**Работа с файлами**

Если данных много, их хранят в файле, а не вводят руками с консоли. Нужно уметь обрабатывать много данных из файла и читать и писать в файл.

**Перенаправление потоков**

Вспомним сначала, как перенаправлять данные.

Если программа matrix.py транспонирует матрицу, которую ей ввели, то можно или вводить матрицу.

Файл matrix.py:

n = int(input())

m = []

for i in range(n):

line = input()

print('--%s--' % line)

m.append(list(map(int, line.split())))

mT = zip(\*m)

print(\*mT)

mT = [list(r) for r in zip(\*m)]

print(\*mT)

print(10/0) # ОШИБКА: Divizion by zero

Пример ввода:

3

1 2 3 4

5 6 7 8

9 10 11 12

Сохраним эти данные в файл test\_matrix.txt и при запуске программы перенаправим данные из этого файла на вход программы matrix.py:

python3 matrix.py < test\_matrix.txt

Знак < означает перенаправление на вход программы. Он работает в любых ОС (Windows, Linux, Mac).

У каждой программе при запуске есть 3 потока: стандартный поток ввода **stdin**, стандартный поток вывода **stdout** и стандартный поток сообщений об ошибках **stderr**.

* **stdin** - в него попадают данные, которые мы вводим с клавиатуры или перенаправляем в программу с помощью < в командной строке. С него читает функция **input()**.
* **stdout** - выводится на экран. В него печатает **print**.
* **stderr** - тоже выводится на экран. В него печатают сообщения об ошибках (exception).

Поток stdout можно перенаправить в файл в командной строке с помощью >.

Поместим транспонируемую матрицу в файл transp.txt, входную матрицу вводим с клавиатуры:

python3 matrix.py > transp.txt

Можно и перенаправить входные данные из файла test\_matrix.txt, и сохранить результат в файле transp.txt:

python3 matrix.py < test\_matrix.txt > transp.txt

Операция > перезаписывает файл. Если нужно добавить данные в конец файла, сохранив в нем старые данные используют >>.

python3 matrix.py >> transp.txt

Данные старые сохранились, добавились новые.

Можно писать и в файл, и выводить на экран. Для этого перенаправьте выходные данные программы на вход программы **tee** (нет в Windows) с помощью |.

python3 matrix.py | tee transp.txt

Сообщения об ошибках тоже можно перенаправить в файл err.log, зная, что stderr - это поток номер 2.:

python3 matrix.py < test\_matrix.txt 2> err.log

Теперь stdout перенаправим в transp.txt, а stderr перенаправим в err.log:

python3 matrix.py < test\_matrix.txt > transp.txt 2> err.log

Перенаправим stderr (поток номер 2) на stdout (поток номер 1) и потом перенаправим stdout в файл transp.txt (перенаправить stderr на stdout это 2>&1):

python3 matrix.py < test\_matrix.txt > transp.txt 2>&1

Перенаправить оба в один файл:

python3 matrix.py < test\_matrix.txt &> transp.txt

**Как прочитать введенный текст по строкам**

Задача: заранее количество строк не известно. Нужно их всех прочитать и все напечатать.

Одну строку читаем **input()**. Много строк - можно сделать цикл for по потоку **stdin**. Для этого нужно имортировать модуль **sys**:

import sys

for line in sys.stdin:

print(line)

Строки перебираются до конца файла. Если вы перенаправляете на stdin из файла, то там есть конец файла. Если вводите с клавиатуры то конец файла с клавиатуры: | Клавиши | OC | |--|--| | Ctrl+Z | Windows | | Ctrl+D | Linux, Mac |

**Как прочитать весь текст сразу**

У потока все его данные можно прочитать функцией **read()**:

import sys

text = sys.stdin.read()

print(text)

**Отдельно прочитать первую строку, потом читать все остальные**

Функция **next()** у любого объекта, который можно перебрать в for дает 1 (очередной) элемент. Прочитаем первую строку с помощью next:

import sys

first = next(sys.stdin)

print('First line is:', first)

print('Others lines:')

for line in sys.stdin:

print(line)

**Открывать файлы в программе**

Можно не перенаправлять файлы на входной поток в командной строке, а открывать файлы в программе по имени.

