Лекция 5.3. Файлы. Ввод‑вывод данных. Файловые операции

# **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')

У функции open много параметров, они указаны в статье "Встроенные функции", 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 мы 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).

>>>

>>> f = open('text.txt')

>>> f.read(1)

'H'

>>> f.read()

'ello world!\nThe end.\n\n'

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

>>>

>>> f = open('text.txt')

>>> for line in f:

... line

...

'Hello world!\n'

'\n'

'The end.\n'

'\n'

Запись в файл

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

>>>

>>> l = [str(i)+str(i-1) for i in range(20)]

>>> l

['0-1', '10', '21', '32', '43', '54', '65', '76', '87', '98', '109', '1110', '1211', '1312', '1413', '1514', '1615', '1716', '1817', '1918']

Откроем файл на запись:

>>>

>>> f = open('text.txt', 'w')

Запись в файл осуществляется с помощью метода write:

>>>

>>> for index in l:

... f.write(index + '\n')

...

4

3

3

3

3

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:

>>>

>>> f.close()

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

>>>

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

>>> l = [line.strip() for line in f]

>>> l

['0-1', '10', '21', '32', '43', '54', '65', '76', '87', '98', '109', '1110', '1211', '1312', '1413', '1514', '1615', '1716', '1817', '1918']

>>> f.close()

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.

# **Ввод-вывод данных. Работа с различными типами файлов**

Мы уже встречались с функцией print(). Она отвечает за вывод данных, по-умолчанию на экран. Если код содержится в файле, то без нее не обойтись. В интерактивном режиме в ряде случаев можно обойтись без нее.

Ввод данных в программу и их вывод важны в программировании. Без ввода программы делали бы одно и то же, исключая случаи, когда в них самих генерируются случайные значения. Вывод позволяет увидеть, использовать, куда-нибудь передать результат работы программы.

Обычно требуется, чтобы программа обрабатывала какой-то диапазон различных входных данных, которые поступают в нее из внешних источников. В качестве последних могут выступать файлы, клавиатура, сеть, выходные данные из другой программы. В свою очередь вывод данных, например, возможен в файл, по сети, в базу данных, на принтер. Однако нередко информацию просто выводят на экран монитора.

Можно сказать, что программа – это открытая система, которая обменивается чем-либо с внешней для нее средой. Если живой организм в основном обменивается веществом и энергией, то программа – данными, информацией.

Вывод данных. Функция print()

Что такое функция в программировании, узнаем позже. Пока будем считать, что print() – это такая команда языка Python, которая выводит то, что в ее скобках на экран.

>>> print(1032)

1032

>>> print(2.34)

2.34

>>> print("Hello")

Hello

В скобках могут быть любые типы данных. Кроме того, количество данных может быть различным:

>>> print("a:", 1)

a: 1

>>> one = 1

>>> two = 2

>>> three = 3

>>> print(one, two, three)

1 2 3

Можно передавать в функцию print() как непосредственно литералы (в данном случае "a:" и 1), так и переменные, вместо которых будут выведены их значения. Аргументы функции (то, что в скобках), разделяются между собой запятыми. В выводе вместо запятых значения разделены пробелом.

Если в скобках стоит выражение, то сначала оно выполняется, после чего print() уже выводит результат данного выражения:

>>> print("hello" + " " + "world")

hello world

>>> print(10 - 2.5/2)

8.75

В print() предусмотрены дополнительные параметры. Например, через параметр sep можно указать отличный от пробела разделитель строк:

>>> print("Mon", "Tue", "Wed", "Thu", "Fri", "Sat", "Sun", sep="-")

Mon-Tue-Wed-Thu-Fri-Sat-Sun

>>> print(1, 2, 3, sep="//")

1//2//3

Параметр end позволяет указывать, что делать, после вывода строки. По-умолчанию происходит переход на новую строку. Однако это действие можно отменить, указав любой другой символ или строку:

>>> print(10, end="")

10>>>

Обычно end используется не в интерактивном режиме, а в скриптах, когда несколько выводов подряд надо разделить не переходом на новую строку, а, скажем, запятыми. Сам переход на новую строку обозначается символом '\n'. Если присвоить это значение параметру end, то никаких изменений в работе функции print вы не увидите, так как это значение и так присвоено по-умолчанию:

>>> print(10, end='\n')

10

>>>

Однако, если надо отступить на одну дополнительную строку после вывода, то можно сделать так:

>>> print(10, end='\n\n')

10

>>>

В функцию print нередко передаются так называемые форматированные строки, хотя по смыслу их правильнее называть строки-шаблоны. Никакого отношения к самому print они не имеют. Когда такая строка находится в скобках print(), интерпретатор сначала согласно заданному в ней формату преобразует ее к обычной строке, после чего передает результат в print().

Форматирование может выполняться в так называемом старом стиле или с помощью строкового метода format. Старый стиль также называют Си-стилем, так как он схож с тем, как происходит вывод на экран в языке C. Рассмотрим пример:

>>> pupil = "Ben"

>>> old = 16

>>> grade = 9.2

>>> print("It's %s, %d. Level: %f" % (pupil, old, grade))

It's Ben, 16. Level: 9.200000

Здесь вместо трех комбинаций символов %s, %d, %f подставляются значения переменных pupil, old, grade. Буквы s, d, f обозначают типы данных – строку, целое число, вещественное число. Если бы требовалось подставить три строки, то во всех случаях использовалось бы сочетание %s.

Хотя в качестве значения переменной grade было указано число 9.2, на экран оно вывелось с дополнительными нулями. Чтобы указать, сколько требуется знаков после запятой, надо перед f поставить точку, после нее указать желаемое количество знаков в дробной части:

>>> print("It's %s, %d. Level: %.1f" % (pupil, old, grade))

It's Ben, 16. Level: 9.2

Теперь посмотрим на метод format():

>>> print("This is a {0}. It's {1}.".format("ball", "red"))

This is a ball. It's red.

