**Файлы**

Программы, которые мы писали до сих пор, требовали повторного ввода данных при каждом запуске, потому что, как только программа заканчивает свою работу, данные для переменных исчезают из оперативной памяти. Если их нужно сберечь между выполнениями программы, требуется запись. Данные записываются в файл, обычно сохраняющийся на диске компьютера.

Когда в программе используется файл, как правило требуется выполнить три шага:

1. **Открыть файл**. В процессе открытия файла создается связь между файлом и программой. Открытие файла вывода обычно создает файл на диске и позволяет программе записать в него данные. Открытие файла ввода позволяет программе прочитать данные из файла.

**2.** Обработать файл. На этом шаге данные либо записываются в файл (если это файл вывода), либо считываются из файла (если это файл ввода).

**3.** Закрыть файл. После использования файла программой его нужно закрыть, тем самым освободить ресурс и разорвать связь файла с программой.

Существует два типа файлов: **текстовые** и **двоичные (бинарные)**. Текстовый файл содержит данные, которые были закодированы в виде текста при помощи такой схемы кодирования, как ASCII или Юникод. Даже если файл содержит числа, эти числа в файле хранятся как набор символов. В результате файл можно открыть и просмотреть в текстовом редакторе, таком как Блокнот.

Двоичный файл содержит данные, которые не были преобразованы в текст. Данные, которые помещены в двоичный файл, предназначены только для чтения программой, и такой файл невозможно просмотреть в текстовом редакторе.

Мы будем рассматривать только текстовые файлы.

**Методы доступа к файлам**

Большинство языков программирования обеспечивает два способа получения доступа к данным в файле:

**последовательный,**

**прямой**или **произвольный**.

Последовательный, как при проигрывании кассет с записью на пленке, выдает порции информации одну за другой. При работе с таким файлом не получится перескочить сразу к нужной части данных, сначала придется прочитать все предыдущие.

При работе с файлом с прямым или произвольным доступом можно перескочить непосредственно к любой порции данных, не читая предыдущие. Как проигрыватель компакт-дисков или МР3-плеер перескакивает сразу к любой песне.

**Открытие файла**

Для открытия файлов в Python существует функция open(). Она создает файловый объект и связывает его с файлом на диске. Общий формат применения функции open():

файловая\_переменная = open(имя\_файла, режим\_доступа)

Здесь

файловая переменная – имя переменной, которая ссылается на файловый объект;

имя\_файла – строковый литерал, задающий имя файла;

режим\_доступа – строковый литерал, задающий режим доступа (чтение, запись, и т.д.), в котором файл будет открыт.

Ниже представлены строковые литералы (символы), используемые для задания режима доступа.

| **Стр. литерал** | **Режим** | **Описание** |
| --- | --- | --- |
| 'r' | Read (чтение) | Открыть файл только для чтения. Такой файл не может быть изменен. |
| 'w' | Write (запись) | Открыть файл для записи. Если файл уже существует, стереть его содержимое. Если файл не существует, он будет создан. |
| 'a' | Append (добавление) | Открыть файл для записи. Данные будут добавлены в конец файла. Если файл не существует, он будет создан. |
| 'r+' | Read + Write | Открыть файл для чтения и записи. В этом режиме происходит частичная перезапись содержимого файла. |
| 'x' | Create (создание) | Создать новый файл. Если файл существует, произойдет ошибка. |

Предположим, файл students.txt содержит данные о студентах, и мы хотим открыть его для чтения.

Это можно сделать с помощью строки кода:

student\_file = **open**('students.txt', 'r')

По умолчанию режим доступа (второй аргумент функции open()) определен для чтения (литерал 'r'), поэтому файл students.txt можно открыть для чтения так:

student\_file = **open**('students.txt')

При работе с текстом на русском языке нужно указать кодировку, для этого служит параметр encoding:

file = **open**('info.txt', 'r', encoding='utf-8')

Предположим, надо создать файл с именем sales.txt и записать в него данные о продажах. Открыть файл на запись можно с помощью строки кода:

sales\_file = **open**('sales.txt', 'w')

**Чтение содержимого файла**

Для чтения содержимого открытого для чтения файла используются три файловых метода:

* read() – читает все содержимое файла;
* readline() – читает одну строку из файла;
* readlines() – читает все содержимое файла и возвращает список строк.

**Запись данных в файлы**

Для записи используются два файловых метода:

* write() – записывает переданную строку в файл;
* writelines() – записывает переданный список строк в файл.

Для записи данных в файл можно также использовать встроенную функцию **print**(). Для этого нужно передать ей еще один именованный аргумент **file,** указывающий на открытый файл. При этом функция **print**() автоматически добавляет переход на новую строку.

output = **open**('philosophers.txt', 'w', encoding='utf-8')

print('Джoн Локк', file=output)

**Закрытие файлов**

После окончания работы с файлом его необходимо закрыть. Для этого есть несколько причин:

* если файл изменялся, это позволит корректно его сохранить;
* если открытый файл потребуется другим программам, ваша программа может его блокировать;
* не стоит держать в памяти лишние, уже не нужные, данные;
* удалить открытый кем-то файл проблематично.

Для закрытия файла используется файловый метод close()

file.close()

**Менеджер контекста**

Как уже сказано, важно своевременно закрывать файлы с помощью метода close(). Закрытие файлов вручную, а также отдача закрытия на откуп среде исполнения, обладают существенным недостатком: если между открытием файла и его закрытием произойдёт ошибка, в лучшем случае файл окажется открыт слишком долго, а в худшем случае часть данных не сохранится.

Хочется иметь возможность автоматически закрывать файл сразу после окончания работы с ним и осуществлять закрытие даже при возникновении ошибки. Файловые объекты уже умеют работать в таком режиме, но для этого их нужно использовать как менеджеры контекста.

Менеджер контекста — объект, реализующий одноименный протокол. Объекты, реализующие этот протокол, позволяют использовать следующий специальный синтаксис:

**with** object **as** name:

# Здесь нам доступен ресурс name.

# Это тело with-блока.

# А здесь ресурс name уже освобождён, даже если в теле with-блока произошла ошибка.

Весь код в теле with-блока работает "в контексте". Чаще всего контекст подразумевает выделение некоего ресурса, например, файла. По выходу из контекста ресурс автоматически освобождается, даже если при выполнении блока возникло исключение.

Как только закончится код, оформленный с отступами в with (аналогичные отступы в циклах или функциях), это будет означать, что контекст закончился, и Python автоматически закроет файл.

**with** open('languages.txt', 'r', encoding='utf-8') **as** file:

**for** line **in** file:

print(line) # автоматическое закрытие файла

print('Файл закрыт')