Лабораторная работа №2

Тема: «Работа с вводом и выводом данных, чтение и запись файлов».

Открытие и закрытие файлов

Python поддерживает множество различных типов файлов, но условно их можно разделить на два виде: текстовые и бинарные. Текстовые файлы – это, к примеру, файлы с расширением cvs, txt, html, в общем, любые файлы, которые сохраняют информацию в текстовом виде. Бинарные файлы – это изображения, аудио и видеофайлы, и т. д. В зависимости от типа файла работа с ним может немного отличаться.

При работе с файлами необходимо соблюдать некоторую последовательность операций:

1. Открытие файла с помощью метода ***open()***
2. Чтение файла с помощью метода ***read()*** или запись в файл посредством метода ***write()***
3. Закрытие файла методом ***close()***

Чтобы начать работу с файлом, его надо открыть с помощью функции ***open()***, которая имеет следующее формальное определение:

```
open(file, mode)
```

Первый параметр функции представляет путь к файлу. Путь файла может быть абсолютным, то есть начинаться с буквы диска, например, *C://somedir/somefile.txt*. Либо он может быть относительным, например, *somedir/somefile.txt* – в этом случае поиск файла будет идти относительно расположения запущенного скрипта Python.

Второй передаваемый аргумент (***mode***) устанавливает режим открытия файла в зависимости от того, что мы собираемся с ним делать. Существует 4 общих режима:

- **r** (Read). Файл открывается для чтения. Если файл не найден, то генерируется исключение FileNotFoundError;
- **w** (Write). Файл открывается для записи. Если файл отсутствует, то он создается. Если подобный файл уже есть, то он создается заново, и, соответственно, старые данные в нем стираются;

- **a** (Append). Файл открывается для дозаписи. Если файл отсутствует, то он создается. Если подобный файл уже есть, то данные записываются в его конец;
- **b** (Binary). Используется для работы с бинарными файлами. Применяется вместе с другими режимами – w или r.

После завершения работы с файлом его обязательно нужно закрыть методом ***close()***. Данный метод освободит все связанные с файлом используемые ресурсы.

Например, откроем для записи текстовый файл `"hello.txt"`:

```
myfile = open("hello.txt", "w")
myfile.close()
```

При открытии файла или в процессе работы с ним мы можем столкнуться с различными исключениями, например, к нему нет доступа и т. д. В этом случае программа выдаст ошибку, а ее выполнение не дойдет до вызова метода ***close***, и, соответственно, файл не будет закрыт.

В этом случае мы можем обрабатывать исключения:

```
try:
    somefile = open("hello.txt", "w")
    try:
        somefile.write("hello world")
    except Exception as e:
        print(e)
    finally:
        somefile.close()
except Exception as ex:
    print(ex)
```

В данном случае вся работа с файлом идет во вложенном блоке ***try***. И если вдруг возникнет какое-либо исключение, то в любом случае в блоке ***finally*** файл будет закрыт.

Однако есть и более удобная конструкция – конструкция ***with***:

```
with open(file, mode) as file_obj:
    инструкции
```

Эта конструкция определяет для открытого файла переменную ***file_obj*** и выполняет набор инструкций. После их выполнения файл автоматически закрывается. Даже если при

выполнении инструкций в блоке ***with*** возникнут какие-либо исключения, то файл все равно закрывается.

Так, перепишем предыдущий пример:

```
with open("hello.txt", "w") as somefile:
    somefile.write("hello world")
```

Запись в текстовый файл

Чтобы открыть текстовый файл на запись, необходимо применить режим **w** (перезапись) или **a** (дозапись). Затем для записи применяется метод ***write(str)***, в который передается записываемая строка. Стоит отметить, что записывается именно строка, поэтому, если нужно записать числа, данные других типов, то их предварительно нужно конвертировать в строку.

Запишем некоторую информацию в файл `"hello.txt"`:

```
with open("hello.txt", "w") as file:
    file.write("hello world")
```

Если мы откроем папку, в которой находится текущий скрипт ***Python***, то увидим там файл hello.txt. Этот файл можно открыть в любом текстовом редакторе и при желании изменить.

