Открытие и закрытие файла

Открытие файла

Для открытия файла существует функция open(). Эта функция создает на основе файла объект, с которым можно выполнять действия. Открыть файл на чтение можно так:

```
file0 = open("text.txt")
file1 = open("text.txt", "r")
file2 = open("text.txt", "rt")
```

По умолчанию файл всегда открывается на чтение, даже если не указать режим открытия. Однако, есть еще другие режимы открытия файла, которые нужно обязательно указывать:

- **"rb"** – открытие бинарного файла на чтение;
- **"w", "wt"** – открытие текстового файла на запись (если в файле содержались данные, они сотрутся, а новые запишутся с начала); записывать данные можно много раз, пока файл не закрыт; записывать можно в конец или в любое место файла, если переместить указатель (об этом чуть дальше), но в этом случае не сохранятся данные, поверх которых производится запись;
- **"wb"** – открытие бинарного файла на запись;
- **"a"** – открытие файла на дозапись в конец.

Если файл, открываемый на запись не существует, он будет создан в папке с программой. Если нет файла, открываемого на чтение, возникнет ошибка:

```
FileNotFoundError: [Errno 2] No such file or directory: 'text.txt'
```

Функция open() имеет несколько необязательных аргументов. Выполним help(open) для получения справки по этой функции.

```
help(open)
```

В консоль выведется довольно большой текст, начало которого такое:

```
Help on built-in function open in module io:

open(file, mode='r', buffering=-1, encoding=None, errors=None,
newline=None, closefd=True, opener=None)
    Open file and return a stream.  Raise OSError upon failure.

    file is either a text or byte string giving the name (and the
path
    if the file isn't in the current working directory) of the file
to
    be opened or an integer file descriptor of the file to be
    wrapped. (If a file descriptor is given, it is closed when the
```

```
    returned I/O object is closed, unless closefd is set to False.)
...
```

Из всех параметров наиболее часто используются следующие:

- file – имя открываемого файла. Это обязательный аргумент. Оно может быть задано с использованием и относительных и абсолютных путей;
- mode – режим открытия, по умолчанию "rt";
- encoding – если работа идёт в текстовом режиме, Python должен получить имя кодировки, чтобы корректно работать с данными. Независимо от кодировки файла, в результате чтения будет возвращаться стандартная юникод-строка Питона;
- newline – символ новой строки, по умолчанию "\n". Чаще всего используется при записи текстовых файлов, для того чтобы заканчивать строки символами отличными от принятых в операционной системе, на которой исполняется программа.

Пожалуйста, прочтите полное описание параметров в документации по функции open.

После окончания работы с файлом его нужно закрыть, иначе в случае аварийного завершения работы программы файл может быть поврежден. Если же программа завершилась штатно, то файл будет закрыт автоматически.

Для закрытия файла есть метод файлового объекта close():

```
file.close()
```

Менеджер контекста
Для того, чтобы файл закрывался автоматически даже в случае ошибок во время выполнения других операций, в языке Python есть блокwith(он называется контекстным менеджером). Его назначение шире, но в нашем случае он даёт закрыть файл после выхода из блока. Его синтаксис такой:

```
with open("input.txt") as in_file:
    ...
```

Блок кода после инструкции with записывается с отступом, который показывает, что операции производятся при открытом файле. Как только код выходит из блока с этим отступом, файл автоматически закрывается.

Пример использования контекстного менеджера:

```
with open('yevgeniy-onegin.txt', 'rt', encoding='utf-8') as f:
    read_data = f.read()
print(read_data[:100])
print(f.closed)
```

```
        Александр Сергеевич Пушкин
Евгений Онегин
```

```
Роман в стихах

    Petri de vanite il avait
True
```

С помощью этой инструкции можно сразу открывать несколько файлов. Прежде, чем что-то делать с информацией из них, не забудем отрезать непечатные символы:

```
with open('costs.txt') as in_file, open('result.txt', 'w') as
out_file:
    read_data = [float(x.strip()) for x in in_file.readlines()]
    print(str(sum(read_data)), file=out_file)
```

И еще: мы преобразовали полученное число в текст, потому что только текст можно записать в файл.

Нужно стараться закрывать файл (или выходить из блока контекстного менеджера) сразу, как он перестал быть нужен. Это делают для того, чтобы дать возможность работать с этим файлом другим программам и программистам.

