Лабораторная работа № 21
РАБОТА С ТЕКСТОВЫМИ ФАЙЛАМИ

Цель работы

Изучить основные функции языка Python, используемые для работы с текстовыми файлами. Научиться создавать файлы, открывать их на чтение и редактирование, обрабатывать содержащуюся в них информацию.

Методические указания

В большинстве случаев информация, поступающая для обработки, представлена в виде готовых файлов, а результаты работы программы требуется записывать в файл, а не выводить на экран. Сама же программа является обработчиком данных, хранящихся в файле.

Файл – это поименованная область памяти на внешнем носителе, предназначенная для хранения информации.

Существуют различные форматы файлов. С точки зрения программиста, бывают файлы двух типов:

1) *текстовые*, содержащие текст, разбитый на строки; они открываются в любом текстовом редакторе (например, Блокноте) и имеют расширения: *.txt*, *.html*, *.csv* и др.;
2) *двоичные*, в которых могут содержаться любые данные и любые коды без ограничений, например рисунки, звуки, видеофильмы и т.д.; эти файлы открываются в специальных программах.

Мы будем работать только с текстовыми файлами, поскольку текстовый формат является наиболее простым и универсальным.

Работа с файлом из программы включает три этапа.

Этап 1. *Открытие файла*. Перед обработкой файл необходимо открыть, то есть сделать его активным для программы. Если файл не открыт, то программа не может к нему обращаться. При открытии файла указывают режим работы: чтение, запись или добавление данных в конец файла. Чаще всего открытый файл блокируется так, что другие программы не могут использовать его.

Этап 2. *Работа с файлом*. После открытия происходит непосредственная работа с файлом: программа выполняет все необходимые операции в соответствии с техническим заданием.

Этап 3. *Закрытие файла*. После обработки файл нужно закрыть, то есть освободить занятые им ресурсы, разорвать связь с программой. Именно при закрытии все последние изменения, сделанные программой в файле, записываются на диск.

В Python для открытия файла используется функция `open`, которой необходимо передать следующие параметры (у функции `open` несколько параметров, с которыми можно ознакомиться в документации, нам важны пока только три рассмотренных):

- `file` – *имя файла* или путь к файлу, если файл находится не в том каталоге, где записана программа;
- `mode` – *режим открытия* файла, определяющий что можно делать с данными из файла: `'r'` – открыть на чтение, `'w'` – открыть на запись, `'a'` – открыть на добавление; основные режимы работы с файлом и их обозначения представлены в табл. 1;
- `encoding` – *наименование кодировки*, используемой при чтении/ записи файла (например, `'utf-8'` или `'cp1251'`); параметр имеет смысл только для текстового режима.

Синтаксис функции открытия файла:

```
open('file', [mode='режим'], [encoding='кодировка'])
```

В квадратные скобки заключены необязательные параметры.

Таблица 1. Режимы открытия файлов

Символ	**Описание**
`t`	Открытие файла в текстовом режиме. Данный режим установлен по умолчанию
`b`	Открытие файла в двоичном режиме
`r`	Открытие файла для чтения. Данный режим установлен по умолчанию
`w`	Открытие файла для записи. Если существующий файл открывается на запись, его содержимое уничтожается, если файл не существует, создается новый
`x`	Открытие файла для записи, причем если файл не существует, то интерпретатор выдает ошибку
`a`	Открытие файла для добавления (`a` – *append*), информация добавляется в конец файла. Если файл не существует, то он создается
`+`	Открытие файла для чтения и записи одновременно

По умолчанию, если не указывать режим открытия, то используется открытие на чтение (`'r'`). Работа с файлами может осуществляться как в текстовом, так и в двоичном режиме. Двоичный/текстовый режимы могут быть скомбинированы с другими: например, режим `'rb'` позволит открыть на чтение бинарный файл, а режим `'wb'` от-

крывает бинарный файл для записи (если файл существует, то его содержимое очищается).

При открытии файла Python по умолчанию использует кодировку, предпочитаемую операционной системой (ОС). Старайтесь указывать кодировку файла явно, например `encoding='utf-8'`, особенно если есть вероятность работы программы в различных ОС.

Открытие файлов связано с потреблением/резервированием ресурсов, поэтому после выполнения необходимых операций его следует закрыть. Метод `close()` закрывает файл и освобождает занятую файловую переменную.

Рассмотрим простой пример чтения всего содержимого файла с помощью метода `read()` и его вывода на экран.

```
file = open('Text.txt', 'r')
contents = file.read()
print(contents)
file.close()
```

Файл *Text.txt* в рассматриваемом примере расположен в рабочем каталоге с программой (рис. 1). На рис. 2 представлены небольшие комментарии к тексту программы.