Теперь допишем в этот файл еще одну строку:

```
with open("hello.txt", "a") as file:
    file.write("\ngood bye, world")
```

Дозапись выглядит как добавление строки к последнему символу в файле, поэтому, если необходимо сделать запись с новой строки, то можно использовать эскейп-последовательность `"\n"`. В итоге файл `hello.txt` будет иметь следующее содержимое:

```
hello world
good bye, world
```

Еще один способ записи в файл представляет стандартный метод ***print()***, который применяется для вывода данных на консоль:

```
with open("hello.txt", "a") as hello_file:
    print("Hello, world", file=hello_file)
```

Для вывода данных в файл в метод ***print*** в качестве второго параметра передается название файла через параметр ***file***. А первый параметр представляет записываемую в файл строку.

Чтение файла

Для чтения файла он открывается с режимом **r** (Read), и затем мы можем считать его содержимое различными методами:

- ***readline()***: считывает одну строку из файла;
- ***read()***: считывает все содержимое файла в одну строку;
- ***readlines()***: считывает все строки файла в список.

Например, считаем выше записанный файл построчно:

```
with open("hello.txt", "r") as file:
    for line in file:
        print(line, end="")
```

Несмотря на то, что мы явно не применяем метод ***readline()*** для чтения каждой строки, при переборе файла этот метод автоматически вызывается для получения каждой новой строки. Поэтому в цикле вручную нет смысла вызывать метод ***readline***. И поскольку строки разделяются символом перевода строки `"\n"`, то чтобы исключить излишнего переноса на другую строку, в функцию ***print*** передается значение `end=""`.

Теперь явным образом вызовем метод ***readline()*** для чтения отдельных строк:

```
with open("hello.txt", "r") as file:
    str1 = file.readline()
    print(str1, end="")
    str2 = file.readline()
    print(str2)
```

Консольный вывод:

```
hello world
good bye, world
```

Метод ***readline*** можно использовать для построчного считывания файла в цикле ***while***:

```
with open("hello.txt", "r") as file:
    line = file.readline()
    while line:
        print(line, end="")
        line = file.readline()
```

Если файл небольшой, то его можно разом считать с помощью метода ***read()***:

```
with open("hello.txt", "r") as file:
    content = file.read()
    print(content)
```

И также применим метод ***readlines()*** для считывания всего файла в список строк:

```
with open("hello.txt", "r") as file:
    contents = file.readlines()
    str1 = contents[0]
    str2 = contents[1]
    print(str1, end="")
    print(str2)
```

При чтении файла мы можем столкнуться с тем, что его кодировка не совпадает с ASCII. В этом случае мы явным образом можем указать кодировку с помощью параметра ***encoding***:

```
filename = "hello.txt"
with open(filename, encoding="utf8") as file:
    text = file.read()
```

Теперь напишем небольшой скрипт, который будет записывать введенный пользователем массив строк и считывать его обратно из файла на консоль:

```
# имя файла
FILENAME = "messages.txt"
# определяем пустой список
messages = list()

for i in range(4):
    message = input("Введите строку " + str(i+1) + ": ")
    messages.append(message + "\n")

# запись списка в файл
with open(FILENAME, "a") as file:
    for message in messages:
        file.write(message)

# считываем сообщения из файла
print("Считанные сообщения")
with open(FILENAME, "r") as file:
    for message in file:
        print(message, end="")
```

Пример работы программы:

```
Введите строку 1: hello
Введите строку 2: world peace
Введите строку 3: great job
Введите строку 4: Python
Считанные сообщения
hello
world peace
great job
Python
```

Файлы CSV

Одним из распространенных файловых форматов, которые хранят в удобном виде информацию, является формат **csv**. Каждая строка в файле csv представляет отдельную запись или строку, которая состоит из отдельных столбцов, разделенных запятыми. Собственно, поэтому формат и называется ***Comma Separated Values***. Но хотя формат **csv** – это формат текстовых файлов, Python для упрощения работы с ним предоставляет специальный встроенный модуль ***csv***.

