Лекция 5.3. Файлы. Ввод‑вывод данных.

**Оглавление**

[**Введение** 2](#_Toc180661905)

[**1.** **Файлы: определения и функционал.** 3](#_Toc180661906)

[**Чтение из файла** 3](#_Toc180661907)

[**Запись в файл** 4](#_Toc180661908)

[**2.** **Работа с различными типами файлов** 6](#_Toc180661909)

[**Работа с файлами Microsoft Office** 6](#_Toc180661910)

[**Чтение документов MS Word** 7](#_Toc180661911)

[**Стилевое оформление** 9](#_Toc180661912)

[**Запись документов MS Word** 13](#_Toc180661913)

[**Добавление заголовков** 14](#_Toc180661914)

[**Добавление разрывов строк и страниц** 14](#_Toc180661915)

[**Добавление изображений** 15](#_Toc180661916)

[**Добавление таблицы** 16](#_Toc180661917)

[**Работа с файлами Excel в виде фреймов Pandas** 17](#_Toc180661918)

[**Как записать Pandas DataFrames в файлы Excel** 18](#_Toc180661919)

[**Как читать и записывать файлы Excel с openpyxl** 19](#_Toc180661920)

[**Запись в файл с pyexcel** 25](#_Toc180661921)

[**Чтение и запись .csv файлов** 26](#_Toc180661922)

[**Финальная проверка данных** 26](#_Toc180661923)

[**Работа с файлами в формате JSON** 27](#_Toc180661924)

[**Чтение** 27](#_Toc180661925)

[**Запись** 29](#_Toc180661926)

[**Что такое Parquet и как с ним работать в Python. Плюсы и минусы Parquet** 32](#_Toc180661927)

[**3.** **Операции с файлами и директориями в Python** 34](#_Toc180661928)

[**Копирование файла** 34](#_Toc180661929)

[**Переименование файла** 34](#_Toc180661930)

[**Удаление файла** 34](#_Toc180661931)

[**Чтение требуемой строки из текстового файла** 34](#_Toc180661932)

[**Перебор файлов в каталоге** 34](#_Toc180661933)

[**Перебор файлов в каталоге по маске** 34](#_Toc180661934)

[**Сравнение файлов** 34](#_Toc180661935)

[**Заключение** 35](#_Toc180661936)

[**Список литературы** 35](#_Toc180661937)

**Файлы и потоки. Ввод-вывод данных**

**Цель:** формирование знаний в отношении файловых операций в языке Python, их принципов работы и методов, основных библиотек для работы с файлами MS Office, JSON, Parquet.

**План лекции:**Введение.1. Файлы: определения и функционал.  
2. Работа с различными типами данных.  
3. Операции с файлами и директориями в Python.Заключение.Список литературы.

# **Введение**

Работа с файлами является одной из ключевых задач в программировании, так как позволяет сохранять, считывать и манипулировать данными, которые могут быть использованы для дальнейшей обработки или анализа. В языке Python встроенные функции и модули предоставляют мощные и гибкие инструменты для работы с файлами различных форматов. Независимо от того, являетесь ли вы начинающим разработчиком или опытным программистом, умение эффективно работать с файлами значительно расширяет ваши возможности в создании разнообразных приложений.

В этой статье мы рассмотрим основные файловые операции в Python, включая чтение, запись, обновление и удаление файлов. Также мы детально изучим работу с популярными форматами данных, такими как CSV, JSON и Parquet. Каждый из этих форматов имеет свои особенности и сферы применения, и понимание их использования позволит вам обрабатывать данные более эффективно и точно.

# **Файлы: определения и функционал.**

В данной статье мы рассмотрим встроенные средства python для работы с файлами: открытие / закрытие, чтение и запись.

Итак, начнем. Прежде, чем работать с файлом, его надо открыть. С этим замечательно справится встроенная функция open:

**f = open('text.txt', 'r')**

Разберем эту строку: ***f*** – имя переменной, оно может быть абсолютно любым; ***open*** – встроенная функция открытия файла; ***'text.txt'*** – имя файла, который мы открываем, однако если файл лежит не в одной директории с вашим *файлом.py*, то нужно указывать весь путь до файла – ***'directory\_1/text.txt'.***

У функции *open* много параметров, нам пока важны 3 аргумента: первый, это имя файла. Путь к файлу может быть относительным или абсолютным. Второй аргумент, это режим, в котором мы будем открывать файл.

|  |  |
| --- | --- |
| Режим | Обозначение |
| 'r' | открытие на чтение (является значением по умолчанию). |
| 'w' | открытие на запись, содержимое файла удаляется, если файла не существует, создается новый. |
| 'x' | открытие на запись, если файла не существует, иначе исключение. |
| 'a' | открытие на дозапись, информация добавляется в конец файла. |
| 'b' | открытие в двоичном режиме. |
| 't' | открытие в текстовом режиме (является значением по умолчанию). |
| '+' | открытие на чтение и запись |

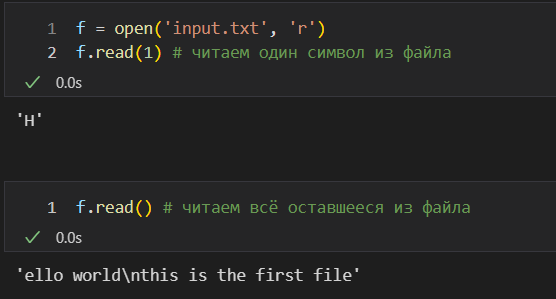
Режимы могут быть объединены, то есть, к примеру, 'rb' – чтение в двоичном режиме. По умолчанию режим равен 'rt'.

И последний аргумент, ***encoding***, нужен только в текстовом режиме чтения файла. Этот аргумент задает кодировку.

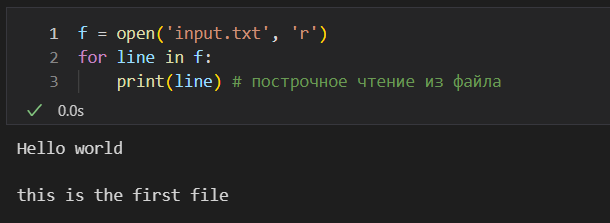
## **Чтение из файла**

Открыли файл, а теперь мы хотим прочитать из него информацию. Для этого есть несколько способов, но большого интереса заслуживают лишь два из них.

Первый – метод ***read***, читающий весь файл целиком, если был вызван без аргументов, и n символов, если был вызван с аргументом (целым числом n).

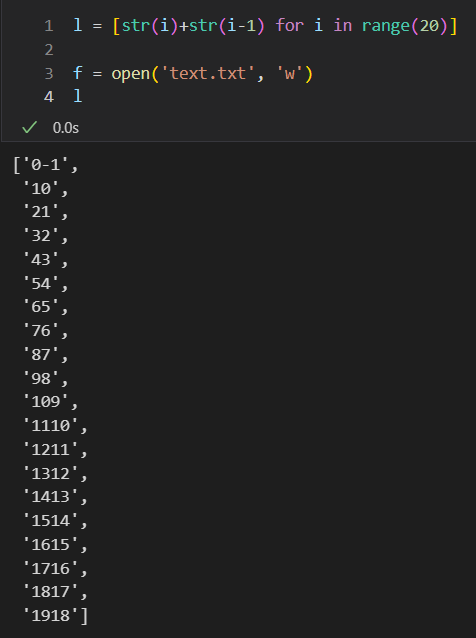


Ещё один способ сделать это – прочитать файл построчно, воспользовавшись циклом for:

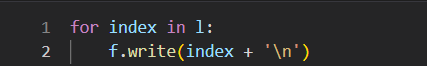


## **Запись в файл**

Откроем файл на запись. Теперь рассмотрим запись в файл. Попробуем записать в файл вот такой вот список:



Запись в файл осуществляется с помощью метода write:

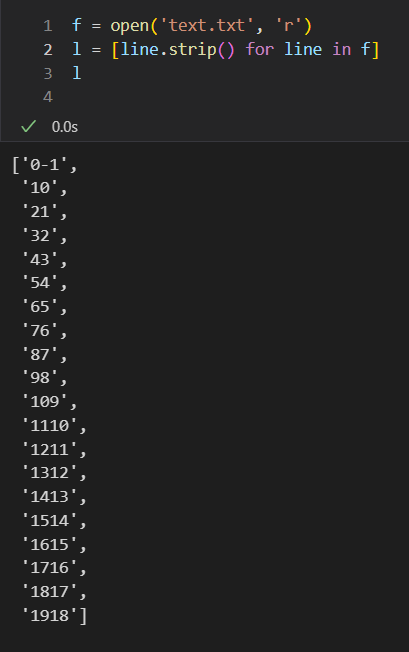


Для тех, кто не понял, что это за цифры, поясню: метод write возвращает число записанных символов.

