

Лабораторная работа №6: «Файлы»

Цель работы:

Дать студентам практический навык в написании программ, в которых выполняются операции с текстовыми файлами – чтение, запись.

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 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 несколько программ, которые будут 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 введен 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.

Например, если в памяти некоторая переменная целого типа хранит в двух байтах значение 731_{10} , то:

- в двоичном представлении это будет: $0000\ 0010\ 1101\ 1011_2$;
- в текстовом: $37\ 33\ 31_{\text{string}}$.

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.

Общий подход к работе с файлами, для всех языков программирования, строится на следующих представлениях:

- в операционной системе имеется файловая подсистема, обеспечивающая надежную работу с файлами в многопользовательском и многозадачном режимах при работе с различными внешними носителями: жесткие диски, флэш, DVD, ...;

- операционная система предоставляет функциональный набор (API-функции), который позволяет работать с файлами на логическом уровне. Программист не занимается управлением внешних устройств, контролем записей о файлах в папках, контролем качества записи, и т.д.

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.

И хотя, по вашему мнению, доступ к файлу из других приложений маловероятен, а число открытых файлов достаточно большое, не стоит испытывать судьбу, только для того, что бы получить сообщение об ошибке, а затем долго и мучительно искать ее причину.

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", encoding="cp1251") 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", encoding="cp1251") 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).

Для правильного понимания того, как может формироваться путь, следует обратиться к литературе, например [1] и провести самостоятельные эксперименты.

Далее рассмотрены модификации программ, написанных к лабораторным работам №1 и №5. В этих модификациях приведены способы работы с файлами.

Дополнение к лабораторной работе №1

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.

Напоминание: При написании программы длинную строку инструкции можно разместить на нескольких строках. Для этого используется символ обратного слэша, за которым, сразу, следует Enter или, например, круглые или квадратные скобки (предпочтительно).

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
```

Текстовый файл

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. Таблица 6.1.

Таблица 6.1. - Режимы работы с файлами

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 не существует, то он создается
'a'	добавление в файл (append)	если файла не существует, 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 не существует, то он создается, существующий файл перезаписывается
'a+'	чтение и запись	если файла не существует, 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) while True:
    line = fi.readline() # чтение строки
    if not line:         # строка пустая
        break           # конец файла и обработки
    elif line=="\n":     # конец строки
        continue        # продолжим чтение строк
    (b, c) = line.split() # разделить строку
```

```

2)      for line in fi:          # для всех строк файла
          if line=="\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, `maxsplit` – количество групп, 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 деления 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'`). Пример использования вы найдете в листинге программы ниже.

Далее представлены два варианта программы и результаты ее работы. Во втором варианте для загрузки данных из файла использована функция модуля `NumPy`, а данные загружаются в двумерный массив (строка выделена жирным шрифтом).

Листинг программы

Первый вариант:

```
# -*- coding: cp1251 -*-
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") #читать файл
fo = open("lab1_pb_ou.txt", "wt") #писать в файл
line = fi.readline()             # Пропустить строки
line = fi.readline()             # заголовка в файле
# 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:          # для всех строк файла
    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()               # закроем файлы
fo.close()

```

Второй вариант:

```

# -*- coding: cp1251 -*-
from math import *
import numpy as np

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(r"lab1_pb_in.txt", "rt") # читать файл
fo = open(r"lab1_pb_ou.txt", "wt") # писать в файл
# Вывести шапку таблицы в файл
fo.write("+=====+=====+=====+=====+\n")
fo.write("I      A      I      X      I      F1      I      F2      I\n")
fo.write("+=====+=====+=====+=====+\n")
# Загрузим данные в массив
ax = np.loadtxt(fi,dtype=np.float, ndmin = 1)
nRow, nCol = ax.shape # Строк и колонок в массиве
for Row in range(nRow): # для всех строк
    a = ax[Row][0]
    x = ax[Row][1]
    fo.write("I {0: .2f} I {1: .2f} I {2: 6.4f} I"
             .format(a, x, f1(a, x)))
    fo.write("{0: 6.4f} I\n".format(f2(x)))
    fo.write("+-----+-----+-----+-----+
             "-----+\n")

```



```
fi.close()    # закроем файлы
fo.close()
```

Замечание: Обратите внимание на то, как выполнен перенос длинных строк. В функции `f1()` в круглые скобки взято все выражение. В функции `f2()` использованы круглые скобки функции `row()`, а в методе записи данных в файл `fo.write()` – круглые скобки метода.

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

