Тема. Обработка данных из текстовых файлов

1. Базовые способы работы с текстовыми фалами

Текстовый файл состоит из символов. Текстовые редакторы выводят на экран текстовый файл как совокупность строк, если встречают символы переноса строк. Таким образом можно сказать, что текстовый файл - это список строк.

Стандартный способ работы с текстовым файлом – получить доступ к нему через файловую переменную с помощью функции open(). Рассмотрим пример.

program 01.py

```
1 file = open('01.py', 'r')
2 text = file.read()
3 file.close()
4 print(text)
```

Эта программа открывает сама себя как текстовый файл и выводит содержимое на консоль (рис.1):

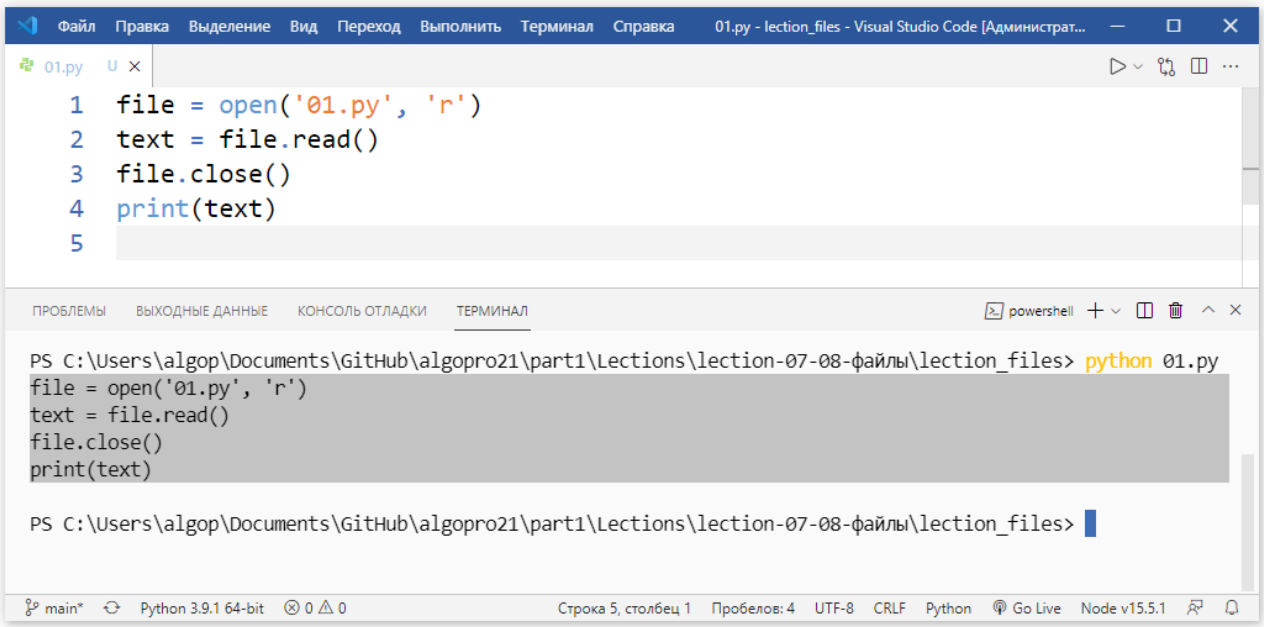

Рис.1. Пример чтения из файла с выводом на экран.

Первым параметром функции open() указывается путь к файлу (абсолютный или относительный). Вторым параметром можно выбрать модификатор доступа к файлу (см. табл. 1). Следует уточнить, что можно указывать имя параметра, например, так: `file = open('01.py', mode='r')`.

В данной программе открывается файл с модификатором «только для чтения» и создаётся файловая переменная file. Это объект, у которого есть

ряд методов для работы с файлом, например, метод read() позволяет прочитать все символы файла сразу – мы их поместили в строковую переменную text. В третьей строке программы был закрыт доступ к файлу, чтобы освободить ресурсы. Затем содержимое строковой переменной file вывели на экран.

Таблица 1.

