Лабораторная работа № 12
Файлы и исключения

1 Цель работы

Изучить файлы и исключения применять полученные знания на практике.

2 Краткая теория

Все программы, которые вы писали до сих пор, считывали данные с клавиатуры. В связи с этим пользователю необходимо было повторять ввод каждый раз, когда программа запускалась. Но это бывает не слишком удобно, особенно если речь идет о большом массиве входных данных. Также все ваши предыдущие программы выводили результаты исключительно на экран. Это нормально, если дело касается нескольких строчек, предназначенных для пользователя программы. Но если выходных данных будет много или они могут потребоваться в дальнейшем для анализа в другой программе, вариант с выводом на экран просто не подойдет. Эффективная работа с файлами решит обе перечисленные проблемы.

Файлы можно назвать относительно постоянным местом хранения информации. Данные, которые записываются в файл в процессе выполнения программы, остаются в нем после завершения ее работы и даже после выключения компьютера. Это позволяет хранить в файлах информацию, которая может быть необходимой на протяжении определенного периода времени или служить источником данных для сторонних программ, запускаемых многократно. Вы наверняка сталкивались в работе с текстовыми файлами, электронными таблицами, изображениями и видео. Да и сами программы на языке Python хранятся в файлах.

Файлы обычно делятся на текстовые и двоичные, или бинарные. В текстовых файлах хранятся исключительно последовательности битов, представляющие собой символы в определенной кодировке, например ASCII или UTF-8. Подобные файлы можно просматривать и редактировать при помощи текстовых редакторов. Все программы, которые вы писали до сих пор, сохранялись на диске в виде текстовых файлов.

Подобно текстовым, двоичные файлы также хранят последовательность битов, представляющую определенные данные. Отличие состоит в том, что эти данные не ограничиваются одними только символами. Файлы, содержащие изображения, звук или видео, являются типичными примерами двоичных файлов. В данной книге мы ограничимся работой с текстовыми файлами, поскольку их легко создавать и просматривать в вашем любимом редакторе. При этом большинство действий и принципов, применяемых к текстовым файлам, прекрасно работают и в отношении двоичных файлов.

2.1 Открытие файлов

Чтобы начать процесс чтения данных из файла, его предварительно следует открыть. То же самое необходимо сделать и перед записью информации в файл. Открыть файл в Python можно при помощи функции open.

Функция open принимает два аргумента. Первый из них указывает имя файла, который необходимо открыть. Второй аргумент также является текстовым и характеризует режим доступа (access mode) к файлу. Мы будем работать с тремя режимами доступа к файлу: на чтение (r), запись (w) и добавление (a).

В качестве выходного значения функция open возвращает файловый объект (file object). При этом функция с аргументами обычно указывается справа от знака присваивания, а переменная, характеризующая файловый объект, – слева, как показано ниже.

```
inf = open("input.txt", "r")
```

После открытия файла к соответствующему ему файловому объекту можно применять методы, позволяющие извлекать информацию. То же самое можно сказать и о записи данных в файл – методы будут другие, а принципы те же. Эти методы мы подробно рассмотрим в следующих разделах лабораторной работы. После завершения чтения или записи файл должен быть закрыт, для чего применяется соответствующий метод close.

2.2 Чтение из файла

Существуют разные методы для чтения данных из открытого ранее файла. Все они могут применяться только при условии, что файл открыт в режиме чтения. Попытка прочитать данные из файла, который открыт на запись или добавление, приведет к возникновению ошибки.

Метод readline позволяет считать одну строку из открытого файла и вернуть ее в качестве строкового значения – подобно тому, как функция input считывает пользовательский ввод с клавиатуры. Каждый очередной вызов метода readline будет возвращать следующую строку из файла. При достижении конца файла метод вернет пустую строку.

Представьте, что у вас есть файл, в котором на каждой строке располагаются числа. В следующем фрагменте кода мы подсчитаем их сумму.

```
# Запрашиваем у пользователя имя файла и открываем его
fname = input("Введите имя файла: ")
inf = open(fname, "r")
# Инициализируем сумму
total = 0
# Подсчитываем сумму
line = inf.readline()
while line != "":
    total += float(line)
    line = inf.readline()
# Закрываем файл
inf.close()
# Отображаем результат
print("Сумма значений в файле", fname, "равна", total)
```

В начале программы мы запрашиваем у пользователя имя файла, который необходимо открыть. После этого указанный файл открывается в режиме чтения, а ссылка на него возвращается в переменную inf. Затем инициализируем общую сумму нулем и считываем первую строку из файла.

