

DIRTY PYTHON 1.0

STONE 13TH PRESENTS

ЛЕКЦИЯ 6



DIRTY
PYTHON

ЧТО У НАС СЕГОДНЯ?

Работа с файлами:

- Для чего вообще нужны эти файлы
- Как их создавать
- Как их читать
- Как их записывать: строками или другими типами
- Поговорим о JSON



ИТАК, ДЛЯ ЧЕГО ЖЕ НАМ НУЖНЫ ФАЙЛЫ, ЕСЛИ МЫ ПИШЕМ КОД?



- Да собственно для того, для чего их и придумали: хранение, запись и получение информации
- Но ведь можно зашить всю инфу в код? Можно, но не всегда это удобно раз, и можно потерять промежуточную информацию – два: как только вы закроете среду разработки вся промежуточная информация потрется, а иногда результат работы кода надо сохранить
- Типичный пример использования: логер – своеобразный журнал того, что делал и когда, который записывается в отдельный файлик.
- Да, конечно, если информации действительно много, то тут обычным txt файликом не обойдешься, надо обращаться к Базам Данных, но иногда этого более чем достаточно



НАЧИНАЕМ РАБОТАТЬ С ФАЙЛОМ



- Для того чтобы создать файл в вашем проекте достаточно выбрать 'создать файл' – дать название и указать расширение.
- Точно так же можно перетащить уже готовый файл в нужную папку.
- А вот чтобы работать с файлом уже в программе, нам понадобиться создать объект, в который мы его будем открывать методом open.

```
# w - write запись
# r - read чтение
# a - append дополнение

file = open('data/file.txt', flag, encoding='UTF-8')
```

Создали переменную file (называете, как вам нравится, но помните про нейминг)
В аргументах open укажем:
путь,
что именно будем делать с файлом: писать,
читать или добавлять
и желательно еще указать кодировку, иначе у вас будет нормально отображаться только латинские буквы

ПУТЬ ФАЙЛА



- Итак, когда мы открываем файл, главное правильно указать откуда его брать. И тут есть два способа

1) Абсолютный: когда мы указываем, где лежит файл, начиная с диска:

/Users/di/Desktop/DirtyPython/dirtyPython1/Lection/file.txt

К преимуществам такого способа указания файла можно отнести то, что на вашем компьютере файл точно будет найден, даже если вы создали проект в проекте, пока делали проект...

К недостаткам – то, что открыв проект на другом компьютере ссылка перестанет работать, хотя текстовый файл может лежать в той же директории, что файл с кодом

2) Относительный: указываем путь относительно корневой папки: file.txt - если прямо в одной папке и лежит, 'data/file.txt' – если вы для файлов сделали отдельную директорию data

Преимущества и недостатки – зеркальные с абсолютным путем: ссылка будет работать в любом месте, но легко запутаться, если у вас несколько уровней вложенности....





ЧТО ДЕЛАТЬ-ТО БУДЕМ?

- После того, как указали путь, обязательно надо прописать, что именно вы собираетесь с файлом делать. И тут у нас есть три основных флага. Внимание! Это не сами методы, а только указание, что мы будем с файлом делать
- ‘w’ - write запись: мы не только запишем в файл, но даже создадим его, если он ранее не существовал, но надо иметь ввиду – файл будет перезаписан, то есть то, что в нем было ранее будет заменено на новое
- ‘r’ – read чтение: так мы показываем, что далее будем только извлекать информацию из файла
- ‘a’ – add дополнение: а вот это добавление информации к уже имеющемуся файлу

WRITE



- Запись в файл осуществляется при помощи метода **write**
- В скобках указывается **строка**, которая будет записана в файл.
- Если передать не строку, а что-то другое, возникнет ошибка.
- После записи файл нужно обязательно закрыть методом `close()`, а то он так и останется висеть открытым и к нему нельзя будет обратиться

```
6 file = open('data/file.txt', 'w', encoding='UTF-8')
7 file.write('123')    I
8 file.close()
9
```

D:\WorkShopPy\.my_venv_3.11.3\Scripts\python.exe D:\WorkShopPy\PhoneBookClasses\file.py

