Тема 10. Работа с текстовыми файлами

Возможность работы с файлами является неотъемлемой частью любого языка программирования. Основными операциями являются: открытие, закрытие, запись и чтение.

10.1. Открытие файла. Метод open

Для начала работы с файлом его нужно открыть. Для этого используется встроенный метод **open()**.

Общий вид выглядит следующим образом:

*f=**open**(‘<путь к файлу (имя)>’, ‘<режим’[,encoding]>)*

Путь к файлу может быть как относительным, так и абсолютным. Если мы укажем только имя файла с расширением, то подразумевается, что файл должен быть в той же папке, что и сама программа. Режимов открытия файла существует несколько, рассмотрим их в виде таблицы.

Режимы открытия файла

Режим	Описание
'r'	Открытие для чтения (значение по умолчанию).
'w'	Открытие для записи. Если файл существует, то его содержимое удаляется, иначе создается новый.
'x'	Открытие для записи, если файла не существует. Иначе вызывается исключение.
'a'	Открытие для продолжения записи. Информация добавляется в конец файла. Если файла нет, то он создастся автоматически.
'b'	Открытие в двоичном режиме.
't'	Открытие в текстовом режиме (значение по умолчанию).
'+'	Открытие на чтение и запись.

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

Последний аргумент **encoding**, нужен только в текстовом режиме чтения файла. Этот аргумент задает кодировку.

Пример:

```
f = open('out.txt','w') # откроет файл для записи, а если его нет, то создаст
f = open('out.txt','a') # откроет файл для продолжения записи
```

10.2. Закрытие файла. Метод close

После работы с файлом в программе его обязательно нужно закрывать, иначе при попытке изменения и сохранения файла другой программой получим сообщение, что файл занят другим приложением, и сохранение выполнить не удастся.

Метод файлового объекта **close()** автоматически закрывает файл, при этом теряется любая несохраненная информация. Работать с файлом (читать, записывать) после этого нельзя.

Python автоматически закрывает файл, если файловый объект, к которому он привязан, присваивается другому файлу.

Пример:

Допустим, у нас есть файловый объект *f*. Чтобы его закрыть, нужно написать

```
f.close()
```

10.3. Запись в файл. Метод write

Метод **write()** записывает любую строку в открытый файл. Важно помнить, что строки в Python могут содержать двоичные данные, а не только текст.

Метод **write()** не добавляет символ переноса строки **'\n'**. Поэтому, если нужен переход на следующую строку, символ переноса строки ставим сами.

При необходимости записать в файл нестроковые данные (допустим числа), нужно обязательно перед записью конвертировать их в строку, иначе интерпретатор выдаст ошибку.

Синтаксис метода write()

f.write(string);

f – наш файловый объект, *string* – строка, записываемая в файл.

Пример:

```
f = open('out.txt', 'w')
f.write('первая строка\n')
f.write('вторая строка\n')
```

Если мы откроем папку с программой, то увидим там файл **out.txt**. Но при его открытии мы обнаружим, что он пустой. Для того, чтобы данные появились в файле, обязательно нужно закрыть этот файл в программе методом **close()**.

```
f = open('out.txt', 'w')
f.write('первая строка\n')
f.write('вторая строка\n')
f.close()
```

Теперь данные в файле появятся, и записаны они будут в 2 строки.

Также можно выводить данные в файл при помощи функции **print**, если передать ей еще один именованный параметр **file**, равный ссылке на открытый файл.

Пример:

```
f = open('out.txt', 'w')
print('первая строка', file=f)
f.close()
```

В отличие от метода **write()**, **print()** не требует перевода данных в строковый тип перед записью в файл. Также метод **print()** автоматически добавляет переносы строк.

10.4. Чтение из файла. Метод read

Так как наша задача разобраться с текстовыми файлами, для чтения из файла мы будем использовать несколько способов.

Способ 1

Подразумевает чтение файла целиком, либо чтение указанного количества байт.

Для его реализации используют метод **read()**. Если скобки пустые, то считывается весь файл с позиции каретки, а если указанно число, то считывается указанное количество байт.

Пример:

Допустим, у нас есть файл **input.txt** со следующим содержимым:

```
privet
python
```

В файле 2 слова, каждое с новой строки.

Выполним код:

```
f = open('input.txt', 'rt')
print(f.read(1))
print(f.read())
```

```
p
rivet
python
```

Обратим внимание на то, что в первый раз был считан один символ, и каретка переместилась тоже на один символ. При втором чтении считалось все, начиная с позиции каретки.

Способ 2

Построчное чтение.

Для реализации построчного чтения используется метод **readline()**.

При каждом вызове метода считывается строка полностью, и указатель переходит на следующую строку.

Допустим, у нас есть файл **input.txt** со следующим содержимым:

```
privet
python
```

Выполним код:

```
f = open('input.txt', 'rt')
print(f.readline())
print(f.readline())
print(f.readline())
f.close()
```

В итоге получим три строки, первые 2 – это строки из файла, а третья – пустая строка. Если строки закончились, программа ошибку не выдаст. Вместо этого будет получать пустые строки:

```
privet

python
```

При выводе на экран появится ненужная пустая строка между нашими выводимыми строками. Это не ошибка. Проблема в том, что каждый раз **print** в конце автоматически ставит переход на новую строку, в то время как в файле тоже есть переводы на новую строку. Чтобы исправить данную ситуацию, зададим **print** следующим образом:

print(s, end='')

Пример вывода 2-х строк без пустой строки между ними:

```
f = open('input.txt', 'rt')
print(f.readline(), end='')
print(f.readline())
f.close()
```

```
privet
python
```

Способ 3

Построчное чтение с использованием цикла.

Для реализации 3-го способа используется цикл **for**. В этом способе не используется метод **read()**.

Работаем с тем же файлом **input.txt**.

Выполним код:

```
f = open('input.txt', 'rt')
for s in f:
    print(s, end='')
```

```
privet
python
```

Способ 4

Автоматическое чтение всего файла с получением списка, состоящего из его строк.

Используем тот же файл с двумя строками:

```
f = open('input.txt','rt')
s = f.readlines()
print(s)
f.close()
```

```
['privet\n', 'python']
```

Обратите внимание, что символ переноса строки **'\n'** остался в конце элемента. Для удаления символов переноса из каждого элемента можно воспользоваться методом обработки строк **rstrip()**.