Файлы

В случае, когда программа должна принимать в обработку и/или выдавать большой объем данных используют файлы.

С точки зрения программы файл можно рассматривать, как последовательность заранее неизвестного количества элементов данных, располагающихся за пределами оперативной памяти компьютера, например, на его жестком диске.

Поскольку программа может непосредственно оперировать только с объектами в оперативной памяти компьютера, для взаимодействия с файлами она вынуждена обращаться к операционной системе. Для связи файлов операционной системы с программой используется объект файлового типа.

Традиционный процесс работы с файлами строится по следующей схеме:

- определяется переменная файлового типа;
- эта переменная связывается с реальным файлом;
- файл открывается с целью его чтения, либо записи или дозаписи в него;
- выполняются операции по обмену данными с файлом;
- файл закрывается; при этом операционная система фиксирует сделанные в нем изменения.

Обычно файлы делят на текстовые и байтовые (бинарные).

Текстовые файлы рассматриваются как содержащие символьные данные, строки. Данные в текстовом файле хранятся в символьном виде, поэтому такой файл может обрабатываться в текстовом редакторе. Программа рассматривает текстовый файл как последовательность физических записей (строк), в которой к записи с номером k можно обратиться, только просмотрев предыдущие k-1 записи, поэтому такой доступ к данным называется последовательным.

При записи или чтении в/из текстового файла всегда передается именно строка, поэтому, если нужно записать числа, данные других типов, то их предварительно нужно конвертировать в строку.

Байтовые файлы рассматриваются как поток байтов. Побайтово считываются, например, файлы изображений.

Работа с бинарными файлами несколько сложнее. Нередко их обрабатывают с помощью специальных модулей Python (pickle, struct).

Открытие файла выполняется с помощью, встроенной в Python функции open().

У функции open много параметров. Пока важны 3 аргумента: первый, это имя файла. Путь к файлу может быть относительным или абсолютным. Второй аргумент, это режим, в котором мы будем открывать файл.

Обычно используются режимы чтения ('r') и записи ('w'). Если файл открыт в режиме чтения, то запись в него невозможна. Можно только считывать данные из него. Если файл открыт в режиме записи, то в него можно только записывать данные, считывать нельзя.

Если файл открывается в режиме 'w', то все данные, которые в нем были до этого, стираются. Файл становится пустым. Если не надо удалять существующие в файле данные, тогда следует использовать вместо режима записи, режим дозаписи ('a').

Если файл отсутствует, то открытие его в режиме 'w' создаст новый файл. Бывают ситуации, когда надо гарантировано создать новый файл, избежав случайной перезаписи данных существующего. В этом случае вместо режима 'w' используется режим 'x'. В нем

всегда создается новый файл для записи. Если указано имя существующего файла, то будет выброшено исключение. Потери данных в уже имеющемся файле не произойдет.

Если при вызове open() второй аргумент не указан, то файл открывается в режиме чтения как текстовый файл. Чтобы открыть файл как байтовый, дополнительно к букве режима чтения/записи добавляется символ 'b'. Буква 't' обозначает текстовый файл. Поскольку это тип файла по умолчанию, то обычно ее не указывают.

Нельзя указывать только тип файла, то есть open("имя_файла", 'b') есть ошибка, даже если файл открывается на чтение. Правильно – open("имя_файла", 'rb'). Только текстовые файлы можно открыть командой open("имя_файла"), потому что и 'r' и 't' подразумеваются по- молчанию.

Режимы могут быть объединены, то есть, к примеру, 'rb' - чтение в двоичном режиме. По умолчанию режим равен 'rt'.

И последний аргумент, encoding, нужен только в текстовом режиме чтения файла. Этот аргумент задает кодировку.

Функция open() возвращает объект файлового типа. Его надо либо сразу связать с переменной (объект файлового типа), чтобы не потерять, либо сразу прочитать.

С помощью файлового метода read() можно прочитать файл целиком или только определенное количество байт. Текстовый файл можно читать по байтам.

Раздаточный материал № 110

```
f1 = open('new_file_1.txt')
print('Читаем и выводим на экран первые 10 байт или символов')
print(f1.read(10))
f1.close()

f1 = open('new_file_1.txt')
print('Читаем и выводим на экран весь файл')
print(f1.read())
f1.close()

f1 = open('new_file_1.txt')
print('Читаем и выводим на экран весь файл с помощью for')
for line in f1:
    print(line, end='')
print(type(f1.read()))
f1.close()
```

Пояснения:

1. Объект файлового типа, т.е. переменная `f1`, после выполнения операции считывания становится пустым, поэтому файл необходимо открывать еще раз, предварительно закрыв его после очередной операции.
2. Объект файлового типа, т.е. переменная `f1`, относится к итераторам, т.е. возможно последовательное извлечение элементов, т.е. сразу в цикле без использования методов чтения.
3. Данные, прочитанные из файла, всегда возвращаются в сценарий в виде строки. Поэтому строку всегда нужно преобразовывать в другой тип, если строка неуместна.
4. Параметр `encoding="utf-8"` задает кодировку читаемого файла.

Раздаточный материал № 111

Метод	Описание
readline()	Чтение файла построчно
readlines()	Считывает сразу все строки и создает список

Запись в файл выполняется с помощью методов write() и writelines(). Во второй можно передать структуру данных. Метод write() возвращает количество записанных символов. Методы write() и writelines() автоматически не ставят символ переноса строки, и это программисту нужно контролировать самостоятельно.

Раздаточный материал № 112

```
print('Запишем в файл структуру данных - список')
l = [' tree', ' four']
f2 = open('data.txt', 'w')
f2.write('one')
f2.write(' two')
f2.writelines(l)
f2.close()
f2 = open('data.txt')
print(f2.read())
print(type(f2.read())) # получаем тип - строка
f2.close()
```

После того как работа с файлом закончена, важно не забывать его закрыть, чтобы освободить место в памяти. Если файл открывается в заголовке цикла, то интерпретатор его закрывает при завершении работы цикла или через какое-то время. Закрывается файл с помощью файлового метода close(). Свойство файлового объекта closed позволяет проверить закрыт ли файл.

Раздаточный материал № 113

```
# содержимое файла data_2.txt:
# зима
# весна
# лето
# осень

nums = []
for i in open('data_2.txt', encoding='UTF-8'):
    nums.append(i[:-1])
print(nums)
print('получаем тип', type(nums))
```

Результат
['зима', 'весна', 'лето', 'осень']

Существует удобный способ записи в файл - использование функции print(). Если передать в необязательный аргумент file объект файлового типа, то поток вывода функции print() перенаправляется из консоли в файл. Преимущество такого подхода заключается в том, что в print() можно передавать не обязательно строковые аргументы — при необходимости функция сама их преобразует к строковому типу.

Фронтальный опрос:

- ✓ Понятие файла и объекта файлового типа.
- ✓ Последовательность действий при работе с файлами.
- ✓ Понятие текстового файла.
- ✓ Понятие байтового файла.
- ✓ Назначение функции open().
- ✓ Назначение файлового метода read().
- ✓ Методы для записи в файл.