Process finished with exit code 0

```
6 file = open('data/file.txt', 'w', encoding='UTF-8')
7 file.write(123)
8
```

Run file x

D:\WorkShopPy\.my_venv_3.11.3\Scripts\python.exe D:\WorkShopPy\PhoneBookClasses\file.py

Traceback (most recent call last):
 File "D:\WorkShopPy\PhoneBookClasses\file.py", line 7, in <module>
 file.write(123)
TypeError: write() argument must be str, not int

WRITE

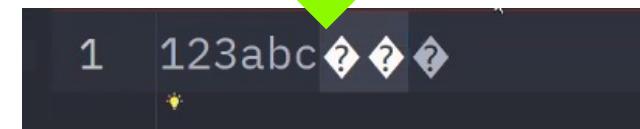
- Если мы несколько раз вызовем write и передадим какие-то строки, то все они успешно запишутся, но в файле они не будут ничем разделены
- И кстати, если вы забудете про encoding и передадите кириллицу, то получите что-то мало вразумительное. И не советую вам лезть в настройки windows

```
6 file = open('data/file.txt', 'w', encoding='UTF-8')
7 file.write('123')
8 file.write('abc')
9 file.write('абв')
10 file.close()
```



```
1 123abcабв
```

```
6 file = open('data/file.txt', 'w')
7 file.write('123')
8 file.write('abc')
9 file.write('абв')
10 file.close()
```



```
1 123abc?????
```

APPEND



- Если же мы не хотим перезаписывать файл полностью, а только что-то в него добавить, то нам надо будет указать 'a' при открытии и использовать метод `append()`
- В него тоже передаем строки!





СЛУЖЕБНЫЕ СИМВОЛЫ

- Если нам хочется, чтобы передаваемые строки не записывались сплошняком, самое время вспомнить про служебные символы (которые можно использовать не только тут, но и при использовании того же принта)
- Для этого нам понадобиться обратный слэш (он обычно рядом с правым enter)
- \n в конце передаваемой строчки обеспечит переход на новую строку
- \t вставка табуляции (4 пробела)
- \' одинарная кавычка, которая не закроет вам случайно строку...
- Ну и другие символы можно посмотреть в табличке

1	123
2	Новая
3	строка \t отступ пробел

Последовательность	Назначение
\n	Новая строка
\\	Символ обратного слеша
'	Апостроф '
"	Кавычка "
\a	Звуковой сигнал
\b	Забой (символ клавиши BackSpace)
\f	Перевод формата
\r	Возврат каретки
\t	Горизонтальная табуляция
\v	Вертикальная табуляция
\xhh	Символ с шестнадцатеричным кодом hh
\ooo	Символ с восьмеричным кодом ooo
\0	Символ Null (не признак конца строки)
\N{id}	Идентификатор ID базы данных Unicode
\uhhhh	16-битный символ Unicode в шестнадцатеричном формате
\Uhhhhhhhhh	32-битный символ Unicode в шестнадцатеричном формате
\другое	Не является экранированной последовательностью (символ \ сохраняется)

```
6 file = open('data/file.txt', 'w', encoding='UTF-8')
7 file.write('123\n')
8 file.close()
9
10 file = open('data/file.txt', 'a', encoding='UTF-8')
11 file.write('Новая\nстрока\tотступ пробел')
12 file.close()
```



МУДРОСТЬ ОТ СТОУНА

ПРОБЕЛ - ОН И В АФРИКЕ ПРОБЕЛ

READ



Если мы захотим поработать с информацией из файла, мы укажем флаг 'r', потом вызовем методом read и присвоим какой-нибудь переменной

И вот теперь можно уже что-то делать – например распечатать принтом.