>>>

>>> print("This is a {1}. It's {0}.".format("white", "cat"))

This is a cat. It's white.

>>>

>>> print("This is a {2}. It's {0} {1}.".format("a", "number", 1))

This is a 1. It's a number.

В строке в фигурных скобках указаны номера данных, которые будут сюда подставлены. Далее к строке применяется метод format(). В его скобках указываются сами данные (можно использовать переменные). На нулевое место подставится первый аргумент метода format(), на место с номером 1 – второй и т. д.

На самом деле возможности метода format существенно шире, и для их изучения понадобился бы отдельный урок. Нам пока будет достаточно этого.

В новых релизах Питона появился третий способ создания форматированных строк – f-строки. Перед их открывающей кавычкой прописывается буква f. В самой строке внутри фигурных скобок записываются выражения на Python, которые исполняются, когда интерпретатор преобразует строку-шаблон в обычную.

>>> a = 10

>>> b = 1.33

>>> c = 'Box'

>>> print(f'qty - {a:5}, goods - {c}')

qty - 10, goods - Box

>>> print(f'price - {b + 0.2:.1f}')

price - 1.5

В примере число 5 после переменной a обозначает количество знакомест, отводимых под вывод значения переменной. В выражении b + 0.2:.1f сначала выполняется сложение, после этого значение округляется до одного знака после запятой.

Ввод данных. Функция input()

За ввод в программу данных с клавиатуры в Python отвечает функция input. Когда вызывается эта функция, программа останавливает свое выполнение и ждет, когда пользователь введет текст. После этого, когда он нажмет Enter, функция input() заберет введенный текст и передаст его программе, которая уже будет обрабатывать его согласно своим алгоритмам.

Если в интерактивном режиме ввести команду input(), то ничего интересного вы не увидите. Компьютер будет ждать, когда вы что-нибудь введете и нажмете Enter или просто нажмете Enter. Если вы что-то ввели, это сразу же отобразиться на экране:

>>> input()

Yes!

'Yes!'

Функция input() передает введенные данные в программу. Их можно присвоить переменной. В этом случае интерпретатор не выводит строку сразу же:

>>> answer = input()

No, it is not.

В данном случае строка сохраняется в переменной answer, и при желании мы можем вывести ее значение на экран:

>>> answer

'No, it is not.'

При использовании функции print() кавычки в выводе опускаются:

>>> print(answer)

No, it is not.

Куда интересней использовать функцию input() в скриптах – файлах с кодом. Рассмотрим такую программу:

name\_user = input()

city\_user = input()

print(f'Вас зовут {name\_user}. Ваш город {city\_user}')

При запуске программы, компьютер ждет, когда будет введена сначала одна строка, потом вторая. Они будут присвоены переменным name\_user и city\_user. После этого значения этих переменных выводятся на экран с помощью форматированного вывода. Пример выполнения скрипта:

Арнольд

Питонск

Вас зовут Арнольд. Ваш город Питонск

Эта программа далека от совершенства. Откуда пользователю знать, что от него хотят? Чтобы не вводить человека в замешательство, для функции input предусмотрен специальный параметр-приглашение. Это приглашение выводится на экран при вызове input(). Усовершенствованная программа может выглядеть так (сразу под ней пример ее выполнения):

name\_user = input('Ваше имя: ')

city\_user = input('Ваш город: ')

print(f'Вас зовут {name\_user}. Ваш город {city\_user}')

Ваше имя: Серый

Ваш город: Белый

Вас зовут Серый. Ваш город Белый

Обратите внимание, что в программу поступает строка. Даже если ввести число, функция input() все равно вернет его строковое представление. Но что делать, если надо получить число? Ответ: использовать функции преобразования типов.

qty = input("Сколько апельсинов? ")

price = input("Цена одного? ")

qty = int(qty)

price = float(price)

summa = qty \* price

print("Заплатите", summa, "руб.")

Сколько апельсинов? 5

Цена одного? 35.80

Заплатите 179.0 руб.

В данном случае с помощью функций int() и float() строковые значения переменных qty и price преобразуются соответственно в целое число и вещественное число. После этого новые численные значения присваиваются тем же переменным.

Программный код можно сократить, если преобразование типов выполнить в тех же строках кода, где вызывается функция input():

qty = int(input("Сколько апельсинов? "))

price = float(input("Цена одного апельсина? "))

summa = qty \* price

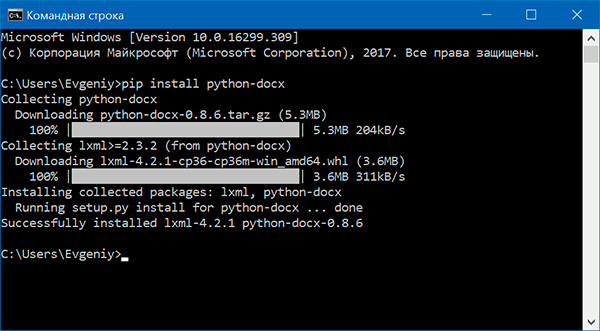
print("Заплатите", summa, "руб.")

Сначала выполняется функция input(). Она возвращает строку, которую функция int() или float() сразу преобразует в число. Только после этого происходит присваивание переменной, то есть она сразу получает численное значение.