В цикле while указано условное выражение, пропускающее в тело цикла только в случае, если считанная строка не является пустой. В самом теле считанная из файла строка преобразуется в число с плавающей запятой и добавляется к общей сумме. После этого из файла считывается следующая строка, и управление снова передается условному выражению цикла.

По достижении конца файла метод readline вернет пустую строку, которая будет сохранена в переменной line. Это приведет к тому, что условное выражение вернет значение False, и цикл завершится. После закрытия файла на экран выводится общая сумма.

Иногда бывает полезно считать все данные из файла за один заход, а не построчно. Это можно сделать при помощи методов read и readlines. При этом метод read возвращает все содержимое файла как одну длинную строку, которую при дальнейшей обработке приходится вручную разбивать на составляющие элементы. Метод readlines, в свою очередь, возвращает содержимое файла в виде списка строк. После завершения чтения можно пройти по созданному списку при помощи цикла и на каждой итерации извлекать по одной строке. В следующем фрагменте кода демонстрируется метод readlines для подсчета суммы значений из исходного файла. Здесь происходит полное считывание содержимого файла вместо получения каждой отдельной строки в цикле.

```
# Запрашиваем у пользователя имя файла и открываем его
fname = input("Введите имя файла: ")
inf = open(fname, "r")
# Инициализируем сумму
total = 0
lines = inf.readlines()
# Подсчитываем сумму
for line in lines:
    total += float(line)
# Закрываем файл
inf.close()
# Отображаем результат
print("Сумма значений в файле", fname, "равна", total)
```

2.3 Символы конца строки

В следующем примере метод readline используется для последовательного вывода всех строк из файла. Каждой строке будет предшествовать ее порядковый номер и двоеточие.

```
# Запрашиваем у пользователя имя файла и открываем его
fname = input("Введите имя файла для отображения: ")
inf = open(fname, "r")
# Инициализируем порядковый номер строки
num = 1
# Отображаем строки из файла, предваряя их порядковыми номерами
line = inf.readline()
while line != "":
    print(f"{num}: {line}")
    # Увеличиваем номер строки и считываем следующую строку
    num += 1
    line = inf.readline()
```

```
# Закрываем файл
inf.close()
```

При запуске данной программы вы будете удивлены тем, что она выведет – после каждой строки будет следовать еще одна строка, пустая. Причина этого в том, что каждая строка в текстовом файле оканчивается одним или несколькими символами конца строки.

Примечание. Символы конца строки и их количество отличаются в разных операционных системах. К счастью, в Python существует автоматическая поддержка этих различий, и текстовый файл, созданный в любой операционной системе, может быть корректно прочитан в Python.

Символы конца строки необходимы для того, чтобы любая программа, считывающая файл, могла без труда определить, где заканчивается одна строка и начинается другая. Если бы не эти символы, при любом считывании данных все содержимое файла оказывалось бы в вашей переменной, а при открытии файла в текстовом редакторе все строки были бы неразрывно связаны.

Маркер окончания строки может быть удален после чтения из файла при помощи метода rstrip. Этот метод, который может быть применен к абсолютно любой строке, избавляет ее от всех примыкающих справа пробельных символов (whitespace character), в число которых входят пробел, символ табуляции и маркеры конца строки. Метод возвращает копию исходной строки без этих специальных символов в конце строки.

Обновленный фрагмент кода программы представлен ниже. Здесь используется метод rstrip для удаления символов конца строки, что позволило избавиться от пустых строк при выводе содержимого файла на экран.

```
# Запрашиваем у пользователя имя файла и открываем его
fname = input("Введите имя файла для отображения: ")
inf = open(fname, "r")
# Инициализируем порядковый номер строки
num = 1
# Отображаем строки из файла, предваряя их порядковыми номерами
line = inf.readline()
while line != "":
    # Удаляем символы конца строки и отображаем строку на экране
    line = line.rstrip()
    print(f"{num}: {line}")
    # Увеличиваем номер строки и считываем следующую строку
    num += 1
    line = inf.readline()
# Закрываем файл
inf.close()
```

2.4 Запись в файл

Когда файл открывается в режиме записи, происходит создание нового файла. В случае если файл с таким именем уже существует, он будет предварительно удален, а все данные, содержащиеся в нем, будут потеряны. Если открыть существующий файл в режиме добавления, любые данные, которые будут записываться в него, будут добавляться в конец файла. При

отсутствии на диске файла попытка его открытия в режиме добавления приведет к созданию нового файла с таким именем.

Для пополнения информацией файла, предварительно открытого на запись или добавление, можно использовать метод write. Этот метод принимает в качестве входного параметра строку, которая будет записана в открытый файл. Данные других типов могут быть преобразованы в строковый тип при помощи функции str. Несколько значений могут быть записаны в файл путем их предварительного склеивания в одну длинную строку или посредством многократного вызова метода write.