После окончания работы с файлом его обязательно нужно закрыть с помощью метода close:



Теперь попробуем воссоздать этот список из получившегося файла. Откроем файл на чтение (надеюсь, вы поняли, как это сделать?), и прочитаем строки.



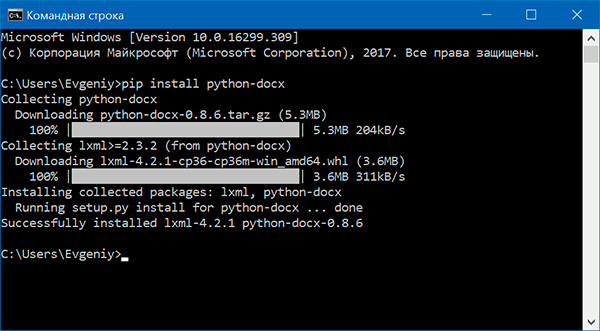
Мы получили тот же список, что и был. В более сложных случаях (словарях, вложенных кортежей и т. д.) алгоритм записи придумать сложнее. Но это и не нужно. В python уже давно придумали средства, такие как pickle или json, позволяющие сохранять в файле сложные структуры.

# **Работа с различными типами файлов**

## **Работа с файлами Microsoft Office**

С помощью модуля python-docx можно создавать и изменять документы MS Word с расширением .docx. Чтобы установить этот модуль, выполняем команду.

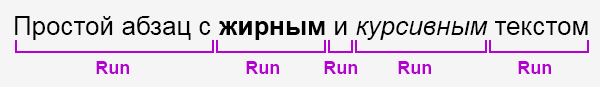
**pip install python-docx**

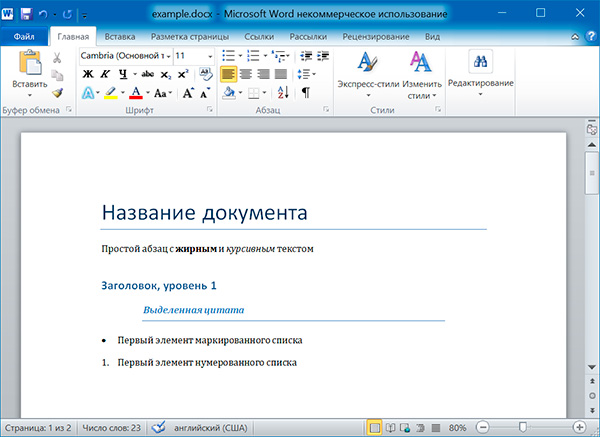


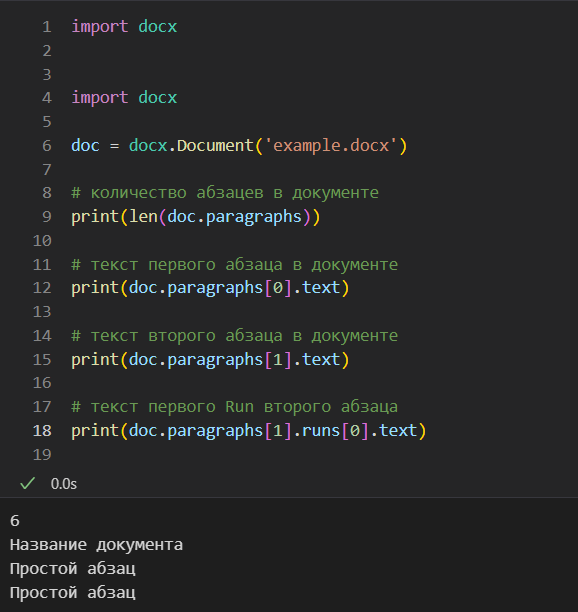
При установке модуля надо вводить python-docx, а не docx (это другой модуль). В то же время при импортировании модуля python-docx следует использовать import docx, а не import python-docx.

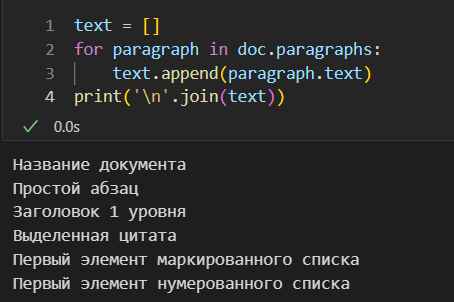
### **Чтение документов MS Word**

Файлы с расширением .docx обладают развитой внутренней структурой. В модуле python-docx эта структура представлена тремя различными типами данных. На самом верхнем уровне объект Document представляет собой весь документ. Объект Document содержит список объектов Paragraph, которые представляют собой абзацы документа. Каждый из абзацев содержит список, состоящий из одного или нескольких объектов Run, представляющих собой фрагменты текста с различными стилями форматирования.







Получаем весь текст из документа: 

### **Стилевое оформление**

В документах MS Word применяются два типа стилей: стили абзацев, которые могут применяться к объектам Paragraph, стили символов, которые могут применяться к объектам Run. Как объектам Paragraph, так и объектам Run можно назначать стили, присваивая их атрибутам style значение в виде строки. Этой строкой должно быть имя стиля. Если для стиля задано значение None, то у объекта Paragraph или Run не будет связанного с ним стиля.

**Стили абзацев**

* Normal
* Body Text
* Body Text 2
* Body Text 3
* Caption
* Heading 1
* Heading 2
* Heading 3
* Heading 4
* Heading 5
* Heading 6
* Heading 7
* Heading 8
* Heading 9
* Intense Quote
* List
* List 2
* List 3
* List Bullet
* List Bullet 2
* List Bullet 3
* List Continue
* List Continue 2
* List Continue 3
* List Number
* List Number 2
* List Number 3
* List Paragraph
* Macro Text
* No Spacing
* Quote
* Subtitle
* TOCHeading
* Title

**Стили символов**

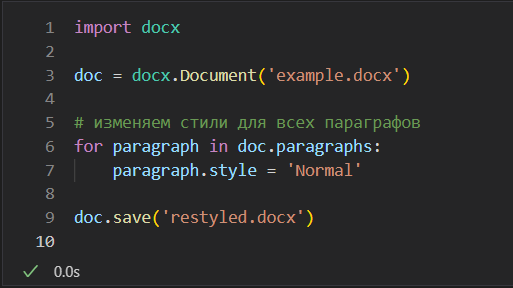
* Emphasis
* Strong
* Book Title
* Default Paragraph Font
* Intense Emphasis
* Subtle Emphasis
* Intense Reference
* Subtle Reference

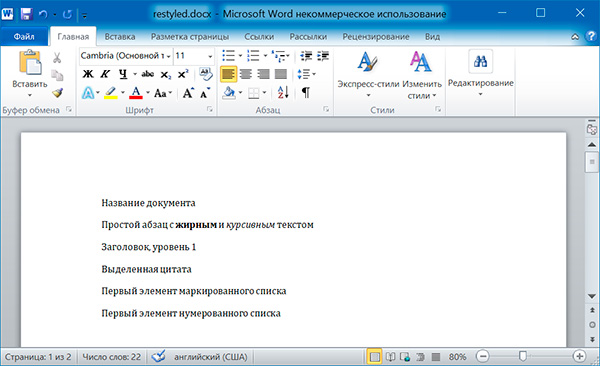
**Атрибуты объекта Run**

Отдельные фрагменты текста, представленные объектами Run, могут подвергаться дополнительному форматированию с помощью атрибутов. Для каждого из этих атрибутов может быть задано одно из трех значений: True (атрибут активизирован), False (атрибут отключен) и None (применяется стиль, установленный для данного объекта Run).