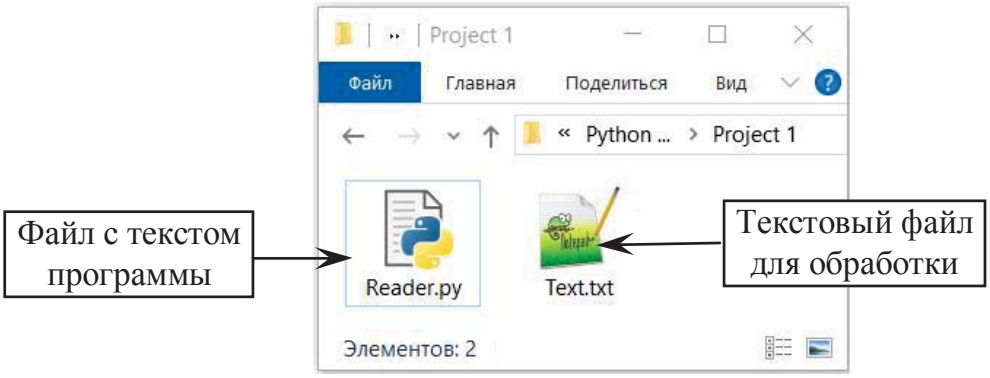

Рис. 1. Расположение программы и текстового файла

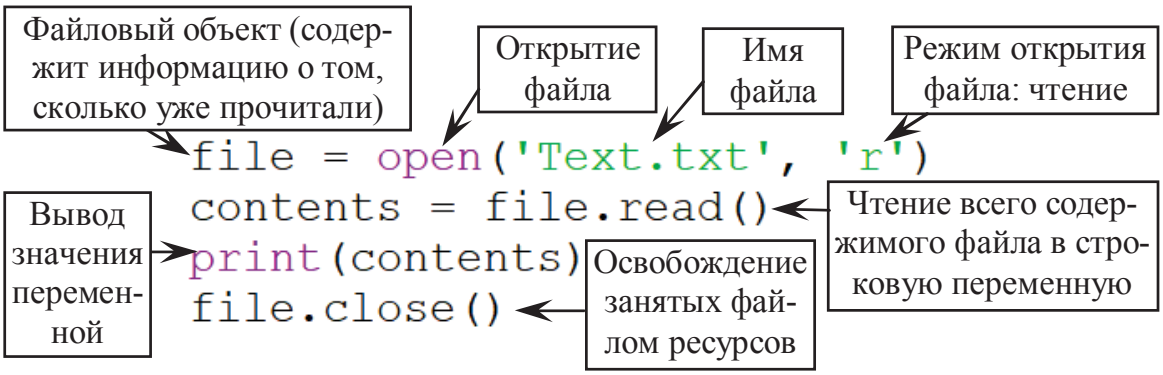

Рис. 2. Текст программы чтения из файла и комментарии к нему

Функция `open()` открывает файл и возвращает специальный *дескриптор* файла (идентификатор), однозначно определяющий, с каким файлом далее будут выполняться операции. После открытия доступен *файловый указатель* (файловый *курсор*) – число, определяющее текущую позицию относительно начала файла. При открытии файла значение указателя будет равно нулю. Когда прочитали весь файл, указатель находится в самом конце файла.

В момент вызова функции `open()` Python ищет указанный файл в текущем рабочем каталоге (в момент запуска программы текущий рабочий каталог там, где сохранена программа).

Строка в функции `open()`, используемая для идентификации файла, может содержать:

- *относительный путь* к файлу, который начинается с текущей директории, например если требуется открыть на чтение файл *Text.txt*, расположенный во внутренней папке *Data* текущего рабочего каталога с программой, то необходимо воспользоваться командой:
  ```
  open('Data/Text.txt', 'r', encoding='cp1251');
  ```
- *абсолютный путь*, который начинается с самой верхней директории файловой системы, например:
  ```
  open('D:/Project/DataFiles/Text.txt', 'r').
  ```

Следует помнить, что после окончания работы программы все открытые ею файлы закроются автоматически. При этом если файл был открыт для записи или добавления, а его закрытие методом `close()` не производится, то внесенные изменения записаны не будут. Поэтому после выполнения необходимых операций **файл обязательно нужно закрыть**.

Во время работы с файлом возможны различные *исключительные ситуации*. *Исключение* – это результат выполнения некорректного оператора, вызывающий прерывание или полное прекращение работы программы. Например, открываемый для чтения или добавления файл не существует, при записи файла диск может заполниться или оказаться недоступным, пользователь может удалить используемый файл во время записи, файл могут переместить и т.д. Эти типы ошибок можно перехватить с помощью обработки исключений, которые целесообразно использовать всегда при работе с файлами.

Обработка исключения заключается в нейтрализации вызвавшей его ошибки. Для обработки исключений в Python используется инструкция `try-except-else-finally`.

Рассмотрим стандартный способ открытия файла с обработкой исключений.

```
try:
    file = open('Text.txt', 'r')
    print(file.read())
    file.close()
except OSError:
    print('Ошибка при работе с файлом!')
else:
    print('Работа с файлом завершена.')
finally:
    print('Работа программы завершена.')
```

В случае отсутствия файла *Text.txt* в каталоге с проектом результатом выполнения программы будет следующее сообщение:

```
Ошибка при работе с файлом!
Работа программы завершена.
```

При наличии файла *Text.txt* в каталоге проекта будет выведено:

```
Это содержимое файла Text.txt.
Работа с файлом завершена.
Работа программы завершена.
```

Пояснение. Код, который потенциально может генерировать исключения, обрамляется в блок `try`, и при генерации исключений управление будет передано в блок `except`, где размещается код для их обработки. После `except` указывается тип исключения. При этом перехватываются как само исключение, так и его потомки. В рассмотренном примере, если файл не может быть открыт, возбуждается исключение `OSError` и его потомки (`FileNotFoundError` и др.). Блок `else` вызывается в том случае, если никакого исключения не произошло. У исключений есть блок `finally`, инструкции которого выполняются в любом случае, было исключение или нет.