Рассмотрим работу модуля на примере:

```
import csv

FILENAME = "users.csv"

users = [
    ["Tom", 28],
    ["Alice", 23],
    ["Bob", 34]
]

with open(FILENAME, "w", newline="") as file:
    writer = csv.writer(file)
    writer.writerows(users)


with open(FILENAME, "a", newline="") as file:
    user = ["Sam", 31]
    writer = csv.writer(file)
    writer.writerow(user)
```

В файл записывается двумерный список – фактически таблица, где каждая строка представляет одного пользователя. А каждый пользователь содержит два поля: имя и возраст. То есть фактически таблица состоит из трех строк и двух столбцов.

При открытии файла на запись в качестве третьего параметра указывается значение `newline=""` – пустая строка позволяет корректно считывать строки из файла вне зависимости от операционной системы.

Для записи нам надо получить объект ***writer***, который возвращается функцией `csv.writer(file)`. В эту функцию передается открытый файл. А собственно запись производится с помощью метода `writer.writerows(users)`. Этот метод принимает набор строк. В нашем случае это двумерный список.

Если необходимо добавить одну запись, которая представляет собой одномерный список, например, `["Sam",  31]`, то в этом случае можно вызвать метод ***writer.writerow(user)***.

В итоге, после выполнения скрипта в той же папке окажется файл `users.csv`, который будет иметь следующее содержимое:

```
Tom,28
Alice,23
Bob,34
Sam,31
```

Для чтения из файла нам нужно создать объект ***reader***:

```
import csv

FILENAME = "users.csv"

with open(FILENAME, "r", newline="") as file:
    reader = csv.reader(file)
    for row in reader:
        print(row[0], " - ", row[1])
```

При получении объекта reader мы можем в цикле перебрать все его строки:

```
Tom  -  28
Alice  -  23
Bob  -  34
Sam - 31
```

Работа со словарями

В примере выше каждая запись или строка представляла собой отдельный список, например, `["Sam", 31]`. Но кроме того, модуль ***csv*** имеет специальные дополнительные возможности для работы со словарями. В частности, функция ***csv.DictWriter()*** возвращает объект ***writer***, который позволяет записывать в файл. А функция ***csv.DictReader()*** возвращает объект ***reader*** для чтения из файла. Например:

```
import csv

FILENAME = "users.csv"

users = [
    {"name": "Tom", "age": 28},
    {"name": "Alice", "age": 23},
    {"name": "Bob", "age": 34}
]

with open(FILENAME, "w", newline="") as file:
    columns = ["name", "age"]
    writer = csv.DictWriter(file, fieldnames=columns)
    writer.writeheader()

    # запись нескольких строк
    writer.writerows(users)

    user = {"name" : "Sam", "age": 41}
    # запись одной строки
    writer.writerow(user)
```

```
with open(FILENAME, "r", newline="") as file:
    reader = csv.DictReader(file)
    for row in reader:
        print(row["name"], "-", row["age"])
```

Запись строк также производится с помощью методов ***writerow()*** и ***writerows()***. Но теперь каждая строка представляет собой отдельный словарь, и кроме того, производится запись и заголовков столбцов с помощью метода ***writeheader()***, а в метод ***csv.DictWriter*** в качестве второго параметра передается набор столбцов.

При чтении строк, используя названия столбцов, мы можем обратиться к отдельным значениям внутри строки: `row["name"]`.

Бинарные файлы

Бинарный файл – это файл, в котором данные хранятся так же, как они хранятся в основной памяти для обработки. Он хранится в двоичном формате вместо символов ASCII. Для работы с ними в ***Python*** необходим встроенный модуль ***pickle***. Этот модуль предоставляет два метода:

- ***dump(obj, file)***: записывает объект **obj** в бинарный файл **file**
- ***load(file)***: считывает данные из бинарного файла в объект.

