**МИНОБРНАУКИ РОССИИ**

**Санкт-Петербургский государственный**

**электротехнический университет**

**«ЛЭТИ» им. В.И. Ульянова (Ленина)**

**Кафедра систем автоматизированного проектирования**

**Отчет**

**по лабораторной работе №2**

**по дисциплине «Объектно-ориентированное программирование»**

**Тема: «Консольная программа для работы с xml и csv файлами»**

|  |  |  |
| --- | --- | --- |
| Студент гр. 3352 |  | Гультяев А.С. |
| Преподаватели |  | Федосеев П.О.  Кулагин М.Ю. |

Санкт-Петербург

2024

# Введение

**Описание задания:** Даны 2 файла-справочника городов. Один файл в формате xml, другой в формате csv.

Необходимо разработать консольное приложение для работы с ними.

После запуска приложение ожидает ввода пути до файла-справочника либо команды на завершение работы (какая-то комбинация клавиш).

По команде завершения работы приложение завершает свою работу.

После ввода пути до файла-справочника приложение формирует сводную статистику:

1. Отображает дублирующиеся записи с количеством повторений.
2. Отображает, сколько в каждом городе: 1, 2, 3, 4 и 5 этажных зданий.
3. Показывает время обработки файла.

После вывода статистики приложение снова ожидает ввода пути до файла-справочника либо команды на завершение работы.

# Теоретическая часть

Для реализации данной работы, мной был выбран объектно-ориентированный язык Python. Был выбрал именно Python, поскольку он предлагает удобный и понятный синтаксис, удобные модули для работы с xml и csv файлами. Конечно, Python является далеко не самым быстром языком программирования, поэтому для этой задачи лучше подошел бы Java или C++, но все таки я остановился на Python.

Одним из условий реализации программы было использование xml и csv файлов. CSV файлы используются для хранения данных в табличном формате, где значения разделены запятыми. Это удобный формат для обмена данными между различными программами, такими как электронные таблицы (например, Excel) и базы данных. CSV часто используется из-за простоты в чтении и редактировании, а также он обычно занимает меньше места в сравнении с XML.

XML файлы используются для хранения и передачи данных в структурированном формате, который может использоваться в веб-приложениях, APIs и конфигурационных файлах. XML Позволяет задавать иерархические структуры данных с помощью тегов и хранить сложные данные и метаданные. Из различий CSV и XML можно выделить:

* Структура: CSV имеет плоскую структуру (таблицы), в то время как XML поддерживает иерархическую структуру (деревья).
* Форматирование: CSV фокусируется на представлении данных, а XML включает в себя разметку для описания структуры данных.
* Читаемость: CSV легче читать и обрабатывать, особенно в табличных приложениях, тогда как XML более сложен из-за своей разметки, но лучше подходит для хранения сложных данных.

В данной работе использовались такие библиотеки, как csv, os, xml, time. Далее подробнее о каждой:

Библиотека csv – встроенная библиотека в Python для работы с csv файлами, в данной работе использовались такие методы, как open() и reader(), где первый метод открывает сам файл, а второй читает информацию оттуда по заданным в аргументах параметрам.

Библиотека os – библиотека для работы с путями. В данной работе использовался базовый функционал для проверки наличия файла по указанному адресу с помощью метода os.path.exists().

Библиотека xml – библиотека для работы с xml файлами, которая в данной работе распаршивает файл с помощью метода parse() и получает корневые элементы с помощью метода getroot().

Помимо данных библиотек использовалась библиотека time, которая предлагает работу со временем. Тут она использовалась для замера времени работы программы.

# Практическая часть.

В практической части было реализовано задание следующим образом:

Для начала был реализован класс Color, который хранит в себе номера цветов, для более комфортного и интуитивно понятного вывода в файл.

Далее началась реализация класса GetInfo, который и выполняет всю основную работу программы, для начала внутри этого класса был написан конструктор, который инициализирует такие атрибуты класса, как path, count\_of\_houses и uniq\_rows, где первый ответственен за путь к файлу, а второй и третий атрибуты являются словарями.