```
+=====+=====+=====+=====+
I   A   I   X   I   F1   I   F2   I
+=====+=====+=====+=====+
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
+-----+-----+-----+-----+
I  4.00 I  3.00 I -1.3266 I 1.0517 I
+-----+-----+-----+-----+
```

Дополнение к лабораторной работе №5

Эту работу выполним в два шага. На первом шаге мы сформируем массив с использованием функции `random()` и сохраним его в текстовом формате. Формирование массива, его запись в файл и чтение из файла выполним с использованием функций модуля `NumPy`.

Для инициализации массива, вычисления среднего значения и дисперсии, а так же корректировки массива, воспользуемся ранее написанными функциями.

Небольшие изменения внесем в функцию генерации массива с тем, что бы можно было получать не только квадратный, но и прямоугольный массив.

Листинг программы

```
import numpy as np
from math import *

def MakeMatr(n, m, a = -5, b = 5):
    """ Инициализация прямоугольной матрицы
    размером nxm псевдослучайными величинами
    в диапазоне [a, b) """
    Matr = a+(b-a)*np.random.random(size = (n,m))
    return Matr
```

```

def MidlDisp(Matr):
    """ Вычисление среднего значения
    Вычисление дисперсии """
    sum = 0
    (nRow, nCol) = Matr.shape # Размеры матрицы
    for Row in range(nRow):
        for Col in range(nCol):
            sum += Matr[Row][Col]
    Midl = sum / Matr.size # Среднее
    Disp = 0
    for Row in range(nRow):
        for Col in range(nCol):
            Disp += (Matr[Row][Col] - Midl)**2
    return (Midl, Disp / (Matr.size - 1))

def MakeCorrect(Matr, avr, sigma):
    """ Замена "отскачивших" значений """
    (nRow, nCol) = Matr.shape # Размеры матрицы
    for Row in range(nRow):
        for Col in range(nCol):
            if abs(abs(Matr[Row][Col]) - avr) > sigma:
                Matr[Row][Col] = avr
    return (Matr)

n,m=input("Введите размер матрицы (N M):").split()
n = int(n)
m = int(m)
tstMatr=MakeMatr(n, m, -7, 7) # готовим матрицу
str_o = 'Resalt after initiation:'
fh_i = open("lab5_pd_in.txt", "wb") # для записи
np.savetxt(fh_i, tstMatr, fmt = ' %8.4f',
            header = str_o) # выгрузим ее в файл
fh_i.close() # закроем файл

# Выполняем задание
# Загрузим данные в массив
fh_i = open("lab5_pd_in.txt", "rt")
MyMatr = np.loadtxt(fh_i, dtype=np.float, ndmin=1)
# Откроем файл для результатов работы программы
fh_o = open(r"lab5_pd_ou.txt", "wb")
# Сохраним исходный массив
str_o = 'Before correction:'
np.savetxt(fh_o, MyMatr, fmt = ' %8.4f',
            header = str_o)

# Вычисляем среднее и дисперсию
mMidl, mDisp = MidlDisp(MyMatr)

# подготовим строки для записи

```

```

str_o = '\nAfter correction:'
fstr=" ".join((' \nMidl= {0:5.4f} Disp= {1:6.4f}',
               'Sigm= {2:6.4f}'))
str_o1 = fstr.format(mMidl, mDisp, sqrt(mDisp))
# Корректируем массив
NMatr = MakeCorrect(MyMatr, mMidl, sqrt(mDisp))
# Сохраняем результат работы в файле в формате:
# Заголовок
# Скорректированный массив
# Завершающая часть
np.savetxt(fh_o, NMatr, fmt = ' %8.4f',
           header = str_o, footer = str_o1)
fh_o.close() # закроем файл

```

Замечание: В этом примере методом `.join()` собрана форматная строка `fstr`, которая использована для сохранения результатов вычисления в файле, см. следующую инструкцию. Пробел в двойных кавычках перед методом – это разделитель, используемый при объединении строк. Вместо пробела можно вставить и комбинацию символов.

Результат работы программы (текстовый файл lab1_pd_ou.txt)

```

# Before correction:
  3.3232    -1.7425    -5.2076
  6.2726    -0.8202     6.1146
  4.6182     0.2746    -2.9510
  2.6626    -3.9889    -5.1610
#
# After correction:
  3.3232    -1.7425     0.2829
  0.2829    -0.8202     0.2829
  0.2829     0.2746    -2.9510
  2.6626    -3.9889     0.2829
#
# Midl= 0.2829 Disp= 17.9449   Sigm= 4.2361

```

Это важно: При просмотре файла через Блокнот записи будут смотреться как одна строка. Откройте файлы через WordPad. В этом случае вы увидите другое представление данных. Сможете ли вы найти причину такой неоднозначности представления. Обратитесь за разъяснением к преподавателю.

Задание к лабораторной работе №6 «Файлы»

Выполнить корректировку программ, написанных для лабораторных работ №1, №4 и №5, с таким условием, что бы ввод данных и вывод результатов работы осуществлялся с использованием файлов.