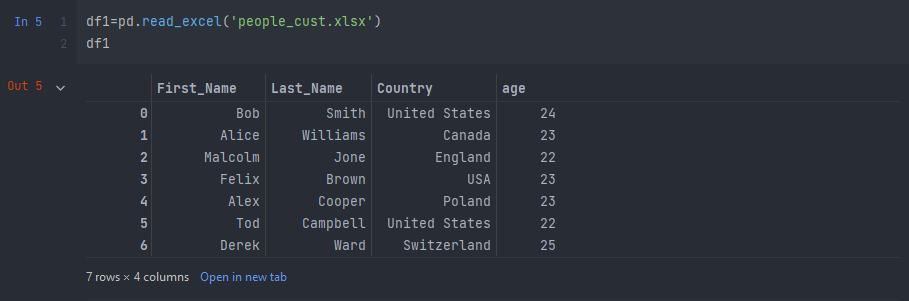
Для работы c такими данными надо установить пакет **openpyxl.**

Загрузка из таблицы .csv осуществляется с помощью команды **pd.read\_csv().** Файл “people-example.csv” в данном случае храниться в текущей папке с исходным файлом .ipynb



*Рис. 12 Импорт данных из .csv*

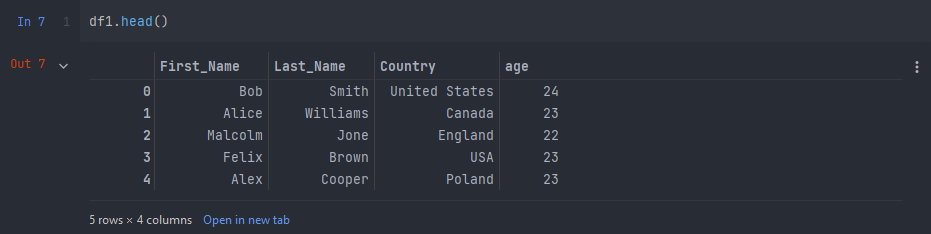
Похожим образом данные считываются из таблицы .xlsx



*Рис. 13 Чтение данных из файла .xlsx*

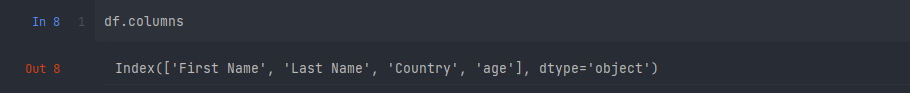
**Изучение данных**

Чтение нескольких первых записей из созданного DataFrame происходит с помощью команды **head()**, см. рис. 14



*Рис. 14 Чтение данных из df1*

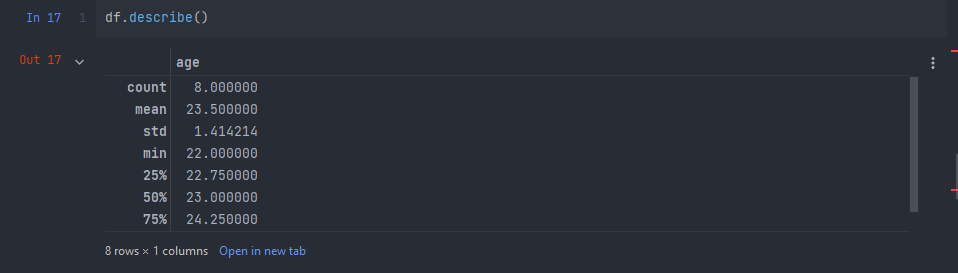
Методом **columns** можно считать название столбцов (полей) таблицы (фрейма), см. рис. 15.



*Рис. 15 Вывод названий столбцов фрейма*

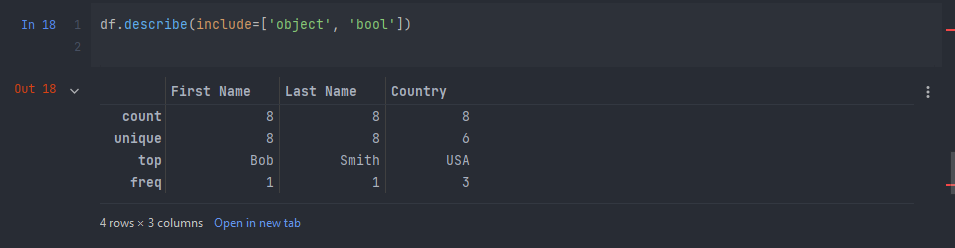
**Описательная статистика**

Команда **describe()**, может вывести характеристики **описательной аналитики** (статистические характеристики: среднее, медиана, мода, среднеквадратичное отклонение, максимальное и минимальное значения), как показано на рис. 16