В отличие от функции print, метод write автоматически не вставляет символ перевода строки при записи в файл. Таким образом, разработчику необходимо самому позаботиться о том, чтобы значения, которые должны находиться в файле на разных строках, были явно разделены символами конца строки. В языке Python используется сочетание символов \n для обозначения маркера конца строки. Эти символы, входящие в список escape-последовательностей (escape sequence), могут присутствовать в строке сами по себе или как ее составная часть.

В следующем фрагменте кода мы запишем числа от единицы до введенного пользователем значения в файл. Операция конкатенации и escape-последовательность \n используются так, чтобы числа располагались строго на своих строках.

```
# Запрашиваем имя файла у пользователя и открываем его на запись
fname = input("В каком файле сохранить последовательность чисел? ")
outf = open(fname, "w")
# Запрашиваем максимальное значение
limit = int(input("Введите максимальное число: "))
# Пишем числа в файл - каждое в своей строке
for num in range(1, limit + 1):
    outf.write(f"{num}\n")
# Закрываем файл
outf.close()
```

2.5 Аргументы командной строки

Обычно компьютерные программы запускаются путем двойного щелчка по соответствующему ярлыку. Но программы также могут быть запущены из командной строки путем ввода соответствующей команды в окно терминала или окно управления командной строкой. Например, во многих операционных системах программу на Python, сохраненную в файле test.py, можно запустить, написав test.py или python test.py в соответствующем окне.

Запуск программы из командной строки открывает дополнительные возможности для передачи ей параметров. Значения, необходимые программе для выполнения своей задачи, могут быть переданы непосредственно в запускающей строке после расширения файла .py. Такая возможность бывает очень полезна при написании скриптов, запускающих другие программы с целью автоматизации каких-либо процессов, а также при создании программ, которые должны запускаться по определенному расписанию.

Все аргументы командной строки, передаваемые в программу, сохраняются во внутренней переменной argv, располагающейся в модуле sys. По своей сути эта переменная является списком, и каждый входной параметр размещается в нем в виде отдельного строкового элемента. При этом любой элемент из списка может быть преобразован в нужный тип данных при помощи стандартных функций вроде int и float. Первым элементом списка argv является имя файла Python, который в данный момент запущен. Следом за ним идут переданные посредством командной строки параметры.

На примере следующей программы мы продемонстрируем доступ к аргументам командной строки непосредственно из кода. Программа начинается с вывода на экран количества элементов в переменной argv и имени запущенного файла. После этого на экран выводятся все переданные программе параметры. Если программа была запущена без параметров, будет показано соответствующее сообщение.

```
# Для доступа к переменным командной строки необходимо импортировать модуль sys
import sys
# Отображаем количество аргументов (включая название файла .py)
print("Программа насчитывает", len(sys.argv), "аргументов командной строки.")
# Выводим на экран имя файла .py
print("Имя запущенного файла .py: ", sys.argv[0])
# Определяем, есть ли другие переданные параметры
if len(sys.argv) > 1:
    # Отображаем все переданные параметры, за исключением имени файла
    print("Остальные аргументы:")
    for i in range(1, len(sys.argv)):
        print(" ", sys.argv[i])
else:
    print("Дополнительных аргументов нет.")
```

Аргументы командной строки можно использовать для передачи программе любых входных значений, которые можно физически ввести в командную строку, то есть строк, целочисленных значений и чисел с плавающей запятой. Эти значения могут быть использованы в программе, как

и любые другие. Взгляните на обновленный пример программы, суммирующей перечисленные в строках числа. Здесь имя файла мы не будем запрашивать у пользователя, а передадим посредством командной строки.

```
# Импортируем системный модуль
import sys
# Отслеживаем, чтобы программа обязательно запускалась с одним переданным параметром
if len(sys.argv) != 2:
    print("Имя файла необходимо передать в качестве аргумента.")
    quit()
# Открываем на чтение файл, имя которого было передано в командной строке
inf = open(sys.argv[1], "r")
# Инициализируем сумму нулем
total = 0
# Суммируем значения в файле
line = inf.readline()
```

```
while line != "":
    total += float(line)
    line = inf.readline()
# Закрываем файл
inf.close()
# Выводим результат
print("Сумма значений в файле", sys.argv[1], "составляет", total)
```

2.6 Исключения

Во время выполнения программы много что может пойти не так: пользователь может ввести строковое значение вместо числового, может возникнуть ошибка деления на ноль, пользователь может запросить на открытие файл, которого не существует, да мало ли что. Все подобные ошибки называются исключениями (exception). По умолчанию программы, написанные на языке Python, прекращают свое выполнение при возникновении исключений. Но мы всегда можем предотвратить такое поведение, если заранее предпримем ряд мер.