Список режимов доступа к файлу в Python.

r	Открывает файл только для чтения. Указатель стоит в начале файла.
rb	Открывает файл для чтения в двоичном формате. Указатель стоит в начале файла.
r+	Открывает файл для чтения и записи. Указатель стоит в начале файла.
rb+	Открывает файл для чтения и записи в двоичном формате. Указатель стоит в начале файла.
w	Открывает файл только для записи. Указатель стоит в начале файла. Создает файл с именем имя_файла, если такового не существует.
wb	Открывает файл для записи в двоичном формате. Указатель стоит в начале файла. Создает файл с именем имя_файла, если такового не существует.
w+	Открывает файл для чтения и записи. Указатель стоит в начале файла. Создает файл с именем имя_файла, если такового не существует.
wb+	Открывает файл для чтения и записи в двоичном формате. Указатель стоит в начале файла. Создает файл с именем имя_файла, если такового не существует.
a	Открывает файл для добавления информации в файл. Указатель стоит в конце файла. Создает файл с именем имя_файла, если такового не существует.
ab	Открывает файл для добавления в двоичном формате. Указатель стоит в конце файла. Создает файл с именем имя_файла, если такового не существует.
a+	Открывает файл для добавления и чтения. Указатель стоит в конце файла. Создает файл с именем имя_файла, если такового не существует.
ab+	Открывает файл для добавления и чтения в двоичном формате. Указатель стоит в конце файла. Создает файл с именем имя_файла, если такового не существует.

Если файл, который нужно считать, находится в той же папке, что и сама программа, то путь к файлу можно не писать. Если в той же директории, где лежит программа, есть папка, в которой находятся необходимые для чтения файлы, то можно использовать относительный путь, например, так:

program 02.py

```
1 file = open('dir_data\\data.txt', 'r')
2 text = file.read()
3 file.close()
4 print(text)
```

Здесь «dir_data» это директория с файлами, и мы открываем из неё файл «data.txt». В первой строке пришлось поставить обратный слеш дважды, так как Python использует его как специальный командный символ.

Например, строка "12\n34" содержит 12, потом символ переноса и далее 34 и при выводе этой строки на экран мы получим такой результат:

```
12
34
```

Когда много иерархически вложенных директорий, то использовать двойные обратные слеши может быть неудобно. Тогда можно применить так называемые «сырые» строки – это такие строки, которые читаются «как есть», то есть без управляющих символов. Для этого нужно перед строкой поставить символ 'r':

program 03.py

```
1  file = open(r'dir_data\data.txt', 'r')
2  text = file.read()
3  file.close()
4  print(text)
```

Запись в текстовый файл осуществляется аналогичным образом, только модификатор доступа к файлу следует выбрать «w». Например, запишем стихотворение, состоящее из нескольких строк в текстовый файл «Bunin.txt»:

program 04.py

```
1  text = '''Ночь, улица, фонарь, аптека,
2  Бессмысленный и тусклый свет.
3  Живи еще хоть четверть века –
4  Все будет так. Исхода нет.
5
6  Умрешь – начнешь опять сначала
7  И повторится все, как встарь:
8  Ночь, ледяная рябь канала,
9  Аптека, улица, фонарь.
10 '''
11
12 fileWrite = open('Bunin.txt', 'w', encoding='utf-8')
13
14 fileWrite.write(text)
```

Обратите внимание на способ создания многострочного текста в программном коде – нужно просто взять весь текст с символами переноса строки в тройные апострофы (или кавычки).

В данной программе файл открывается для записи модификатором «w», если уже был такой файл, то всё его содержимое стирается, затем все символы из строки text, включая и символы переноса, записываются в файл. Обратите внимание, что в качестве третьего аргумента можно указать кодировку символов. Мы указали utf-8 для поддержки кириллицы. После запуска этой программы в текущей папке будет создан текстовый файл с этим стихотворением.

Теперь можно исследовать другие способы работы с файлом. Давайте из этого файла считаем первую строку:

program 05.py

```
1  with open('Bunin.txt', 'r', encoding='utf-8') as file:
2      line = file.readline()
3
4  print(line)
```

Обратите внимание, что файл можно открывать с помощью оператора with. Тогда в самом конце работы с файлом можно его не закрывать - это произойдёт автоматически при выходе из оператора with. Во второй строке программного кода мы используем метод readline() – он позволяет считать из файла только одну строку. Если запустить эту программу, то на экран будет выведена только первая строка стихотворения.