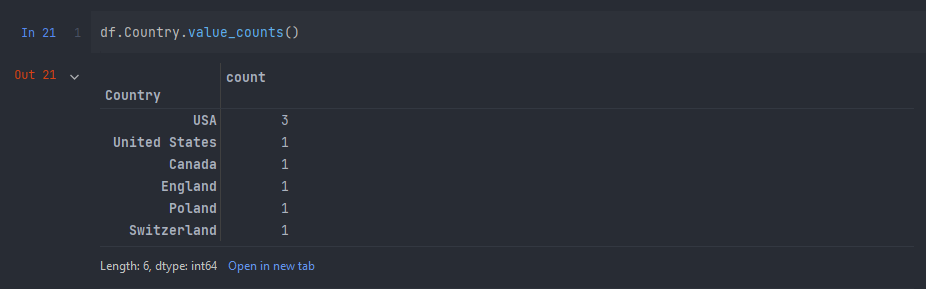
*Рис. 16 Описательная статистика*

Статистика по другим переменным

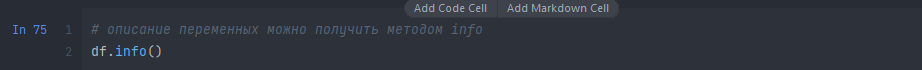


*Рис. 17 Вызов статистики*

Команда **value\_counts()** определяет количество значений в столбце.

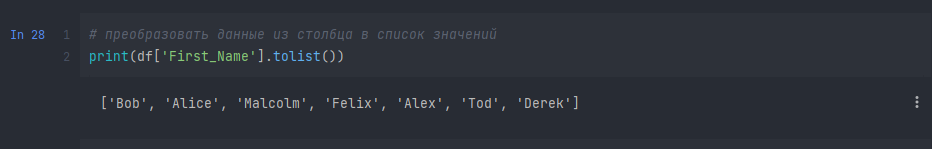


Описание переменных можно получить при помощи команды **info(),** см. рис. 18.



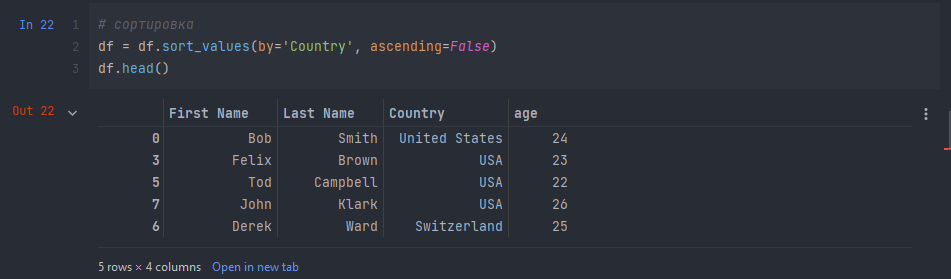
*Рис. 18 Вызов метода info()*

Преобразование данных в список значений (рис. 19):



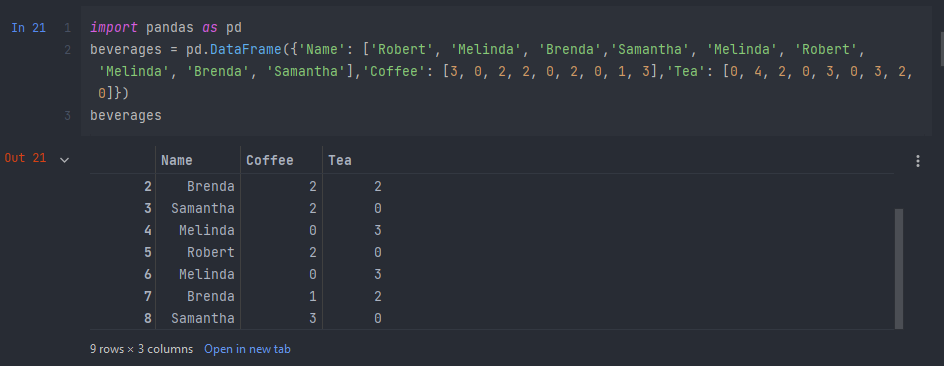
*Рис. 19 Преобразование данных в список*

Сортировка данных показана на рис. 20.