Разработчик имеет возможность указать программе, в каком месте может возникнуть исключение, чтобы была возможность его перехватить и соответствующим образом обработать, тем самым защитив программу от аварийного завершения. Для этого в языке Python предусмотрено два ключевых слова, с которыми мы раньше не сталкивались: это try и except. Код, который может вызвать возникновение исключения, предусмотрительно помещается в блок try, следом за которым располагается альтернативный блок except. Когда в блоке try возникает исключение, выполнение программы немедленно переносится в соответствующий блок except без выполнения оставшихся инструкций в блоке try.

В каждом блоке except может быть явно указан тип исключения, для перехвата которого предназначен этот блок. Это можно сделать, указав необходимый тип исключения непосредственно после ключевого слова except. В этом случае только указанный тип исключения будет обрабатываться при помощи данного блока. Без указания конкретного типа исключения блок except будет перехватывать все возникающие исключения, которые ранее не были обработаны другими блоками except, принадлежащими тому же блоку try. Необходимо четко понимать, что инструкции из блока except будут выполнены только в случае возникновения исключения. Если же выполнение блока try пройдет нормально, следующие за ним блоки except будут просто пропущены, и выполнение программы продолжится с первой строки кода, следующей за последним блоком except соответствующего блока try.

Все программы, которые мы писали до сих пор в этой лабораторной работе, аварийно завершали свою работу при вводе пользователем несуществующего имени файла. Это объясняется тем, что в данный момент возникало исключение FileNotFoundError, которое не было перехвачено. В следующем фрагменте кода мы исправим это упущение и обработаем данное исключение с выводом соответствующего сообщения. После указанного фрагмента можно добавить любой код для обработки информации из открытого файла.

```
# Запрашиваем имя файла у пользователя
fname = input("Введите имя файла: ")
# Попытка открыть файл
try:
    inf = open(fname, "r")
except FileNotFoundError:
    # Показываем сообщение и выходим из программы, если возникла проблема при
открытии файла
    print("Невозможно открыть файл '%s'. Выходим...")
    quit()
```

Данная версия программы будет аккуратно закрываться с предупреждением, если файл, к которому хочет обратиться пользователь, не существует по указанному адресу. И хотя иногда такое поведение программы будет приемлемым, чаще мы захотим дать пользователю возможность повторно ввести имя файла для открытия. При этом и со второй попытки у него может не получиться ввести правильное имя файла. Организуем цикл, успешный выход из которого возможен только после ввода корректного имени файла. Обратите внимание, что оба блока – try и except – находятся внутри цикла while.

```
# Запрашиваем имя файла у пользователя
fname = input("Введите имя файла: ")
file_opened = False
while file_opened == False:
    # Попытка открыть файл
    try:
        inf = open(fname, "r")
        file_opened = True
    except FileNotFoundError:
        # Показываем сообщение и запрашиваем имя файла повторно
        print("Файл '%s' не найден. Попробуйте еще.")
        fname = input("Введите имя файла: ")
```

В начале программы у пользователя запрашивается имя файла. После этого переменной file_opened присваивается значение False, и цикл запускается в первый раз. В теле цикла размещается блок try с двумя строками кода. В первой из них выполняется попытка открытия файла. Если файла нет, будет сгенерировано исключение типа FileNotFoundError, в результате чего выполнение программы продолжится в соответствующем блоке except, тогда как вторая строка в блоке try так и останется невыполненной. В блоке except на экран выводится сообщение об ошибке и производится повторный запрос имени файла у пользователя.

Выполнение программы возвращается к первой строке цикла, где происходит проверка условия file_opened == False. Это условие возвращает истину, и программа вновь входит в блок try, пытаясь открыть файл, имя которого пользователь ввел повторно. Если и такого файла не существует, все повторится согласно описанному выше алгоритму. Если же файл найти удалось, исключения не возникает, и выполнение программы переходит ко второй строке в блоке try, где переменной file_opened присваивается значение True. В результате блок except пропускается, цикл возвращается на начало и тут же завершается, поскольку условие цикла не выполняется.

Концепция, описанная в данной лабораторной работе, может быть использована для идентификации и обработки всех возможных исключений, которые могут возникать в процессе выполнения программы. Наличие блоков try и except позволит вам уберечь свою программу от аварийного завершения и должным образом обрабатывать все возникающие ошибки.

3 Порядок выполнения работы

Получить задание для выполнения лабораторной работы (раздел 4) согласно своему варианту (по журналу). Разработать программу.