**Открыть файл**

**open** (file, mode='r', buffering=None, encoding=None, errors=None, newline=None, closefd=True)

* file - это имя файла;
* mode - как отрывать файл (чтени, запись, добавление)

| **Мода** | **Как открывает** |
| --- | --- |
| 'r' | открытие на чтение (является значением по умолчанию). |
| 'w' | открытие на запись, содержимое файла удаляется, если файла не существует, создается новый. |
| 'x' | открытие на запись, если файла не существует, иначе исключение. |
| 'a' | открытие на запись, информация добавляется в конец файла. |
| 'b' | открытие в двоичном режиме. |
| 't' | открытие в текстовом режиме (является значением по умолчанию). |
| '+' | открытие на чтение и запись |

**Надо закрывать файлы**

Если вы прочитали файл, то его нужно **обязательно закрыть**. Закрываем методом **close()**:

**Читать из открытого файла**

Так же, как из stdin:

f = open('test\_matrix.txt', 'r')

for line in f:

print(line)

f.close()

**Писать в файл**

Писать в файл можно методом **write()** или функцией **print()**, указав аргумент *file=открытыйфайл*

fin = open('test\_matrix.txt', 'r')

fout = open('transp.txt', 'w')

for line in fin:

print(line, file=fout)

fin.close()

fout.close()

**with .. as**

* Программисты иногда забывают закрывать файлы. Это плохо.
* Если пока мы пишем произошла в программе ошибка, то программа закончит работать, но сама файл не закроет. Придется писать специальный код, чтобы обрабатывать ошибки. Это сложно.

Можно сделать проще - использовать конструкцию **with .. as** которая сама закроет файл в конце работы или если произошла ошибка:

with open('test\_matrix.txt', 'r') as fin:

for line in fin:

print(line)

# тут файл test\_matrix.txt будет уже закрыт

Если нужно из одного файла читать, а в другой писать, то пишут так:

with open('test\_matrix.txt', 'r') as fin, open('transp.txt', 'w') as fout:

for line in fin:

print(line, file=fout, end='')

Можно вместо открытия файла написать, что fin берет данные с клавиатуры, а fout печатает на экран:

import sys

fin = sys.stdin

fout = sys.stdout

**Задачи**

**0. Перенаправления**

Повторите перенаправления:

* только читать данные из файла
* только писать результат в файл
* и читать данные из файла, и писать результат в файл.

**1. Матрица из файла**

Сохраните данные в файл test.txt:

3

1 2 3 4

5 6 7 8

9 10 11 12

Откройте в программе файл с этим именем, прочтайте из него матрицу в формате

* количество строк
* строки с целыми числами через пробел

Транспонируйте матрицу и напечатайте ее на экран.

**1.b Сохранить в файл**

В предыдущей задаче откройте файл и напечатайте в него транспонируемую матрицу.

**Прочитать матрицу, заранее количество строк не известно**

Дана матрица в файле в формате:

1 2 3 4

5 6 7 8

9 10 11 12

**БЕЗ количества строк**

Напечатать транспорируемую матрицу.

**Убрать из строки пробельные символы**

Иногда в строке **в начале и в конце** есть лишние пробелы, табуляции и концы строк. Чтобы их убрать используйте метод **strip()**

s = ' abc xyz \n'

s = s.strip()

print(s) # abc xyz (пробелы в середине функция НЕ удаляет)

**Проверить, что строка пустая**

def check\_string(s):

if s:

print('В строке что-то есть')

else:

print('Пустая строка')

check\_string('hello') # В строке что-то есть

check\_string('') # Пустая строка

**Эти строки равны?**

Проверить, что строки равны:

s1 = 'abc'

s2 = 'abc'

if s1 == s2:

print('Строки равны')

**3. Обработать данные из файла**

Возьмите файл (tr.log)[<http://acm.mipt.ru/twiki/pub/Cintro/PythonListTask/tr.log>] и найдите:

1. Сколько времени длился тест?
2. Какие названия действий есть в тесте?
3. Сколько раз встретилось действие get\_pdf?
4. Сколько раз встретилось действие get\_pdf в % от всех действий?
5. Сколько раз встретилось действие add\_outcome у клиента номер n (n - ввести с клавиатуры)?
6. Когда первый раз встретилось действие?
7. Минимальная длительность действия login.
8. Максимальная длительность действия main.
9. Средняя длительность действия get\_pdf.

Модификация: сделать для всех действий.

**Дополнительная задача**

Напечатать для каждого действия из файла : действие, минимальная длительность, максимальная длительность, средняя длительность.