```
#Pandas: анализ данных на Python

#Pandas - это высокоуровневая Python библиотека для анализа данных. Почему её называют высокоуровневой,
#потому что построена она поверх более низкоуровневой библиотеки NumPy (написана на Си),
#что является большим плюсом в производительности. В экосистеме Python,
#pandas является наиболее продвинутой и быстроразвивающейся библиотекой для обработки и анализа данных.

#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,
#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.

import pandas
my_series = pandas.Series([5, 6, 7, 8, 9, 10])
my_series

#Результат действий
# 0     5
# 1     6
# 2     7
# 3     8
# 4     9
# 5    10
# dtype: int64

#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.

my_series.index
```

```
#Результат
# RangeIndex(start=0, stop=6, step=1)

my_series.values
#Результат
# array([ 5,  6,  7,  8,  9, 10], dtype=int64)

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

my_series[4]

#Результат
# 9

#Индексы можно задавать явно:
my_series2 = pandas.Series([5, 6, 7, 8, 9, 10], index=['a', 'b', 'c', 'd', 'e', 'f'])
my_series2['f']

#Результат
# 10

#Делать выборку по нескольким индексам и осуществлять групповое присваивание:

my_series2[['a', 'b', 'f']]
#Результат
# a     5
# b     6
# f    10
# dtype: int64

my_series2[['a', 'b', 'f']] = 0
my_series2
#Результат
# a    0
# b    0
# c    7
# d    8
# e    9
# f    0
# dtype: int64

#Можно также фаильтровать Series как душе захочется, а также применять математические
операции и многое другое:

my_series2[my_series2 > 0]
# c    7
# d    8
# e    9
# dtype: int64

my_series2[my_series2 > 0] * 2
#c    14
```

```
#d    16
#e    18
#dtype: int64

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

my_series3 = pandas.Series({'a': 5, 'b': 6, 'c': 7, 'd': 8})
my_series3

#Результат
# a    5
# b    6
# c    7
# d    8
# dtype: int64

'd' in my_series3
#Результат
True

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

my_series3.name = 'numbers'
my_series3.index.name = 'letters'
my_series3

#Результат
# letters
# a    5
# b    6
# c    7
# d    8
# Name: numbers, dtype: int64


#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:
#Получается очень симпотичная табличка

df = pandas.DataFrame({
    'country': ['Kazakhstan', 'Russia', 'Belarus', 'Ukraine'],
    'population': [17.04, 143.5, 9.5, 45.5],
    'square': [2724902, 17125191, 207600, 603628]
    })
df
```

```
#Результат
#      country      population    square
# 0  Kazakhstan       17.04       2724902
# 1      Russia      143.50       17125191
# 2     Belarus        9.50       207600
# 3     Ukraine       45.50       603628


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

df['country']

#Результат
#0    Kazakhstan
#1        Russia
#2       Belarus
#3       Ukraine
#Name: country, dtype: object

type(df['country'])

#Результат
#pandas.core.series.Series

#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.

df.columns

#Результат
#Index(['country', 'population', 'square'], dtype='object')

df.index
#Результат
#RangeIndex(start=0, stop=4, step=1)

#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:

df = pandas.DataFrame({
    'country': ['Kazakhstan', 'Russia', 'Belarus', 'Ukraine'],
    'population': [17.04, 143.5, 9.5, 45.5],
    'square': [2724902, 17125191, 207600, 603628]
}, index=['KZ', 'RU', 'BY', 'UA'])

df
```

```
#Результат
#        country     population     square
# KZ   Kazakhstan       17.04      2724902
# RU      Russia      143.50       17125191
# BY     Belarus        9.50       207600
# UA     Ukraine       45.50       603628

df.index = ['KZ', 'RU', 'BY', 'UA']
df.index.name = 'Country Code'
df

#Результат
#                 country    population   square
#Country Code
#KZ            Kazakhstan       17.04     2724902
#RU                Russia      143.50     17125191
#BY               Belarus        9.50     207600
#UA               Ukraine       45.50     603628

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

df['country']
#Результат
#Country Code
#KZ    Kazakhstan
#RU        Russia
#BY       Belarus
#UA       Ukraine
#Name: country, dtype: object

#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)

df.loc['KZ']
#Результат
#country       Kazakhstan
#population         17.04
#square           2724902
#Name: KZ, dtype: object

df.iloc[0]
#Результат
#country       Kazakhstan
#population         17.04
#square           2724902
#Name: KZ, dtype: object

#Можно делать выборку по индексу и интересующим колонкам:

df.loc[['KZ', 'RU'], 'population']
```

```
#Результат
#Country Code
#KZ     17.04
#RU    143.50
#Name: population, dtype: float64

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

df.loc['KZ':'BY', :]
#Результат
#                 country    population    square
#Country Code
#KZ            Kazakhstan       17.04     2724902
#RU                Russia      143.50     17125191
#BY               Belarus        9.50     207600

#Сбросить индексы можно вот так:

df.reset_index()
#Результат
#  Country Code     country  population    square
#0           KZ  Kazakhstan       17.04   2724902
#1           RU      Russia      143.50  17125191
#2           BY     Belarus        9.50    207600
#3           UA     Ukraine       45.50    603628

#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:

df['density'] = df['population'] / df['square'] * 1000000
df
#Результат
#                 country  population    square    density
#Country Code
#KZ            Kazakhstan       17.04   2724902   6.253436
#RU                Russia      143.50  17125191   8.379469
#BY               Belarus        9.50    207600  45.761079
#UA               Ukraine       45.50    603628  75.377550

#Не нравится новый столбец? Не проблема, удалим его:

df.drop(['density'], axis='columns')
#Результат
#                 country  population    square
#Country Code
#KZ            Kazakhstan       17.04   2724902
#RU                Russia      143.50  17125191
#BY               Belarus        9.50    207600
#UA               Ukraine       45.50    603628
```

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

#Переименовывать столбцы нужно через метод rename:

df = df.rename(columns={'Country Code': 'country_code'})
df
#Результат
#  country_code     country  population    square
#0           KZ  Kazakhstan       17.04   2724902
#1           RU      Russia      143.50  17125191
#2           BY     Belarus        9.50    207600
#3           UA     Ukraine       45.50    603628

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

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

#Для визуального анализа данных, pandas использует библиотеку matplotlib.

#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.

#В pandas очень удобно анализировать временные ряды.

#Более подробно с представленными выше "+" можно познакомиться
#с помощью официальной документации - http://pandas.pydata.org/pandas-
docs/stable/timeseries.html#offset-aliases
```