Чтение файла
Чтобы работать с данными, их нужно сначала прочитать. Пока у нас есть только файловый объект, созданный из открытого файла.

Читать данные можно в двух режимах: как текст и как поток байт.

Чтение текстового файла

Прежде всего нужно понять, что текстовый файл состоит из строк.

Строка

Строка – это последовательность из некоторого количества символов, в том числе и нулевого, не содержащая символа перевода строки ("\n"), но им завершающаяся.

То есть в конце файла всегда должна находиться пустая строка. Конечно, можно не поставить последним символом перевод строки, но тогда не всякая программа обработает такой файл правильно.

Итак, перечислим способы чтения из текстового файла:

- метод read() – считывает весь текст из файла в одну строку python;
- метод readline() – считывает очередную строку из файла, при следующем вызове будет считана следующая строка; поскольку файл – это итератор, прочитать его можно только один раз; если данные нужно использовать многократно, их нужно сохранить в какую-либо структуру данных;
- метод readlines() – считывает все строки файла в список строк.

Если файл, который вы читаете, создавали не вы сами, то нужно предполагать, что он оформлен в соответствии со стандартами и заканчивается пустой строкой. Поэтому считывать файл одной строкой, а затем разбивать на части методом split('\n') – плохая практика.

Действительно, считаем файл в список разными способами:

```
Файл text.txtJingle bells,
jingle bells
Jingle all the way.
```

Посмотрите, чем отличается вывод первым способом:

```
with open("text.txt") as file:
    print(file.read().split('\n'))
# ['Jingle bells,', 'jingle bells', 'Jingle all the way.', '']
```

...и вторым:

```
with open("text.txt") as file:
    print(file.readlines())
# ['Jingle bells,\n', 'jingle bells\n', 'Jingle all the way.\n']
```

Количество элементов в списке получилось разным. От символа перевода строки во втором случае можно избавиться методом rstrip():

```
with open("text.txt") as file:
    print(list(map(str.rstrip, file.readlines())))
# ['Jingle bells,', 'jingle bells', 'Jingle all the way.']
```

или даже так:

```
with open("text.txt") as file:
    print(list(map(str.rstrip, file)))
# ['Jingle bells,', 'jingle bells', 'Jingle all the way.']
```

Файл является итератором, поэтому проходить по нему можно, даже не считывая строки в список. Экономится память.

Еще одним важным моментом является понимание, что каждая строка завершается переводом строки. Если об этом забыть, то можно получить лишние пустые строки при выводе:

```
with open("../text.txt") as file:
    for line in file.readlines():
        print(line)
```

В примере файл находится не в папке с программой, а в родительском каталоге, поэтому в пути появились две точки, означающие переход на уровень вверх.

Получим:

```
Jingle bells,

jingle bells

Jingle all the way.
```

В конце каждой строки стоял символ, который перевел курсор на следующую строку, и еще у функции print() именованный параметр end, у которого осталось значение по умолчанию "\n", перевел строку. Получились лишние строки.

Чтение в байтовом режиме

Прочитаем первые 10 байт из файла "Евгений Онегин" в байтовом режиме (при открытии файла из проводника может оказаться, что символы, которые вы видите, отличаются от кириллицы; это значит, что кодировка файла отличается от той, что ожидает ваша операционная система; это не влияет на работу программы):

```
with open("yevgeniy-onegin.txt", "rb") as in_file:
    print(in_file.read(10))
```

Вот, что мы увидим:

```
b'\xd0\x90\xd0\xbb\xd0\xb5\xd0\xba\xd1\x81'
```

Перед строкой стоит модификатор b. Он говорит о том, что перед нами поток байт. Далее записаны значения этих байт в 16-ричном формате, о чем говорит специальный символ \x. Поток байт в языке Python представляется классом bytes. Если вы открываете файл для чтения в бинарном режиме, результат метода `read()` имеет тип bytes.

```
with open("yevgeniy-onegin.txt", "rb") as in_file:
    text = in_file.read()
print(type(text))
print(text[5])
```

Получим:

```
<class 'bytes'>
181
```

Действительно, байт с индексом 5 имеет значение $b5_{16}=181$.

Рассмотрим чтение в байтовом режиме чуть подробнее на примере файла, содержащего строки на русском и английском языках и цифры.