## 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.

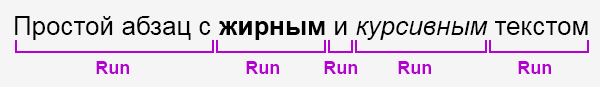
> pip install python-docx

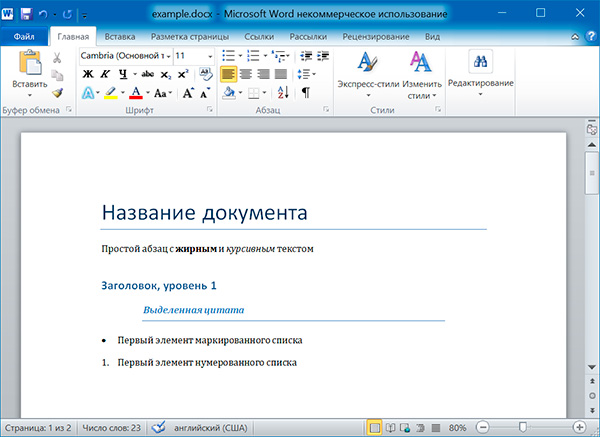


При установке модуля надо вводить python-docx, а не docx (это другой модуль). В то же время при импортировании модуля python-docx следует использовать import docx, а не import python-docx.

## 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.





import docx

doc = docx.Document('example.docx')

# количество абзацев в документе

print(len(doc.paragraphs))

# текст первого абзаца в документе

print(doc.paragraphs[0].text)

# текст второго абзаца в документе

print(doc.paragraphs[1].text)

# текст первого Run второго абзаца

print(doc.paragraphs[1].runs[0].text)Копировать

6

Название документа

Простой абзац с жирным и курсивным текстом

Простой абзац с Копировать

Получаем весь текст из документа:

text = []

for paragraph in doc.paragraphs:

text.append(paragraph.text)

print('\n'.join(text))Копировать

Название документа

Простой абзац с жирным и курсивным текстом

Заголовок, уровень 1

Выделенная цитата

Первый элемент маркированного списка

Первый элемент нумерованного спискаКопировать

### 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

paragraph.style = 'Quote'

run.style = 'Book Title'Копировать

### 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:

import docx

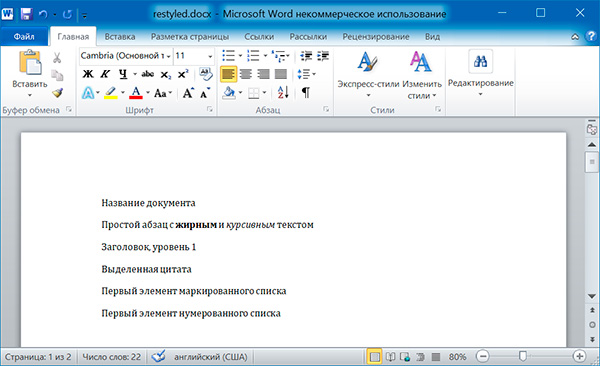
doc = docx.Document('example.docx')

# изменяем стили для всех параграфов

for paragraph in doc.paragraphs:

paragraph.style = 'Normal'

doc.save('restyled.docx')Копировать



А теперь восстановим все как было:

import docx

os.chdir('C:\\example')

doc1 = docx.Document('example.docx')

doc2 = docx.Document('restyled.docx')

# получаем из первого документа стили всех абзацев

styles = []

for paragraph in doc1.paragraphs:

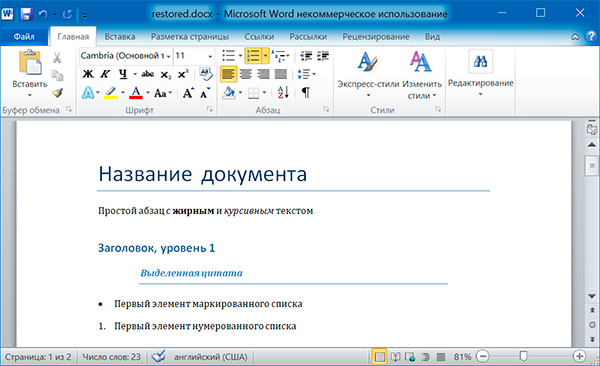
styles.append(paragraph.style)

# применяем стили ко всем абзацам второго документа

for i in range(len(doc2.paragraphs)):

doc2.paragraphs[i].style = styles[i]

doc2.save('restored.docx')Копировать



Изменим форматирвание объектов Run второго абзаца:

import docx

doc = docx.Document('example.docx')

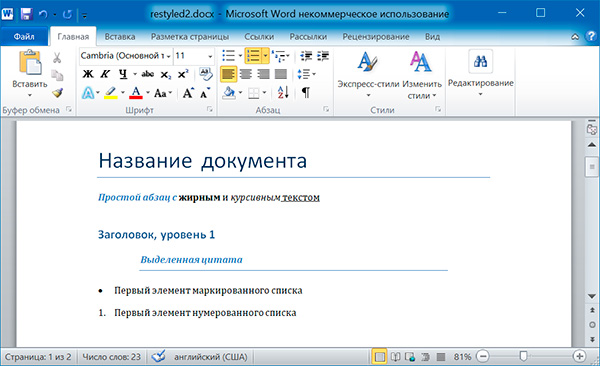
# добавляем стиль символов для runs[0]

doc.paragraphs[1].runs[0].style = 'Intense Emphasis'

# добавляем подчеркивание для runs[4]

doc.paragraphs[1].runs[4].underline = True

doc.save('restyled2.docx')Копировать



## Запись докуменов 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:

import docx

doc = docx.Document()

# добавляем первый параграф

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

# добавляем еще два параграфа

par1 = doc.add\_paragraph('Это второй абзац.')

par2 = doc.add\_paragraph('Это третий абзац.')

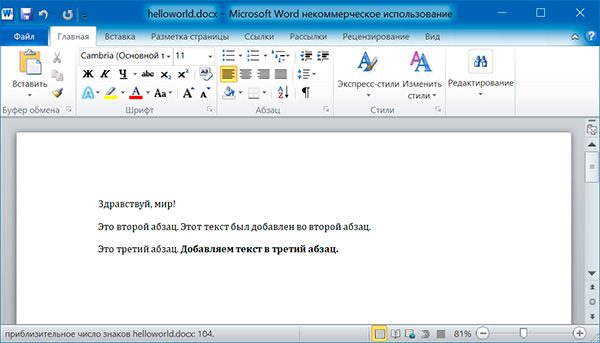
# добавляем текст во второй параграф

par1.add\_run(' Этот текст был добавлен во второй абзац.')

