Лекция №16.

Тема: **Файловый тип данных. Часть 1.**

Особенности файлового типа. Файловый объект. Стандартные операции над файлами. Обработка ошибок ввода-вывода. Специальные операции с файлами, каталогами, логическими дисками. Сериализация объектов.

Ключевые слова: файл, файловый объект, дескриптор файла, поток, имя файла, атрибут файла, режим работы с файлом.

Keywords: file, file object, file descriptor, stream, filename, file attribute, file mode.

1 Особенности файлового типа. Файловый объект

Файловый тип данных вводится для обработки информации, которая хранится в долговременной памяти (файл), и для выполнения операций ввода-вывода на устройства (стандартный ввод/вывод, потоковые сокеты и др).

Файловый объект (поток) представляет файл в программе на языке Python.

Особенности файлового типа:

1. Файл в языке Python понимается как упорядоченная последовательность компонентов – байт или Unicode-символов.
2. Все компоненты файла считаются пронумерованными. Начальный компонент имеет нулевой номер. Однако количество компонентов заранее не оговаривается.
3. С файловым объектом связано понятие текущего указателя файла – адреса текущего компонента. Если операции над файлами выполняются покомпонентно, то в действии участвует тот компонент, который обозначен текущим указателем. При создании файлового объекта неявно создается скрытый объект, где хранится текущий указатель.
4. При открытии файла операционная система устанавливает каждому открываемому файлу обработчик файла с определённым номером. Этот обработчик осуществляет операции обмена данными через буфер ввода-вывода. Номер обработчика – дескриптор файла (см. https://en.wikipedia.org/wiki/File_descriptor).
5. В Python отсутствуют операторы для работы с файлами. Обработка файлов выполняется с помощью функций, включенных в модуль io.
6. Достоинства работы с файлами: доступ к большим объёмам постоянной памяти, возможность многократного считывания информации их долговременной памяти, возможность вывода объёмных результатов работы программы с последующим изучением.
7. Недостатки работы с долговременной памятью: выполнение требует больших временных затрат, чем выполнение аналогичных операций в оперативной памяти, при потере входных файлов необходимо их создание.
8. Доступ к файлам может быть последовательным (текущий указатель перемещается последовательно) или прямым (текущий указатель перемещается произвольно).
9. Существует 3 категории файловых объектов в языке Python:

- текстовые;
- двоичные;
- буферизированные двоичные.

2 Стандартные операции над файлами

Стандартные операции над файлами (модуль io) можно разбить на 4 группы:

1. Установочные и завершающие операции.

open(...) – открывает файл и возвращает поток, связанный с этим файлом.

Формат функции:

```
open(file, mode='r', buffering=-1, encoding=None, errors=None, newline=None, closefd=True) → file object
```

Рассмотрим основные параметры функции:

- file – абсолютный или относительный путь к файлу (правила определения имени файла для различных платформ см. в https://en.wikipedia.org/wiki/Filename).
- mode – необязательный литерал, задающий режим работы с файлом (см. Примечание 1).
- buffering – необязательное целочисленное значение, задающее политику буферизации данных (см. Примечание 2),
- encoding – необязательный литерал, задающий кодировку текстового файла (по умолчанию используется системная кодировка),

Описание остальных параметров функции open может быть получено с помощью интерактивной подсказки в IDLE

```
>>> help(open)
```

Примечание 1. Допустимые режимы работы с файлами представлены в табл. 1.

Таблица 1 – Режимы работы с файлами

Режим	Описание	Возможные значения mode
t (text)	Текстовый режим (по умолчанию). При работе с файлом в текстовом режиме, файловые методы принимают и возвращают объекты типа str.	-
b (binary)	Двоичный режим. При работе с файлом в двоичном режиме, файловые методы принимают и возвращают объекты типа bytes.	-
r (read)	Режим чтения данных (по умолчанию). При отсутствии файла возбуждается исключение IOError.	r, rt, rb
w (write)	Режим записи данных. В случае отсутствия файла – создает файл, в случае присутствия – перезаписывает файл.	w, wt, wb
a (append)	Режим добавления данных. В случае отсутствия файла – создает файл.	a, at, ab
x (create)	Режим создания файла для записи. При отсутствии файла возбуждается исключение IOError.	x, xt, xb
+	Режим редактирования файла (чтение и запись). После открытия файла указатель устанавливается на начало файла.	r+, rt+, rb+ w+, wt+, wb+ x+, xt+, xb+

Примечание 2. Буферизация данных позволяет повысить производительность при работе с файлом. Возможные значения необязательного параметра buffering представлены в табл. 2.

Таблица 2 – Типы политики буферизации при работе с файлами

Значение buffering	Описание
0	Отсутствие буферизации данных (доступно только при работе с двоичными файлами)
1	Построчная буферизация данных (доступно только при работе с текстовыми файлами)
>1	Полная буферизация, значение определяет размер буфера
-1	Значение по умолчанию: размер буфера устанавливается исходя из допустимого размера буфера устройства, с которым ассоциирован файл, и размера системного буфера, который обычно равен 4096 или 8192 байт; если файл связан с терминалом, то выполняется построчная буферизация.

Функция open(…) возвращает потоковый объект (file object), с помощью которого производится дальнейшая работа с файлом. Тип объекта зависит от выбранного режима работы с файлом. В таблице 3 представлены возможные типы потоковых объектов.

Таблица 3 – Типы возвращаемых объектов функцией open(...)