Общий синтаксис конструкции для обработки исключений:

```
try
    # код, который может генерировать исключения
except <тип исключения>:
    # обработчик исключения
else:
    # вызывается, если исключения не произошло
finally:
    # инструкции, выполняемые в любом случае,
    # было исключение или нет
```

Чтение из файла

Для чтения информации из файла существует несколько методов, но большего интереса заслуживают лишь некоторые из них.

Рассмотрим, каким образом читается информация с помощью методов чтения `read()`, `readline()` и `readlines()` на примере текстового файла *ThisPython.txt*, содержащего несколько строк из «Дзен Питона», иллюстрирующего идеологию и особенности данного языка (рис. 3).

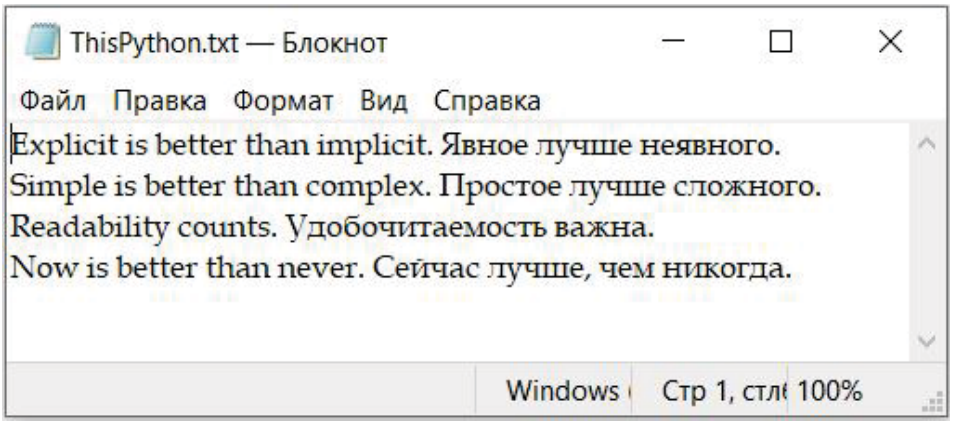

Рис. 3. Содержимое текстового файла

Метод **`read()`**, вызванный без аргументов, позволяет прочитать содержимое файла целиком. Если файл слишком большой, он может не поместиться в памяти.

Пример. Вывод содержимого текстового файла *ThisPython.txt*, содержащего текст на английском и русском языках:

```
file = open('ThisPython.txt', 'r', encoding='cp1251')
print(file.read())
file.close()
```

Результат работы программы:

```
Explicit is better than implicit. Явное лучше неявного.
Simple is better than complex. Простое лучше сложного.
Readability counts. Удобочитаемость важна.
Now is better than never. Сейчас лучше, чем никогда.
```

В методе `read()` можно указать конкретное количество информации, которое требуется прочитать. В случае работы с текстовым файлом при вызове **`read(n)`** будет прочитано `n` символов.

Когда прочитан весь файл, указатель находится в самом конце. Если попробовать методом `read()` прочитать информацию из файла еще раз, то уже ничего считано не будет. Для того чтобы прочитать файл повторно, его можно закрыть и открыть заново, а можно исполь-

зовать метод **`seek()`** и перенести указатель на начало файла, используя `seek(0)`. После этого указатель будет равен нулю, и файл можно читать заново.

Метод **`tell()`** возвращает текущую позицию указателя в файле относительно его начала.

Текущая позиция указателя – это позиция (количество байт), с которой будет осуществляться следующее чтение/запись.

Пример. Чтение информации из файла с использованием методов `read(n)`, `tell()`, `seek()`:

```
f = open('ThisPython.txt')
print(f.read(10))            # Explicit i
print(f.tell())              # 10
print(f.read(15))            # s better than i
print(f.tell())              # 25
print(f.read(8))             # mplicit.
f.seek(0)
print(f.read(6))             # Explic
f.close()
```

В рассмотренном примере выводимая информация содержится в комментариях.

Так как текстовый файл представляет последовательность символов, разбитых на строки, то из специальных символов в них могут находиться символы перехода на новую строку.

Метод **`readline()`** возвращает строку от текущей позиции указателя до символа переноса строки. Так, при выполнении программы

```
f = open('ThisPython.txt', 'r')
print(f.readline())
f.close()
```

на экран будет выведено только содержимое первой строки из текстового файла.

Когда файловый курсор указывает на конец файла, метод `readline()` возвращает пустую строку, которая воспринимается как ложное логическое значение. Эту особенность метода `readline()` можно использовать при чтении текстовых файлов с неизвестным количеством строк, например,

```
f = open('ThisPython.txt', 'r')
while True:
    s = f.readline()
    print(s, end='')
```

```
        if not s:
            break
    f.close()
```

При таком подходе в случае считывания пустой строки цикл заканчивается с помощью оператора `break`.

Метод **`readlines()`** возвращает список строк, разбитых по символу перевода строки. Этот метод удобно использовать, если требуется прочитать сразу все строки, разбить их по символу перевода строки и записать все это, например, в список.