# добавляем текст в третий параграф

par2.add\_run(' Добавляем текст в третий абзац.').bold = True

doc.save('helloworld.docx')Копировать



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

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

### Добавление заголовков

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

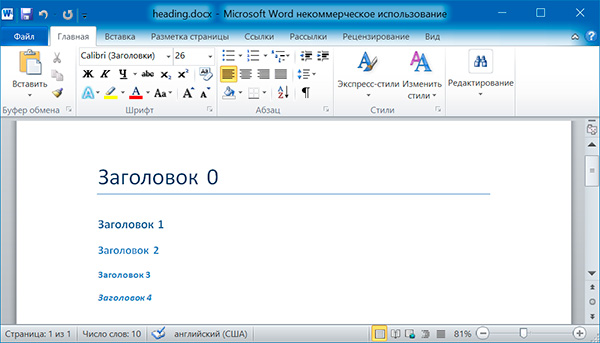
doc.add\_heading('Заголовок 0', 0)

doc.add\_heading('Заголовок 1', 1)

doc.add\_heading('Заголовок 2', 2)

doc.add\_heading('Заголовок 3', 3)

doc.add\_heading('Заголовок 4', 4)Копировать



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:

import docx

doc = docx.Document()

doc.add\_paragraph('Это первая страница')

doc.paragraphs[0].runs[0].add\_break(docx.enum.text.WD\_BREAK.PAGE)

doc.add\_paragraph('Это вторая страница')

doc.save('pages.docx')Копировать

### Добавление изображений

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

import docx

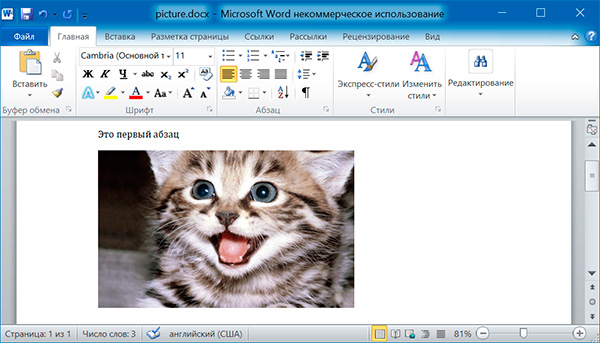
doc = docx.Document()

doc.add\_paragraph('Это первый абзац')

doc.add\_picture('kitten.jpg', width = docx.shared.Cm(10))

doc.save('picture.docx')Копировать

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



### Добавление таблицы

import docx

doc = docx.Document()

# добавляем таблицу 3x3

table = doc.add\_table(rows = 3, cols = 3)

# применяем стиль для таблицы

table.style = 'Table Grid'

# заполняем таблицу данными

for row in range(3):

for col in range(3):

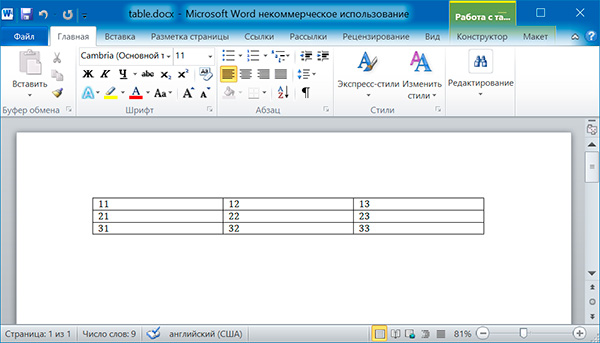
# получаем ячейку таблицы

cell = table.cell(row, col)

# записываем в ячейку данные

cell.text = str(row + 1) + str(col + 1)

doc.save('table.docx')Копировать



import docx

doc = docx.Document('table.docx')

# получаем первую таблицу в документе

table = doc.tables[0]

# читаем данные из таблицы

for row in table.rows:

string = ''

for cell in row.cells:

string = string + cell.text + ' '

print(string)Копировать

11 12 13

21 22 23

31 32 33

## Загрузить файлы Excel в виде фреймов Pandas

Все, среда настроена, вы готовы начать импорт ваших файлов.

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():

# импорт библиотеки pandas

import pandas as pd

# Загружаем ваш файл в переменную `file` / вместо 'example' укажите название свого файла из текущей директории

file = 'example.xlsx'

# Загружаем spreadsheet в объект pandas

xl = pd.ExcelFile(file)

# Печатаем название листов в данном файле

print(xl.sheet\_names)

# Загрузить лист в DataFrame по его имени: df1

df1 = xl.parse('Sheet1')

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:

# Импорт библиотеки pandas

import pandas as pd

# Загрузить csv файл

df = pd.read\_csv("example.csv")

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

# Указать writer библиотеки

writer = pd.ExcelWriter('example.xlsx', engine='xlsxwriter')

# Записать ваш DataFrame в файл

yourData.to\_excel(writer, 'Sheet1')

# Сохраним результат

writer.save()

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:

# Запишите DataFrame в csv

df.to\_csv("example.csv")

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).

## Пакеты для разбора файлов Excel и обратной записи с помощью Python

Помимо библиотеки Pandas, который вы будете использовать очень часто для загрузки своих данных, вы также можете использовать другие библиотеки для получения ваших данных в Python. Наш обзор основан [на этой странице](http://www.python-excel.org/?ref=codecamp.ru) со списком доступных библиотек, которые вы можете использовать для работы с файлами Excel в Python.

Далее вы увидите, как использовать эти библиотеки с помощью некоторых реальных, но упрощенных примеров.

