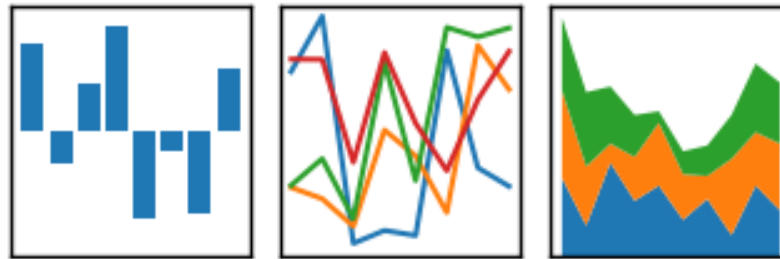


[Блог \(/ru/\)](#) [Архивы \(/ru/archives/\)](#) [Контакты \(/ru/p/contacts/\)](#) [In English \(/en/\)](#)

Введение в pandas: анализ данных на Python

📅 4 Март 2017, Python (/ru/category/python/), 26751 просмотров

pandas
 $y_{it} = \beta' x_{it} + \mu_i + \epsilon_{it}$



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 мне 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. Также надеюсь, что читателям блога заметка поможет в решении их собственных задач с помощью pandas, и послужит небольшим введением в возможности этой библиотеки.

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.

```
>>> import pandas as pd
>>> my_series = pd.Series([5, 6, 7, 8, 9, 10])
>>> my_series
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
RangeIndex(start=0, stop=6, step=1)
>>> my_series.values
array([ 5,  6,  7,  8,  9, 10], dtype=int64)
```

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

```
>>> my_series[4]
9
```

Индексы можно задавать явно:

```
>>> my_series2 = pd.Series([5, 6, 7, 8, 9, 10], index=['a', 'b', 'c', 'd', 'e', 'f'])
>>> my_series2['f']
10
```

Делать выборку по нескольким индексам и осуществлять групповое присваивание:

```
>>> my_series2[['a', 'b', 'f']]
a      5
b      6
f     10
dtype: int64
>>> my_series2[['a', 'b', 'f']] = 0
>>> my_series2
a      0
b      0
c      7
d      8
e      9
f      0
dtype: int64
```

Фильтровать Series как душе заблагорассудится, 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 = pd.Series({'a': 5, 'b': 6, 'c': 7, 'd': 8})
>>> my_series3
a    5
b    6
c    7
d    8
dtype: int64
>>> 'd' in my_series3
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
letters
a    5
b    6
c    7
d    8
Name: numbers, dtype: int64
```

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

```
>>> my_series3.index = ['A', 'B', 'C', 'D']
>>> my_series3
A      5
B      6
C      7
D      8
Name: numbers, dtype: int64
```

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:

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

	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']
0    Kazakhstan
1         Russia
2         Belarus
3         Ukraine
Name: country, dtype: object
>>> type(df['country'])
<class '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
Index([u'country', u'population', u'square'], dtype='object')
>>> df.index
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 или "на лету":


```
>>> df = pd.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
```

	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
```

	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']
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']
country      Kazakhstan
population      17.04
square      2724902
Name: KZ, dtype: object
>>> df.iloc[0]
country      Kazakhstan
population      17.04
square      2724902
Name: KZ, dtype: object
```

Можно делать выборку по индексу и интересующим колонкам:

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

	country	population	square
Country Code			
KZ	Kazakhstan	17.04	2724902
RU	Russia	143.50	17125191
BY	Belarus	9.50	207600

Фильтровать DataFrame с помощью т.н. булевых массивов:

```
>>> df[df.population > 10][['country', 'square']]
```

	country	square
Country Code		
KZ	Kazakhstan	2724902
RU	Russia	17125191
UA	Ukraine	603628

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:

```
>>> df.reset_index()
   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
   Country Code  country  population  square  density
0            KZ  Kazakhstan    17.04  2724902    6.253436
1            RU    Russia    143.50  17125191    8.379469
2            BY    Belarus     9.50   207600   45.761079
3            UA    Ukraine    45.50   603628   75.377550
```

Не нравится новый столбец? Не проблема, удалим его:

```
>>> df.drop(['density'], axis='columns')
      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']`.