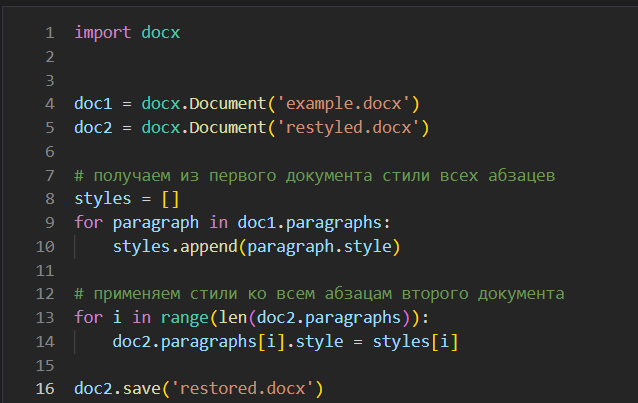
* *bold* — Полужирное начертание
* *underline* — Подчеркнутый текст
* *italic* — Курсивное начертание
* *strike* — Зачеркнутый текст

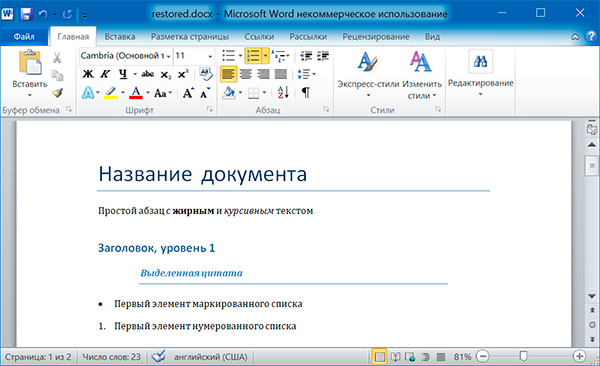
Изменим стили для всех параграфов нашего документа:



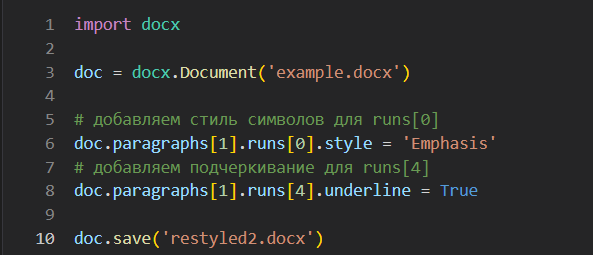


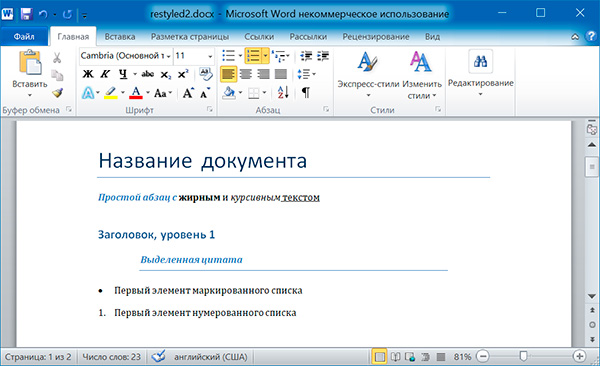
А теперь восстановим все как было:





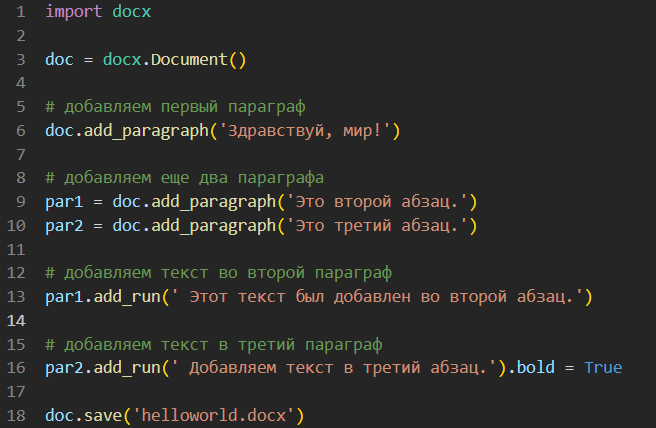
Изменим форматирование объектов Run второго абзаца:

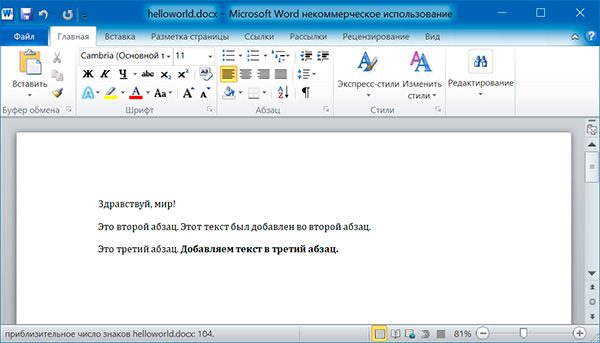




### **Запись документов MS Word**

Добавление абзацев осуществляется вызовом метода add\_paragraph() объекта Document. Для добавления текста в конец существующего абзаца, надо вызвать метод add\_run() объекта Paragraph:





Оба метода, add\_paragraph() и add\_run() принимают необязательный второй аргумент, содержащий строку стиля, например:

doc.add\_paragraph('Здравствуй, мир!', 'Title')

### **Добавление заголовков**

Вызов метода add\_heading() приводит к добавлению абзаца, отформатированного в соответствии с одним из возможных стилей заголовков:

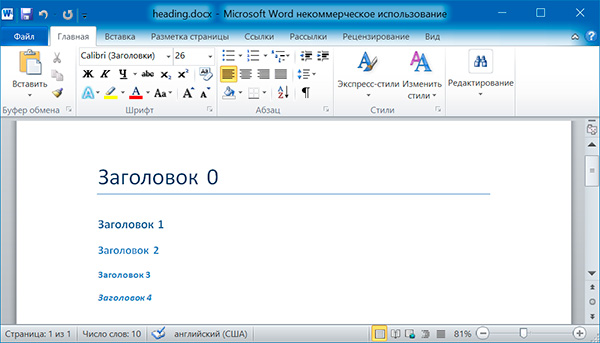
doc.add\_heading('Заголовок 0', 0)

doc.add\_heading('Заголовок 1', 1)

doc.add\_heading('Заголовок 2', 2)

doc.add\_heading('Заголовок 3', 3)

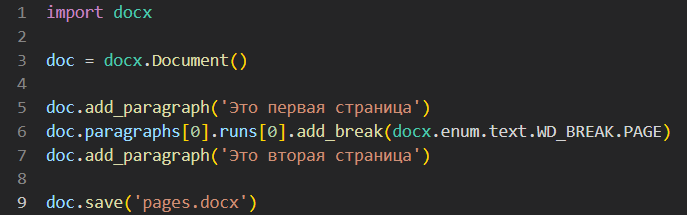
doc.add\_heading('Заголовок 4', 4)



Аргументами метода add\_heading() являются строка текста и целое число от 0 до 4. Значению 0 соответствует стиль заголовка Title.

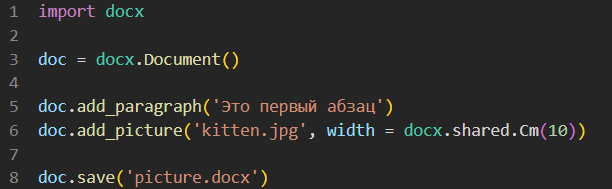
### **Добавление разрывов строк и страниц**

Чтобы добавить разрыв строки (а не добавлять новый абзац), нужно вызвать метод add\_break() объекта Run. Если же требуется добавить разрыв страницы, то методу add\_break() надо передать значение docx.enum.text.WD\_BREAK.PAGE в качестве единственного аргумента:

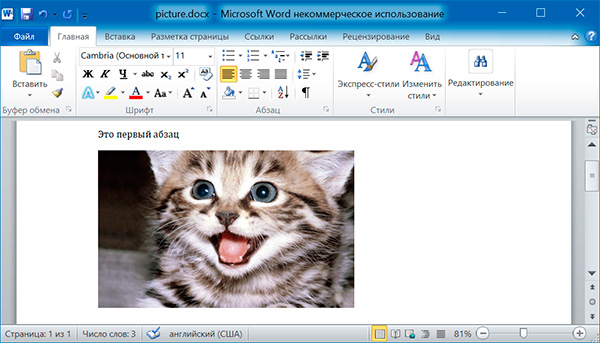


### **Добавление изображений**

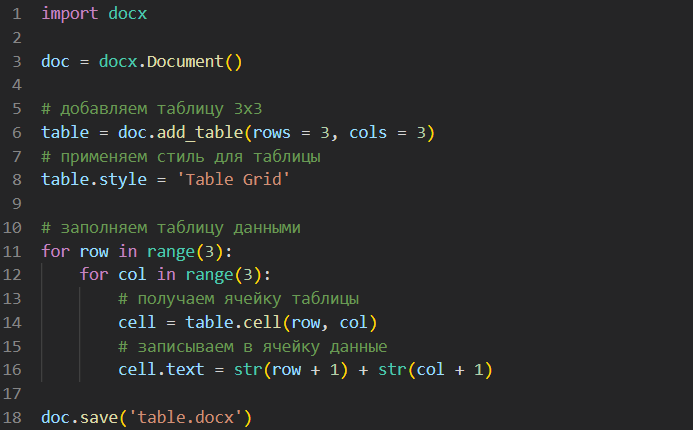
Метод add\_picture() объекта Document позволяет добавлять изображения в конце документа. Например, добавим в конец документа изображение kitten.jpg шириной 10 сантиметров:

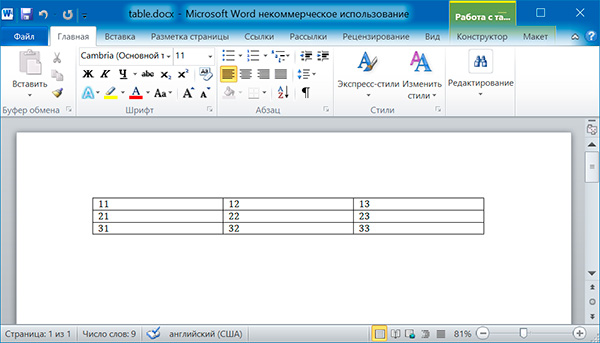


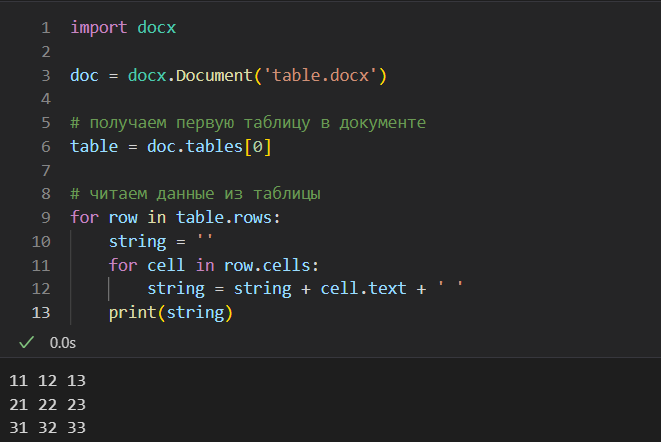
Именованные аргументы width и height задают ширину и высоту изображения. Если их опустить, то значения этих аргументов будут определяться размерами самого изображения.



### **Добавление таблицы**





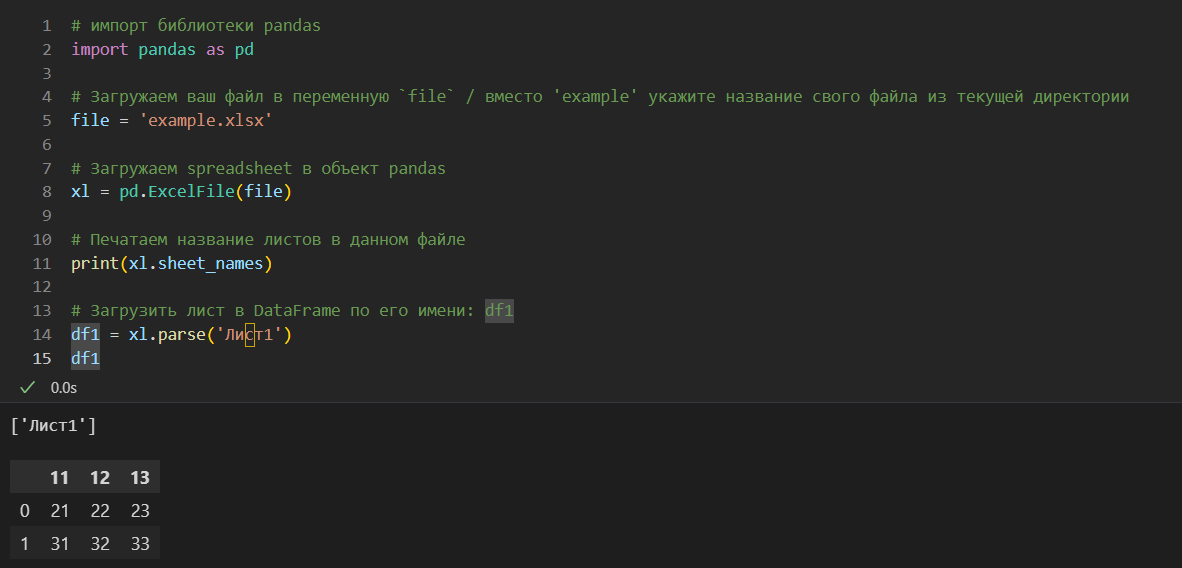


## **Работа с файлами Excel в виде фреймов Pandas**

Один из способов, который вы часто используете для импорта ваших файлов для обработки данных, — с помощью библиотеки Pandas. Она основана на NumPy и предоставляет простые в использовании структуры данных и инструменты анализа данных Python.

Эта мощная и гибкая библиотека очень часто используется дата-инженерами для передачи своих данных в структуры данных, очень выразительных для их анализа.

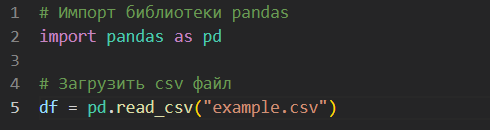
Если у вас уже есть Pandas, доступные через Anaconda, вы можете просто загрузить свои файлы в Pandas DataFrames с помощью pd.Excelfile():



Если вы не установили Anaconda, просто выполните pip install pandas, чтобы установить библиотеку Pandas в вашей среде, а затем выполните команды, которые включены в фрагмент кода выше.

Проще простого, да?

Для чтения в файлах .csv у вас есть аналогичная функция для загрузки данных в DataFrame: read\_csv(). Вот пример того, как вы можете использовать эту функцию:



Разделитель, который будет учитывать эта функция, по умолчанию является запятой, но вы можете указать альтернативный разделитель, если хотите. Перейдите к [документации](http://pandas.pydata.org/pandas-docs/stable/generated/pandas.read_csv.html?ref=codecamp.ru), чтобы узнать, какие другие аргументы вы можете указать для успешного импорта!

Обратите внимание, что есть также функции read\_table() и read\_fwf() для чтения файлов и таблиц с фиксированной шириной в формате DataFrames с общим разделителем. Для первой функции разделителем по умолчанию является вкладка, но вы можете снова переопределить это, а также указать альтернативный символ-разделитель. Более того, есть и другие функции, которые вы можете использовать для получения данных в DataFrames: вы можете найти их [здесь](http://pandas.pydata.org/pandas-docs/stable/api.html?ref=codecamp.ru#input-output).

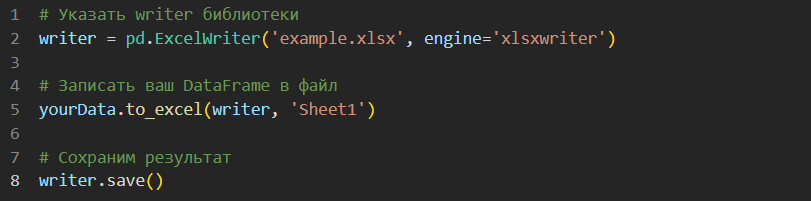
### **Как записать Pandas DataFrames в файлы Excel**

Допустим, что после анализа данных вы хотите записать данные обратно в новый файл. Есть также способ записать ваши Pandas DataFrames обратно в файлы с помощью функции to\_excel().