Пример считывания строк из текстового файла в список методом `readlines()`:

```
f = open('ThisPython.txt', 'r')
print(f.readlines())
f.close()
```

Результат работы программы:

```
['Explicit is better than implicit. Явное лучше неявного.\n', 'Simple is better than complex. Простое лучше сложного.\n', 'Readability counts. Удобочитаемость важна.\n', 'Now is better than never. Сейчас лучше, чем никогда.\n']
```

Рассмотрим вариант построчного вывода информации из файла в стиле Python:

```
for s in open('ThisPython.txt', 'r'):
    print(s, end='')
```

Этот вариант обычно рекомендуют к использованию. При таком способе чтения информации из файла его не нужно закрывать. Обратите внимание, что переход на новую строку в функции `print` отключен (`end=''`), потому что при чтении символы перевода строки «`\n`» в конце строк файла сохраняются.

Запись в файл

Метод **`write(s)`** позволяет записать передаваемую ему текстовую строку `s` в файл, открытый предварительно для записи или добавления (с использованием режимов `'w'` и `'a'` соответственно). Метод `write(s)` возвращает количество символов, которые были записаны. Если записывалась байтовая информация, то это будет количество байт, а не количество символов.

Пример добавления текстовой строки в существующий файл. Добавить в конец файла *ThisPython.txt* ещё одну строку из «Дзен Питона»:

```
f = open('ThisPython.txt', 'a')
f.write('Errors should never pass silently.
       Ошибки никогда не должны замалчиваться.\n')
f.close()
```

Содержимое файла после добавления строки показано на рис. 4.

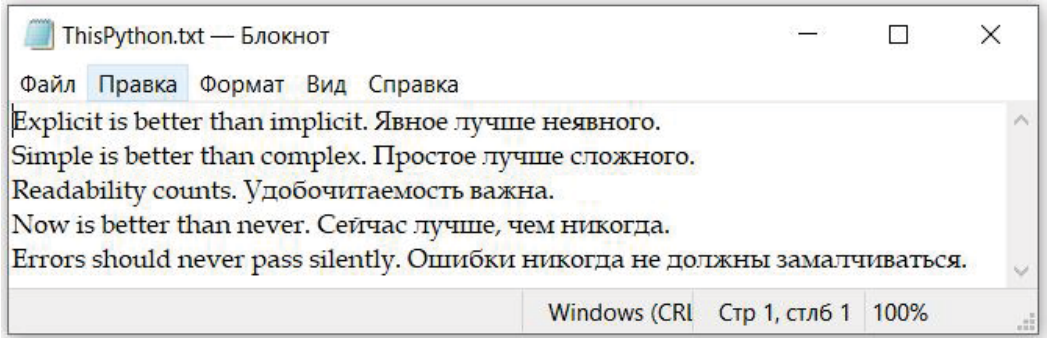

Рис. 4. Содержимое файла после добавления строки

Если требуется отобразить количество символов, записанных в файл, то запись в файл можно поместить внутрь оператора вывода:

```
print(f.write('Errors should never pass silently.
       Ошибки никогда не должны замалчиваться. \n'))
```

В ходе выполнения этого оператора на экране отобразится число 75, показывающее количество записанных в файл символов.

Если нужно записать в файл числовые данные, их сначала преобразуют в строку, например так:

```
f = open('Data.txt', 'w')
a = 34
b = -102.5678
f.write(str(a) + '\n' + '{:7.2f} \n'.format(b))
f.close()
```

При таком способе записи символ перехода на новую строку «\n» автоматически не добавляется, его добавляют в конце строки вручную. В результате выполнения этой программы в каталоге с проектом будет создан файл *Data.txt* в первой строке которого будет записано число «34», а во второй строке – число «–102.57».

Пример. Заполните список *N* случайными целыми числами в диапазоне [–10; 10]. Сохраните список в файле. Дополните файл строкой с информацией о сумме всех чисел и количестве отрицательных.

```
from random import randrange  # подключить randrange

n = int(input('Введите количество чисел '))
lst = [randrange(-10, 11) for _ in range(n)]

f = open('Data.txt', 'w')   # открыть файл для записи
cnt = 0                     # счётчик количества отрицательных
for num in lst:
    s = str(num)    # конвертировать число в строку
    f.write(s + '\n')     # записать строку в файл
    if num < 0:
        cnt += 1    # подсчёт количества отрицательных
f.write('Сумма чисел: ' + str(sum(lst))+'\n')
f.write('Количество отрицательных: ' + str(cnt)+'\n')
f.close()           # закрыть файл
```

Для вывода результатов работы программы использовалась встроенная функция `print()`. Синтаксис функции:

```
print(*objects, sep=' ', end='\n',
      file=sys.stdout, flush=False)
```

Она печатает набор объектов `objects`, разделенных запятой. При печати все объекты преобразуются в строки. Параметры функции:

- `sep` – разделитель при выводе нескольких объектов (по умолчанию – пробел);
- `end` – строка, завершающая вывод (по умолчанию – перенос строки);
- `file` – объект вывода (по умолчанию – терминал).

Таким образом, если не указывать значение параметра `file`, то выводимые значения отобразятся на экране. Если же указать в значении параметра `file` файловый указатель, то выводимые функцией **`print()`** объекты будут записаны в файл (открытый предварительно для записи или добавления), связанный с файловым указателем.

