GeekBrains

Работа с инструментами и файлами

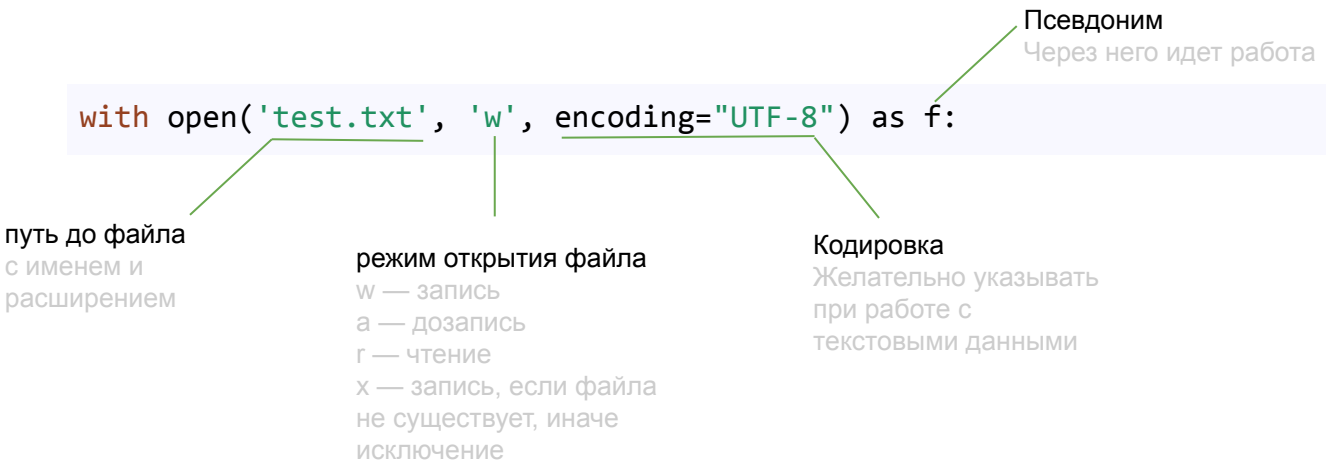
О принципах работы с файлами, о создании многофайловых проектов и особенностях импорта модулей, расширяющих возможности приложения

Файлы

Принципы работы с файлами

Чтение и запись данных в файл — классическая задача в любом языке программирования. В файлах могут храниться базы данных, мультимедийная и другая информация. И для работы с каждым типом данных у Python есть дополнительные модули.

Начнем с классических текстовых файлов. Прежде чем начать работу с файлом, его необходимо открыть. Чаще всего для этого используют оператор **with** и функцию **open**:



Принципы работы с файлами

Функция **open** создаёт блок, внутри которого будет происходить вся работа с файлом. Как только блок закончится, файл автоматически закроется.

Основные команды для чтения содержимого файла:

```
with open('test.txt', 'w', encoding="UTF-8") as f:
    f.read()            # Читаем весь файл
    f.read(5)           # Читаем 5 символов с текущей позиции
    f.readline()        # Читаем целую строку из файла
    f.readlines()       # Зачитывает файл построчно в список

    for line in f:      # Читаем построчно файл
        print(line)     # и обрабатываем каждую строку отдельно
```

Для записи используется метод:

```
f.write(<data_to_write>)
```

Принципы работы с файлами

У любого файла есть также свойства, которые позволят получить полезную информацию:

```
    f.closed            # признак закрытого файла (True - закрыт, False - открыт)
    f.mode              # режим открытия
    f.name              # имя файла
```

Данные свойства особенно полезны при работе с файлом внутри класса. Файл открывается с помощью обычной функции open в конструкторе:

```
class Database:
    def __init__(self, filename):
        self.file = open(filename, 'a+')    # Открываем файл на чтение и дозапись

    def __del__(self):
        self.file.close()                   # Закрываем в деструкторе
```

Файлы CSV

Модуль для работы и объект reader

CSV — один из самых популярных форматов для хранения большого набора данных. Данные записываются построчно через заранее определённый разделитель. Причём разделитель может быть разным в разных файлах.

Для удобной работы с CSV-файлами есть модуль, который необходимо подключить к проекту:

```
1 import csv
```

Далее также создается объект, который содержит данные файла и дополнительные методы и свойства для обработки.

```
reader = csv.reader('file.csv', delimiter=",")  # объект чтения CSV файла
```

путь до файла
с именем и
расширением

Разделитель
установленный в файле между
данными одной строки

Чтение файла. Объект DictReader

Полученный объект reader является итерируемым, а значит, мы можем спокойно пройтись по его содержимому с помощью цикла for. На каждой итерации будем получать одну строку данных в виде списка:

```
for row in reader:
  print(row)
```

Ещё одна разновидность — объект DictReader. У него в отличие от reader происходит соотношение имён столбцов к их содержимому. И мы можем обращаться на каждой итерации цикла к нужной колонке по её имени. Наименования столбцов хранятся в свойстве fieldnames:

```
    reader = csv.DictReader(f, delimiter=';')
    field_names = reader.fieldnames             # Наименования столбцов
    print(field_names)
    for row in reader:
        print(row['name'], row['address'])      # выводим только нужное
```