Но, прежде чем использовать эту функцию, убедитесь, что у вас установлен XlsxWriter, если вы хотите записать свои данные в несколько листов в файле .xlsx:

# Установим `XlsxWriter`

**pip install XlsxWriter**



Обратите внимание, что в приведенном выше фрагменте кода вы используете объект ExcelWriter для вывода DataFrame.

Иными словами, вы передаете переменную Writer в функцию to\_excel() и также указываете имя листа. Таким образом, вы добавляете лист с данными в существующую рабочую книгу: вы можете использовать ExcelWriter для сохранения нескольких (немного) разных DataFrames в одной рабочей книге.

Все это означает, что если вы просто хотите сохранить один DataFrame в файл, вы также можете обойтись без установки пакета XlsxWriter. Затем вы просто не указываете аргумент движка, который вы передаете в функцию pd.ExcelWriter(). Остальные шаги остаются прежними.

Аналогично функциям, которые вы использовали для чтения в файлах .csv, у вас также есть функция to\_csv() для записи результатов обратно в файл, разделенный запятыми. Он снова работает так же, как когда вы использовали его для чтения в файле:



Если вы хотите иметь файл, разделенный табуляцией, вы также можете передать \ t аргументу sep. Обратите внимание, что есть другие функции, которые вы можете использовать для вывода ваших файлов. Вы можете найти их все [здесь](http://pandas.pydata.org/pandas-docs/stable/api.html?ref=codecamp.ru#id12).

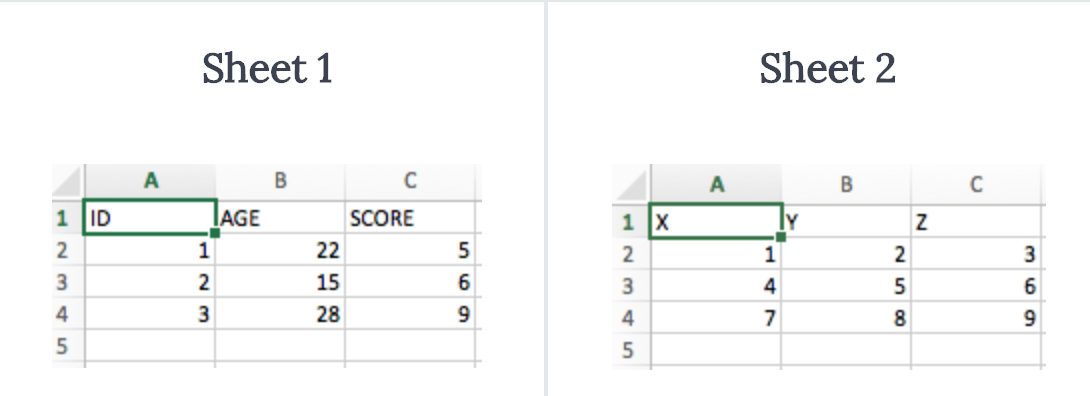
### **Как читать и записывать файлы Excel с openpyxl**

Этот пакет обычно рекомендуется, если вы хотите читать и записывать файлы .xlsx, xlsm, xltx и xltm.

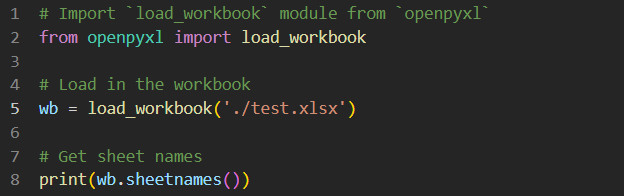
Установите openpyxl с помощью pip: вы видели, как это сделать в предыдущем разделе.

Теперь, когда вы установили openpyxl, вы можете загружать данные. Но что это за данные?

Доспутим Excel с данными, которые вы пытаетесь загрузить в Python, содержит следующие листы:

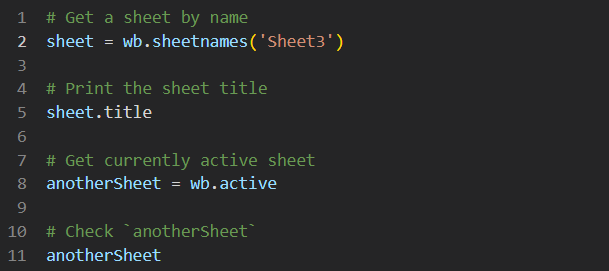
[](https://lh6.googleusercontent.com/AN4J5Y1fcovTQKNLpdF5Hj6nGGfQ4Yc2By1WJEKjHLeH_j6rs3bKMN2NiuGWdRgexukJZGfkpYqMpqcsyUA3M0k-2vmDLszPRspOP66whhxZAAzASJJD8rbtY0yTJDQzxVJi122M)

Функция load\_workbook() принимает имя файла в качестве аргумента и возвращает объект рабочей книги, который представляет файл. Вы можете проверить это, запустив type (wb). Убедитесь, что вы находитесь в том каталоге, где находится ваша таблица, иначе вы получите error при импорте.

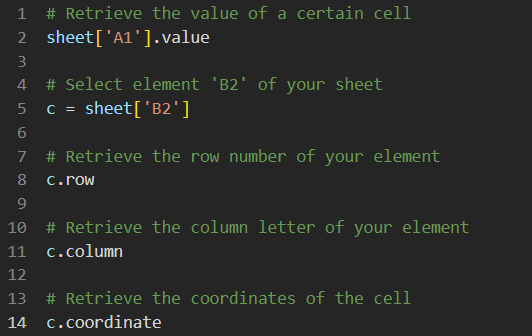
  
Помните, что вы можете изменить рабочий каталог с помощью os.chdir().

Вы видите, что фрагмент кода выше возвращает имена листов книги, загруженной в Python.Можете использовать эту информацию, чтобы также получить отдельные листы рабочей книги.

Вы также можете проверить, какой лист в настоящее время активен с wb.active. Как видно из кода ниже, вы можете использовать его для загрузки другого листа из вашей книги:

  
На первый взгляд, с этими объектами рабочего листа вы не сможете многое сделать. Однако вы можете извлечь значения из определенных ячеек на листе вашей книги, используя квадратные скобки [], в которые вы передаете точную ячейку, из которой вы хотите получить значение.

Обратите внимание, что это похоже на выбор, получение и индексирование массивов NumPy и Pandas DataFrames, но это не все, что вам нужно сделать, чтобы получить значение. Вам нужно добавить атрибут value:

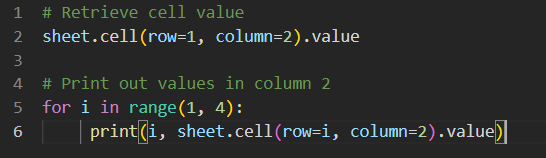
  
Как вы можете видеть, помимо значения, есть и другие атрибуты, которые вы можете использовать для проверки вашей ячейки, а именно: row, column и coordinate.

Атрибут row вернет 2;

Добавление атрибута column к c даст вам ‘B’

coordinate вернет ‘B2’.

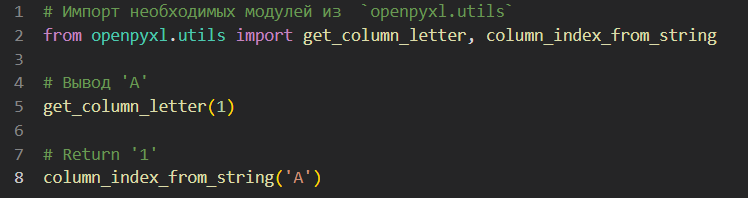
Вы также можете получить значения ячеек с помощью функции cell(). Передайте row и column, добавьте к этим аргументам значения, соответствующие значениям ячейки, которую вы хотите получить, и, конечно же, не забудьте добавить атрибут value:

  
Обратите внимание, что если вы не укажете атрибут value, вы получите <Cell Sheet3.B1>, который ничего не говорит о значении, которое содержится в этой конкретной ячейке.

Вы видите, что вы используете цикл for с помощью функции range(), чтобы помочь вам распечатать значения строк, имеющих значения в столбце 2. Если эти конкретные ячейки пусты, вы просто вернете None. Если вы хотите узнать больше о циклах for, пройдите наш курс Intermediate Python для Data Science.