### Использование виртуальных сред

Общий совет для установки — делать это в Python virtualenv без системных пакетов. Вы можете использовать virtualenv для создания изолированных сред Python: он создает папку, содержащую все необходимые исполняемые файлы для использования пакетов, которые потребуются проекту Python.

Чтобы начать работать с virtualenv, вам сначала нужно установить его. Затем перейдите в каталог, в который вы хотите поместить свой проект. Создайте virtualenv в этой папке и загрузите в определенную версию Python, если вам это нужно. Затем вы активируете виртуальную среду. После этого вы можете начать загрузку в другие библиотеки, начать работать с ними и т. д.

Совет: не забудьте деактивировать среду, когда закончите!

# Install virtualenv

$ pip install virtualenv

# Go to the folder of your project

$ cd my\_folder

# Create a virtual environment `venv`

$ virtualenv venv

# Indicate the Python interpreter to use for `venv`

$ virtualenv -p /usr/bin/python2.7 venv

# Activate `venv`

$ source venv/bin/activate

# Deactivate `venv`

$ deactivate

Обратите внимание, что виртуальная среда может показаться немного проблемной на первый взгляд, когда вы только начинаете работать с данными с Python. И, особенно если у вас есть только один проект, вы можете не понять, зачем вам вообще нужна виртуальная среда.

С ней будет гораздо легче, когда у вас одновременно запущено несколько проектов, и вы не хотите, чтобы они использовали одну и ту же установку Python. Или когда ваши проекты имеют противоречащие друг другу требования, виртуальная среда пригодится!

Теперь вы можете, наконец, начать установку и импорт библиотек, о которых вы читали, и загрузить их в таблицу.

### 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.

Общий совет для установки этой библиотеки — делать это в виртуальной среде Python без системных библиотек. Вы можете использовать виртуальную среду для создания изолированных сред Python: она создает папку, которая содержит все необходимые исполняемые файлы для использования библиотек, которые потребуются проекту Python.

Перейдите в каталог, в котором находится ваш проект, и повторно активируйте виртуальную среду venv. Затем продолжите установку openpyxl с pip, чтобы убедиться, что вы можете читать и записывать файлы с ним:

# Активируйте virtualenv

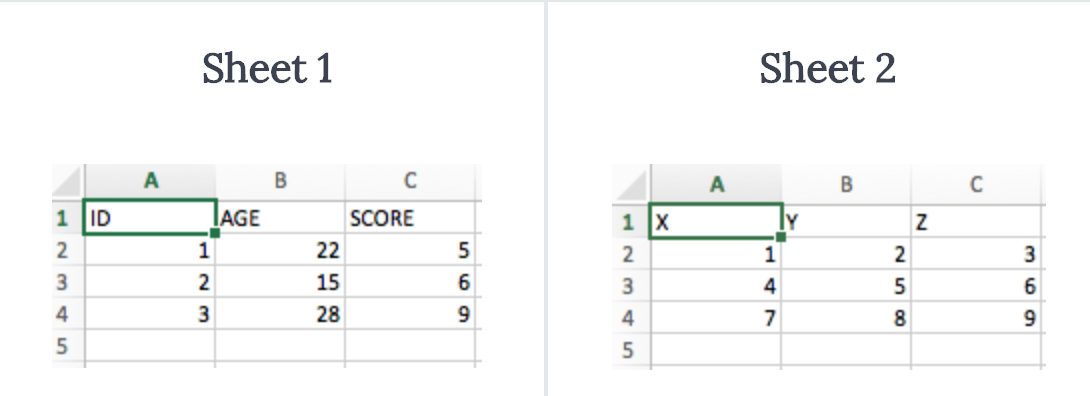
$ source activate venv

# Установим `openpyxl` в `venv`

$ pip install openpyxl

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.

# Import `load\_workbook` module from `openpyxl`

from openpyxl import load\_workbook

# Load in the workbook

wb = load\_workbook('./test.xlsx')

# Get sheet names

print(wb.get\_sheet\_names())

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:

# Get a sheet by name

sheet = wb.get\_sheet\_by\_name('Sheet3')

# Print the sheet title

sheet.title

# Get currently active sheet

anotherSheet = wb.active

# Check `anotherSheet`

anotherSheet

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:

# Retrieve the value of a certain cell

sheet['A1'].value

# Select element 'B2' of your sheet

c = sheet['B2']

# Retrieve the row number of your element

c.row

# Retrieve the column letter of your element

c.column

# Retrieve the coordinates of the cell

c.coordinate

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:

# Retrieve cell value

sheet.cell(row=1, column=2).value

# Print out values in column 2

for i in range(1, 4):

print(i, sheet.cell(row=i, column=2).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:

# Импорт необходимых модулей из `openpyxl.utils`

from openpyxl.utils import get\_column\_letter, column\_index\_from\_string

# Вывод 'A'

get\_column\_letter(1)

# Return '1'

column\_index\_from\_string('A')

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.

# Напечатать строчку за строчкой

for cellObj in sheet['A1':'C3']:

for cell in cellObj:

print(cells.coordinate, cells.value)

print('--- END ---')

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.

# Вывести максимальное количество строк

sheet.max\_row

# Вывести максимальное количество колонок

sheet.max\_column

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:

# Import `pandas`

import pandas as pd

# конвертировать Лист в DataFrame

df = pd.DataFrame(sheet.values)

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

# Put the sheet values in `data`

data = sheet.values

# Indicate the columns in the sheet values

cols = next(data)[1:]

# Convert your data to a list

data = list(data)

# Read in the data at index 0 for the indices

idx = [r[0] for r in data]

# Slice the data at index 1

data = (islice(r, 1, None) for r in data)

# Make your DataFrame

df = pd.DataFrame(data, index=idx, columns=cols)

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:

# Import `dataframe\_to\_rows`

from openpyxl.utils.dataframe import dataframe\_to\_rows

# Initialize a workbook

wb = Workbook()

# Get the worksheet in the active workbook

ws = wb.active

# Append the rows of the DataFrame to your worksheet

for r in dataframe\_to\_rows(df, index=True, header=True):

ws.append(r)

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.