Давайте организуем цикл while, чтобы считать все строки из файла:

program 06.py

```
1  file = open('Bunin.txt', 'r', encoding="utf-8")
2
3  while True:
4      line = file.readline()
5      if not line:
6          break
7      print(line)  # выводит с двумя переносами
8
9  file.close()
```

В данной программе цикл работает до тех пор, пока считываются строки. Обратите внимание, что строки считываются из файла вместе с их завершающим символом – символом переноса, поэтому, когда выполняется функция `print(line)`, то каждая выводится строка с двумя переносами (сама функция print() тоже добавляет символ переноса):

Пример вывода

```
Ночь, улица, фонарь, аптека,

Бессмысленный и тусклый свет.

Живи еще хоть четверть века –

Все будет так. Исхода нет.



Умрешь – начнешь опять сначала

И повторится все, как встарь:

Ночь, ледяная рябь канала,

Аптека, улица, фонарь.
```

Чтобы исправить это положение, можно удалять последний символ переноса в каждой строке или добавить все строки по очереди в список строк, а затем объединить их в одну строку и уже её вывести:

program 07.py

```
1  file = open('Bunin.txt', 'r', encoding="utf-8")
2
3  lines = []  # создаём пустой список
4  while True:
5      line = file.readline()
6      if not line:
7          break
8      lines.append(line)  # добавляем строку в список
9
10 print(''.join(lines))  # объединяем список в строку
11
12 file.close()
```

Теперь уже вывод содержимого файла будет без лишних пустых строк:

Пример вывода

```
Ночь, улица, фонарь, аптека,
Бессмысленный и тусклый свет.
Живи еще хоть четверть века –
Все будет так. Исхода нет.

Умрешь – начнешь опять сначала
И повторится все, как встарь:
Ночь, ледяная рябь канала,
Аптека, улица, фонарь.
```

Предусмотрен и штатный способ чтения всех строк файла в список с помощью метода readlines():

program 08.py

```
1  file = open('Bunin.txt', 'r', encoding="utf-8")
2  lines = file.readlines()
3  file.close()
4
5  print(''.join(lines))
```

Если требуется добавить строки к существующему файлу, то используется модификатор доступа «a»: ():

program 09.py

```
1  file = open('Bunin.txt', 'a', encoding='utf-8')
2  file.write('\nИван Бунин\n')
```

Апробируйте работу этой программы и проверьте изменения, произошедшие с файлом.

2. Способы обработки данных из текстовых фалов

Предположим есть файл numbers.txt с целочисленными значениями

numbers.txt

```
12 33 100
20 30 40
-99 80 0
```

Найдём сумму чисел из первой строки, для чего считаем первую строку, затем методом split() разделим её на список строк по пробелу, потом запустим цикл и просуммируем все числа из первой строки:

program 10.py

```
1  file = open('numbers.txt', 'r')
2  line = file.readline()
3  file.close()
4
5  lst = line.split()
6  summa = 0
7  for item in lst:
8      summa += int(item)
9  print(summa)
```

Используя функцию map() эту же самую задачу можно реализовать несколько короче:

program 11.py

```
1  file = open('numbers.txt', 'r')
2  line = file.readline()
3  file.close()
4
5  numbers = list(map(int, line.split()))
6  print(sum(numbers))
```

Или ещё короче:

program 12.py

```
1  with open('numbers.txt') as file:
2      print(sum(map(int, file.readline().split())))
```

Обратите внимание, модификатор доступа «r» можно вообще не указывать, так как файл по умолчанию и так открывается в режиме «только для чтения».

Давайте теперь попробуем вывести на экран числа из последней строки файла, отсортированные в обратном порядке:

program 13.py

```
1  with open('numbers.txt') as file:
2      lines = file.readlines()
3
```

```
lst = list(map(int, lines[-1].split()))

print(*sorted(lst, reverse=True))
```

В данной программе мы считываем все строки файла в список lines. Затем берём только последнюю строку lines[-1], её переводим в список чисел функцией map и сортируем его в обратном порядке с помощью модификатора reverse. Символ звёздочка используется в последней строке программного кода в функции print чтобы вывести элементы списка без скобок и запятых через пробел.