Обратите внимание: print распечатал с учетом служебных символов

Если мы попробуем распечатать объект file, в который мы передавали путь – то получим совсем другое – ссылка на сам файл, если еще не почистили

```
15 file = open('data/file.txt', 'r', encoding='UTF-8')
16 data = file.read()
17 file.close()
18 print(data)
19 print(file)
```

```
Run file x
D:\WorkShopPy\my_venv_3.11.3\Scripts\python.exe D:\WorkShopPy\PhoneBookClasses\file.py
123
Новая
строка отступ пробел
<_io.TextIOWrapper name='data/file.txt' mode='r' encoding='UTF-8'>
```

READLINE



- Кроме метода `read`, который читает весь файл целиком, еще есть `readline()` – читающий по строчкам, что иногда прямо очень удобно
- Но если вы обратили внимание в терминале распечаталось с лишними пустыми строками – это как раз из-за `\n`
- Кстати, посмотреть, как выглядит строка на самом деле можно при помощи `__repr__`

The screenshot shows the PyCharm IDE interface. In the top editor window, there is Python code:

```
file = open('data/file.txt', 'r', encoding='UTF-8')
data1 = file.readline()
data2 = file.readline()
data3 = file.readline()
file.close()

print(data1)
print(data2)
print(data3)
# my_list = [data]
```

In the bottom terminal window, the output of the run command is shown:

```
D:\WorkShopPy\my_venv_3.11.3\Scripts\python.exe D:\WorkShopPy\PhoneBookClasses\file.py
123
Новая
Т
```

Below the terminal, status bar text indicates: строка отступ пробел.

```
22 print(data1.__repr__())
23 print(data2.__repr__())
24 print(data3.__repr__())
25 print(data4.__repr__())
```

READLINES



- Метод `readlines` возвращает список из строк файла. И именно его чаще всего и используют.
 - А потом уже куда как удобнее работать со списком: проходится циклом и делать, что угодно
 - В принципе тоже самое можно получить и при помощи `readline` и цикла, но согласитесь запись справа куда лаконичнее. Но `readline` удобно использовать, когда вам например нужна, каждая вторая или третья строка

```
15     file = open('data/file.txt', 'r', encoding='UTF-8')
16     data_list = []
17     while True:
18         elem = file.readline()
19         if elem != '':
20             data_list.append(elem)
21         else:
22             break
```



CLOSE



FILE HANDLING IN PYTHON

- Поработал – закрой, потому как пока файл открыт, никто с ним не сможет поработать кроме вас
- Более того, система не даст вам этот файл удалить, пока он открыт – можно попасть в просак.
- Но Python не был бы таким няшным языком, если бы не было способа обезопасить себя, поэтому у нас есть контекстный менеджер

USE OPEN()



USE CLOSED



USE WITH OPEN()

КОНТЕКСТНЫЙ МЕНЕДЖЕР



- Вместо того, чтобы открывать файл `open`, а потом закрывать `close` – используем конструкцию `with open(path, 'flag', encoding='UTF-8') as file:`
- Можно и не `file`, а любое другое наименование, главное, что внутри тела `with open` вы будете обращаться именно к этой переменной
- Внутри тела (обозначив табуляцией) прописывается все, что вы хотите с файлом сделать
- И да, кстати, неожиданно, но в режиме '`w`' метод `print` можно использовать для записи в файл

```
15 with open('data/file.txt', 'r', encoding='UTF-8') as file:  
16     data_list = []  
17     file.seek()  
18     data_list = file.readline()  
19     print(data_list)  
20
```

```
with open('data/file.txt', 'w', encoding='UTF-8') as file:  
    print('Здарова, заебал', file=file)
```

СДЕЛАЕМ-КА СЛОВАРЬ!



- Если наш файл содержит однотипные данные, да еще и разделенные определенным символом (например ;), то довольно удобно преобразовывать такие данные для дальнейшей работы в словарь, содержащий в качестве ключа, например, номер строки, а в качестве значения – другой словарь с содержимым строки... Звучит путано, но в коде это довольно удобно: у всего есть имя, по которому можно обратиться

```
Scratches and Consoles
15 with open('data/file.txt', 'r', encoding='UTF-8') as file:
16     phone_book = file.readlines()
17
18 print(phone_book)
19
20
```

Вот тут мы получили список из строк, внутри которых имя, телефон и комментарий разделены ;

```
Run file x
D:\WorkShopPy\.my_venv_3.11.3\Scripts\python.exe D:\WorkShopPy\PhoneBookClasses\file.py
['Панфилов Кирилл;+79094512021;Семинары GB\n', 'Андрей Беляев;888999000;Кент Стоуна\n', 'Диана;0000000001;По
```