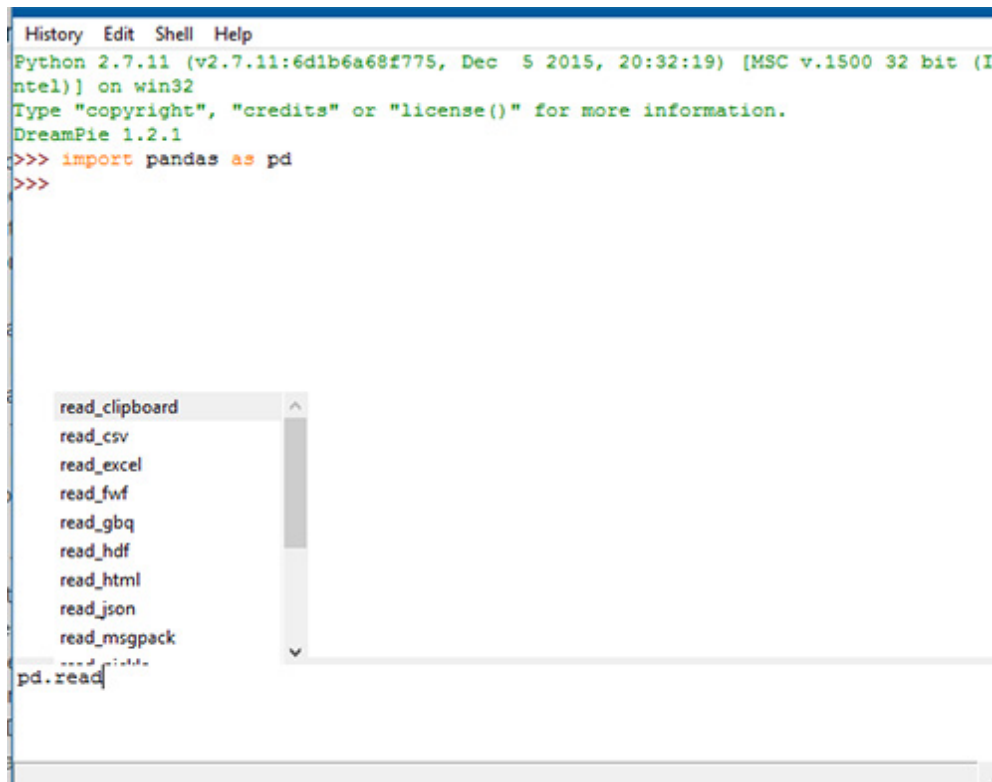
Переименовывать столбцы нужно через метод `rename`:

```
>>> df = df.rename(columns={'Country Code': 'country_code'})
>>> df
   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.

Чтение и запись данных

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



```
History Edit Shell Help
Python 2.7.11 (v2.7.11:6d1b6a68f775, Dec 5 2015, 20:32:19) [MSC v.1500 32 bit (I
ntel)] on win32
Type "copyright", "credits" or "license()" for more information.
DreamPie 1.2.1
>>> import pandas as pd
>>>

read_clipboard
read_csv
read_excel
read_fwf
read_gbq
read_hdf
read_html
read_json
read_msgpack
read_pickle
pd.read
```

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

```
>>> df.to_csv('filename.csv')
```

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`.

```
>>> df = pd.read_csv('filename.csv', sep=',')
```

Аргумент `sep` указывает разделитель 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. Скачать CSV файл можно тут (<https://yadi.sk/d/TfhJdE2k3EyALt>).

```
>>> titanic_df = pd.read_csv('titanic.csv')
>>> print(titanic_df.head())
```

	PassengerID	Name	PClass	Age	\
0	1	Allen, Miss Elisabeth Walton	1st	29.00	
1	2	Allison, Miss Helen Loraine	1st	2.00	
2	3	Allison, Mr Hudson Joshua Creighton	1st	30.00	
3	4	Allison, Mrs Hudson JC (Bessie Waldo Daniels)	1st	25.00	
4	5	Allison, Master Hudson Trevor	1st	0.92	

	Sex	Survived	SexCode
0	female	1	1
1	female	0	1
2	male	0	0
3	female	0	1
4	male	1	0

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

```
>>> print(titanic_df.groupby(['Sex', 'Survived'])['PassengerID'].count())
```

Sex	Survived	
female	0	154
	1	308
male	0	709
	1	142

Name: PassengerID, dtype: int64

А теперь проанализируем в разрезе класса кабины:


```
>>> print(titanic_df.groupby(['PClass', 'Survived'])['PassengerID'].count())
PClass  Survived
*       0         1
1st     0        129
        1        193
2nd     0        160
        1        119
3rd     0        573
        1        138
Name: PassengerID, dtype: int64
```

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:

```
>>> titanic_df = pd.read_csv('titanic.csv')
>>> pvt = titanic_df.pivot_table(index=['Sex'], columns=['PClass'], values='Name', aggfunc='count')
```

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

```
>>> print(pvt.loc['female', ['1st', '2nd', '3rd']])
PClass
1st      143.0
2nd      107.0
3rd      212.0
Name: female, dtype: float64
```

Всё очень просто.

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).