Давайте попробуем создать файл по некоторому шаблону. Пусть требуется сделать файл, в котором построчно выводятся данные в два столбика – в первом id участника соревнований, а во втором возраст участника. Столбики нужно разделить символом табуляции.

На начальном этапе будем выводить данные на экран, позже, после отладки программы выведем их в файл.

Данные по возрасту будем генерировать случайным образом, а сам список будем создавать с помощью генератора списков:

program 14.py

```
from random import randint

count = 10  # всего участников
age_min = 10  # от
age_max = 99  # до
ages = [randint(age_min, age_max) for _ in range(count)]

print("id\tage")
for i, item in enumerate(ages):
    print(f"{i+1}\t{item}")
```

Пример вывода

```
id      age
1       15
2       34
3       36
4       23
5       48
6       49
7       17
8       29
9       68
10      81
```

Можно данные про каждого участника хранить в одном объекте, например, в кортеже. Пока у нас будет маленький кортеж, состоящий только из

двух полей - id и age, но программа будет работать в точности, как и предыдущая:

program 15.py

```
from random import randint

count = 10  # всего участников
age_min = 10  # от
age_max = 99  # до
members = [(i+1, randint(age_min, age_max)) for i in range(count)]

print("id\tage")
for item in members:
    print(f"{item[0]}\t{item[1]}")
```

Однако преимущество хранения данных про каждого отдельного участника в одном объекте состоит в том, что можно будет с каждым объектом работать по отдельности, изменять какие-то данные объектов, искать объекты по id, фильтровать или сортировать объекты по какому-нибудь признаку.

Давайте попробуем выбрать участников старше 50 лет и сохранить их в файл older50.txt:

program 16.py

```
from random import randint


def gen_members(count=10):  # параметр по умолчанию
    age_min, age_max = 10, 99
    return [(i+1, randint(age_min, age_max)) for i in range(count)]


def save_to_file(name_file, lst):
    with open(name_file, 'w', encoding='utf-8') as file:
        file.write(f"id\tage\n")
        for item in lst:
            file.write(f"{item[0]}\t{item[1]}\n")


members = gen_members()  # сгенерировать список участников

print("id\tage")
for item in members:
    print(f"{item[0]}\t{item[1]}")  # вывести на экран

older50 = list(filter(lambda item: item[1] > 50, members))

save_to_file('older50.txt', older50)  # сохранить в файл
```

Всю программу для удобства мы разбили на функции, каждая выполняет одно определённое действие. Для начала генерируем список участников функцией gen_members(), потом выводим результаты на экран для контроля, далее с помощью функции filter() отбираем только тех участников, которые соответствуют установленному анонимной функцией правилу (`lambda item: item[1] > 50`). Анонимная функция проверяет второй параметр каждого объекта, выбираются только те, которые возвращают значение True при проверке условия `item[1] > 50`. Все отобранные участники размещаются в списке older50, который отправляется в функцию save_to_file() для сохранения в файл.

После запуска программы на исполнение можно сравнить полученные результаты на экране и в файле:

Пример вывода на экран

```
id      age
1       85
2       45
3       85
4       15
5       50
6       75
7       22
8       65
9       20
10      82
```

Пример вывода в файл older50.txt

```
id  age
1   85
3   85
6   75
8   65
10  82
```

Теперь давайте попробуем отсортировать участников соревнований по возрасту и сохранить в файл sorted_age.txt:

program 17.py

```
1  from random import randint
2
3
4  def gen_members(count=10):  # параметр по умолчанию
5      age_min, age_max = 10, 99
6      return [(i+1, randint(age_min, age_max)) for i in range(count)]
7
8
9  def save_to_file(name_file, lst):
10     with open(name_file, 'w', encoding='utf-8') as file:
11         file.write("id\tage\n")
12         for item in lst:
```

```
            file.write(f"{item[0]}\t{item[1]}\n")


members = gen_members()  # сгенерировать список участников
sorted_age = sorted(members, key=lambda item: item[1])
save_to_file('sorted.txt', sorted_age)  # сохранить в файл
```

В функции sorted в качестве первого параметра обозначен сортируемый список объектов (кортежей), а в качестве второго параметра анонимная функция, которая указывает по какому именно параметру следует сортировать.

