Основы работы с файлами в Python

261

Михаил Свинцов
автор курса «Full-stack веб-разработчик на Python»

Взаимодействие с файловой системой позволяет хранить информацию, полученную в результате работы программы. Михаил Свинцов из SkillFactory расскажет о базовой функциональности языка программирования Python для работы с файлами.

Встроенные средства Python

Основа для работы с файлами — built-in функция `open()`

```
open(file, mode="rt")
```

Эта функция имеет два аргумента. Аргумент `file` принимает строку, в которой содержится путь к файлу. Второй аргумент, `mode`, позволяет указать режим, в котором необходимо работать с файлом. По умолчанию этот аргумент принимает значение «rt», с которым, и с некоторыми другими, можно ознакомиться в таблице ниже

Режим	Что позволяет делать?
"r"	Чтение из файла (по умолчанию)
"w"	Запись в файл. Если файл не существует, то он будет создан
"x"	Запись в файл. Если файл не существует, то будет вызвано исключение
"a"	Запись в файл. В отличие от предыдущих режимов, не стирает данные, которые уже хранятся в файле, а добавляет новые в конец
"t"	Открывает как текстовый файл (по умолчанию)
"b"	Открывает как двоичный файл
"+"	Позволяет работать с файлом как в режиме чтения, так и в режиме записи

Эти режимы могут быть скомбинированы. Например, «rb» открывает двоичный файл для чтения. Комбинируя «r+» или «w+» можно добиться открытия файла в режиме и чтения, и записи одновременно с одним отличием — первый режим вызовет исключение, если файла не существует, а работа во втором режиме в таком случае создаст его.

Начать саму работу с файлом можно с помощью объекта класса `io.TextIOWrapper`, который возвращается функцией `open()`. У этого объекта есть несколько атрибутов, через которые можно получить информацию

- `name` — название файла;
- `mode` — режим, в котором этот файл открыт;
- `closed` — возвращает `True`, если файл был закрыт.

По завершении работы с файлом его необходимо закрыть при помощи метода `close()`

```
f = open("examp.le", "w")
//  работа с файлом
f.close()
```

Однако более pythonic way стиль работы с файлом встроенными средствами заключается в использовании конструкции `with .. as ..`, которая работает как менеджер создания контекста. Написанный выше пример можно переписать с ее помощью

```
with open("examp.le", "w") as f:
// работа с файлом
```

Главное отличие заключается в том, что python самостоятельно закрывает файл, и разработчику нет необходимости помнить об этом. И бонусом к этому не будут вызваны исключения при открытии файла (например, если файл не существует).

Чтение из файла

При открытии файла в режимах, допускающих чтение, можно использовать несколько подходов.

Работа со строками в Python. Готовимся к собеседованию: вспоминаем азы
tproger.ru

Для начала можно прочитать файл целиком и все данные, находящиеся в нем, записать в одну строку.

```
with open("examp.le", "r") as f:
    text = f.read()
```

Используя эту функцию с целочисленным аргументом, можно прочитать определенное количество символов.

```
with open("examp.le", "r") as f:
    part = f.read(16)
```

При этом будут получены только первые 16 символов текста. Важно понимать, что при применении этой функции несколько раз подряд будет считываться часть за частью этого текста — виртуальный курсор будет сдвигаться на считанную часть текста. Его можно сдвинуть на определенную позицию, при необходимости воспользовавшись методом `seek()`.

```
with open("examp.le", "r") as f: # 'Hello, world!'
    first_part = f.read(8)       # 'Hello, w'
    f.seek(4)
    second_part = f.read(8)      # 'o, world'
```

Другой способ заключается в считывании файла построчно.
Метод `readline()` считывает строку и, также как и с методом `read()`, сдвигает курсор — только теперь уже на целую строку. Применение этого метода несколько раз будет приводить к считыванию нескольких строк. Схожий с этим способом, другой метод позволяет прочитать файл целиком, но по строкам, записав их в список. Этот список можно использовать, например, в качестве итерируемого объекта в цикле.

```
with open("examp.le", "r") as f:
    for line in f.readlines():
        print(line)
```

Однако и здесь существует более pythonic way. Он заключается в том, что сам объект `io.TextIOWrapper` имеет итератор, возвращающий строку за строкой. Благодаря этому нет необходимости считывать файл целиком, сохраняя его в список, а можно динамически по строкам считывать файл. И делать это лаконично.

```
with open("examp.le", "r") as f:
    for line in f:
        print(line)
```

Запись в файл

Функциональность внесения данных в файл не зависит от режима — добавление данных или перезаписывание файла. В выполнении этой операции также существует несколько подходов.

Самый простой и логичный — использование функции `write()`

```
with open("examp.le", "w") as f:
    f.write(some_string_data)
```

Важно, что в качестве аргумента функции могут быть переданы только строки. Если необходимо записать другого рода информацию, то ее

необходимо явно привести к строковому типу, используя методы `__str__(self)` для объектов или форматированные строки.

Есть возможность записать в файл большой объем данных, если он может быть представлен в виде списка строк.

```
with open("examp.le", "w") as f:
    f.writelines(list_of_strings)
```

Здесь есть еще один нюанс, связанный с тем, что функции `write()` и `writelines()` автоматически не ставят символ переноса строки, и это разработчику нужно контролировать самостоятельно. Существует еще один, менее известный, способ, но, возможно, самый удобный из представленных. И как бы не было странно, он заключается в использовании функции `print()`. Сначала это утверждение может показаться странным, потому что общеизвестно, что с помощью нее происходит вывод в консоль. И это правда. Но если передать в необязательный аргумент `file` объект типа `io.TextIOWrapper`, каким и является объект файла, с которым мы работаем, то поток вывода функции `print()` перенаправляется из консоли в файл.

```
with open("examp.le", "w") as f:
    print(some_data, file=f)
```

Сила такого подхода заключается в том, что в `print()` можно передавать не обязательно строковые аргументы — при необходимости функция сама их преобразует к строковому типу.

На этом знакомство с базовой функциональностью работы с файлами можно закончить. Вместе с этим стоит сказать, что возможности языка Python им не ограничивается. Существует большое количество библиотек, которые позволяют работать с файлами определенных типов, а также допускают более тесное взаимодействие с файловой системой. И в совокупности они предоставляют разработчикам легкий и комфортный способ работы с файлами.