При открытии бинарного файла на чтение или запись также надо учитывать, что нам нужно применять режим `"b"` в дополнение к режиму записи (`"w"`) или чтения (`"r"`). Допустим, надо сохранить два объекта:

```
import pickle

FILENAME = "user.dat"

name = "Tom"
age = 19

with open(FILENAME, "wb") as file:
    pickle.dump(name, file)
    pickle.dump(age, file)

with open(FILENAME, "rb") as file:
    name = pickle.load(file)
    age = pickle.load(file)
    print("Имя:", name, "\tВозраст:", age)
```

С помощью функции ***dump*** последовательно записываются два объекта. Поэтому при чтении файла также последовательно посредством функции ***load*** мы можем считать эти объекты. Консольный вывод программы:

```
Имя: Tom        Возраст: 28
```

Подобным образом мы можем сохранять и извлекать из файла наборы объектов:

```
import pickle

FILENAME = "users.dat"

users = [
    ["Tom", 28, True],
    ["Alice", 23, False],
    ["Bob", 34, False]
]

with open(FILENAME, "wb") as file:
    pickle.dump(users, file)


with open(FILENAME, "rb") as file:
    users_from_file = pickle.load(file)
    for user in users_from_file:
        print("Имя:",    user[0],    "\tВозраст:",    user[1],
"\tЖенат(замужем):", user[2])
```

В зависимости от того, какой объект мы записывали функцией ***dump***, тот же объект будет возвращен функцией ***load*** при считывании файла.

Консольный вывод:

```
Имя: Tom      Возраст: 28    Женат(замужем): True
Имя: Alice    Возраст: 23    Женат(замужем): False
Имя: Bob      Возраст: 34    Женат(замужем): False
```

В отличие от JSON, pickle – это протокол, который позволяет сериализовать сложные объекты Python. Он специфичен для Python и не может использоваться для связи с приложениями, написанными на других языках. Десериализация данных pickle, поступающих из ненадежного источника, может привести к выполнению произвольного кода.

Фундаментальные различия между протоколами pickle и JSON:

1. JSON – это формат сериализации текста, pickle – это формат бинарной сериализации.
2. JSON удобочитаем для человека, pickle – нет.
3. JSON совместим и широко используется за пределами экосистемы Python, в то время как pickle зависит от Python.

4. JSON по умолчанию может представлять только подмножество встроенных типов Python и никаких пользовательских классов; pickle может представлять чрезвычайно большое количество типов Python.
5. В отличие от pickle, десериализация ненадежного JSON сама по себе не создает уязвимости при выполнении произвольного кода.

Модуль shelve

Для работы с бинарными файлами в ***Python*** может применяться еще один модуль – ***shelve***. Он сохраняет объекты в файл с определенным ключом. Затем по этому ключу может извлечь ранее сохраненный объект из файла. Процесс работы с данными через модуль ***shelve*** напоминает работу со словарями, которые также используют ключи для сохранения и извлечения объектов.

Для открытия файла модуль ***shelve*** использует функцию ***open()***:

```
open(путь_к_файлу[,flag="c"[,protocol=None[,writeback=False]]])
```

Параметр ***flag*** может принимать следующие значения:

- c: файл открывается для чтения и записи (значение по умолчанию). Если файл не существует, то он создается;
- r: файл открывается только для чтения;
- w: файл открывается для записи;
- n: файл открывается для записи. Если файл не существует, то он создается. Если он существует, то он перезаписывается.

Для закрытия подключения к файлу вызывается метод ***close()***:

```
import shelve
d = shelve.open(filename)
d.close()
```

Также можно открывать файл с помощью оператора ***with***. Сохраним и считаем в файл несколько объектов:

```
import shelve

FILENAME = "states2"
with shelve.open(FILENAME) as states:
    states["London"] = "Great Britain"
    states["Paris"] = "France"
    states["Berlin"] = "Germany"
    states["Madrid"] = "Spain"

with shelve.open(FILENAME) as states:
    print(states["London"])
    print(states["Madrid"])
```

Запись данных предполагает установку значения для определенного ключа:

```
states["London"] = "Great Britain"
```

А чтение из файла эквивалентно получению значения по ключу:

```
print(states["London"])
```

В качестве ключей используются строковые значения.