Пример использования функции `print()` для записи в файл:

```
f = open('Data.txt', 'w')   # открыть файл для записи
a, b = 3, 89
print(a, '+', b, '=', a + b, file=f)
f.close()                   # закрыть файл
```

В результате выполнения программы будет создан файл *Data.txt* со следующим содержимым:

```
3 + 89 = 92
```

Контекстные менеджеры

В Python для упрощения кода по выделению и высвобождению ресурсов предусмотрены специальные объекты – *менеджеры контекста*, которые могут самостоятельно следить за использованием ресурсов. Менеджер контекста для работы с файлом вызывается с использованием конструкции `with…as`:

```
with open('Data.txt', 'r') as file:
    for s in file:
        print(s, end='')
```

В первой строке файл *Data.txt* открывается в режиме чтения (`'r'`) и связывается с файловым указателем `file`. Затем в цикле перебираются все строки из файла, каждая из них по очереди попадает в переменную `s` и выводится на экран. Закрывать файл командой `close()` при использовании контекстного менеджера не нужно, он закроется автоматически после окончания цикла.

Использование контекстного менеджера при работе с файлами не защищает от возникновения исключительных ситуаций, связанных, например, с его отсутствием, недостатком места на диске при записи, ограничением прав доступа к файлу. Поэтому и при использовании конструкции `with…as` также рекомендуется использовать обработку исключительных ситуаций.

Пример использования контекстного менеджера при чтении строк из файла с обработкой исключений:

```
try:
    with open('Data.txt', 'r') as file:
        for s in file:
            print(s, end='')
except Exception as e:
    print('Ошибка при работе с файлом!', e)
else:
    print('Работа с файлом завершена.')
finally:
    print('Работа программы завершена.')
```

Пример выполнения задания

Сформировать текстовый файл *Data.txt* из N строк со случайным количеством (от 10^2 до 10^6) заглавных букв A, B, C, D, E, F, G латинского алфавита. Необходимо найти строку, содержащую наименьшее количество букв G (если таких строк несколько, надо взять ту, которая находится в файле раньше; строки, не содержащие букву G, не участ-

вуют в поиске), и определить, какая буква встречается в этой строке чаще всего. Если таких букв несколько, надо взять ту, которая позже стоит в алфавите. В файл *Result.txt* выведите следующую информацию, полученную в ходе обработки текста из файла *Data.txt*:

1) номер строки, содержащей наименьшее количество букв G;
2) буква, встречающаяся в этой строке чаще всего;
3) по каждой букве найденной строки вывести информацию о количестве её повторений в строке.

На рис. 5 приведен пример сгенерированного файла *Data.txt* (с небольшим количеством букв в строке) и файла *Result.txt*, содержащего информацию, полученную при анализе файла *Data.txt* при N=6.

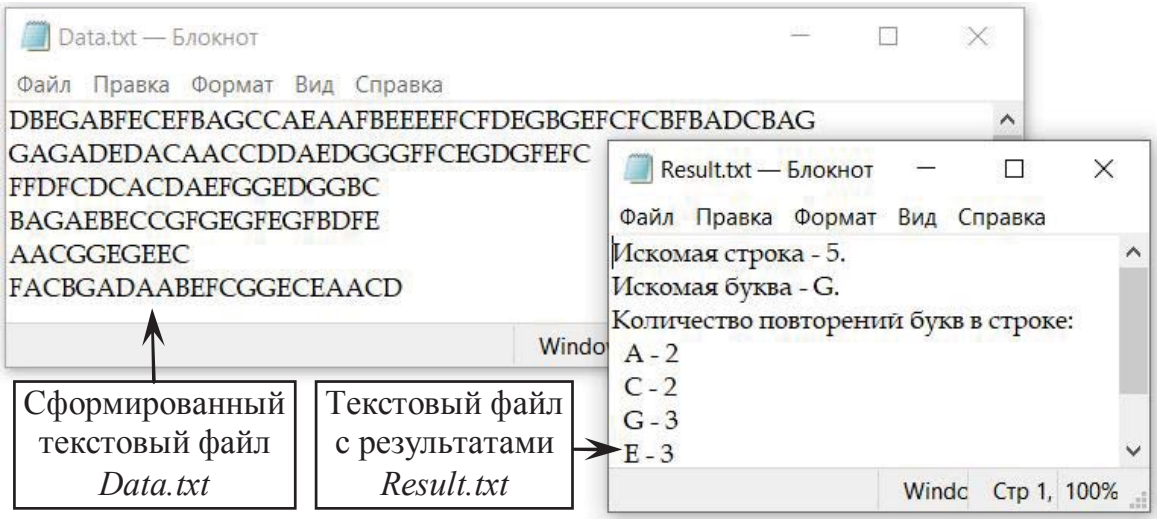

Рис. 5. Пример сформированного текстового файла *Data.txt* и файла *Result.txt* с результатами работы программы

В сформированном файле *Data.txt* в первой строке 5 букв G, во второй – 7, в третьей – 4, в четвёртой – 5, в пятой и шестой – 3. Искомая строка – пятая, так как она находится в файле раньше, чем шестая. В пятой строке буквы A и C встречаются 2 раза, буквы G и E –3 раза. Чаще других в пятой строке встречаются буквы G и E, выбираем букву G, так как она стоит в алфавите позже.