После чего была написана функция get\_path(), внутри которой происходит ввод от пользователя пути к файлу. Тут происходит проверка на то, существует ли такой файл, является ли файл необходимого типа, и предлагается пользователю продолжить или завершить работу программы.

Затем был реализован метод get\_count\_of\_houses(), который проводит проверку на то, не является ли поле uniq.rows пустым, если нет, то идет проход по всем ключам данного словаря и заполняется словарь count\_of\_houses, который хранит в себе название города и количество этажей, после чего при нахождении похожего ключа увеличивает значение домов на 1. В конце метода происходит сортировка данного словаря.

Далее реализован метод uniq(), который так же проверяет то, не является ли пустым, далее идет ветвление, которое проводит проверку на то, какого типа является файл .csv или .xml, и в зависимости от ответа используются разные библиотеки, но суть остается одна: открывается файл, происходит получение элементов, которые заносятся в словарь, в ключе которого хранится сама по себе строка, а в качестве элемента содержится количество таких элементов. Это делается для проверки на дубликаты.

После чего реализованы два метода для вывода out\_duplicates() и out\_count\_of\_houses(), которые по своей сути имеют одну реализацию: производится взятие header строк (а точнее не взятие, а просто инициализация, поскольку все файлы известны заранее и этим можно пренебречь), затем высчитывается максимальная длина для каждой header строки, которая используется для красивого вывода. После этого происходит провека в методе out\_duplicates() на то, сколько схожих значений, если это значение более 1, то выводится город, улица, номер дома, количество этажей и количество дубликатов. В случае с out\_count\_of\_houses() выводится город, количество этажей от 1 до 5 для каждого города (поскольку ранее мы отсортировали словарь, можно сделать красивый вывод, где город выводится на 3 этаже).

Ну и в конце был реализован метод, который использует все вышеприведенные функции. А именно, в бесконечном цикле производится создание объекта класса GetInfo, после чего замеряется начальное время и выполняются все приведенные выше функции. Затем замеряется итоговое время и выводится разность конечного и начального времени в секундах. После чего пользователю предлагается продолжить или завершить работу.

# Примеры работы программы:

Рисунок 1. Исправный пример работы программы с csv файлом.