4 Задания для выполнения работы

В большинстве заданий из данной лабораторной работы вам придется работать с файлами. В каких-то из них содержание файлов будет не важно, в других вам придется заранее создать и заполнить их при помощи любого текстового редактора. Будут в этой лабораторной работе и задания на чтение конкретных данных, таких как список слов, имен или химических элементов. Эти наборы данных можно скачать по следующим ссылкам:

1. Список химических элементов, включающий атомный номер, символ и название каждого элемента: https://disk.yandex.ru/d/HPIwcVk6THl5zw.
2. Список из более чем 200 000 английских слов: https://disk.yandex.ru/d/MYepBAkM_50fTg.
3. Коллекция файлов, содержащая 100 наиболее часто используемых имен для мальчиков и девочек с 1900 по 2012 год: https://disk.yandex.ru/d/cQB_Y-FJKGOZpA.

Задание 1. Отображаем начало файла.

В операционных системах на базе Unix обычно присутствует утилита с названием head. Она выводит первые десять строк содержимого файла, имя которого передается в качестве аргумента командной строки. Напишите программу на Python, имитирующую поведение этой утилиты. Если файла, указанного пользователем, не существует, или не задан аргумент командной строки, необходимо вывести соответствующее сообщение об ошибке.

Задание 2. Отображаем конец файла.

Продолжая тему предыдущего задания, в тех же операционных системах на базе Unix обычно есть и утилита с названием tail, которая отображает последние десять строк содержимого файла, имя которого передается в качестве аргумента командной строки. Реализуйте программу, которая будет делать то же самое. Так же, как и в задании 1, в случае отсутствия файла, указанного пользователем, или аргумента командной строки вам нужно вывести соответствующее сообщение. Данную задачу можно решить сразу несколькими способами. Например, можно все содержимое файла целиком загрузить в список и затем выбрать из него последние десять элементов. А можно дважды прочитать содержимое файла: первый раз, чтобы посчитать количество строк, а второй – чтобы отобразить последние десять из них. При этом оба перечисленных подхода нежелательны, если речь идет о файлах достаточного объема. Существует решение, требующее единственного чтения

файла и сохранения всех десяти строк за раз. В качестве дополнительного задания разработайте такой алгоритм.

Задание 3. Сцепляем файлы.

Продолжаем тему операционных систем на базе Unix, в которых обычно также есть утилита с названием cat, что является сокращением от concatenate (сцепить). Эта утилита выводит на экран объединенное содержимое нескольких файлов, имена которых передаются ей в качестве аргументов командной строки. При этом файлы сцепляются в том порядке, в котором указаны в аргументах. Напишите программу на Python, имитирующую работу этой утилиты. В процессе работы программа должна выдавать сообщения о том, какие файлы открыть не удается, и переходить к следующим файлам. Если программа была запущена без аргументов командной строки, на экран должно быть выведено соответствующее сообщение об ошибке.

Задание 4. Нумеруем строки в файле.

Напишите программу, которая будет считывать содержимое файла, добавлять к считанным строкам порядковый номер и сохранять их в таком виде в новом файле. Имя исходного файла необходимо запросить у пользователя, так же, как и имя целевого файла. Каждая строка в созданном файле должна начинаться с ее номера, двоеточия и пробела, после чего должен идти текст строки из исходного файла.

Задание 5. Самое длинное слово в файле.

В данном задании вы должны написать программу, которая будет находить самое длинное слово в файле. В качестве результата программа должна выводить на экран длину самого длинного слова и все слова такой длины. Для простоты принимайте за значимые буквы любые непробельные символы, включая цифры и знаки препинания.

Задание 6. Частота букв в файле.

Одна из техник декодирования простейших алгоритмов шифрования заключается в применении частотного анализа. Иными словами, вы просто анализируете зашифрованный текст, подсчитывая частоту употребления всех букв. Затем можно использовать операции подстановки для замены наиболее популярных символов на часто используемые в языке буквы (в английском это, например, буквы E и T).

Напишите программу, которая будет способствовать дешифрации текста путем вывода на экран частоты появления разных букв. При этом пробелы, знаки препинания и цифры должны быть проигнорированы. Также не должен учитываться регистр, то есть символы a и A должны восприниматься как одна буква. Имя файла для анализа пользователь должен передавать программе посредством аргумента командной строки. Если программе не удастся открыть файл для анализа или аргументов командной строки будет слишком много, на экране должно быть отображено соответствующее сообщение об ошибке.

Задание 7. Частота слов в файле.

Разработайте программу, которая будет показывать слово (или слова), чаще остальных встречающееся в текстовом файле. Сначала пользователь

должен ввести имя файла для обработки. После этого вы должны открыть файл и проанализировать его построчно, разделив при этом строки по словам и исключив из них пробелы и знаки препинания. Также при подсчете частоты появления слов в файле вам стоит игнорировать регистры.