В программе при формировании текстового файла предварительно задаётся количество строк, которые будут записаны в файл. Для каждой строки:

1) случайным образом с помощью функции `randrange()` из модуля `random` определяется число образующих её символов;
2) формируется список из букв, который с помощью метода `join()` преобразуется в строку `s`;
3) полученная строка `s` записывается в файл *Data.txt*.

После формирования текстового файла он открывается для чтения. Для каждой строки определяются значения переменных: 1) `curr` – количество вхождений в неё буквы G; 2) `line` – порядковый номер строки в файле. Если в строке была обнаружена буква G и число её вхождений в строку меньше, чем в других строчках файла, то в переменной `number` запоминаем номер найденной строки, саму строку – в переменной `found`, а минимальное количество её вхождений – в переменной `minG`.

Значение переменной `number`, равное нулю, говорит о том, что ни в одной строке файла *Data.txt* не оказалось буквы G. В этом случае в файл *Result.txt* запишется соответствующее сообщение.

Если в файле *Data.txt* нашлась строка `found`, содержащая наименьшее количество букв G, то из её символов формируется частотный словарь `D`, в котором для каждой буквы определяется количество её вхождений в строку. Далее находится `mx` – максимальное значение в словаре и из ключей, соответствующих этому значению, формируется список `ans`. Если в списке `ans` окажется несколько букв, то нас будет интересовать та, что стоит в алфавите позже. Поэтому из отсортированного списка `ans` берётся последний элемент.

В файл *Result.txt* записываются найденные по заданию значения:

1) `number` – номер строки с наименьшим количеством букв G;
2) `ans[-1]` – буква, встречающаяся в этой строке чаще всего;
3) содержимое словаря `D` в виде пары ключ-значение.

Код программы:

```
from random import randrange

# Заполнение файла Data.txt
n = int(input('Введите количество строк '))
f = open('Data.txt', 'w')
for _ in range(n):
    cnt = randrange(100, 10**6+1) # число символов в строке
    s = ''.join([chr(randrange(ord('A'), ord('G')+1))
                 for _ in range(cnt)])
    f.write(s + '\n')
f.close()

# Поиск первой строки с наименьшим количеством букв G
f = open('Data.txt', 'r')
minG = 1_000_000
line, number = 0, 0
for s in f:
    line = line + 1
    curr = s.count('G')
    if 0 < curr < minG:
```

```
            minG, found, number = curr, s, line
    f.close()

    # Заполнение файла Result.txt
    f = open('Result.txt', 'w')
    if number == 0:
        f.write('В файле нет строк, содержащих букву G.\n')
    else:
        # Формирование частотного словаря
        D = {}
        for c in found:
            if c.isalpha():
                if c in D:
                    D[c] += 1
                else:
                    D[c] = 1
        # Поиск в словаре буквы, которая встречается чаще всего
        mx = max(D.values())
        ans = [key for key, value in D.items() if mx == value]
        ans.sort()
        # Вывод результатов в файл
        f.write('Искомая строка - {}.\n'.format(number))
        f.write('Искомая буква - {}.\n'.format(ans[-1]))
        f.write('Количество повторений букв в строке:\n')
        for c in D:
            f.write('  {} - {}\n'.format(c, D[c]))
    f.close()
```

Контрольные вопросы

1. Что такое файл? Какие типы файлов бывают?
2. Что такое файловая переменная?
3. Почему для работы с файлом используется не имя файла, а файловая переменная?
4. Какие режимы открытия файла вам известны?
5. Для чего необходимо закрывать файл, открытый для чтения или записи?
6. С помощью каких функций/ методов можно прочитать информацию из текстового файла?
7. Каким образом записать текстовую информацию в файл?
8. Каким образом при записи текстовой информации в файл определить количество записанных символов?
9. Каким образом установить файловый указатель в начало файла?
10. Для чего используется метод `tell()`?
11. Почему при работе с файлами удобно использовать контекстные менеджеры?
12. Какие исключительные ситуации могут возникнуть при работе с файлом и каким образом их можно обработать?

Задания

Во всех вариантах требуется написать программу с использованием метода нисходящего пошагового проектирования. Программа должна работать в двух режимах, номер которого определяется при запуске программы:

1) *режим формирования* файла *Data.txt* из *N* строк со случайным количеством (от 10 до 100) символов из некоторого набора;
2) *режим обработки* файла *Data.txt* в соответствии с заданием.

Всю выходную информацию записывать в файл *Result.txt*. Логически законченные участки вычислений необходимо оформить в виде подпрограмм. В программе предусмотреть обработку исключительных ситуаций, которые могут возникнуть при работе с файлами.

Возрастающей подпоследовательностью будем называть последовательность символов, расположенных в порядке увеличения их номера в кодовой таблице символов ASCII.

Убывающей подпоследовательностью будем называть последовательность символов, расположенных в порядке уменьшения их номера в кодовой таблице символов ASCII.

Под *числом* будем понимать последовательность подряд идущих цифр.

Под *расстоянием* между буквами будем понимать количество символов, расположенных между ними.

Палиндромом называется последовательность символов, которая читается в обе стороны одинаково.

Вариант 1. Сформировать текстовый файл *Data.txt* из заглавных букв *A*, *B*, *C*, *D* латинского алфавита и десятичных цифр. Найти номер строки, содержащей самую длинную подцепочку из символов *C* (если таких строк несколько, то выбрать ту, которая находится в файле позже), и определить в этой строке наибольшее число.