Запись в файл

Для записи в файл необходимо создать объект другого класса:

```
1 writer = csv.writer(f, delimiter=';')
```

Записываются данные с помощью метода **writerow**, который принимает данные в виде списка:

```
1  writer = csv.writer(f, delimiter=';')
2  writer.writerow([183, 'Школа № 83', 'Ленина-91'])
```

Запись в файл

Также существует объект DictWriter, где данные можно записать в виде словаря. Параметры при создании этого объекта немного другие:

```
with open('data-2.csv', 'a', encoding="UTF-8") as f:
    fieldnames = ['num', 'name', 'address']
    writer = csv.DictWriter(f, fieldnames=fieldnames)
    writer.writerow({'num': 184,
                     'name': 'Тестовая',
                     'address': 'Пушкинская-90'})
```

Если файл совсем пустой, необходимо вначале заполнить заголовки и только потом записывать данные:

```
with open('data-2.csv', 'a', encoding="UTF-8") as f:
    fieldnames = ['num', 'name', 'address']
    writer = csv.DictWriter(f, fieldnames=fieldnames)
    writer.writeheader()                # Записывает заголовки в файл из fieldnames
    ...
```

JSON

Что такое JSON

JSON (JavaScript Object Notation) — формат данных, который стал очень популярным ввиду удобства использования в разных языках. Очень похож на словарь в Python.

Данные можно хранить в файлах с расширением *.json. В языке Python работу с этими данными упрощает модуль json.


```json
{
  "coord": {
    "lon": -122.08,
    "lat": 37.39
  },
  "weather": [
    {
      "id": 800,
      "main": "Clear",
      "description": "clear sky",
      "icon": "01d"
    }
  ],
  "base": "stations",
  "main": {
    "temp": 282.55,
    "feels_like": 281.86,
    "temp_min": 280.37,
    "temp_max": 284.26,
    "pressure": 1023,
    "humidity": 100
  },
  "visibility": 10000,
  "wind": {
    "speed": 1.5,
    "deg": 350
  },
  "clouds": {
    "all": 1
  },
  "dt": 1560350645,
  "sys": {
    "type": 1,
    "id": 5122,
    "message": 0.0139,
    "country": "US",
    "sunrise": 1560343627,
    "sunset": 1560396563
  },
  "timezone": -25200,
  "id": 420006353,
  "name": "Mountain View",
  "cod": 200
}
```

Чтение JSON-данных

Файл открывается стандартным способом, и данные его считываются с помощью метода .**load()**:

```
1 import json
2
3 with open('data.json', 'r') as f:    # Читаем json файл (данные структурированы)
4     info = json.load(f)
```

В info у нас попадает содержимое прочитанного файла целиком в виде словаря.

Также модуль json позволяет считывать строки и текстовые файлы, содержащие в себе json, чтобы конвертировать содержимое в словарь:

```
1 with open('test.txt', 'r') as f:
2     content = f.read()              # считываем обычный текстовый файл
3     info = json.loads(content)      # получаем json
```

Запись в JSON

Записать данные в файл формата JSON также достаточно легко. Для этого в модуле json есть два метода:

1) json.dump() — записывает словарь в файл формата json

```
1 to_json = {'one': 1, 'two': 2, 'three': 3}   # Подготовим словарь
2
3 with open('new.json', 'w') as f:
4     json.dump(to_json, f)                    # Записываем словарь в json
```

2) json.dumps() — записывает json-данные в текстовый файл

```
1 to_json = {'one': 1, 'two': 2, 'three': 3}   # Подготовим словарь
2
3 with open('new.txt', 'w') as f:
4     f.write(json.dumps(to_json))             # Записываем в текстовый файл
```

Проблема с кодировкой и дополнительные возможности

Иногда при записи в json-файл возникает проблема с кодированием символов, отличных от латиницы. Также иногда бывает нужно при записи в текстовый файл сохранить структуру и расположение данных для более удобного визуального восприятия.

Для решения этих вопросов можно добавить дополнительные параметры:

- ensure_ascii=False — устраняет проблему с кириллицей
- indent = 4 — отступы размером 4 как в json
- sort_keys=True — сортирует по ключам перед записью в файл

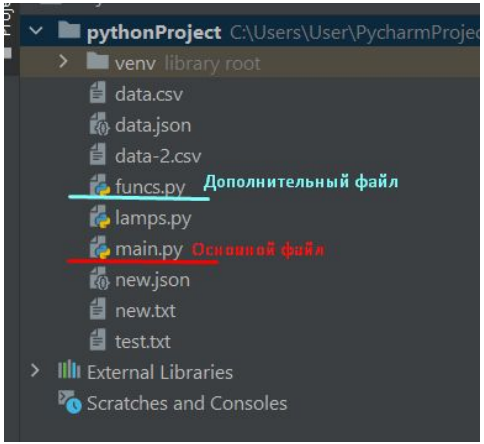
```
1 with open('new.txt', 'w') as f:
2     f.write(json.dumps(to_json, indent=4, ensure_ascii=False, sort_keys=True))
```

Файлы модулей

Свои модули

Python — модульный язык. Чтобы получить какую-либо функциональность, мы подключаем соответствующий модуль. Но помимо использования чужих модулей мы можем создавать свои. Обычно в эти модули можно выносить функции и классы, которые потом используются в основном коде.