Прежде чем закрыть этот раздел, не забудьте отключить виртуальную среду, когда закончите!

### Чтение и форматирование Excel-файлов: xlrd

Эта библиотека идеально подходит для чтения и форматирования данных из Excel с расширением xls или xlsx.

# Import `xlrd`

import xlrd

# Open a workbook

workbook = xlrd.open\_workbook('example.xls')

# Loads only current sheets to memory

workbook = xlrd.open\_workbook('example.xls', on\_demand = True)

Когда вам не нужны данные из всей Excel-книги, вы можете использовать функции sheet\_by\_name() или sheet\_by\_index() для получения листов, которые вы хотите получить в своём анализе.

# Load a specific sheet by name

worksheet = workbook.sheet\_by\_name('Sheet1')

# Load a specific sheet by index

worksheet = workbook.sheet\_by\_index(0)

# Retrieve the value from cell at indices (0,0)

sheet.cell(0, 0).value

Также можно получить значение в определённых ячейках с вашего листа.

Перейдите к xlwt и xlutils, чтобы узнать больше о том, как они относятся к библиотеке [xlrd](http://xlrd.readthedocs.io/en/latest/?ref=codecamp.ru).

### Запись данных в Excel-файлы с xlwt

Если вы хотите создать таблицу со своими данными, вы можете использовать не только библиотеку XlsWriter, но и xlwt. xlwt идеально подходит для записи данных и форматирования информации в файлах с расширением .xls

Когда вы вручную создаёте файл:

# Import `xlwt`

import xlwt

# Initialize a workbook

book = xlwt.Workbook(encoding="utf-8")

# Add a sheet to the workbook

sheet1 = book.add\_sheet("Python Sheet 1")

# Write to the sheet of the workbook

sheet1.write(0, 0, "This is the First Cell of the First Sheet")

# Save the workbook

book.save("spreadsheet.xls")

Если вы хотите записать данные в файл, но не хотите делать все самостоятельно, вы всегда можете прибегнуть к циклу for, чтобы автоматизировать весь процесс. Составьте сценарий, в котором вы создаёте книгу и в которую добавляете лист. Укажите список со столбцами и один со значениями, которые будут заполнены на листе.

Далее у вас есть цикл for, который гарантирует, что все значения попадают в файл: вы говорите, что для каждого элемента в диапазоне от 0 до 4 (5 не включительно) вы собираетесь что-то делать. Вы будете заполнять значения построчно. Для этого вы указываете элемент строки, который появляется в каждом цикле. Далее у вас есть еще один цикл for, который будет проходить по столбцам вашего листа. Вы говорите, что для каждой строки на листе, вы будете смотреть на столбцы, которые идут с ним, и вы будете заполнять значение для каждого столбца в строке. Заполнив все столбцы строки значениями, вы перейдете к следующей строке, пока не останется строк.

# Initialize a workbook

book = xlwt.Workbook()

# Add a sheet to the workbook

sheet1 = book.add\_sheet("Sheet1")

# The data

cols = ["A", "B", "C", "D", "E"]

txt = [0,1,2,3,4]

# Loop over the rows and columns and fill in the values

for num in range(5):

row = sheet1.row(num)

for index, col in enumerate(cols):

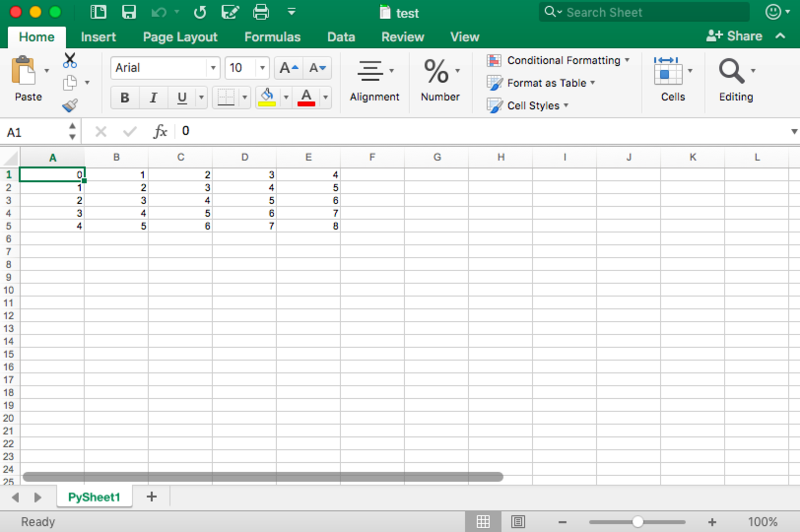
value = txt[index] + num

row.write(index, value)

# Save the result

book.save("test.xls")

На скриншоте ниже представлен результат выполнения этого кода:

[](https://lh6.googleusercontent.com/9lij2EHvLSLsPvrUwSItw3nsBn81a9sMFD-Kn0rUlRAyt180ux3GgjGQ15c5o9W4lCl_OKfFcthrjIH0z7IsaoaxFT8gE0ZLJKonqaWyYII_tY-bJDV2-xp1GmPYKxvOAmiUo6OR)

Теперь, когда вы увидели, как xlrd и xlwt работают друг с другом, пришло время взглянуть на библиотеку, которая тесно связана с этими двумя: xlutils.

### Сборник утилит: xlutils

Эта библиотека — сборник утилит, для которого требуются и xlrd и xlwt, и которая может копировать, изменять и фильтровать существующие данные. О том, как пользоваться этими командами рассказано в разделе по openpyxl.

Вернитесь в раздел openpyxl, чтобы получить больше информации о том, как использовать этот пакет для получения данных в Python.