Вариант 2. Сформировать текстовый файл *Data.txt* из заглавных букв *X*, *Y*, *Z* латинского алфавита и десятичных цифр. Найти номер строки, в которой находится самое большое число, и определить в этой строке максимальное количество подряд идущих букв *Y*. Если строк с максимальным числом несколько, то выбрать ту, которая находится в файле раньше.

Вариант 3. Сформировать текстовый файл *Data.txt* из заглавных букв *A*, *B*, *C*, *D, E, F* латинского алфавита и десятичных цифр. Найти номер строки, содержащей самую длинную подцепочку из букв, не включающую символ *C* (если таких строк несколько, то выбрать ту,

которая находится в файле раньше), и определить в этой строке наименьшее число, кратное трём.

Вариант 4. Сформировать текстовый файл *Data.txt* из заглавных букв латинского алфавита. Найти номер строки, содержащей самую длинную возрастающую последовательность символов (если таких строк несколько, то выбрать ту, которая находится в файле раньше), и определить в этой строке максимальное количество идущих подряд символов, среди которых каждые два соседних различны.

Вариант 5. Сформировать текстовый файл *Data.txt* из заглавных букв *A*, *B*, *C*, *D*, *E*, *F* латинского алфавита и десятичных цифр. Найти номер строки, в которой находится самое большое нечётное число, и определить в ней самую длинную возрастающую последовательность из букв. Если строк с максимальным нечётным числом несколько, то выбрать ту, которая находится в файле позже.

Вариант 6. Сформировать текстовый файл *Data.txt* из заглавных букв *A*, *B*, *C*, *D* латинского алфавита и десятичных цифр. В тех строках, где букв меньше, чем цифр, определить размах между числами (разницу между максимальным и минимальным числами).

Вариант 7. Сформировать текстовый файл *Data.txt* из заглавных букв латинского алфавита и десятичных цифр. Найти номер строки, в которой находится самое большое чётное число, и определить в ней длину самой длинной подцепочки, не содержащей гласных букв. Если строк с максимальным чётным числом несколько, то выбрать ту, которая находится в файле раньше.

Вариант 8. Сформировать текстовый файл *Data.txt* из заглавных букв *A*, *B*, *C*, *D* латинского алфавита. Найти номер строки, содержащей самую длинную подцепочку из символов *B* (если таких строк несколько, то выбрать ту, которая находится в файле раньше), и определить максимальное расстояние между одинаковыми буквами в этой строке.

Вариант 9. Сформировать текстовый файл *Data.txt* из заглавных букв *X*, *Y*, *Z* латинского алфавита. Определить номер строки с самым длинным палиндромом. Для каждой буквы найденной строки определить частоту её появления в строке. Если строк с самым длинным палиндромом несколько, то выбрать ту, которая находится в файле раньше.

Вариант 10. Сформировать текстовый файл *Data.txt* из заглавных букв латинского алфавита. Найти номер строки, содержащей самую длинную возрастающую цепочку символов, и для каждой буквы этой строки определить частоту её появления в строке. Если таких строк несколько, то выбрать ту, которая находится в файле позже.

Вариант 11. Сформировать текстовый файл *Data.txt* из заглавных букв *A*, *B*, *C*, *D*, *E*, *F* латинского алфавита и пробела. Найти номер строки, в которой больше всего слов, которые начинаются и заканчиваются на одну и ту же букву. Если таких строк несколько, то выбрать ту, которая находится в файле раньше. Найти в найденной строке процентное соотношение между гласными и согласными буквами.

Вариант 12. Сформировать текстовый файл *Data.txt* из заглавных букв *A*, *B*, *C*, *D*, *E*, *F* латинского алфавита и десятичных цифр. Найти номер строки, в которой находится самое большое число, кратное трём, и определить в ней самую длинную убывающую цепочку символов. Если строк с максимальным числом, кратным трём, несколько, то выбрать ту, которая находится в файле раньше.

Вариант 13. Сформировать текстовый файл *Data.txt* из заглавных букв латинского алфавита и десятичных цифр. Найти номер строки, содержащей самую длинную возрастающую последовательность из букв (если таких строк несколько, то выбрать ту, которая находится в файле раньше), и определить в этой строке наибольшее число.

Вариант 14. Сформировать текстовый файл *Data.txt* из заглавных букв от *A* до *K* латинского алфавита и пробела. Найти номер строки с самым длинным словом (если таких строк несколько, то выбрать ту, которая находится в файле раньше). В найденной строке поменять порядок следования слов на обратный.

Вариант 15. Сформировать текстовый файл *Data.txt* из заглавных букв от *A* до *F* латинского алфавита и пробела. Определить во всём тексте *Букву_1*, с которой чаще всего начинаются слова, и *Букву_2*, на которую чаще всего заканчиваются слова (если вариантов начала и окончания несколько, то взять букву, которая позже встречается в алфавите). Вывести те слова из текста, которые начинаются на *Букву_1* и заканчиваются на *Букву_2*.

Вариант 16. Сформировать текстовый файл *Data.txt* из заглавных и строчных букв *V…Z*, *v…z* латинского алфавита и пробела. Найти номер строки с самым длинным словом (если таких строк несколько, то выбрать ту, которая находится в файле позже) и определить в этой строке процентное соотношение между строчными и заглавными буквами.