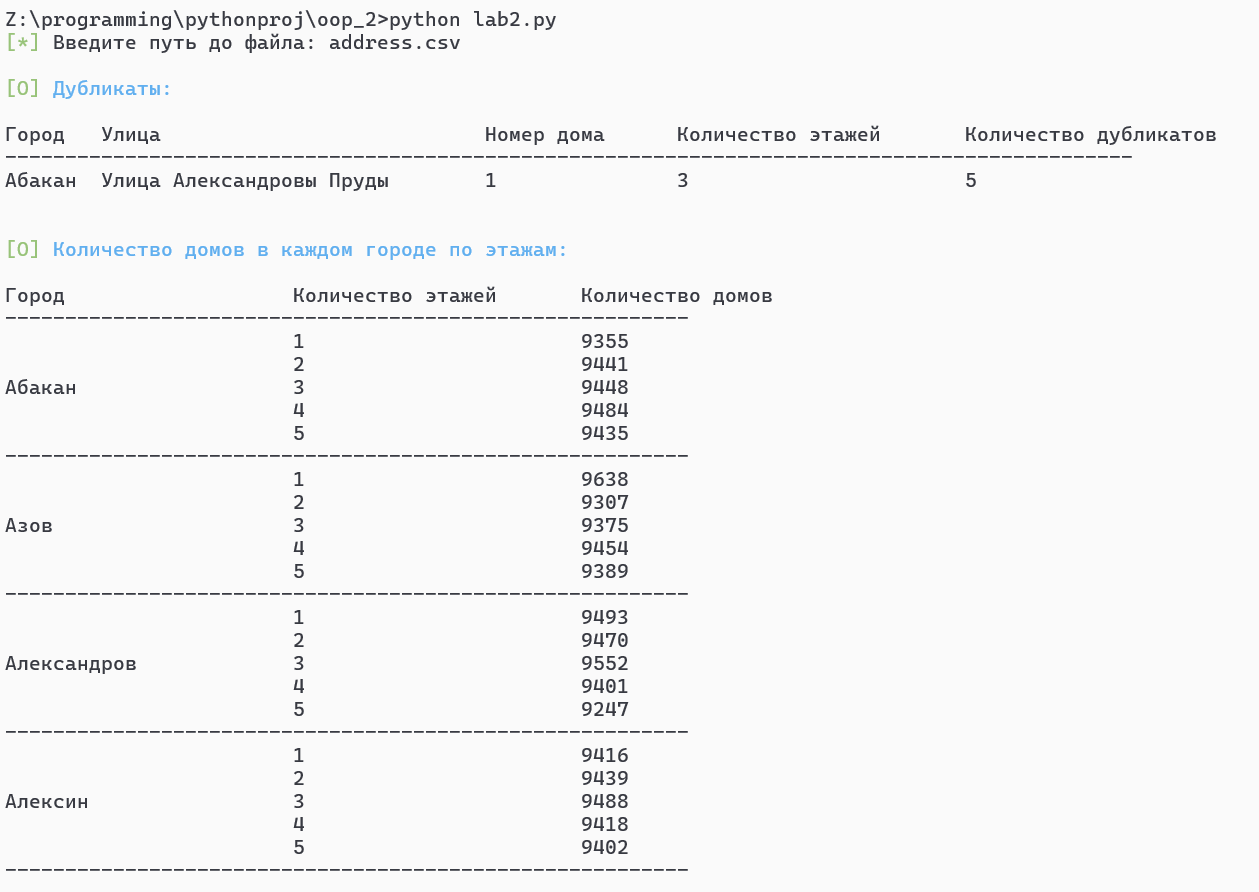


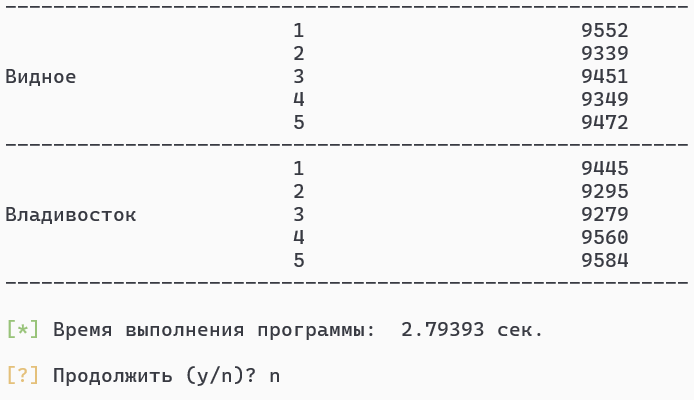
Рисунок 2. Исправный пример работы с csv файлом (продолжение). 

Рисунок 3. Первый пример работы с ошибкой.

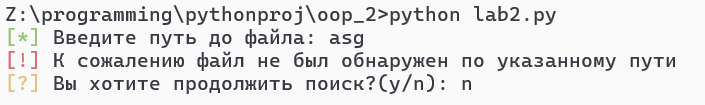


Рисунок 4. Второй пример работы с ошибкой.

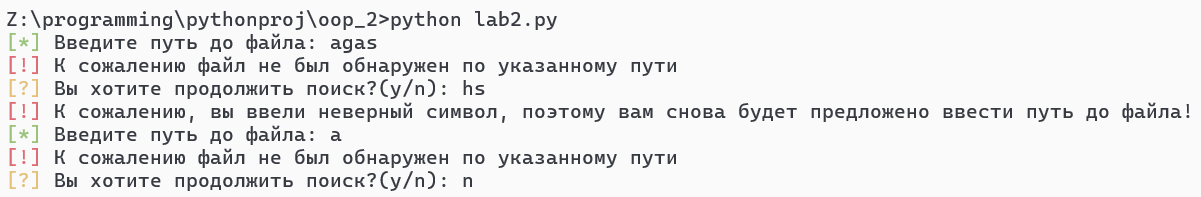


Рисунок 5. Третий пример работы с ошибкой.