STRIP



- Самые внимательные заметили, что в нашем полученном списке у всех значений, кроме последних имеется в конце служебный символ \n, который хорошо бы обрезать.
- Можно, конечно, сделать через срезы и условия, но у строк есть замечательные методы:
- strip убирает лишние пробелы и служебные символы с двух сторон строки
- rstrip – справа, а lstrip - слева

```
file.txt
file.py
External Libraries
Scratches and Consoles

15     with open('data/file.txt', 'r', encoding='UTF-8') as file:
16         phone_book = file.readlines()
17
18     print(phone_book)
19     new_book = []
20     for contact in phone_book:
21         new_book.append(contact.strip())
22
23     print(new_book)
24

for contact in phone_book

Run file.py
D:\WorkShopPy\.my_venv_3.11.3\Scripts\python.exe D:\WorkShopPy\PhoneBookClasses\file.py
['Панфилов Кирилл;+79094512021;Семинары GB\n', 'Андрей Беляев;888999000;Кент Стоуна\n', 'Диана;0000000001;По конспектам\n', 'Панфилов Кирилл;+79094512021;Семинары GB', 'Андрей Беляев;888999000;Кент Стоуна', 'Диана;0000000001;По конспектам']
```

Обратите внимание, что при использовании strip у первого и второго элемента удалились лишние символы, а у последнего и так все было хорошо

И ВОТ НАКОНЕЦ-ТО СЛОВАРЬ



- И вот теперь мы можем создать словарик разбив наши строковые элементы из списка при помощи `split`
- Да и `enumerate` пригодился и мы использовали как ключ, а внутрь передали словарь, где ключ определили сами, а в качестве значений поочередно брали сплитованные части самого элемента списка

```
14
15 with open('data/file.txt', 'r', encoding='UTF-8') as file:
16     phone_book = file.readlines()
17
18 print(phone_book)
19 new_book = {}
20 for i, contact in enumerate(phone_book, 1):
21     contact = contact.strip().split(';')
22     new_book[i] = {'name': contact[0], 'phone': contact[1], 'comment': contact[2]}
23
24 print(new_book)
25
```

D:\WorkShopPy\.my_venv_3.11.3\Scripts\python.exe D:\WorkShopPy\PhoneBookClasses\file.py

[{'name': 'Панфилов Кирилл;+79094512021;Семинары GB', 'phone': '888999000', 'comment': 'Кент Стоуна'}, {'name': 'Андрей Беляев;888999000;Кент Стоуна', 'phone': '8889990001', 'comment': 'По конспектам'}, {'name': 'Диана;0000000001;По конспектам'}]

Process finished with exit code 0

СЛОВАРЬ, ПАРСИНГ И JSON



- Почему мы решили использовать словарь? Потому что это удобно при парсинге текста
- Про парсинг мы поговорим чуть позже, но в двух словах.
- Итак, парсинг – это процесс сбора данных с последующей их обработкой и анализом...
- И мы с вами это только что сделали: получили список, убрали лишнее, сделали удобный словарик.
- Но ведь не для телефонной книги это используют? Разумеется, сейчас парсинг используют для обработки данных получаемых с сайтов. А это отдельная и большая тема
- И, конечно, есть целая куча библиотек для парсинга (например, Beautiful Soup)