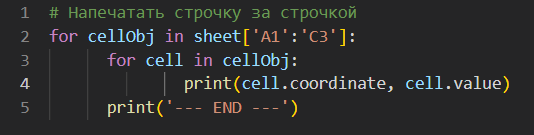
Есть специальные функции, которые вы можете вызывать для получения некоторых других значений, например, get\_column\_letter() и column\_index\_from\_string.

Две функции указывают примерно то, что вы можете получить, используя их, но лучше сделать их четче: хотя вы можете извлечь букву столбца с предшествующего, вы можете сделать обратное или получить адрес столбца, когда вы задаёте букву последнему. Вы можете увидеть, как это работает ниже:

  
Вы уже получили значения для строк, которые имеют значения в определенном столбце, но что вам нужно сделать, если вы хотите распечатать строки вашего файла, не сосредотачиваясь только на одном столбце? Использовать другой цикл, конечно!

Например, вы говорите, что хотите сфокусироваться на области между «А1» и «С3», где первая указывает на левый верхний угол, а вторая — на правый нижний угол области, на которой вы хотите сфокусироваться.

Эта область будет так называемым cellObj, который вы видите в первой строке кода ниже. Затем вы говорите, что для каждой ячейки, которая находится в этой области, вы печатаете координату и значение, которое содержится в этой ячейке. После конца каждой строки вы печатаете сообщение, которое указывает, что строка этой области cellObj напечатана.

  
Еще раз обратите внимание, что выбор области очень похож на выбор, получение и индексирование списка и элементов массива NumPy, где вы также используете [] и **:** для указания области, значения которой вы хотите получить. Кроме того, вышеприведенный цикл также хорошо использует атрибуты ячейки!

Чтобы сделать вышеприведенное объяснение и код наглядным, вы можете проверить результат, который вы получите после завершения цикла:

('A1', u'M')

('B1', u'N')

('C1', u'O')

--- END ---

('A2', 10L)

('B2', 11L)

('C2', 12L)

--- END ---

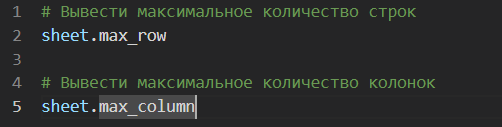
('A3', 14L)

('B3', 15L)

('C3', 16L)

--- END ---

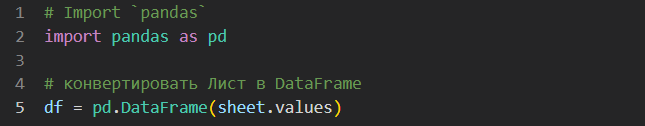
Наконец, есть некоторые атрибуты, которые вы можете использовать для проверки результата вашего импорта, а именно max\_row и max\_column. Эти атрибуты, конечно, и так  — общие способы проверки правильности загрузки данных, но они все равно полезны.

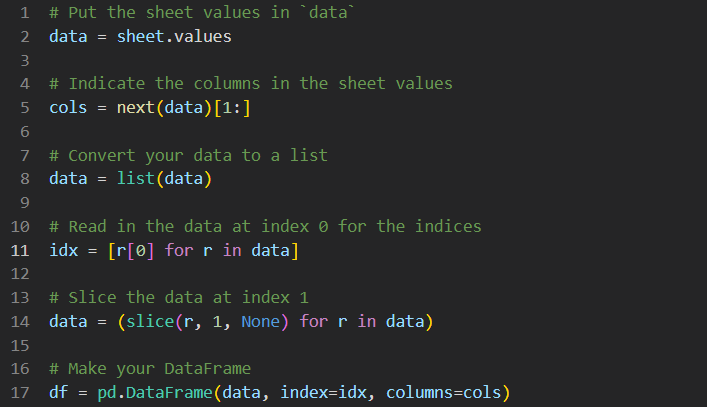


Наверное, вы думаете, что такой способ работы с этими файлами сложноват, особенно если вы еще хотите манипулировать данными.

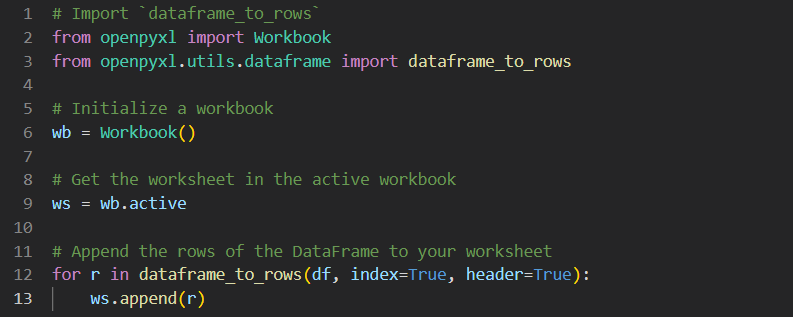
Должно быть что-то попроще, верно? Так и есть!

Openpyxl поддерживает Pandas DataFrames! Вы можете использовать функцию DataFrame() из библиотеки Pandas, чтобы поместить значения листа в DataFrame:

  
Если вы хотите указать заголовки и индексы, вам нужно добавить немного больше кода:

  
Затем вы можете начать манипулировать данными со всеми функциями, которые предлагает библиотека Pandas. Но помните, что вы находитесь в виртуальной среде, поэтому, если библиотека еще не представлена, вам нужно будет установить ее снова через pip.

Чтобы записать ваши Pandas DataFrames обратно в файл Excel, вы можете легко использовать функцию dataframe\_to\_rows() из модуля utils:

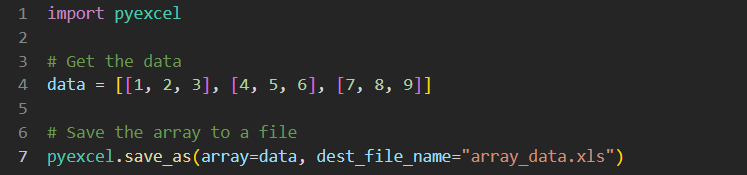
Но это точно не все! Библиотека openpyxl предлагает вам высокую гибкость при записи ваших данных обратно в файлы Excel, изменении стилей ячеек или использовании режима write-only. Эту библиотеку обязательно нужно знать, когда вы часто работаете с электронными таблицами.

Совет: читайте больше о том, как вы можете изменить стили ячеек, перейти в режим write-only или как библиотека работает с NumPy.

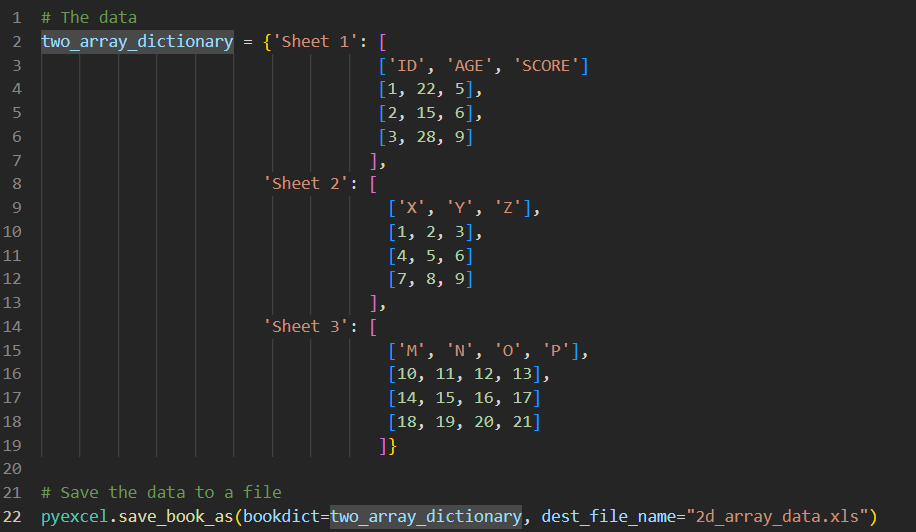
Теперь давайте также рассмотрим некоторые другие библиотеки, которые вы можете использовать для получения данных вашей электронной таблицы в Python.