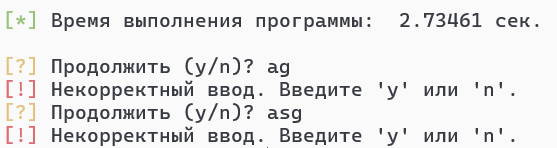


Рисунок 6. Четвертый пример работы с ошибкой.

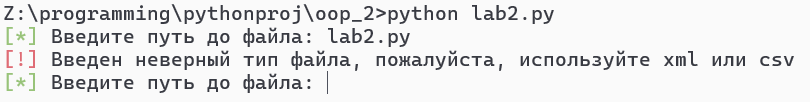


Рисунок 7. Удачный пример работы с xml файлом.

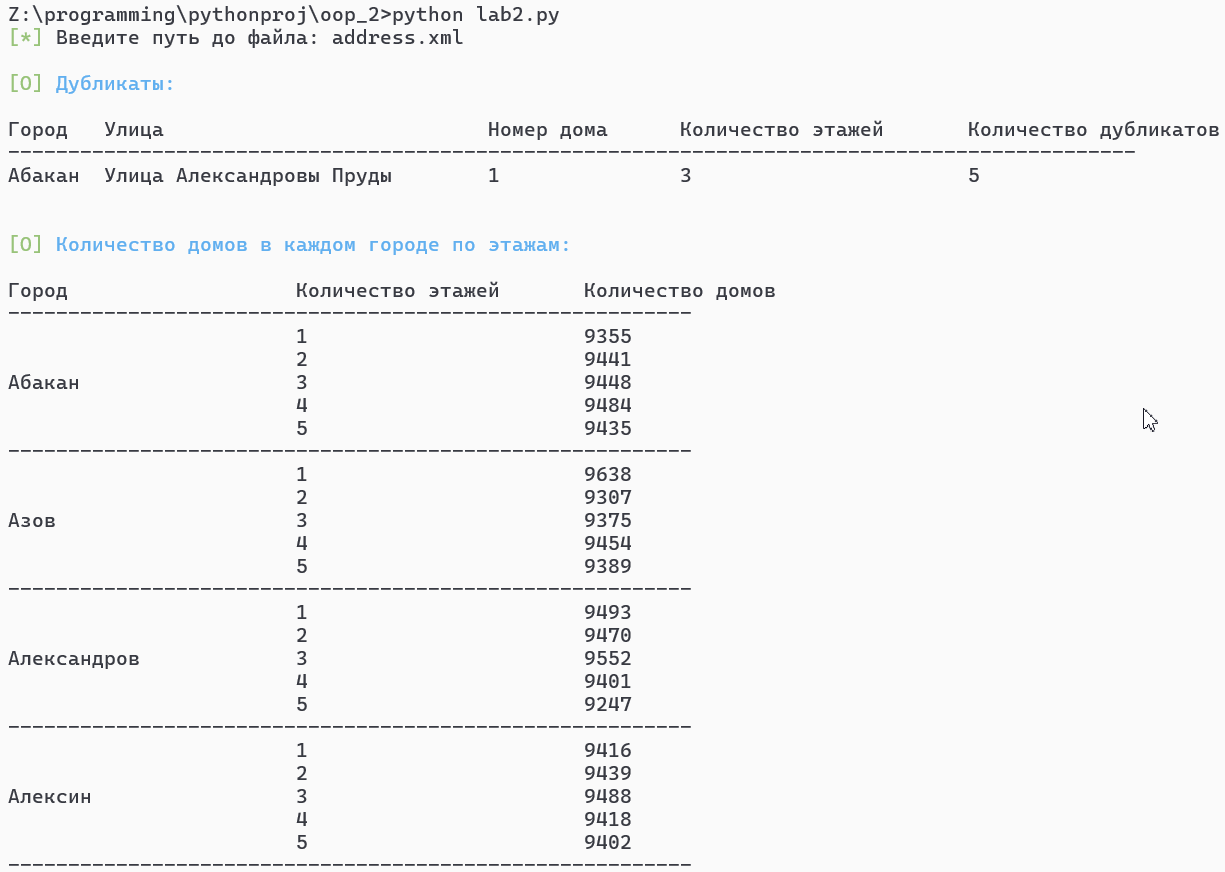
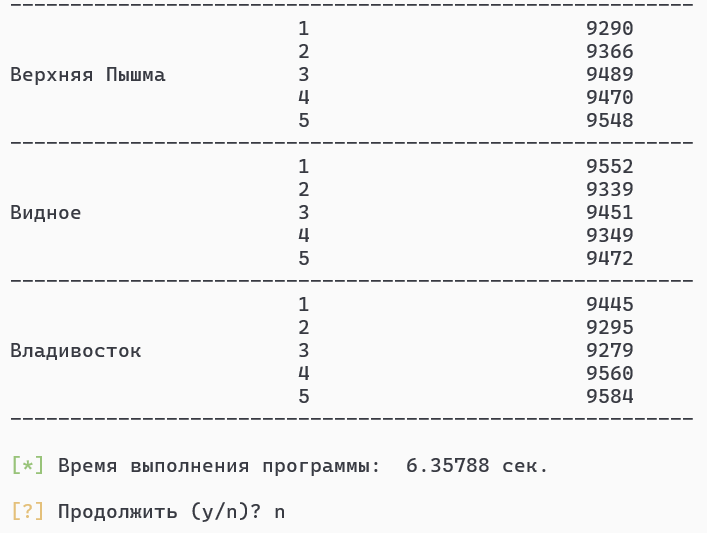