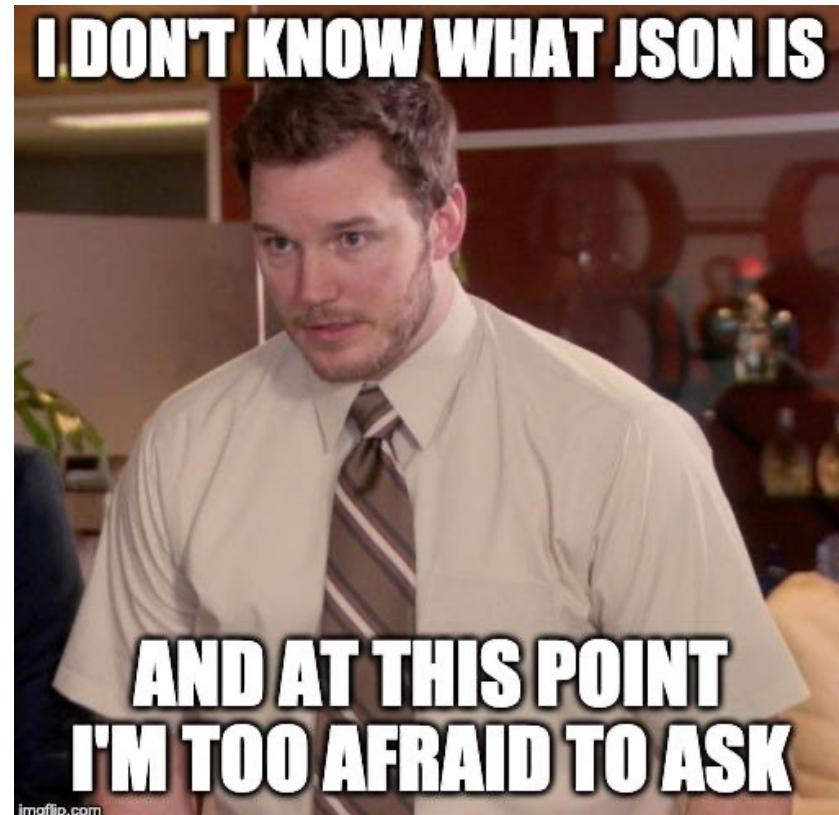


meme-arsenal.ru

JSON



- Одним из самых удобных форматов работы с данными является JSON, который очень и очень похож на словарь
- JSON (англ. JavaScript Object Notation, обычно произносится как /'dʒeɪsən/] — текстовый формат обмена данными, основанный на JavaScript. Как и многие другие текстовые форматы, JSON легко читается людьми.
- Несмотря на происхождение от JavaScript, формат считается независимым от языка и может использоваться практически любым языком программирования.



JSON



- Попробуем записать свой собственный файл с расширением json
- Хорошим тоном считается не прописывать адрес напрямую, а создать для него переменную. Такие переменные будут константами, и потому их имена пишут большими буквами.
- Теперь во всех контекстных менеджерах можно просто писать PATH для файла txt, JSON_PATH для нашего json файла
- Единственное, для работы с json файлами, нам понадобиться библиотека - не забудьте импортировать ее import json
- И два основным метода – dump для записи и load для чтения

```
13
14 PATH = 'data/file.txt'
15 JSON_PATH = 'data/text.json'
16 with open(PATH, 'r', encoding='UTF-8') as file:
17     phone_book = file.readlines()
18
```

```
18 with open(JSON_PATH, 'w', encoding='UTF-8') as file:
19     json.dump(new_book, file)
20
21 with open(JSON_PATH, 'r', encoding='UTF-8') as file:
22     json_data = json.load(file)
23
24 print(json_data)
```



ЕЩЕ НЕМНОГО О JSON

- Файловыми методами write и add можно записать только строки
- А вот json позволяет записать любой встроенный тип данных – int, list, dict, set за исключением tuple
- Чтобы json нормально работал с кириллицей при записи необходимо указать ensure_ascii=False

```
17  
18     with open(JSON_PATH, 'w', encoding='UTF-8') as file:  
19         json.dump(new_book, file, ensure_ascii=False)  
20
```

ИЧТО ДАЛЬШЕ?



D1RTY
PYTH0N

КОД, КОТОРЫЙ
МЫ ПИШЕМ НА
DIRTY PYTHON



ДОМАШКА
GB



И ЧТО ДАЛЬШЕ?

- ПЕРЕВАРИВАЙТЕ
- ПРОЙДИТЕСЬ ПО КОДУ РУКАМИ
- СДЕЛАЕМ ВАМ ПАРУ ЗАДАНИЙ ПОПРОШЕ
- И ОДНУ ЗАДАЧКУ ПРЯМ ИНТЕРЕСНУЮ



D1RTY
РУТНОН