Задание 8. Сумма чисел.

Напишите программу, которая будет суммировать все числа, введенные пользователем, игнорируя при этом нечисловой ввод. Выводите на экран текущую сумму чисел после каждого очередного ввода. Ввод пользователем значения, не являющегося числовым, должен приводить к появлению соответствующего предупреждения, после чего пользователю должно быть предложено ввести следующее число. Выход из программы будет осуществляться путем пропуска ввода. Удостоверьтесь, что ваша программа корректно обрабатывает целочисленные значения и числа с плавающей запятой.

Подсказка. В данном задании вам придется поработать не с файлами, а с исключениями.

Задание 9. Буквенные и числовые оценки.

Напишите программу, выполняющую перевод из буквенных оценок в числовые и обратно. Программа должна позволять пользователю вводить несколько значений для перевода – по одному в каждой строке. Для начала предпримите попытку сконвертировать введенное пользователем значение из числового в буквенное. Если возникнет исключение, попробуйте выполнить обратное преобразование – из буквенного в числовое. Если и эта попытка окончится неудачей, выведите предупреждение о том, что введенное значение не является допустимым. Пусть ваша программа конвертирует оценки до тех пор, пока пользователь не оставит ввод пустым.

Задание 10. Удаляем комментарии.

В языке Python для создания комментариев в коде используется символ #. Комментарий начинается с этого символа и продолжается до конца строки – без возможности остановить его раньше.

В данном задании вам предстоит написать программу, которая будет удалять все комментарии из исходного файла с кодом на языке Python. Пройдите по всем строкам в файле на предмет поиска символа #. Обнаружив его, программа должна удалить все содержимое, начиная с этого символа и до конца строки. Для простоты не будем рассматривать ситуации, когда знак решетки встречается в середине строки. Сохраните новое содержимое в созданном файле. Имена файла источника и файла назначения должны быть запрошены у пользователя. Удостоверьтесь в том, что программа корректно обрабатывает возможные ошибки при работе с обоими файлами.

Задание 11. Случайный пароль из двух слов.

Создание пароля посредством генерирования случайных символов может обернуться сложностью в запоминании полученной относительно надежной последовательности. Некоторые системы создания паролей рекомендуют сцеплять вместе два слова на английском языке, тем самым

упрощая запоминание заветного ряда символов – правда, в ущерб его надежности.

Напишите программу, которая будет открывать файл со списком слов, случайным образом выбирать два из них и сцеплять вместе для получения итогового пароля. При создании пароля исходите из следующего требования: он должен состоять минимум из восьми символов и максимум из десяти, а каждое из используемых слов должно быть длиной хотя бы в три буквы. Кроме того, сделайте заглавными первые буквы обоих слов, чтобы легко можно было понять, где заканчивается одно и начинается другое. По завершении процесса полученный пароль должен быть отображен на экране.

Задание 12. Странные слова.

Ученикам, желающим запомнить правила написания слов в английском языке, часто напоминают следующее рифмованное одностишие: «I before E except after C» (I перед E, если не после C). Это правило позволяет запомнить, в какой последовательности писать буквы I и E, идущие в слове одна за другой, а именно: буква I должна предшествовать букве E, если непосредственно перед ними не стоит буква C. Если стоит – порядок гласных будет обратным. Примеры слов, на которые действует это правило: believe, chief, fierce, friend, ceiling и receipt. Но есть и исключения из этого правила, и одним из них является слово weird (странный).

Напишите программу, которая будет построчно обрабатывать текстовый файл. В каждой строке может присутствовать много слов, а может и не быть ни одного. Слова, в которых буквы E и I не соседствуют друг с другом, обработке подвергать не следует. Если же такое соседство присутствует, необходимо проверить, соответствует ли написание анализируемого слова указанному выше правилу. Создайте и выведите на экран два списка. В первом должны располагаться слова, следующие правилу, а во втором – нарушающие его. При этом списки не должны содержать повторяющиеся слова. Также отобразите на экране длину каждого списка, чтобы пользователю было понятно, сколько слов в файле не отвечает правилу.

Задание 13. Что за химический элемент.

Напишите программу, которая будет считывать файл, содержащий информацию о химических элементах, и сохранять ее в более подходящей для этого структуре данных. После этого пользователь должен ввести значение. Если введенное значение окажется целочисленным, программа должна вывести на экран обозначение и название химического элемента с введенным количеством протонов. При вводе пользователем строки необходимо отобразить количество протонов элемента с введенным пользователем обозначением или названием. Если введенное пользователем значение не соответствует ни одному из элементов в файле, необходимо вывести соответствующее сообщение об ошибке. Позвольте пользователю вводить значения до тех пор, пока он не оставит ввод пустым.