При чтении данных, если запрашиваемый ключ отсутствует, то генерируется исключение. В этом случае перед получением мы можем проверять на наличие ключа с помощью оператора ***in***:

```
with shelve.open(FILENAME) as states:
    key = "Brussels"
    if key in states:
        print(states[key])
```

Также мы можем использовать метод ***get()***. Первый параметр метода – ключ, по которому следует получить значение, а второй – значение по умолчанию, которое возвращается, если ключ не найден.

```
with shelve.open(FILENAME) as states:
    state = states.get("Brussels", "Undefined")
    print(state)
```

Используя цикл ***for***, можно перебрать все значения из файла:

```
    with shelve.open(FILENAME) as states:
        for key in states:
            print(key," - ", states[key])
```

Метод ***keys()*** возвращает все ключи из файла, а метод ***values()*** – все значения:

```
    with shelve.open(FILENAME) as states:

        for city in states.keys():
            print(city, end=" ")        # London Paris Berlin Madrid
        print()
        for country in states.values():
            print(country, end=" ")    # Great Britain France Germany
Spain
```

Еще один метод ***items()*** возвращает набор кортежей. Каждый кортеж содержит ключ и значение.

```
    with shelve.open(FILENAME) as states:

        for state in states.items():
            print(state)
```

Консольный вывод:

```
    ("London", "Great Britain")
    ("Paris", "France")
    ("Berlin", "Germany")
    ("Madrid", "Spain")
```

Обновление данных

Для изменения данных достаточно присвоить по ключу новое значение, а для добавления данных – определить новый ключ:

```
import shelve

FILENAME = "states2"
with shelve.open(FILENAME) as states:
    states["London"] = "Great Britain"
    states["Paris"] = "France"
    states["Berlin"] = "Germany"
    states["Madrid"] = "Spain"

with shelve.open(FILENAME) as states:

    states["London"] = "United Kingdom"
    states["Brussels"] = "Belgium"
    for key in states:
        print(key, " - ", states[key])
```

Удаление данных

Для удаления с одновременным получением можно использовать функцию ***pop()***, в которую передается ключ элемента и значение по умолчанию, если ключ не найден:

```
with shelve.open(FILENAME) as states:

    state = states.pop("London", "NotFound")
    print(state)
```

Также для удаления может применяться оператор ***del***:

```
with shelve.open(FILENAME) as states:
    del states["Madrid"]    # удаляем объект с ключом Madrid
```

Для удаления всех элементов можно использовать метод ***clear()***:

```
with shelve.open(FILENAME) as states:

    states.clear()
```

Модуль OS и работа с файловой системой

Ряд возможностей по работе с каталогами и файлами предоставляет встроенный модуль ***os***. Хотя он содержит много функций, рассмотрим только основные из них:

- mkdir(): создает новую папку;
- rmdir(): удаляет папку;
- rename(): переименовывает файл;
- remove(): удаляет файл.

Создание и удаление папки

Для создания папки применяется функция ***mkdir()***, в которую передается путь к создаваемой папке:

```
import os

# путь относительно текущего скрипта
os.mkdir("hello")
# абсолютный путь
os.mkdir("c://somedir")
os.mkdir("c://somedir/hello")
```

Для удаления папки используется функция ***rmdir()***, в которую передается путь к удаляемой папке:

```
import os

# путь относительно текущего скрипта
os.rmdir("hello")
# абсолютный путь
os.rmdir("c://somedir/hello")
```

Переименование файла

Для переименования вызывается функция ***rename(source, target)***, первый параметр которой – путь к исходному файлу, а второй – новое имя файла. В качестве путей могут использоваться как абсолютные, так и относительные. Например, пусть в папке *C://SomeDir/* располагается файл ***somefile.txt***. Переименуем его в файл ***"hello.txt"***:

```
import os

os.rename("C://SomeDir/somefile.txt", "C://SomeDir/hello.txt")
```

Удаление файла

Для удаления вызывается функция ***remove()***, в которую передается путь к файлу:

```
import os

os.remove("C://SomeDir/hello.txt")
```

Существование файла

Если мы попытаемся открыть файл, который не существует, то ***Python*** выбросит исключение ***FileNotFoundError***. Для отлова исключения мы можем использовать конструкцию `try...except`. Однако можно уже до открытия файла проверить, существует ли он или нет с помощью метода ***os.path.exists(path)***. В этот метод передается путь, который необходимо проверить:

```
filename = input("Введите путь к файлу: ")
if os.path.exists(filename):
    print("Указанный файл существует")
else:
    print("Файл не существует")
```