### Использование pyexcel для чтения .xls или .xlsx файлов

Еще одна библиотека, которую можно использовать для чтения данных электронных таблиц в Python — это pyexcel; Python Wrapper, который предоставляет один API для чтения, записи и работы с данными в файлах .csv, .ods, .xls, .xlsx и .xlsm. Конечно, для этого урока вы просто сосредоточитесь на файлах .xls и .xls.

Чтобы получить ваши данные в массиве, вы можете использовать функцию get\_array(), которая содержится в пакете pyexcel:

# Import `pyexcel`

import pyexcel

# Get an array from the data

my\_array = pyexcel.get\_array(file\_name="test.xls")

Вы также можете получить свои данные в упорядоченном словаре списков. Вы можете использовать функцию get\_dict():

# Import `OrderedDict` module

from pyexcel.\_compact import OrderedDict

# Get your data in an ordered dictionary of lists

my\_dict = pyexcel.get\_dict(file\_name="test.xls", name\_columns\_by\_row=0)

# Get your data in a dictionary of 2D arrays

book\_dict = pyexcel.get\_book\_dict(file\_name="test.xls")

Здесь видно, что если вы хотите получить словарь двумерных массивов или получить все листы рабочей книги в одном словаре, вы можете прибегнуть к get\_book\_dict().

Помните, что эти две структуры данных, которые были упомянуты выше, массивы и словари вашей таблицы, позволяют вам создавать DataFrames ваших данных с помощью pd.DataFrame(). Это облегчит обработку данных.

Кроме того, вы можете просто получить записи из таблицы с помощью pyexcel благодаря функции get\_records(). Просто передайте аргумент file\_name в функцию, и вы получите список словарей:

# Retrieve the records of the file

records = pyexcel.get\_records(file\_name="test.xls")

Чтобы узнать, как управлять списками Python, ознакомьтесь с [примерами из документации о списках Python](https://codecamp.ru/documentation/python/209/list?ref=codecamp.ru).

### 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:

# Get the data

data = [[1, 2, 3], [4, 5, 6], [7, 8, 9]]

# Save the array to a file

pyexcel.save\_as(array=data, dest\_file\_name="array\_data.xls")

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:

# The data

2d\_array\_dictionary = {'Sheet 1': [

['ID', 'AGE', 'SCORE']

[1, 22, 5],

[2, 15, 6],

[3, 28, 9]

],

'Sheet 2': [

['X', 'Y', 'Z'],

[1, 2, 3],

[4, 5, 6]

[7, 8, 9]

],

'Sheet 3': [

['M', 'N', 'O', 'P'],

[10, 11, 12, 13],

[14, 15, 16, 17]

[18, 19, 20, 21]

]}

# Save the data to a file

pyexcel.save\_book\_as(bookdict=2d\_array\_dictionary, dest\_file\_name="2d\_array\_data.xls")

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. Вы можете прочитать все об этом здесь (https://pyexcel.readthedocs.io/en/latest/tutorial\_data\_conversion.html?ref=codecamp.ru#save-an-array-to-an-excel-sheet).

### 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:

# import `csv`

import csv

# Read in csv file

for row in csv.reader(open('data.csv'), delimiter=','):

print(row)

# Write csv file

data = [[1, 2, 3], [4, 5, 6], [7, 8, 9]]

outfile = open('data.csv', 'w')

writer = csv.writer(outfile, delimiter=';', quotechar='"')

writer.writerows(data)

outfile.close()

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:

# Check the first entries of the DataFrame

df1.head()

# Check the last entries of the DataFrame

df1.tail()

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

# Inspect the shape

data.shape

# Inspect the number of dimensions

data.ndim

# Inspect the data type

data.dtype

## Что дальше?

Поздравляем! Вы успешно прошли наш урок и научились читать файлы Excel на Python.

Если вы хотите продолжить работу над этой темой, попробуйте воспользоваться [***PyXll***](https://www.enthought.com/products/pyxll/introduction/?ref=codecamp.ru), который позволяет писать функции в Python и вызывать их в Excel.

# 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:

{

"access": [

"switchport mode access",

"switchport access vlan",

"switchport nonegotiate",

"spanning-tree portfast",

"spanning-tree bpduguard enable"

],

"trunk": [

"switchport trunk encapsulation dot1q",

"switchport mode trunk",

"switchport trunk native vlan 999",

"switchport trunk allowed vlan"

]

}

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):

import json

with open('sw\_templates.json') as f:

templates = json.load(f)

print(templates)

for section, commands in templates.items():

print(section)

print('\n'.join(commands))

Вывод будет таким:

$ python json\_read\_load.py

{'access': ['switchport mode access', 'switchport access vlan', 'switchport nonegotiate', 'spanning-tree portfast', 'spanning-tree bpduguard enable'], 'trunk': ['switchport trunk encapsulation dot1q', 'switchport mode trunk', 'switchport trunk native vlan 999', 'switchport trunk allowed vlan']}

access

switchport mode access

switchport access vlan

switchport nonegotiate

spanning-tree portfast

spanning-tree bpduguard enable

trunk

switchport trunk encapsulation dot1q

switchport mode trunk

switchport trunk native vlan 999

switchport trunk allowed vlan

### json.loads

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

import json

with open('sw\_templates.json') as f:

file\_content = f.read()

templates = json.loads(file\_content)

print(templates)

for section, commands in templates.items():

print(section)

print('\n'.join(commands))

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):

import json

trunk\_template = [

'switchport trunk encapsulation dot1q', 'switchport mode trunk',

'switchport trunk native vlan 999', 'switchport trunk allowed vlan'

]

access\_template = [

'switchport mode access', 'switchport access vlan',

'switchport nonegotiate', 'spanning-tree portfast',

'spanning-tree bpduguard enable'