Если вам потребуется сортировать в обратном порядке, то можете использовать ключ reverse:

```
sorted(members, key=lambda item: item[1], reverse=True)
```

Не обязательно в файл сохранять по одной строчке, можно использоваться метод сохранения списка строк:

program 18.py

```
from random import randint


def gen_members(count=10):  # параметр по умолчанию
    age_min, age_max = 10, 99
    return [(i+1, randint(age_min, age_max)) for i in range(count)]


def save_to_file(name_file, lst):
    lines = ["id\tage\n"]
    for item in lst:
        lines += [f"{item[0]}\t{item[1]}\n"]
    with open(name_file, 'w', encoding='utf-8') as file:
        file.writelines(lines)


members = gen_members()  # сгенерировать список участников
sorted_age = sorted(members, key=lambda item: item[1])
save_to_file('sorted.txt', sorted_age)  # сохранить в файл
```

В данном случае мы создаём список строк будущего файла и сразу инициализируем его одним элементом - первой строкой файла:

```
lines = ["id\tage\n"]
```

далее в цикле for добавляем в этот список строки с участниками

```
lines += [f"{item[0]}\t{item[1]}\n"]
```

и уже потом выводим в файл весь список методом writelines().

Задания для самостоятельного исполнения по теме «Текстовые файлы»

Разработать программы на языке Python, все программы и сопутствующие файлы загрузить в свой репозитарий. Оформляйте бизнес-логику программ в виде самостоятельных функций, например, функция проверки на палиндромность может быть такой: is_palindrom(line) и возвращать значение логического типа.

Задание 1.

Пусть дан входной файл numbers.txt (количество строк и столбцов с числами заранее неизвестно):

Пример входного файла numbers.txt

```
12 33 100
20 30 40
-99 80 0
```

Напишите программу, которая найдёт сумму всех чисел из этого файла.

Задание 2.

Напишите программу, которая может генерировать файл numbers.txt с целыми числами из диапазона [-100; 100] по заданным входным параметрам – размер по ширине (количество чисел в строке) и размер по высоте – количество строк с числами в файле.

Пользователь в ходе диалога с программой может задать количество строк и столбцов с числами (прямоугольную матрицу), а программа в ответ сгенерирует все числа и сохранит их в файл numbers.txt.

Пример выходного файла numbers.txt
При заданных параметрах: ширина = 6, высота = 4

```
12 33 -100 15 666 7
20 30 40 34 -12 -89
9 81 0 -4 77 -12
-49 -5 12 -69 57 -33
```

Задание 3.

Напишите программу, которая будет считывать файл numbers.txt, сортировать считанные строки по заданному столбцу по возрастанию (номер столбца вводит пользователь), и сохраняет отсортированные строки в файл numbers_sorted.txt.

Пример входного файла numbers.txt

```
12 33 -100 15 666 7
```

```
20 30 40 34 -12 -89
9 81 0 -4 77 -12
-49 -5 12 -69 57 -33
```

Пример выходного файла numbers.txt
Если пользователь выбрал столбец 2

```
-49 -5 12 -69 57 -33
20 30 40 34 -12 -89
12 33 -100 15 666 7
9 81 0 -4 77 -12
```

Задание 4.

Напишите программу, которая будет считывать строки из файла input.txt и проверять их на палиндромность. Все палиндромы нужно сохранить в файле output.txt.

Пример входного файла input.txt

```
Аргентина манит негра
А роза упала на лапу Азора
Топот
Карапуз
О, лета тело!
Лег на храм, и дивен и невидим архангел…
Искать такси?
Пишу проги на Питоне
Хочу стать тестировщиком.
```

Пример выходного файла input.txt

```
Аргентина манит негра
А роза упала на лапу Азора
Топот
О, лета тело!
Лег на храм, и дивен и невидим архангел…
Искать такси?
```

Список палиндромов во входном файле должен содержать 15-20 строк, можете найти палиндромы в сети…