Требования к выполнению лабораторной работы №2

1. Изучите теоретическую часть ко второй лабораторной работе:
    a. Теоретическая часть ко второй лабораторной работе.
    b. Лекция №2.
2. Создайте новый проект.
3. Запустите примеры из лабораторной работы.
4. Выполните задание согласно вашему варианту:
    a. Вычислите свой вариант (*согласно формуле ниже*). Если сделали не свой вариант => работа не засчитывается.
    b. Каждое задание представляет собой отдельный скрипт формата: `lab_{`**`номер_ЛР`**`}_task_{`**`номер_задания`**`}_{`**`номер_варианта`**`}.py`, пример: `lab_2_1_2.py`
    c. Отправьте выполненное задание в ОРИОКС (*раздел Домашние задания*).

Формат защиты лабораторных работ:

1. Продемонстрируйте выполненные задания.
2. Ответьте на вопросы по вашему коду.
3. При необходимости выполните дополнительное (*дополнительные*) задания от преподавателя.
4. Ответьте (*устно*) преподавателю на контрольные вопросы.

Список вопросов

1. Перечислите основные режимы работы с файлами. Чем они отличаются?
2. Перечислите основные методы считывания данных из текстового файла. Чем они отличаются?
3. Для чего нужен параметр *encoding* в функции *open*?
4. В чём отличие текстового файла от бинарного?
5. Какие данные обычно хранят в формате *csv*?
6. Специфика работы библиотеки pickle.
7. Отличия pickle от JSON.

Задания

Во всех заданиях необходимо проверять корректность вводимых данных и выводить соответствующие сообщения об ошибках.

№ Варианта = *номер_студенческого* % 2 + 1

Вариант №1

Задание №1. Сгенерируйте файл `ip.log` состоящий из 10 000 IPv4 адресов. Каждый IP адрес должен находиться на новой строке.

Пример:

```
192.168.1.1
95.162.15.20
…
10.0.0.1
```

Задание №2. Пользователь с клавиатуры вводит маску подсети [1]. Напишите скрипт, который считывает файл `ip.log` (*из первого задания*), для каждого IP адреса применяет маску подсети и записывает в файл `ip_solve.log` адреса сети [2].

Пример:

IP-адрес	`192.168.1.2 (формат из файла)` **`11000000 10101000 00000001 00000010`** `(binary)`
Маска подсети	`255.255.254.0 (ввод с клавиатуры)` **`11111111 11111111 11111110 00000000`** `(binary)`
Адрес сети	`192.168.0.0 (запись в файл)` **`11000000 10101000 00000000 00000000`** `(binary)`

Задание №3. В файле *players.csv* записан (*формируется студентом вручную*) протокол турнира в формате:

```
Спортсмен;Количество побед;Доп. показатель
Иванов;10;256
Петров;30;1000
Медведев;30;1100
...
Сидоров;20;300
```

Необходимо записать в текстовый файл `results.csv` результаты турнира.

Распределение мест зависит от следующих показателей:

1. Количество побед (чем больше, тем выше участник турнира).
2. Дополнительный показатель (учитывается для участников, набравших одинаковое количество побед, чем выше показатель, тем выше участник).

Пример:

Входной файл *players.csv (заголовок с названиями полей – обязателен)*

```
Спортсмен;Количество побед;Доп. показатель
Иванов;10;256
Петров;30;1000
Медведев;30;1100
Сидоров;20;300
Уткин;10;256
Васин;5;100
```

Выходной файл *results.csv (заголовок с названиями полей – обязателен)*:

```
Спортсмен;Место
Медведев;1
Петров;2
Сидоров;3
Иванов;4
Уткин;4
Васин;6
```

Примечание: если участники имеют одинаковое количество побед и доп. показателей, то они делят место.

Задание №4. Внутри запускаемого проекта (*должен использоваться относительный путь*) создайте директорию example. Внутри директории example сгенерируйте 100 файлов размером от 1 Кб до 100 Кб (*размер каждого файла задаётся случайно*).

