# Лабораторная работа №7:
# «Файлы»

## Постановка задачи

В предыдущих заданиях необходимые для программы данные, вводились с клавиатуры, а результат выводился на экран монитора. Очевидно, что и при отладке программ, и при вводе большого объема данных в программу такой подход требует значительных трудозатрат и потому непригоден. Наиболее подходящее решение – это ввод данных из файла и вывод результатов работы в файл. При этом входные данные подготавливаются однократно и с должным многообразием, а результаты работы можно анализировать многократно.

Напишем программу, которая будет считывать входные данные из заранее подготовленного текстового файла, и выводить результат в текстовый файл. Для этой цели воспользуемся программами, написанными в предыдущих заданиях.

## Теоретическое введение

Файловый тип данных введён в языках программирования для работы с внешними устройствами – файлами на диске, портами ввода/вывода, принтерами и т.д. Файловый тип подразделяют на текстовый и бинарный (двоичный).

Текстовый файл содержит данные типа строка (str), а бинарный – типа bytes.

Доступ к файлам может быть последовательным или прямым. При последовательном доступе каждый следующий элемент может быть прочитан только после выполнения аналогичной операции с предыдущим элементом. При прямом доступе операция чтения (записи) может быть выполнена для произвольного элемента с заданным адресом.

Текстовые файлы относятся к файлам с последовательным доступом. Они предназначены для хранения информации строкового типа. При этом ввод и вывод информации сопровождается преобразованием типов данных. При выводе в текстовый файл данные преобразуются из внутреннего представления в символы, а при вводе выполняется обратное преобразование.

Бинарные файлы относятся к файлам с прямым доступом. В них хранится информация в двоичном виде. При записи или чтении бинарного файла информация не подвергается дополнительному преобразованию.

Для организации работы с файлами, при программировании на языке высокого уровня, выполняются, как правило, четыре шага:

- создается объект файла. Для этого используются подпрограммы, которые связывают имя файла, задаваемое пользователем, с переменной, которая хранит ссылку на специально создаваемую операционной системой структуру. Эта структура содержит

информацию о файле, о буфере данных, через который будет проходить обмен между программой и файлом и о текущем состоянии процесса обмена данными;

- задается режим обмена, в котором будет происходить работа с файлом: будет ли это режим чтения, записи, добавления или какой-либо совмещенный режим, например, чтение и запись. Этот шаг реализуется либо после создания объекта файла, либо в процессе выполнения первого шага;
- производится запись или чтение данных. Процесс обмена данными между программой и файлом состоит в обмене данными между программой и буфером данных под управлением файловой подсистемы. При записи данных в буфер, файловая подсистема контролирует процесс записи и при заполнении буфера до некоторого уровня выполняет запись данных в файл, а буфер очищается, разрешая программе продолжать запись. При чтении данных из буфера файловая подсистема контролирует объем данных в буфере и, при необходимости, выполняет подкачку свежих данных из файла;
- выполняется операция закрытия файла. При этом остаток данных, находящийся в буфере записывается в файл и файл закрывается.

Операция закрытия файла обязательно должна выполняться, если файл был открыт на запись. Если файл не закрыть, то при завершении программы, ресурсы, выделенные операционной системой, будут закрыты. В этом случае может возникнуть состояние, когда файл окажется пуст (чаще всего) либо будет содержать не всю информацию, которую в него пытались записать. Это зависит от размера буфера, который в современных ОС может быть достаточно большим.

Файлы, открытые на чтение, так же необходимо закрывать. Для этого есть две причины:

- открытый на чтение файл блокируется и другие приложения не получают к нему доступа;
- и в программе, и в операционной системе есть ограничение на число открытых файлов.

В объектно-ориентированном языке программирования, как это, например, реализовано в Python, часть описанных шагов может быть реализована в скрытой форме.

Например, при создании файлового объекта первый и второй шаги выполняются в одной инструкции:

```
fh = open(<Имя_файла> [, mode = <mod>])
```

где `fh`– переменная, хранящая ссылку на файловый объект, `<Имя_файла>` – абсолютный или относительный путь и имя файла, `mode=<mod>`– режим в котором открывается файл: запись, чтение, добавление, …

Язык Python поддерживает протокол менеджеров контекста. Этот протокол гарантирует правильное закрытие файла в независимости от того,

произошло исключение (ошибка) внутри блока кода или нет. Например, следующий код открывает файл на запись, записывает в файл строки, закрывает файл, а затем вновь открывает его, выводит текст на экран и закрывает файл. Обратите внимание на то, что операция закрытия файла в явном виде в коде отсутствует:

```
with open(r"lab6.txt","w") as fh:
fh.write("Меркурий\n") # Запись строк в файл
fh.write("Венера\n")
fh.write("Земля\n")
# В этом месте файл fh закрыт автоматически
with open(r"lab.6.txt","r") as fh:
print(fh.read())
# В этом месте файл fh так же закрыт
```

При создании объекта файла, необходимо указывать путь и имя существующего, либо будущего файла. Путь к файлу можно задавать как относительно текущей рабочей папки, так и абсолютно. При этом под относительным путем понимается путь относительно текущей рабочей папки, а под текущей рабочей папкой понимается папка, в которой находится пользователь в момент запуска файла (запускаемый файл может находиться в другой папке).