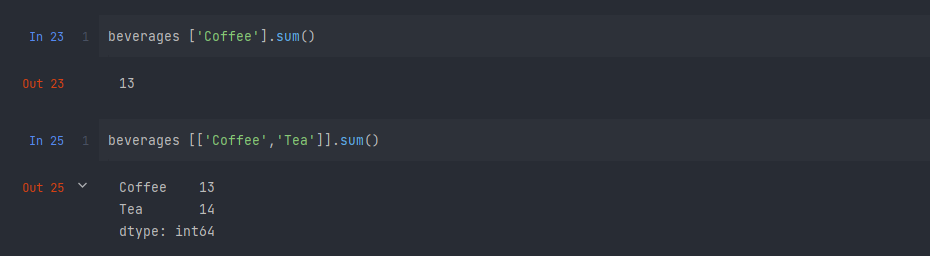


*Рис. 20. Сортировка данных*

Вызов команд **sum(), mean(), groupby()** приведены ниже на рис. 21, 22, 23



*Рис. 21 Создание фрейма данных*



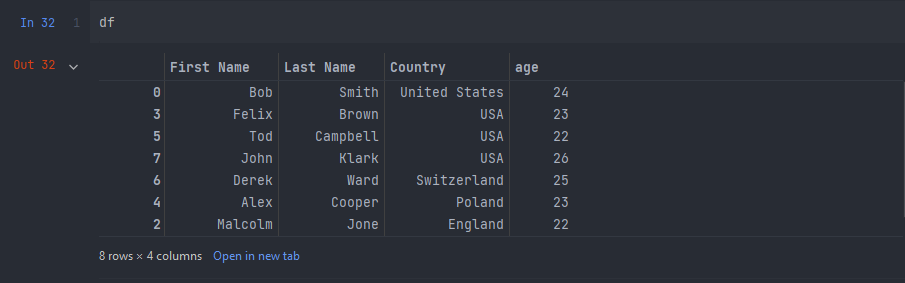
*Рис. 22 Вызов функции sum()*



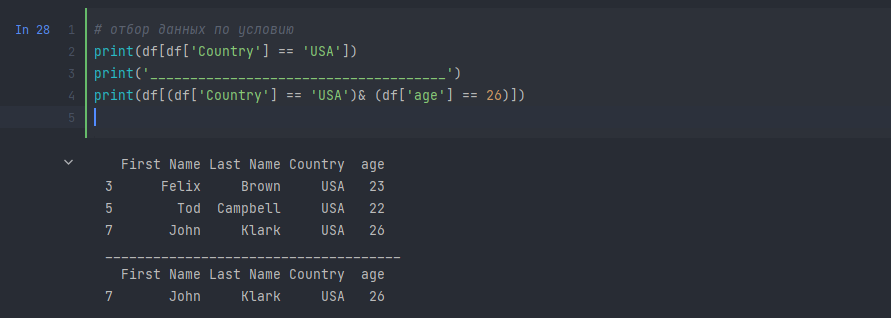
*Рис. 23 Применение группировки данных*

Отбор данных по условию (см. рис. 24):