Файл data.bin:

```
раз
two
345
```

Откроем файл в байтовом режиме, прочитаем и выведем поток байт:

```
f = open("data.bin", "rb")
text = f.read()
print(text)  # b'\xd1\x80\xd0\xb0\xd0\xb7\ntwo\n345\n'
f.close()
```

Префикс b указывает, что дальше идет поток байт.
\x – означает, что далее идут два символа, определяющие число в 16-м виде. Например \xd0. В нашем случае это коды символов кириллицы.
\n – это символ перевода строки, в Linux это символ с кодом 10.

Полученная строка – не что иное, как массив байт, к элементам которого можно обращаться по индексу. А еще можно преобразовать в список чисел – числовых кодов символов в кодировочной таблице:

```
f = open("data.bin", "rb")
text = f.read()
print(*map(str, text))  # 209 128 208 176 208 183 10 116 119 111 10
51 52 53 10
f.close()
```

Или так:

```
f = open("data.bin", "rb")
for i in f.read():
    print(i)
f.close()
```

Можно увидеть, что для кодирования каждого символа кириллицы потребовалось 2 байта.

Байтовую строку можно декодировать с помощью метода .decode(), а закодировать строку в нужной кодировке с помощью метода .encode():

```
line = "Эники-бэники ели вареники"
encoding_line = line.encode("utf8")
print(encoding_line)
"""
b'\xd0\xad\xd0\xbd\xd0\xb8\xd0\xba\xd0\xb8-
\xd0\xb1\xd1\x8d\xd0\xbd\xd0\xb8\xd0\xba\xd0\xb8
\xd0\xb5\xd0\xbb\xd0\xb8
\xd0\xb2\xd0\xb0\xd1\x80\xd0\xb5\xd0\xbd\xd0\xb8\xd0\xba\xd0\xb8'
"""
decoding_line = encoding_line.decode()
print(decoding_line)  # Эники-бэники ели вареники
```

Запись в файл

Для записи в файл также есть два режима: w (если файл существовал, его содержимое будет потеряно) и a — запись идет в конец файла.

После выбора режима можно также ввести и символ "+". Посмотрите в документации по функции open, что означает такая конструкция.

Один из способов записать информацию в файл — метод write. Если мы хотим сделать запись в середину файла, должны сначала переместить курсор на место предполагаемой записи (метод seek), а уже потом записывать (метод write).

```
f = open("files/example.txt", 'w')
# выводим количество записанных символов
print(f.write('123\n456'))
# выводим позицию указателя
print(f.seek(3))
# записываем 34352, начиная с того места, где стоит указатель
print(f.write('34352'))
f.close()
f = open("files/example.txt", 'r')
print(f.read())
f.close()
7
3
5
12334352
```

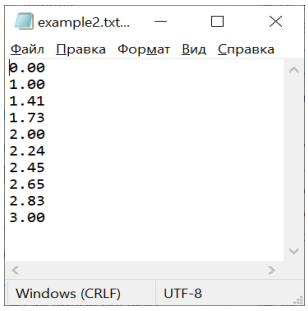
Построчно пройдитесь по приведенной программе, чтобы лучше понять, как она работает. Заметьте, что если мы используем seek, то данные при записи все равно

будут стерты. Этот процесс похож на рисование фломастером: рисуя поверх, вы закрашиваете старый рисунок.

Второй способ записи в файл — стандартная функция print. Для этого применяется именованный параметр file, в который передается открытый на запись файловый объект.

Например:

```
out_file = open("files/example2.txt", 'w')
for i in range(10):
    print(f"{i ** 0.5:.02f}", file=out_file)
out_file.close()
```



Для записи данных в бинарный файл в функцию `write()` надо передать переменную типа bytes. Например, так:

```
f = open("files/ex_bin.dt", 'wb')
data = [1, 2, 3, 4, 5]
f.write(bytes(data))
f.close()
```

Обратите внимание: в примере мы преобразовываем список в поток байт, чтобы потом записать его в файл.

Поэкспериментируем с файлом data.bin, который мы рассматривали выше. Мы получили числовые коды символов, попробуем их снова перевести в поток байт и записать в новый файл:

```
result = [209, 128, 208, 176, 208, 183, 10, 116, 119, 111, 10, 51,
52, 53, 10]
f = open("result.bin", "wb")
f.write(bytes(result))
f.close()
```

Обратите внимание, что функции bytes() мы передали список. Все получилось, в файл result.bin записалось:

раз
two
345