Задание 14. Книги без буквы E.

Истории литературы известен случай написания романа объемом около 50 тыс. слов, в котором ни разу не была употреблена самая популярная в английском алфавите буква E. Название его – «Gadsby».

Напишите программу, которая будет считывать список слов из файла и собирать статистику о том, в каком проценте слов используется каждая буква алфавита. Выведите результат для всех 26 букв английского алфавита и отдельно отметьте букву, которая встречалась в словах наиболее редко. В вашей программе должны игнорироваться знаки препинания и регистр символов.

Примечание. Липограмма представляет собой литературный прием, состоящий в написании текста без использования одной из букв (или группы букв). Часто отвергнутой буквой является одна из распространенных гласных, хотя это условие и не обязательно. Например, в стихотворении Эдгара Аллана По «Ворон» («The Raven»), состоящем из более тысячи слов, ни разу не встречается буква Z, что делает его полноценной липограммой. Еще одним примером липограммы является роман «Исчезание» («La Disparition»). Во французской и английской версиях этого произведения общим объемом примерно в 300 страниц не употребляется буква E, за исключением фамилии автора.

Задание 15. Популярные детские имена.

Набор данных, содержащий детские имена, состоит более чем из 200 файлов, в каждом из которых, помимо сотни имен, указано количество названных тем или иным именем детей. При этом файлы отсортированы по убыванию популярности имен. Для каждого года присутствует по два файла: в одном перечислены мужские имена, в другом – женские. Совокупный набор данных содержит информацию для всех лет, начиная с 1900-го и заканчивая 2012-м.

Напишите программу, которая будет считывать по одному все файлы из набора данных и выделять имена, которые были лидерами по частоте использования как минимум в одном году. На выходе должно получиться два списка: в одном из них будут присутствовать наиболее популярные имена для мальчиков, во втором – для девочек. При этом списки не должны содержать повторяющиеся имена.

Задание 16. Универсальные имена.

Некоторые имена, такие как Бен, Джонатан и Эндрю, подходят только для мальчиков, другие – Ребекка или Флора – только для девочек. Но есть и универсальные имена наподобие Крис или Алекс, которые могут носить и мальчики, и девочки.

Напишите программу, которая будет выводить на экран имена, использованные для мальчиков и девочек в указанном пользователем году. Если в этом году универсальных имен не было, нужно известить об этом пользователя. Кроме того, если за указанный пользователем год не было данных по именам, выведите соответствующее сообщение об ошибке. Дополнительные детали по хранению имен в файлах – в задании 15.

Задание 17. Самые популярные имена за период.

Напишите программу, которая будет определять самые популярные имена для детей в выбранном пользователем периоде. Используйте базу данных из задания 15. Позвольте пользователю выбрать первый и последний год анализируемого диапазона. В результате программа должна вывести на экран мужское и женское имена, которые были чаще остальных даны детям в заданный период времени.

Задание 18. Имена без повторов.

Продолжаем использовать базу имен из задания 15. Проходя по файлам, выберите имена без дублирования отдельно для мальчиков и для девочек и выведите их на экран. Разумеется, повторяющихся имен в этих списках быть не должно.

Задание 19. Проверяем правильность написания.

Автоматическая проверка орфографии не помешала бы многим из нас. В данном задании мы напишем простую программу, сверяющую слова из текстового файла со словарем. Неправильно написанными будем считать все слова, которых не нашлось в словаре.

Имя файла, в котором требуется выполнить орфографическую проверку, пользователь должен передать при помощи аргумента командной строки. В случае отсутствия аргумента должна выдаваться соответствующая ошибка. Сообщение об ошибке также должно появляться, если не удается открыть указанный пользователем файл.

Подсказка. Конечно, вы могли бы загрузить все слова из текста в список и анализировать их при помощи функции оператора in. Но эта операция будет выполняться довольно долго. Гораздо быстрее в Python выполняется проверка на наличие определенного ключа в словаре или значения в наборе. Если вы остановитесь на варианте со словарем, ключами должны стать слова, а значениями – ноль или любое другое число, поскольку их мы в данном случае использовать не будем.

Задание 20. Повторяющиеся слова.

Проверка орфографии – лишь составная часть расширенного текстового анализа на предмет наличия ошибок. Одной из самых распространенных ошибок в текстах является повторение слов. Например, автор может по ошибке дважды подряд написать одно слово, как в следующем примере:

At least one value must be entered
entered in order to compute the average.

Некоторые текстовые процессоры умеют распознавать такой вид ошибок при выполнении текстового анализа.