Рисунок 8. Удачный пример работы с xml файлом (продолжение).



# Код программы

import csv

import os

import xml.etree.ElementTree as ET

import time

class Color:

Reset = '\033[0m'

Red = '\033[91m'

Green = '\033[92m'

Yellow = '\033[93m'

Blue = '\033[94m'

class GetInfo:

def \_\_init\_\_(self):

self.path = self.get\_path()

self.count\_of\_houses = dict()

self.uniq\_rows = dict()

@staticmethod

def get\_path():

while True:

path\_to\_csv\_file = input("{}[\*]{} Введите путь до файла: ".format(Color.Green, Color.Reset))

if os.path.exists(path\_to\_csv\_file):

if path\_to\_csv\_file.endswith('.csv') or path\_to\_csv\_file.endswith('.xml'):

return path\_to\_csv\_file

else:

print("{}[!]{} Введен неверный тип файла, пожалуйста, используйте xml или csv".

format(Color.Red, Color.Reset))

else:

query = input("{}[!]{} К сожалению файл не был обнаружен по указанному пути\n"

"{}[?]{} Вы хотите продолжить поиск?(y/n): ".format(Color.Red,

Color.Reset,

Color.Yellow,

Color.Reset))

if query == 'y':

continue

elif query == 'n':

return None

else:

print("{}[!]{} К сожалению, вы ввели неверный символ, "

"поэтому вам снова будет предложено ввести путь до файла!".format(Color.Red, Color.Reset))

def get\_count\_of\_houses(self):

if self.uniq\_rows:

for k in self.uniq\_rows:

key = (k[0], k[3])

self.count\_of\_houses[key] = self.count\_of\_houses.setdefault(key, 0) + 1

self.count\_of\_houses = dict(sorted(self.count\_of\_houses.items()))

def uniq(self):

if self.path is not None:

if self.path.endswith(".csv"):

with open(self.path, 'r', encoding="utf-8") as f:

next(f)

spam\_reader = csv.reader(f, delimiter=";", quotechar='"')

for row in spam\_reader:

row\_tuple = tuple(row)

self.uniq\_rows[row\_tuple] = self.uniq\_rows.setdefault(row\_tuple, 0) + 1

else:

tree = ET.parse(self.path)

root = tree.getroot()

for item in root.findall("item"):

city = item.get('city')

street = item.get('street')

house = item.get('house')

floor = item.get('floor')

row\_tuple = (city, street, house, floor)

if all(row\_tuple):

self.uniq\_rows[row\_tuple] = self.uniq\_rows.setdefault(row\_tuple, 0) + 1

def out\_duplicates(self):