Далее рассмотрена модификация программы, написанной к лабораторной работе №1. Приведены способы работы с файлами.

В этой работе мы учились записывать выражения на языке Python. Внесем следующее изменение в нашу программу:

- подготовим текстовый файл с исходными данными;
- используя инструкции для работы с текстовым файлом, прочитаем записанные строки и выполним вычисления;
- результат вычисления запишем в текстовый файл в виде таблицы.

Вычисляемые выражения оформим в виде функций:

```
def f1(a,x):
    y = (tan(x**2/2-1)**2+(2*cos(x-pi/6))
        /(1/2+sin(a)**2))
    return y
def f2(x):
    y = pow(2, log(3-cos(pi/4+2*x),3+sin(x))
        /(1+tan(2*x/pi)**2))
    return y
```

*Текстовыйфайл*

Установим следующий формат текстового файла:

- две строки – это шапка, в которой указано назначение столбцов. Эти строки снабдим символом комментария, который используется в Python: '#';
- два столбца – это данные, для которых будут проводиться вычисления.

Пример:
```
# a x
#------------
-2 -2
 0 -2
...
```
Используем упрощенную схему работы с текстовым файлом.

***Создать файловый объект***:

`fh = open(<Имя_файла> [, mode = <mod>])`,

где `fh`– переменная, хранящая ссылку на создаваемый файловый объект, `<Имя_файла>`– абсолютный или относительный путь и имя файла, `mode=<mod>`– режим в котором открывается файл. Вместо `<mod>`подставляется один из символов, см. таблицу.

| mod | Режим | Примечание |
|---|---|---|
| 'r' | чтение файла (read) | файл должен существовать, если файл не существует, то возбуждается исключение: FileNotFoundError |
| 'w' | запись в файл (write) | если файла несуществует, то он создается |
| 'a' | добавление в файл (append) | если файла несуществует, то он создается, запись выполняется в конец файла |
| 'r+' | чтение и запись | файл должен существовать, если файл не существует, то возбуждается исключение: FileNotFoundError |
| 'w+' | чтение и запись | если файла несуществует, то он создается, существующий файл перезаписывается |
| 'a+' | чтение и запись | если файла несуществует, то он создается, запись выполняется в конец файла |
| 'x' | создать файл для записи | если файл существует, то возбуждается исключение: FileExistsError |
| 'x+' | создать файл для чтения и записи | если файл существует, то возбуждается исключение: FileExistsError |

Вместе с указанием режима может следовать модификатор, определяющий режим открытия файла: `t` – текстовый или `b` – бинарный.

Для решения нашей задачи нам необходимо открыть два файла. Один файл будет содержать информацию для расчета выражений и будет открыт на чтение, а второй – для вывода результатов расчета – будет открыт для записи:

```
fi = open("lab1_pb_in.txt", mode = "rt")
fo = open("lab1_pb_ou.txt", mode = "wt")
```

***Читать из файла:***

Чтение файла можно организовать по-разному, например, считывать по строкам (метод `readline()`) или считать весь файл в буфер (метод `readlines()`) и затем обрабатывать строки. Обратим внимание на то, что

строки заканчиваются символом конца строки (`'\n'`). Метод `readline()` читает строку, включая и символ конца строки.

При чтении по строкам итерацию (чтение следующей строки) можно выполнять через цикл `for`. Рассмотрим два примера:

```
1) whileTrue:
    line = fi.readline() # чтение строки
    ifnotline: # строка пустая
        break # конец файла и обработки
    elifline=="\n": # конец строки
        continue # продолжим чтение строк
    (b, c) = line.split() # разделить строку
2) forlineinfi: # для всех строк файла
    ifline=="\n": # конец строки
        continue # продолжим чтение строк
    (b, c) = line.split("\t") # разделить строку
```

В первом примере итерации выполняются в цикле `while` при выполнении инструкции чтения строки. Считанное значение сохраняется в переменной `line`, и проверяется на то, что получено не пустое значение. Если инструкция `fi.readline()` вернет пустую строку, то это значит, что прочитан конец файла (`EOF`) и дальнейшую обработку можно прекратить (`break`). Кроме этого проверяется, что строка содержит информацию. Если строка не содержит информации, то в ней будет только символ конца строки. В этом случае обработку следует продолжить с чтения следующей строки (`continue`).