Задание №5. Пользователь с клавиатуры вводит два целых (*left* и *right*), положительных числа от 1 до 100. Необходимо вывести (*в консоль*) количество файлов в директории example (из задания №4), размер которых (в Кб) находится между *left* и *right* заданными пользователем (*левая и правая граница включительно*).

Задание №6. Создайте файл `article_rus.txt` и заполните его текстом любого художественного произведения на русском языке (*ограничение – размер файла должен быть не менее 10 Кб*). Необходимо определить частоту (*в долях*) повторяемости каждой кириллической буквы в тексте (*остальные символы игнорировать*), отсортировать в порядке убывания частоты, результат записать в файл `article_rus_solve.txt` в формате: {*буква}: {частота}*.

Пример:

о: 0.095

е: 0.074

…

Примечание: символы в нижнем и верхнем регистре учитывать как один символ.

Вариант №2

Задание №1. Сгенерируйте файл `mask.log` состоящий из всех возможных масок подсети. Каждая маска должна находиться на новой строке.

Пример:

```
255.255.255.255
255.255.255.254
…
000.000.000.000
```

Примечание: маски должны генерироваться скриптом, а не заданы в коде (*в любом виде*).

Задание №2. Пользователь с клавиатуры вводит IPv4 адрес. Напишите скрипт, который считывает файл `mask.log` (*из первого задания*), для каждой строки применяет маску подсети и записывает в файл `ip_solve.log` адреса сети [2].

Пример:

IP-адрес	`192.168.1.2 (ввод с клавиатуры)` **`11000000 10101000 00000001 00000010`** `(binary)`
Маска подсети	`255.255.254.0 (формат из файла)` **`11111111 11111111 11111110 00000000`** `(binary)`
Адрес сети	`192.168.0.0 (запись в файл)` **`11000000 10101000 00000000 00000000`** `(binary)`

Задание №3. В файле *store.csv* записана (*формируется студентом вручную*) информация о товарах на складе:

```
Товар;Категория;Стоимость
Шоколад;Сладости;200.50
Яблоко;Фрукты;99.99
...
Молоко;Молоко;300
```

Необходимо записать в текстовый файл `categories.csv` информацию о категориях: *Категория => Общая стоимость товаров категории*.

Пример:

Входной файл *store.csv (заголовок с названиями полей – обязателен)*

```
Товар;Категория;Стоимость
Шоколад;Сладости;200.50
Яблоко;Фрукты;99.99
Бананы;Фрукты;10
Молоко;Молоко;300
```

Выходной файл *categories.csv (заголовок с названиями полей – обязателен)*:

```
Категория;Стоимость
Сладости;200.50
Фрукты;109.99
Молоко;300
```

Примечание: в столбце «Стоимость» все значения > 0 и имеют не более двух знаков после запятой.

Задание №4. Внутри запускаемого проекта (*должен использоваться относительный путь*) создайте директорию example. Внутри директории example сгенерируйте 100 файлов с различными расширениями (*на ваш выбор, не менее 10 различных расширений, размер файла не важен*).

Задание №5. Пользователь с клавиатуры вводит расширение файла. Необходимо вывести (*в консоль*) количество файлов в директории example (из задания №4), расширение которых совпадает с заданным пользователем.

Задание №6. Создайте файл `article_eng.txt` и заполните его текстом любого художественного произведения на английском языке (*ограничение – размер файла должен быть не менее 10 Кб*). Необходимо определить частоту (*в долях*) повторяемости каждой латинской буквы в тексте (*остальные символы игнорировать*), отсортировать в порядке убывания частоты, результат записать в файл `article_eng_solve.txt` в формате: {*буква}: {частота}*.

Пример:

o: 0.095

e: 0.074

…

Примечание: символы в нижнем и верхнем регистре учитывать как один символ.

Дополнительное задание (*необязательное*)

Турист Петя решил посетить N экскурсий за N дней. Стоимость экскурсий в разные дни различается (см. CSV файл). Помогите Пете составить план посещения экскурсий (1 экскурсия в день) так, чтобы потратить наименьшую сумму. Если таких вариантов несколько, выберите любой из них.

Примеры входных файлов *input_*.csv*, выходных соответственно *output_*.csv* доступны по *ссылке*.