print("\n{}[O]{} Дубликаты: {}\n".format(Color.Green, Color.Blue, Color.Reset))

duplicates\_found = False

headers = ["Город", "Улица", "Номер дома", "Количество этажей", "Количество дубликатов"]

max\_lengths = [len(header) for header in headers]

for k, v in self.uniq\_rows.items():

if v > 1:

duplicates\_found = True

for i, item in enumerate(k):

max\_lengths[i] = max(max\_lengths[i], len(item))

header\_row = "\t".join(header.ljust(max\_lengths[i]) for i, header in enumerate(headers))

print(header\_row)

for k, v in self.uniq\_rows.items():

if v > 1:

print(f"{(max\_lengths[0] + 4) \* "-"}"

f"{(max\_lengths[1] + 4) \* "-"}"

f"{(max\_lengths[2] + 4) \* "-"}"

f"{(max\_lengths[3] + 4) \* "-"}"

f"{max\_lengths[4] \* "-"}")

print("\t".join(str(k[i]).ljust(max\_lengths[i]) for i in range(len(k))), f"\t{v}")

if not duplicates\_found:

print("{}[!]{} Дубликаты не были найдены".format(Color.Red, Color.Reset))

print("\n")

def out\_count\_of\_houses(self):

print("{}[O]{} Количество домов в каждом городе по этажам: {}\n".format(Color.Green, Color.Blue, Color.Reset))

if self.count\_of\_houses:

headers = ["Город", "Количество этажей", "Количество домов"]

max\_lengths = [len(header) for header in headers]

for (city, floors), count in self.count\_of\_houses.items():

max\_lengths[0] = max(max\_lengths[0], len(city))

max\_lengths[1] = max(max\_lengths[1], len(str(floors)))

max\_lengths[2] = max(max\_lengths[2], len(str(count)))

header\_row = "\t".join(header.ljust(max\_lengths[i]) for i, header in enumerate(headers))

print(header\_row)

prev\_city = ""

for (city, floors), count in self.count\_of\_houses.items():

if city != prev\_city:

print(f"{(max\_lengths[0] + 4) \* "-"}"

f"{(max\_lengths[1] + 4) \* "-"}"

f"{max\_lengths[2] \* "-"}")

if int(floors) == 3:

print(f"{str(city.ljust(max\_lengths[0]))}\t"

f"{str(floors).ljust(max\_lengths[1])}\t"

f"{str(count).ljust(max\_lengths[2])}")

else:

print(f"{''.ljust(max\_lengths[0])}\t"

f"{str(floors).ljust(max\_lengths[1])}\t"

f"{str(count).ljust(max\_lengths[2])}")

prev\_city = city

print(f"{(max\_lengths[0] + 4) \* "-"}"

f"{(max\_lengths[1] + 4) \* "-"}"

f"{max\_lengths[2] \* "-"}\n")

else:

print("{}[!]{} Файл оказался пустым".format(Color.Red, Color.Reset))

def process():

r = 1

while r:

proc = GetInfo()

if proc.path:

start\_time = time.time()

proc.uniq()

proc.get\_count\_of\_houses()

proc.out\_duplicates()

proc.out\_count\_of\_houses()

end\_time = time.time()

print("{}[\*]{} Время выполнения программы: ".format(Color.Green, Color.Reset),

round(end\_time - start\_time, 5), "сек.\n")

while True:

r = input("{}[?]{} Продолжить (y/n)? ".format(Color.Yellow, Color.Reset))

if r.lower() == 'y':

r = 1

break

elif r.lower() == 'n':

r = 0

break

else:

print("{}[!]{} Некорректный ввод. Введите 'y' или 'n'.".format(Color.Red, Color.Reset))

else:

break

if \_\_name\_\_ == '\_\_main\_\_':

process()

# Заключение

В данной лабораторной работе была проделана работа по работе с csv и xml файлами. Были изучены новые методы работы и рассмотрены разделы ООП, а именно разбиение на классы, создание конструктора и другое.

# Ссылка на репозиторий GitHub

<https://github.com/ZamniProg/oop_2>