**Исходный фрейм**

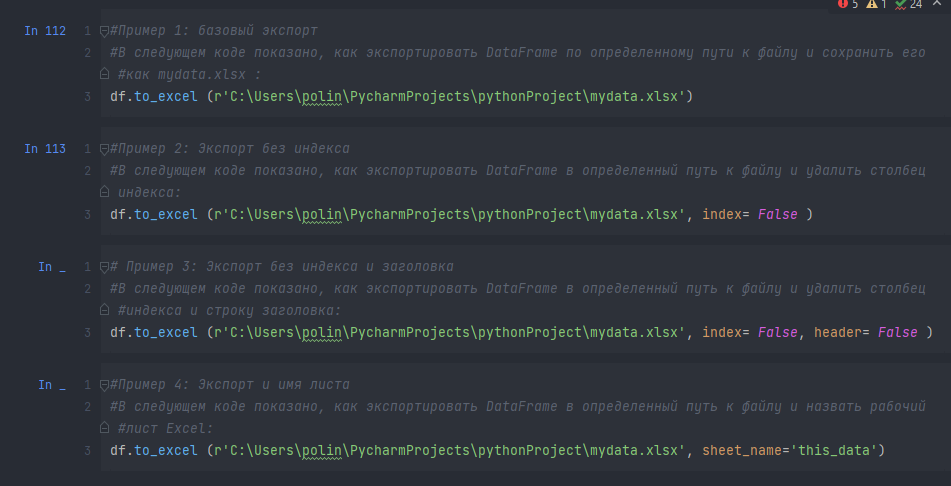


**Результат**



*Рис. 24 Условная выборка*

Экспорт данных из фрейма в файл MS Excel показан на рис. 25



*Рис. 25 Экспорт данных в формат MS Excel*

# Задания

Создайте новый файл блокнота **LAB\_2\_Name.ipynb.**

## Использование команд библиотеки NumPy

Подключите библиотеку numpy

1. Создайте одномерный массив my\_array, в котором по порядку все числа до 100, кратные 5
2. Из полученного массива получить массив с кратными 10 числами
3. Получить двумерный массив с двумя столбцами
4. Получите набор my\_second\_array, в котором все элементы меньше предыдущего исходного на 5
5. Соедините массивы, полученные в п. 3.1 и 4.1, горизонтально. Используйте .T для транспонирования матрицы.
6. Разделите 2 и 3 строку на 4 верхних строки my\_array. Используйте .T для транспонирования матрицы
7. Оставить только те значения, которые не изменились после деления
8. Снова получить трехстрочный двумерный массив и поменять местами первую и третью строки
9. Получить среднее, медиану и стандартное отклонение для каждой строки, используя команды np.mean(), np.median(), np.std().
10. Сделать нормализацию на максимальное значение вдоль строк, поделив каждый элемент массива на max() вдоль строк.

## Использование команд библиотеки pandas

Подключите библиотеку pandas

* 1. Импортируйте данные из файла “Global Population Trends(2016-2022).csv”. Описание данных смотрите ниже.

Изучите полученные данные:

* 1. Считайте несколько первых записей таблицы (head())
  2. Определите размерность данного фрейма (shape())
  3. Выведите название заголовков столбцов фрейма (columns)
  4. Получите описание всех переменных (полей) (info())
  5. Удалите строки, не содержащие значения в столбце «Total Population» - “-”

df1=df[df['Total Population']!='-']

* 1. Получите описательную статистику по переменным (например, max(), min(), describe())
  2. Выполните сортировку данных по нескольким критериям
  3. Экспорт фрейма в MS Excel

## Результаты

Результаты оформить в виде Jupyetet блокнота. Сделать оформление, заголовки, подписи, комментарии к коду

**Описание данных Global Population Trends(2016-2022).csv**

[**https://www.kaggle.com/datasets/alitaqi000/global-population-trends2016-2022**](https://www.kaggle.com/datasets/alitaqi000/global-population-trends2016-2022)

Набор данных о глобальных демографических тенденциях:

Изучите обширную коллекцию ключевых демографических показателей со всего мира. Этот набор данных охватывает общую численность населения, распределение между городскими и сельскими районами, ожидаемую продолжительность жизни, уровень рождаемости и смертности, уровень рождаемости, уровень детской смертности и темпы роста. Узнайте больше о глобальной демографической динамике и лежащих в ее основе факторах. Идеально подходит для исследований, визуализации и анализа.

**Содержание:**

1) Общая численность населения

2)Городское и сельское население

3) Плотность населения (Плотность населения — это численность среднегодового населения, деленная на площадь суши в квадратных километрах)

4) Ожидаемая продолжительность жизни (Ожидаемая продолжительность жизни при рождении указывает, сколько лет прожил бы новорожденный, если бы преобладающие тенденции смертности на момент его рождения оставались неизменными на протяжении всей его жизни)

5) Коэффициенты рождаемости и смертности (общий коэффициент указывает количество смертей, произошедших в течение года, на 1000 человек населения по оценкам на середину года)

6) Коэффициент рождаемости (Общий коэффициент рождаемости представляет собой количество детей, которые родились бы у женщины, если бы она дожила до конца детородного возраста и родила детей в соответствии с возрастными коэффициентами рождаемости указанного года.)

7) Уровень младенческой смертности (Уровень младенческой смертности — это количество младенцев, умирающих до достижения возраста одного года, на 1000 живорождений в данном году)

8) Скорость роста