Чтобы создать свой модуль, можно создать дополнительный python-файл. И в него вынести свои классы и функции.



А потом мы можем импортировать этот файл в наш основной код, чтобы все его содержимое стало доступно:

```
1 import funcs
```

Чтобы использовать любой объект из импортированного модуля, необходимо сначала указать сам модуль, а затем объект внутри:

```
1 result = funcs.expression(5, 1, 3)
```

Варианты импортов

Мы можем импортировать объекты из модуля так, чтобы обращаться к ним напрямую, минуя имя самого модуля:

```
1 from funcs import *
```

Звёздочка означает импорт абсолютно всех объектов. Такой импорт опасен, так как может произойти конфликт имён. Лучше использовать первый вариант.

Но иногда можно импортировать из модуля конкретные объекты:

```
1 from funcs import expression, circle_square
```

В таком случае будут доступны только данные объекты и ничего больше.

Проблема импортов

Когда выполняется инструкция import, выполняется весь код, находящийся внутри импортируемого файла. Если там будут какие-либо инструкции, то они тоже выполнятся:

```
1 # Файл main.py
2
3 import funcs
```

```
1 # Файл func.py
2
3 def circle_square(radius):
4     return 2 * 3.14 * radius
5
6 result = circle_square(20)
7 print(result)
```

При запуске **main.py** мы увидим в консоли результат вызова функции из **func.py**.

Проблема импортов

Чтобы этого избежать, необходимо разграничить запуск файла как отдельного приложения и как модуля. Для этого логика, которую необходимо выполнять только при непосредственном запуске файла, должна быть помещена в специальный блок:

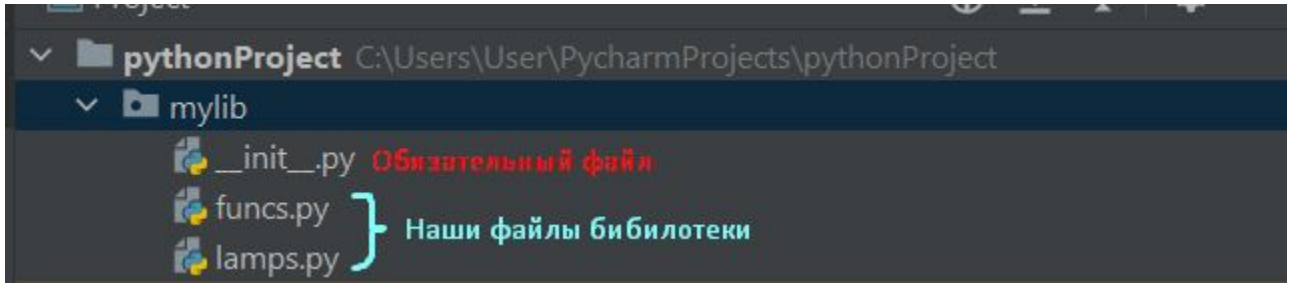
```python
# Файл func.py

def circle_square(radius):
    return 2 * 3.14 * radius


if __name__ == '__main__':          # Выполняться при import не будет
    result = circle_square(20)
    print(result)
```

Свои библиотеки

Когда файлов становится много, возникает необходимость объединить их в пакеты в отдельной папке. В этом случае поместите все ваши дополнительные файлы в одну директорию и дополнительно создайте внутри файл с названием __init__.py Содержимое файла должно оставаться пустым:



Во время импорта указывайте к файлу с учётом его расположения в отдельной папке:

```
1 from mylib.funcs import expression
```

Практическое задание

Задание: подсчитайте количество уникальных слов в текстовом файле и запишите их в другой файл в алфавитном порядке.

1. Создайте в текстовом редакторе текстовый файл по имени “data.txt” и запишите в него любой произвольный текст. Текст можно скопировать из любого источника, например из веб-страницы.
2. **Создайте функцию read_file.** Функция должна принимать имя файла, прочитать его и вернуть список уникальных слов из этого файла в нижнем регистре.
   Например:
   words = read_file('data.txt')
   # ['артемий', 'лиге', 'соперники', 'буллитов', 'серии', 'гагарина', 'реализовал', 'форвард', 'время', 'кубок', 'выступает', ...]
   Создайте функцию save_file. Функция должна принимать имя файла для записи и список уникальных слов. В функции подсчитайте количество уникальных слов и запишите его в файл вместе со всеми словами отсортированными в алфавитном порядке.

Например:
save_file('count.txt', words)
 # Содержимое файла count.txt
Всего уникальных слов: 67
==========
артемий
буллит
буллитов
в
видео
витязь
вместе
время
встречи
выиграл
выступает
...

3. Вынесите функции в отдельный файл, который нужно будет подключать с помощью import.

Вопросы?

Вопросы?

Вопросы?