### **Запись в файл с pyexcel**

С помощью этой библиотеки можно не только загружать данные в массивы, вы также можете экспортировать свои массивы обратно в таблицу. Используйте функцию save\_as() и передайте массив и имя файла назначения в аргумент dest\_file\_name:

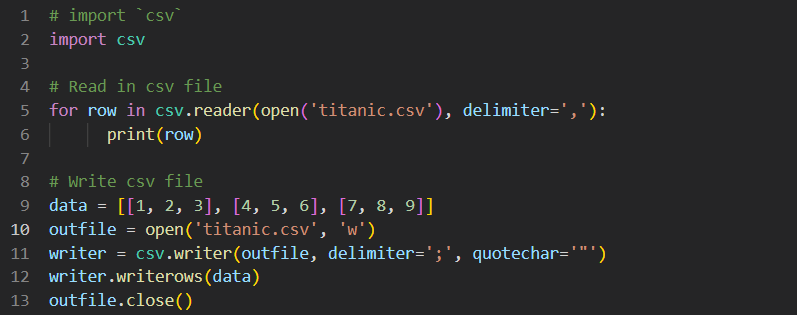
Обратите внимание, что если вы хотите указать разделитель, вы можете добавить аргумент dest\_delimiter и передать символ, который вы хотите использовать в качестве разделителя между "".

Однако если у вас есть словарь, вам нужно использовать функцию save\_book\_as(). Передайте двумерный словарь в bookdict и укажите имя файла:

  
При использовании кода, напечатанного в приведенном выше примере, важно помнить, что порядок ваших данных в словаре не будет сохранен. Если вы не хотите этого, вам нужно сделать небольшой обход.

### **Чтение и запись .csv файлов**

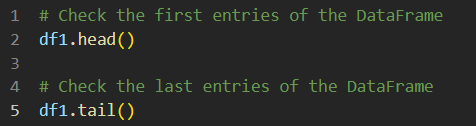
Если вы все еще ищете библиотеки, которые позволяют загружать и записывать данные в файлы .csv, кроме Pandas, лучше всего использовать пакет csv:



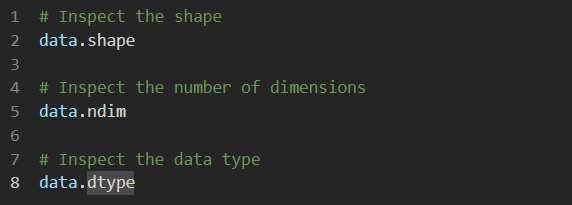
Обратите внимание, что в пакете NumPy есть функция genfromtxt(), которая позволяет загружать данные, содержащиеся в файлах .csv, в массивы, которые затем можно поместить в DataFrames.

### **Финальная проверка данных**

Когда у вас есть данные, не забудьте последний шаг: проверить, правильно ли загружены данные. Если вы поместили свои данные в DataFrame, вы можете легко и быстро проверить, был ли импорт успешным, выполнив следующие команды:



Если у вас есть данные в массиве, вы можете проверить их, используя следующие атрибуты массива: shape, ndim, dtype и т.д .:



## **Работа с файлами в формате JSON**

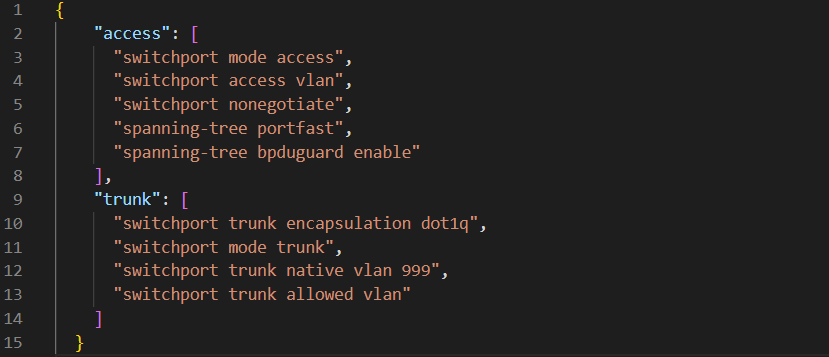
**JSON (JavaScript Object Notation)** - это текстовый формат для хранения и обмена данными.

[JSON](https://ru.wikipedia.org/wiki/JSON) по синтаксису очень похож на Python и достаточно удобен для восприятия.

Как и в случае с CSV, в Python есть модуль, который позволяет легко записывать и читать данные в формате JSON.

### **Чтение**

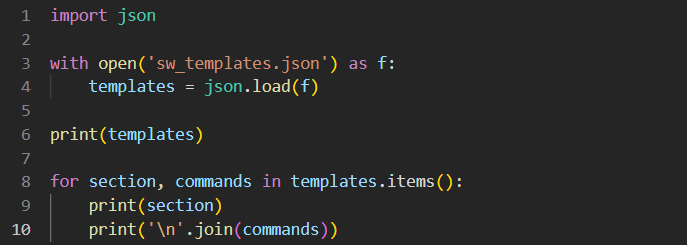
Файл sw\_templates.json:

Для чтения в модуле json есть два метода:

* json.load - метод считывает файл в формате JSON и возвращает объекты Python
* json.loads - метод считывает строку в формате JSON и возвращает объекты Python

#### **json.load**

Чтение файла в формате JSON в объект Python (файл json\_read\_load.py):

Вывод будет таким:

#### **json.loads**

Считывание строки в формате JSON в объект Python (файл json\_read\_loads.py):

Результат будет аналогичен предыдущему выводу.

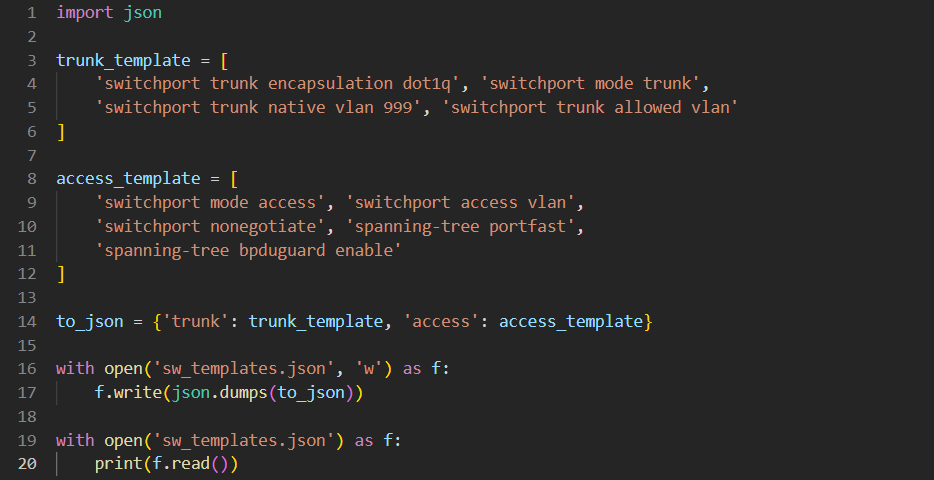
### **Запись**

Запись файла в формате JSON также осуществляется достаточно легко. Для записи информации в формате JSON в модуле json также два метода:

* json.dump - метод записывает объект Python в файл в формате JSON
* json.dumps - метод возвращает строку в формате JSON

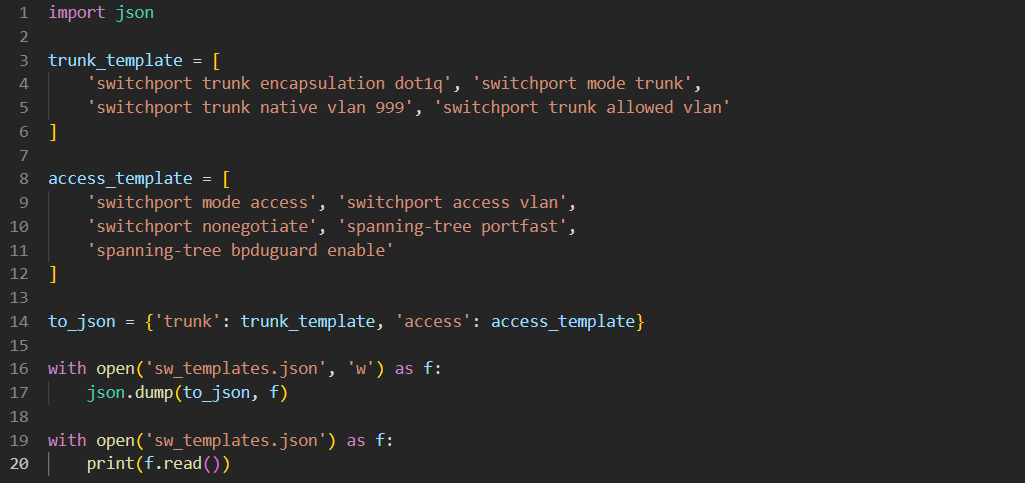
#### **json.dumps**

Преобразование объекта в строку в формате JSON (json\_write\_dumps.py):

Метод json.dumps подходит для ситуаций, когда надо вернуть строку в формате JSON. Например, чтобы передать ее API.