Режим	Тип потокового объекта
Режим чтения / записи / добавления / редактирования данных в текстовом формате	io.TextIOWrapper
Режим чтения двоичный данных	io.BufferedReader
Режим записи / добавления двоичных данных	io.BufferedWriter
Режим редактирования двоичных данных	io.BufferedRandom

close( ) – сбрасывает содержимое буфера на диск и закрывает файл.

Основные причины, вызывающие необходимость явного закрытия файла:

1. Буферизация данных при записи в файл может привести к неожиданным эффектам и возникновению ошибок.
2. Операционная система ограничивает количество одновременно открытых файлов.
3. Одновременный доступ к файлу на чтение и на запись требует синхронизации файловых операций. Буферизация данных при записи в файл может привести к тому, что запись уже произошла, а данных в файле еще нет.

flush( ) – сбрасывает содержимое буфера на диск (возможна блокировка файла на чтение).

2. Операции ввода-вывода.

write(data) – записывает данные в файл.

В двоичном режиме функция write() в качестве параметра принимает объект типа bytes или bytearray, в текстовом режиме – объект типа str. Функция возвращает количество записанных байт (для двоичного режима) или символов (для текстового режима).

writelines(lines) – записывает последовательность в файл.

В двоичном режиме функция writelines() в качестве параметра принимает последовательность объектов типа bytes или bytearray, в текстовом режиме – последовательность объектов типа str.

read(size=-1) – читает данные из файла.

Возвращает size символов (для текстового режима) или size байт (для двоичного режима). Если параметр size не указан или равен -1, то вызывается функция readall( ).

readall() – читает все данные из файла.

readline(size=-1) – читает данные из файла построчно.

Возвращает объект типа str (для текстового режима) или объект типа bytearray (для двоичного режима). Для двоичного режима допускается только один разделитель строк – b'\n'. Если указан параметр size и не равен -1, то считывание будет выполняться до тех пор, пока не встретится символ новой строки, символ конца файла или из файла не будет прочитано указанное количество символов / байт.

readlines( ) – читает все данные из файла в список.

Возвращает список, каждый элемент которого является объектом типа str (для текстового режима) или объектом bytearray (для двоичного режима).

3. Операции перемещения по файлу.

tell( ) – определяет позицию указателя файла.

seek(offset, whence=SEEK_SET) – задаёт позицию указателя файла.

Рассмотрим параметры функции:

- offset – смещение относительно whence.
- whence – смещение относительно трех возможных позиций: SEEK_SET (0) – начало файла (значение по умолчанию); SEEK_CUR (1) – текущая позиция указателя; SEEK_END (2) – конец файла.

truncate (size=None) – усекает файл до указанного количества символов / байт или до текущей позиции, если параметр size не определен.

4. Специальные операции, включающие операции с файлами, каталогами, логическими дисками (будут описаны позже).

3 Примеры использования стандартных файловых функций

Рассмотрим основные стандартные файловые функции на простых примерах.

Пример №1. Работа с базовыми файловыми функциями.

```
>>> # Текстовый файл открывается в режиме записи. Содержимое файла очищается.
>>> f = open('test.txt', 'w')
>>> f.write('Это первая строка файла\n')
24
>>> f.close()
>>> # Текстовый файл открывается в режиме добавления данных. Содержимое файла не очищается, к нему добавляются новые компоненты.
>>> f = open('test.txt', 'a')
>>> f.write('Это вторая строка файла\n')
24
>>> f.close()
>>> # Текстовый файл открывается в режиме чтения данных.
>>> f = open('test.txt', 'r')
>>> f.read(24)
'Это первая строка файла\n'
>>> f.read()
'Это вторая строка файла\n'
```

Содержимое файла test.txt:

```
Это первая строка файла
Это вторая строка файла
```

Пример №2. Организация произвольного доступа к файлу.

```
>>> import random as rnd
>>> # Двоичный файл открывается в режиме записи. Содержимое файла очищается.
>>> f = open('test.txt', 'wb')
>>> N = 25
>>> A = bytearray([rnd.randint(0, 256) for i in range(N)])
>>> # Запись двоичных данных в файл.
>>> f.write(A)
25
>>> # Перемещение указателя файла на 10 позицию.
>>> f.seek(10)
10
>>> # Запись двоичных данных в файл.
>>> f.write(b' Python ')
8
>>> f.close()
>>> # Двоичный файл открывается в режиме чтения.
>>> f = open('test.txt', 'rb')
>>> # Чтение двоичных данных из файла.
>>> f.read()
b'\xc9dB\x1f\xd9/\x02\x1ff\x1e Python \xba3\xaa\x1c/\xd6\xda'
>>> f.close()
```

Пример №3. Итерационная обработка данных в файле.

```
>>> # Текстовый файл открывается в режиме записи. Содержимое
файла очищается.
>>> f = open('test.txt', 'w')
>>> str_list = []
>>> str_list.append('The first string' + '\n')
>>> str_list.append('The second string' + '\n')
>>> str_list.append('The third string' + '\n')
>>> str_list.append('The fourth string' + '\n')
>>> str_list.append('The fifth string' + '\n')
>>> # Построчная запись данных в файл.
>>> f.writelines(str_list)
>>> f.close()
>>> # Текстовый файл открывается в режиме чтения данных.
>>> f = open('test.txt', 'r')
>>> # Построчное чтение данных из файла.
>>> for line in f:
     print(line)

The first string

The second string

The third string

The fourth string

The fifth string

>>> f.close()
```