Вариант 17. Сформировать текстовый файл *Data.txt* из заглавных букв латинского алфавита. Найти номер строки, содержащей самую длинную убывающую цепочку символов (если таких строк несколько, то выбрать ту, которая находится в файле раньше), и определить символ, который чаще всего встречается в этой строке между двумя оди-

наковыми символами. Если таких символов несколько, вывести тот, который стоит в алфавите позже.

Вариант 18. Сформировать текстовый файл *Data.txt* из заглавных и строчных букв *V…Z*, *v…z* латинского алфавита и пробела. Найти номер строки с самым коротким словом (если таких строк несколько, то выбрать ту, которая находится в файле раньше) и определить символ, который чаще всего встречается в этой строке между двумя одинаковыми символами (без учета регистра). Если таких символов несколько, вывести тот, который стоит в алфавите позже.

Вариант 19. Сформировать текстовый файл *Data.txt* из заглавных букв латинского алфавита и десятичных цифр. Найти номер строки, в которой находится самое большое количество чётных чисел (если таких строк несколько, то выбрать ту, которая находится в файле раньше), и для каждой буквы этой строки определить частоту её появления в строке.

Вариант 20. Сформировать текстовый файл *Data.txt* из заглавных и строчных букв *X*, *Y*, *Z*, *x*, *y*, *z* латинского алфавита и пробела. Найти номер строки с самым коротким словом (если таких строк несколько, то выбрать ту, которая находится в файле раньше) и вывести все слова-палиндромы, содержащиеся в этой строке (без учета регистра).

Вариант 21. Сформировать текстовый файл *Data.txt* из заглавных и строчных букв *A*, *B*, *C*, *a*, *b*, *c* латинского алфавита и пробела. Найти номер строки с самым длинным словом (если таких строк несколько, то выбрать ту, которая находится в файле позже) и для каждой буквы этой строки определить частоту её появления в строке (без учета регистра).

Вариант 22. Сформировать текстовый файл *Data.txt* из заглавных букв латинского алфавита и десятичных цифр. В тех строках, где букв меньше цифр, определить количество чётных чисел и минимальное чётное число. В выходных строках записывать через пробел номер найденной строки, количество чётных чисел и минимальное чётное число в этой строке.

Вариант 23. Сформировать текстовый файл *Data.txt* из заглавных букв *A*, *B*, *C*, *D* латинского алфавита. Найти номер строки, содержащей самую длинную цепочку вида ABDABDABD... (состоящей из фрагментов ABD, последний фрагмент может быть неполным), и для каждой буквы этой строки определить частоту её появления в строке. Если искомых строк несколько, то выбрать ту, которая находится в файле раньше.

Вариант 24. Сформировать текстовый файл *Data.txt* из заглавных букв *A*, *B*, *C*, *D*, *E*, *F* латинского алфавита. Найти номер строки, содер-

жащей самую длинную подцепочку из букв, не включающую символ *C* (если таких строк несколько, то выбрать ту, которая находится в файле позже), и определить максимальное расстояние между одинаковыми буквами в этой строке.

Вариант 25. Сформировать текстовый файл *Data.txt* из заглавных букв латинского алфавита и десятичных цифр. Найти номер строки, в которой находится самое маленькое количество чётных чисел (если таких строк несколько, то выбрать ту, которая находится в файле раньше), и определить в этой строке процентное соотношение между буквами и цифрами.

Вариант 26. Сформировать текстовый файл *Data.txt* из заглавных букв *A*, *B*, *C*, *D, E, F* латинского алфавита и десятичных цифр. Найти номер строки, содержащей самую длинную подцепочку из букв, не включающую символ *A* (если таких строк несколько, то выбрать ту, которая находится в файле позже), и определить в этой строке наибольшее число, в котором цифры образуют возрастающую последовательность.

Вариант 27. Сформировать текстовый файл *Data.txt* из заглавных букв *A*, *B*, *C* латинского алфавита и десятичных цифр. Найти номер строки, содержащей самую длинную подцепочку из символов *B* (если таких строк несколько, то выбрать ту, которая находится в файле позже), и определить в этой строке наибольшее число, в котором цифры образуют убывающую последовательность.

Вариант 28. Сформировать текстовый файл *Data.txt* из заглавных букв латинского алфавита. Найти номер строки, содержащей самую длинную подцепочку из разных символов (каждый символ цепочки встречается не более одного раза). Если таких строк несколько, то выбрать ту, которая находится в файле раньше. Определить три символа, которые чаще всего встречаются в этой строке.

Вариант 29. Сформировать текстовый файл *Data.txt* из заглавных и строчных букв латинского алфавита. Найти номер строки, в которой находится самое большое количество гласных букв (если таких строк несколько, то выбрать ту, которая находится в файле раньше), и для этой строки вывести в алфавитном порядке те латинские буквы (без учёта регистра), которые в неё не вошли.

Вариант 30. Сформировать текстовый файл *Data.txt* из заглавных букв латинского алфавита. Найти номер строки, содержащей самую длинную возрастающую последовательность символов (если таких строк несколько, то выбрать ту, которая находится в файле позже). Определить три символа, которые чаще всего встречаются в этой строке.