*Замечание:* Если строка не содержит информацию (в строке нет символов, которые можно было бы визуализировать), то в ней есть символ конца строки. Если строка пустая, то в ней НЕТ НИКАКИХ символов.

Во втором примере итерации (чтение строк) выполняются циклом `for`. Строки по очереди считываются в переменную `line`. В этом случае нет необходимости контролировать конец файла (`EOF`) .

Дальнейшая обработка считанной информации выполняется в соответствии с форматом записи данных.

В нашем примере мы поместили в начале файла две строки, описывающие данные, которые следуют за ними, а сами данные разместили по строкам в форме двух столбцов. Разделителем между столбцами может выступать знак табуляции или несколько пробелов.

При чтении такого файла поступим следующим образом:

- прочитаем две строки без обработки (пропустим эти строки). Эти строки – памятка, которой можно воспользоваться при подготовке файла в текстовом редакторе;
- в цикле читаем строку, и расщепляем ее для получения данных.

Формат записи метода расщепления (разделения) следующий:

`str.split(sep=None, maxsplit=-1)`,
где `str`– строка символов, `sep`– разделитель, а `maxsplit`– количество груп, на которые делится строка.

Если указан разделитель `sep`(sep – от separate – разнимать) и количество групп `maxsplit`, то строка будет разделена на `maxsplit + 1` части. Если `maxsplit`не указан или равен -1, то число частей, на которые будет поделена строка неограниченно. Пример разделения строки:

```
'1,2,3'.split(',',maxsplit=1) # ['1', '2,3']
```

Мы можем не указывать параметры в методе `split()`, поскольку при расщеплении будет формироваться список из двух значений (в строке два столбца), а разделителем мы выбрали пробелы или знак табуляции.

В левой части инструкции мы укажем две переменные, которые примут значения, полученные при расщеплении строки.

```
b, c = line.split()
```

Полученные при расщеплении значения будут строкового типа и для дальнейшего их использования необходимо выполнить преобразование к вещественному типу (`float`).

***Записывать в файл:***

Для записи в файл будем использовать метод`write(<Данные>)`. Под данными тут выступает строка, в том числе и форматная строка, содержащая знаки форматирования. При записи строки в файл метод`write()` не добавляет символа конца строки. Для того, что бы следующие данные записывались с новой строки, необходимо, что бы текущая строка завершалась символом конца строки (`'\n'`).

## Листингпрограммы

```
from math import *
def f1(a,x):
    y = (tan(x**2/2-1)**2+(2*cos(x-pi/6))
        /(1/2+sin(a)**2))
    return y
def f2(x):
    y = pow(2, log(3-cos(pi/4+2*x),3+sin(x))
        /(1+tan(2*x/pi)**2))
    return y
fi = open("lab1_pb_in.txt", "rt") #читатьфайл
fo = open("lab1_pb_ou.txt", "wt") #писатьвфайл
line = fi.readline() # Пропуститьстроки
line = fi.readline() # заголовкавфайле
# Вывести шапку таблицы в файл
fo.write("+======+======+========+========+\n")
fo.write("I A I X I F1 I F2 I\n")
    fo.write("+======+======+========+========+\n")
for line in fi: # длявсехстрокфайла
```

```
    if line=="\n":
        continue
    (b, c) = line.split() # расщепить
    a = float(b) # привести к вещест. типу
    x = float(c)
# Вывод в файл
    fo.write("I {0: .1f} I {1: .1f} I {2: 5.4f} I"
        .format(a, x, f1(a, x)))
    fo.write("{0: 6.4f} I\n".format(f2(x)))
    fo.write("+------+------+---------+""--------+\n")
fi.close() # закроемфайлы
fo.close()
```

**Результат работы программы(текстовый файл lab1_pb_ou.txt)**

```
+=======+=======+=========+========+
I A I X I F1 I F2I
+=======+=======+=========+========+
I -2.00 I -2.00 I 1.1970 I 1.1184 I
+-------+-------+---------+--------+
I 0.00 I -2.00 I -0.8347 I 1.1184 I
+-------+-------+---------+--------+
I 0.00 I 0.00 I 5.8896 I 1.6880 I
+-------+-------+---------+--------+
I 2.00 I 0.00 I 3.7309 I 1.6880 I
+-------+-------+---------+--------+
I 1.50 I 0.50 I 2.7712 I 1.7955 I
+-------+-------+---------+--------+
I4.00  I 3.00  I -1.3266 I 1.0517 I
+-------+-------+---------+--------+
```

## Задание к лабораторной работе №7 «Файлы»

Выполнить корректировку программы, написанной для лабораторной работы №1, чтобы ввод данных и вывод результатов работы осуществлялся с использованием файлов.