В данном задании вам предстоит написать программу для определения наличия дублей слов в тексте. При нахождении повтора на экран должен выводиться номер строки и дублирующееся слово. Удостоверьтесь, что программа корректно обрабатывает случаи, когда повторяющиеся слова находятся на разных строках, как в предыдущем примере. Имя файла для анализа должно быть передано программе в качестве единственного аргумента командной строки. При отсутствии аргумента или невозможности

открыть указанный файл на экране должно появляться соответствующее сообщение об ошибке.

Задание 21. Редактирование текста в файле.

Перед публикацией текста или документа обычно принято удалять или изменять в них служебную информацию.

В данном задании вам необходимо написать программу, которая будет заменять все служебные слова в тексте на символы звездочек (по количеству символов в словах). Вы должны осуществлять регистрозависимый поиск служебных слов в тексте, даже если эти слова входят в состав других слов. Список служебных слов должен храниться в отдельном файле. Сохраните отредактированную версию исходного файла в новом файле. Имена исходного файла, файла со служебными словами и нового файла должны быть введены пользователем.

Примечание. Вам может пригодиться метод replace для работы со строками при решении этого задания. Информацию о работе данного метода можно найти в интернете.

В качестве дополнительного задания расширьте свою программу таким образом, чтобы она выполняла замену служебных слов вне зависимости от того, какой регистр символов используется в тексте. Например, если в списке служебных слов будет присутствовать слово exam, то все следующие варианты слов должны быть заменены звездочками: exam, Exam, ExaM и EXAM.

Задание 22. Пропущенные комментарии.

При написании функций хорошей практикой считается предварение ее блоком комментариев с описанием назначения функции, ее входных параметров и возвращаемого значения. Но некоторые разработчики просто не пишут комментарии к своим функциям. Другие честно собираются написать их когда-нибудь в будущем, но руки так и не доходят.

Напишите программу, которая будет проходить по файлу с исходным кодом на Python и искать функции, не снабженные блоком комментариев. Можно принять за аксиому, что строка, начинающаяся со слова def, следом за которым идет пробел, будет считаться началом функции. И если функция документирована, предшествующая строчка должна начинаться со знака #. Перечислите названия всех функций, не снабженных комментариями, вместе с именем файла и номером строки, с которой начинается объявление функции.

Одно или несколько имен файлов с кодом на языке Python пользователь должен передать в функцию в качестве аргументов командной строки. Для файлов, которые не существуют или не могут быть открыты, должны выдаваться соответствующие предупреждения, после чего должна быть продолжена обработка остальных файлов.

Задание 23. Строки фиксированной длины.

Ширина окна терминала обычно составляет 80 символов, хотя есть более широкие и узкие терминалы. Это затрудняет отображение текстов, разбитых на абзацы. Бывает, что строки в исходном файле оказываются слишком длинными и автоматически переносятся, тем самым затрудняя чтение, или

слишком короткими, что приводит к недостаточному заполнению свободного места на экране.

Напишите программу, которая будет открывать файл и выводить его на экран с постоянной длиной строк. Если в исходном файле строка оказывается слишком длинной, «лишние» слова должны быть перенесены на следующую строку, а если слишком короткой, слова со следующей строки должны перекочевать в конец текущей до полного ее заполнения. Например, следующий отрывок из «Алисы в Стране чудес»:

Alice was
beginning to get very tired of sitting by her
sister
on the bank, and of having nothing to do: once
or twice she had peeped into the book her sister
was reading, but it had
no
pictures or conversations in it, "and what is
the use of a book," thought Alice, "without
pictures or conversations?"

в отформатированном виде с длиной строки в 50 символов будет выглядеть так:

Alice was beginning to get very tired of sitting
by her sister on the bank, and of having nothing
to do: once or twice she had peeped into the book
her sister was reading, but it had no pictures or
conversations in it, "and what is the use of a
book," thought Alice, "without pictures or
conversations?"

Убедитесь, что ваша программа корректно обрабатывает тексты, в которых присутствуют абзацы. Идентифицировать конец одного абзаца и начало другого можно по наличию пустой строки в тексте после удаления символа конца строки.

Подсказка. Используйте константу для задания максимальной ширины строк. Это позволит вам легко адаптировать программу под любые окна.

Задание 24. Слова с шестью гласными в ряд.

Существует как минимум одно слово в английском языке, содержащее все гласные буквы в том порядке, в котором они расположены в алфавите, а именно A, E, I, O, U и Y. Напишите программу, которая будет просматривать текстовый файл на предмет поиска и отображения таких слов. Имя исходного файла должно быть запрошено у пользователя. Если имя файла окажется неверным или возникнут иные ошибки при чтении файла, выведите соответствующее сообщение об ошибке.