Урок 6. Работа с файлами

1. Не используя библиотеки для парсинга, распарсить (получить определённые данные) файл логов web-сервера `nginx_logs.txt` (https://github.com/elastic/examples/raw/master/Common%20Data%20Formats/nginx_logs/nginx_logs) — получить список кортежей вида: `(<remote_addr>, <request_type>, <requested_resource>)`. Например:

[
...
('141.138.90.60', 'GET', '/downloads/product_2'),
('141.138.90.60', 'GET', '/downloads/product_2'),
('173.255.199.22', 'GET', '/downloads/product_2'),
...
]

Примечание: код должен работать даже с файлами, размер которых превышает объем ОЗУ компьютера.

2. * (вместо 1) Найти IP адрес спамера и количество отправленных им запросов по данным файла логов из предыдущего задания.

Примечания: спамер — это клиент, отправивший больше всех запросов; код должен работать даже с файлами, размер которых превышает объем ОЗУ компьютера.

3. Есть два файла: в одном хранятся ФИО пользователей сайта, а в другом — данные об их хобби. Известно, что при хранении данных используется принцип: одна строка — один пользователь, разделитель между значениями — запятая. Написать код, загружающий данные из обоих файлов и формирующий из них словарь: ключи — ФИО, значения — данные о хобби. Сохранить словарь в файл. Проверить сохранённые данные. Если в файле, хранящем данные о хобби, меньше записей, чем в файле с ФИО, задаём в словаре значение None. Если наоборот — выходим из скрипта с кодом «1». При решении задачи считать, что объём данных в файлах во много раз меньше объема ОЗУ.

Фрагмент файла с данными о пользователях (users.csv):

```
Иванов,Иван,Иванович
Петров,Петр,Петрович
```

Фрагмент файла с данными о хобби (hobby.csv):

```
скалолазание,охота
горные лыжи
```

4. * (вместо 3) Решить задачу 3 для ситуации, когда объём данных в файлах превышает объём ОЗУ (разумеется, не нужно реально создавать такие большие файлы, это просто задел на будущее проекта). Также реализовать парсинг данных из файлов — получить отдельно фамилию, имя и отчество для пользователей и название каждого хобби: преобразовать в какой-нибудь контейнерный тип (список, кортеж, множество, словарь). Обосновать выбор типа. Подумать, какие могут возникнуть проблемы при парсинге. В словаре должны храниться данные, полученные в результате парсинга.

5. ** (вместо 4) Решить задачу 4 и реализовать интерфейс командной строки, чтобы можно было задать путь к обоим исходным файлам и путь к выходному файлу со словарём. Проверить работу скрипта для случая, когда все файлы находятся в разных папках.

6. Реализовать простую систему хранения данных о суммах продаж булочной. Должно быть два скрипта с интерфейсом командной строки: для записи данных и для вывода на экран записанных данных. При записи передавать из командной строки значение суммы продаж. Для чтения данных реализовать в командной строке следующую логику:

- просто запуск скрипта — выводить все записи;
- запуск скрипта с одним параметром-числом — выводить все записи с номера, равного этому числу, до конца;
- запуск скрипта с двумя числами — выводить записи, начиная с номера, равного первому числу, по номер, равный второму числу, включительно.

Подумать, как избежать чтения всего файла при реализации второго и третьего случаев.

Данные хранить в файле bakery.csv в кодировке utf-8. Нумерация записей начинается с 1. Примеры запуска скриптов:

```
python add_sale.py 5978,5
python add_sale.py 8914,3
python add_sale.py 7879,1
python add_sale.py 1573,7
python show_sales.py
5978,5
8914,3
7879,1
1573,7
python show_sales.py 3
7879,1
1573,7
python show_sales.py 1 3
5978,5
8914,3
7879,1
```

7. * (вместо 6) Добавить возможность редактирования данных при помощи отдельного скрипта: передаём ему номер записи и новое значение. При этом файл не должен читаться целиком — обязательное требование. Предусмотреть ситуацию, когда пользователь вводит номер записи, которой не существует.

Задачи со * предназначены для продвинутых учеников, которым мало сделать обычное задание.