]

to\_json = {'trunk': trunk\_template, 'access': access\_template}

with open('sw\_templates.json', 'w') as f:

f.write(json.dumps(to\_json))

with open('sw\_templates.json') as f:

print(f.read())

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):

import json

trunk\_template = [

'switchport trunk encapsulation dot1q', 'switchport mode trunk',

'switchport trunk native vlan 999', 'switchport trunk allowed vlan'

]

access\_template = [

'switchport mode access', 'switchport access vlan',

'switchport nonegotiate', 'spanning-tree portfast',

'spanning-tree bpduguard enable'

]

to\_json = {'trunk': trunk\_template, 'access': access\_template}

with open('sw\_templates.json', 'w') as f:

json.dump(to\_json, f)

with open('sw\_templates.json') as f:

print(f.read())

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):

import json

trunk\_template = [

'switchport trunk encapsulation dot1q', 'switchport mode trunk',

'switchport trunk native vlan 999', 'switchport trunk allowed vlan'

]

access\_template = [

'switchport mode access', 'switchport access vlan',

'switchport nonegotiate', 'spanning-tree portfast',

'spanning-tree bpduguard enable'

]

to\_json = {'trunk': trunk\_template, 'access': access\_template}

with open('sw\_templates.json', 'w') as f:

json.dump(to\_json, f, sort\_keys=True, indent=2)

with open('sw\_templates.json') as f:

print(f.read())

Теперь содержимое файла sw\_templates.json выглядит так:

{

"access": [

"switchport mode access",

"switchport access vlan",

"switchport nonegotiate",

"spanning-tree portfast",

"spanning-tree bpduguard enable"

],

"trunk": [

"switchport trunk encapsulation dot1q",

"switchport mode trunk",

"switchport trunk native vlan 999",

"switchport trunk allowed vlan"

]

}

### 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:

In [1]: import json

In [2]: trunk\_template = ('switchport trunk encapsulation dot1q',

...: 'switchport mode trunk',

...: 'switchport trunk native vlan 999',

...: 'switchport trunk allowed vlan')

In [3]: print(type(trunk\_template))

<class 'tuple'>

In [4]: with open('trunk\_template.json', 'w') as f:

...: json.dump(trunk\_template, f, sort\_keys=True, indent=2)

...:

In [5]: cat trunk\_template.json

[

"switchport trunk encapsulation dot1q",

"switchport mode trunk",

"switchport trunk native vlan 999",

"switchport trunk allowed vlan"

]

In [6]: templates = json.load(open('trunk\_template.json'))

In [7]: type(templates)

Out[7]: list

In [8]: print(templates)

['switchport trunk encapsulation dot1q', 'switchport mode trunk', 'switchport trunk native vlan 999', 'switchport trunk allowed vlan']

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 |

### Ограничение по типам данных

В формат JSON нельзя записать словарь, у которого ключи - кортежи:

In [23]: to\_json = {('trunk', 'cisco'): trunk\_template, 'access': access\_template}

In [24]: with open('sw\_templates.json', 'w') as f:

...: json.dump(to\_json, f)

...:

...

TypeError: key ('trunk', 'cisco') is not a string

С помощью дополнительного параметра можно игнорировать подобные ключи:

In [25]: to\_json = {('trunk', 'cisco'): trunk\_template, 'access': access\_template}

In [26]: with open('sw\_templates.json', 'w') as f:

...: json.dump(to\_json, f, skipkeys=True)

...:

...:

In [27]: cat sw\_templates.json

{"access": ["switchport mode access", "switchport access vlan", "switchport nonegotiate", "spanning-tree portfast", "spanning-tree bpduguard enable"]}

Кроме того, в JSON ключами словаря могут быть только строки. Но, если в словаре Python использовались числа, ошибки не будет. Вместо этого выполнится конвертация чисел в строки:

In [28]: d = {1: 100, 2: 200}

In [29]: json.dumps(d)

Out[29]: '{"1": 100, "2": 200}'

# 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.

# Какие инструменты, кроме Python, поддерживают формат хранения данных Parquet

Помимо Python, формат хранения данных Parquet поддерживается множеством инструментов и языков программирования. Вот несколько популярных инструментов, которые поддерживают формат Parquet:

* **Apache Arrow:** Apache Arrow является универсальной инфраструктурой для обработки данных в памяти и включает поддержку формата Parquet. Он предоставляет набор API для работы с данными Parquet на разных языках, включая C++, Java, JavaScript, Go, Ruby и многие другие.
* **Apache Spark:** Apache Spark — это мощный фреймворк для обработки больших объемов данных и аналитики. Он предоставляет встроенную поддержку для формата Parquet и позволяет эффективно работать с данными в этом формате. Spark поддерживает несколько языков программирования, включая Scala, Java, Python и R.
* **Apache Hive:** Apache Hive — это инфраструктура для обработки данных и выполнения аналитических запросов. Hive также поддерживает формат Parquet и предоставляет возможность создания таблиц и выполнения запросов на основе данных Parquet с использованием языка запросов HiveQL.
* **Apache Impala:** Apache Impala — это высокопроизводительный SQL-движок, разработанный для анализа данных в формате Parquet и других форматах. Он позволяет выполнить сложные SQL-запросы на данных Parquet с низкой задержкой.
* **Dask:** Dask — это фреймворк для параллельных вычислений в Python. Он имеет интеграцию с форматом Parquet и позволяет распределять обработку данных Parquet на кластерах и многопоточных системах.
* **Presto:** Presto — это распределенный SQL-движок, который обеспечивает быструю обработку запросов на больших объемах данных. Presto также поддерживает формат Parquet и может выполнять запросы на данных, хранящихся в Parquet-файлах.

Это лишь несколько примеров инструментов и фреймворков, которые поддерживают формат хранения данных Parquet. Parquet широко используется в экосистеме больших данных, поэтому многие другие инструменты также имеют поддержку этого формата.

# **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/