```
>>> import pandas as pd
>>> df = pd.read_csv('apple.csv', index_col='Date', parse_dates=True)
>>> df = df.sort_index()
>>> print(df.info())
<class 'pandas.core.frame.DataFrame'>
DatetimeIndex: 1258 entries, 2017-02-22 to 2012-02-23
Data columns (total 6 columns):
Open            1258 non-null float64
High            1258 non-null float64
Low             1258 non-null float64
Close           1258 non-null float64
Volume          1258 non-null int64
Adj Close       1258 non-null float64
dtypes: float64(5), int64(1)
memory usage: 68.8 KB
```

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):

```
>>> df.loc['2012-Feb', 'Close'].mean()
528.4820021999999
```

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

```
>>> df.loc['2012-Feb':'2015-Feb', 'Close'].mean()  
430.43968317018414
```

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

```
>>> df.resample('W')['Close'].mean()
```

```
Date
```

2012-02-26	519.399979
2012-03-04	538.652008
2012-03-11	536.254004
2012-03-18	576.161993
2012-03-25	600.990001
2012-04-01	609.698003
2012-04-08	626.484993
2012-04-15	623.773999
2012-04-22	591.718002
2012-04-29	590.536005
2012-05-06	579.831995
2012-05-13	568.814001
2012-05-20	543.593996
2012-05-27	563.283995
2012-06-03	572.539994
2012-06-10	570.124002
2012-06-17	573.029991
2012-06-24	583.739993
2012-07-01	574.070004
2012-07-08	601.937489
2012-07-15	606.080008
2012-07-22	607.746011
2012-07-29	587.951999
2012-08-05	607.217999
2012-08-12	621.150003
2012-08-19	635.394003
2012-08-26	663.185999
2012-09-02	670.611995
2012-09-09	675.477503
2012-09-16	673.476007

```
...
2016-08-07    105.934003
2016-08-14    108.258000
2016-08-21    109.304001
2016-08-28    107.980000
2016-09-04    106.676001
2016-09-11    106.177498
2016-09-18    111.129999
2016-09-25    113.606001
2016-10-02    113.029999
2016-10-09    113.303999
2016-10-16    116.860000
2016-10-23    117.160001
2016-10-30    115.938000
2016-11-06    111.057999
2016-11-13    109.714000
2016-11-20    108.563999
2016-11-27    111.637503
2016-12-04    110.587999
2016-12-11    111.231999
2016-12-18    115.094002
2016-12-25    116.691998
2017-01-01    116.642502
2017-01-08    116.672501
2017-01-15    119.228000
2017-01-22    119.942499
2017-01-29    121.164000
2017-02-05    125.867999
2017-02-12    131.679996
2017-02-19    134.978000
2017-02-26    136.904999
Freq: W-SUN, Name: Close, dtype: float64
```

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>).

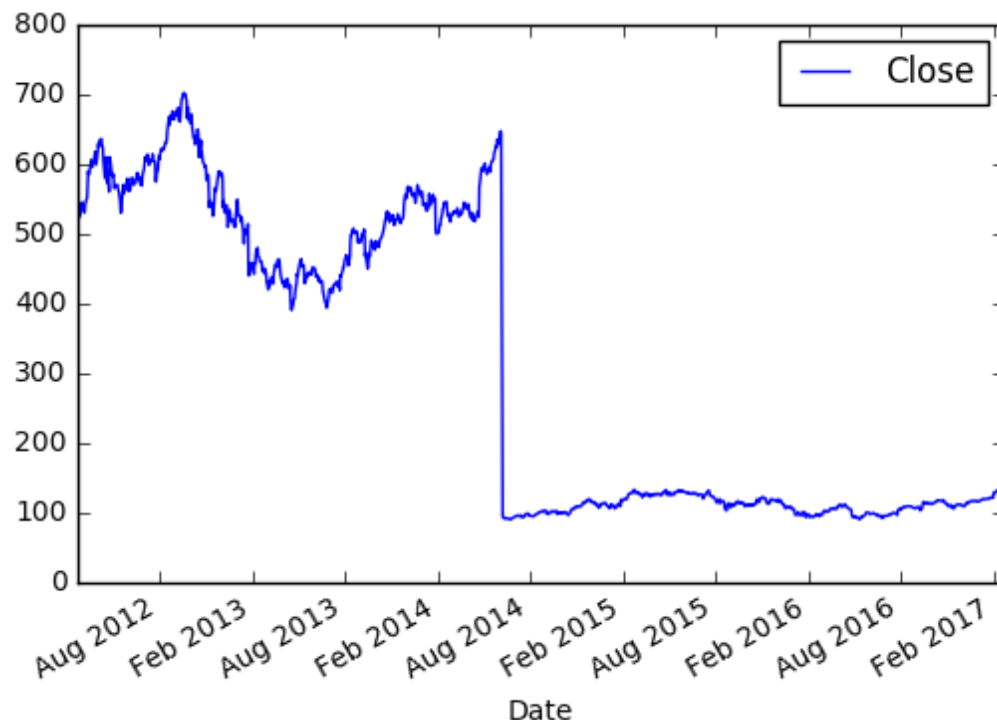
Визуализация данных в pandas

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.