#### **json.dump**

Запись объекта Python в файл в формате JSON (файл json\_write\_dump.py):

Когда нужно записать информацию в формате JSON в файл, лучше использовать метод dump.

#### **Дополнительные параметры методов записи**

Методам dump и dumps можно передавать дополнительные параметры для управления форматом вывода.

По умолчанию эти методы записывают информацию в компактном представлении. Как правило, когда данные используются другими программами, визуальное представление данных не важно. Если же данные в файле нужно будет считать человеку, такой формат не очень удобно воспринимать.

К счастью, модуль json позволяет управлять подобными вещами.

Передав дополнительные параметры методу dump (или методу dumps), можно получить более удобный для чтения вывод (файл json\_write\_indent.py):



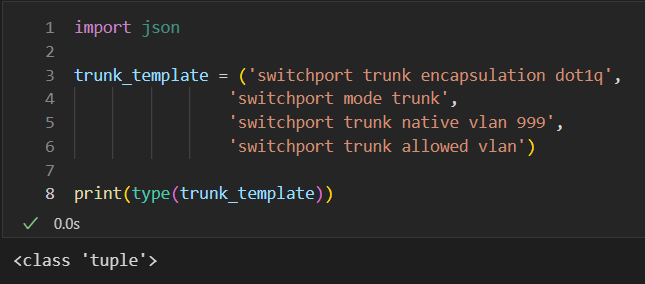
Теперь содержимое файла sw\_templates.json выглядит так:



#### **Изменение типа данных**

Еще один важный аспект преобразования данных в формат JSON: данные не всегда будут того же типа, что исходные данные в Python.

Например, кортежи при записи в JSON превращаются в списки:



Так происходит из-за того, что в JSON используются другие типы данных и не для всех типов данных Python есть соответствия.

Таблица конвертации данных Python в JSON:

|  |  |
| --- | --- |
| Python | JSON |
| dict | object |
| list, tuple | array |
| str | string |
| int, float | number |
| True | true |
| False | false |
| None | null |

Таблица конвертации JSON в данные Python:

|  |  |
| --- | --- |
| JSON | Python |
| object | dict |
| array | list |
| string | str |
| number (int) | int |
| number (real) | float |
| true | True |
| false | False |
| null | None |

## **Что такое Parquet и как с ним работать в Python. Плюсы и минусы Parquet**

Parquet — это формат хранения данных, разработанный для эффективного анализа больших объемов данных. Он обеспечивает высокую производительность чтения и записи, а также поддерживает сжатие данных, что позволяет сэкономить место на диске. В Python существует несколько библиотек для работы с форматом Parquet, наиболее популярной из них является pyarrow.

Для работы с Parquet в Python вам потребуется установить библиотеку pyarrow. Вы можете установить ее с помощью pip командой:

**pip install pyarrow**

После установки вы сможете импортировать необходимые модули и начать работу с данными в формате Parquet.

Вот пример базовой работы с Parquet в Python, используя pyarrow:

import pyarrow.parquet as pq

# Чтение данных из файла Parquet

table = pq.read\_table('example.parquet')

df = table.to\_pandas() # Преобразование таблицы Parquet в объект pandas DataFrame

# Запись данных в файл Parquet

table = pq.Table.from\_pandas(df)

pq.write\_table(table, 'example.parquet')

**Плюсы формата данных Parquet:**

* Эффективное сжатие данных: Parquet поддерживает различные алгоритмы сжатия, что позволяет снизить потребление дискового пространства.
* Высокая производительность чтения и записи: благодаря структуре данных и эффективной компрессии, Parquet обеспечивает быстрое чтение и запись данных.
* Поддержка разных типов данных: формат Parquet поддерживает различные типы данных, включая числовые, строковые, даты/время и многие другие.
* Поддержка схемы данных: Parquet хранит информацию о схеме данных в самом файле, что позволяет легко выполнять операции с данными без необходимости загрузки всего файла.

**Минусы формата данных Parquet:**

* Неудобство для небольших наборов данных: Parquet наиболее эффективен при работе с большими объемами данных. Для небольших наборов данных его использование может быть избыточным.
* Значительные затраты на запись: запись данных в формате Parquet может занимать больше времени и ресурсов по сравнению с другими форматами данных.

В целом, Parquet является мощным форматом данных для хранения и анализа больших объемов данных в Python. Он может быть особенно полезен в случаях, когда требуется высокая производительность и эффективное использование дискового пространства.

# **Операции с файлами и директориями в Python**

## **Копирование файла**

import shutil

shutil.copyfile("C:\\mydoc.doc", "C:\\My Documents\\mydoc\_2.doc")

## **Переименование файла**

import os

os.rename("/home/user/testfile.txt", "/home/user/test.txt")

## **Удаление файла**

import os

os.remove("/home/user/testfile.txt")

## **Чтение требуемой строки из текстового файла**

Чтобы прочесть строку под определенным номером — можно воспользоваться как стандартным чтением файла в лист, так и использовать модуль linecache:

line = linecache.getline("C:\\boot.ini", 2)

# or

line = open("C:\\boot.ini").readlines()[1]

## **Перебор файлов в каталоге**

for filename in os.listdir("../plugins"):

print(filename)

## **Перебор файлов в каталоге по маске**

import glob

for filename in glob.glob("../plugins/\*.zip"):

print(filename)

## **Сравнение файлов**

Сравнивать файлы можно как по содержимому, так и по их свойствам, что значительно быстрее.

Оба варианта возможны при помощи filecmp

import filecmp

similar = filecmp.cmp('C:\\file1.txt', 'C:\\file2.txt')

print(similar)

# **Заключение**

Работа с файлами в Python предоставляет разработчикам мощные инструменты для управления данными. В ходе этой статьи мы подробно рассмотрели основные файловые операции, которые являются фундаментальными для любой программы, работающей с файлами. Мы также изучили особенности работы с файлами в форматах CSV, JSON и Parquet, каждый из которых имеет свои преимущества и применяется в различных сценариях.

Файлы CSV, благодаря своей простоте и универсальности, широко используются для хранения табличных данных и передачи информации между различными системами. JSON, с его гибкой структурой и возможностью представления сложных данных, является незаменимым инструментом для веб-разработки и взаимодействия с API. Формат Parquet, оптимизированный для аналитических задач и больших объемов данных, становится все более популярным в областях, связанных с большими данными и машинным обучением.

Понимание и умение работать с этими форматами данных позволяет создавать более эффективные и производительные приложения, а также обеспечивает лучшую интеграцию с различными системами и платформами. Надеемся, что эта статья поможет вам улучшить ваши навыки работы с файлами в Python и расширит ваш арсенал инструментов для решения разнообразных задач в программировании.

# **Список литературы**

1. https://tokmakov.msk.ru/blog/item/78

2. https://blog.aspose.com/ru/words/create-word-documents-using-python/

3. https://habr.com/ru/articles/456534/

4. https://infozone.pro/python-parquet-example-advantage-and-disadvantage-tools/

5. https://habr.com/ru/companies/wrike/articles/279797/

6. https://pyneng.readthedocs.io/ru/latest/book/17\_serialization/json.html

7. https://foxford.ru/wiki/informatika/operatsii-s-faylami-i-direktoriyami-v-python?utm\_referrer=https%3A%2F%2Fwww.google.com%2F

8. https://ru.hexlet.io/courses/python-pandas/lessons/excel-workflow/theory\_unit

9. https://habr.com/ru/articles/775548/

10. https://www.codecamp.ru/blog/python-excel-tutorial/