```
>>> import matplotlib.pyplot as plt
>>> new_sample_df = df.loc['2012-Feb':'2017-Feb', ['Close']]
>>> new_sample_df.plot()
>>> plt.show()
```

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 ;)

Эта заметка демонстрирует лишь малую часть возможностей pandas. Со своей стороны я постараюсь по мере своих сил обновлять и дополнять её.

Полезные ссылки

- pandas cheatsheet (https://github.com/pandas-dev/pandas/blob/master/doc/cheatsheet/Pandas_Cheat_Sheet.pdf)
- Официальная документация pandas (<http://pandas.pydata.org/pandas-docs/stable/index.html>)
- Почему Python (<https://khashtamov.com/2016/06/why-python/>)
- Python Data Science Handbook (<https://github.com/jakevdp/PythonDataScienceHandbook>)



Не так давно я открыл канал в Telegram на тему разработки программного обеспечения, называется он **DevBrain**. На канале я делюсь своими мыслями , а также ресурсами (статьями, видео, презентациями, книжными рекомендациями и т.д.) о кодинге, методологиях разработки и многом другом с чем нам, разработчикам, приходится сталкиваться каждый день в работе.

На канале уже более 1000 участников, поэтому присоединяйтесь и Вы — DevBrain (<https://t.me/devbrain>)

14 Комментариев Personal Blog of Adil Khashtamov

 Войти ▾ Рекомендовать 4  Поделиться

Лучшее в начале ▾



Присоединиться к обсуждению...

ВОЙТИ С ПОМОЩЬЮ

ИЛИ ЧЕРЕЗ DISQUS (?)

Имя **Natasha Ermakova** • 18 дней назад

спасибо, что подошли с душой к изложению материала

1 ^ | ▾ • Ответить • Поделиться ›

**Adil** Модератор ➔ Natasha Ermakova • 18 дней назад

Спасибо вам за отзыв)

^ | ▾ • Ответить • Поделиться ›

**Роман** • месяц назад

Адиль, спасибо большое за подробный конспект с примерами, он мне помог сдвинуться с мертвой точки в моей задаче. Я хочу написать телеграм бота который будет забирать из таблицы с персоналом кто какими проектами занимается и вечером рабочего дня спрашивать конкретного сотрудника какое время он на каждый свой проект сегодня потратил и записывать это в новую сводную табличку по трудозатратам.

я в питоне полный ноль но по 5-10 строк кода в день двигаюсь к своей задаче, с ботом в телеграме вроде разобрался начал работать с таблицами и тут ваш конспект очень пригодился я смог достать из таблицы строку по конкретному человеку со всеми его проектами, но дальше не могу понять могу ли строке присвоить переменную что бы её передать в сообщение бота пользователю или я могу например не оборачивая ни во что строку сразу её передать в сообщение бота

Интересные записи:

- Почему Python? (</ru/why-python/>)
- Pyenv: удобный менеджер версий python (</ru/pyenv-python/>)
- Разворачиваем Django приложение в production на примере Telegram бота (</ru/how-to-deploy-django-app/>)
- Python-RQ: очередь задач на базе Redis (</ru/python-rq-howto/>)
- Django Channels: работа с WebSocket и не только (</ru/django-channels-websocket/>)
- Как написать Telegram бота: практическое руководство (</ru/create-telegram-bot-in-python/>)
- Руководство по работе с HTTP в Python. Библиотека requests (</ru/python-requests/>)
- Celery: начинаем правильно (</ru/celery-best-practices/>)
- Что нового появилось в Django Channels? (</ru/django-channels-new-features/>)
- Итоги первой встречи Python программистов в Алматы (</ru/ala-py-first/>)
- Участие в подкасте TalkPython (</ru/talkpython-podcast/>)

Работает на Django 1.11.6 & Python 3.6.1 © 2015 — 2017