Министерство науки и высшего образования Российской Федерации

Федеральное государственное бюджетное образовательное учреждение

высшего образования

«Ульяновский государственный технический университет»

Кафедра «Вычислительная техника»

Разработка профессиональных приложений

(название дисциплины)

**Отчёт по лабораторной работе №4**

«Классы»

(название (тема) работы)

**Вариант №13**

Выполнил:

студент группы ИВТАПбд-21

Молчанов А. В.

Проверил(а):

преподаватель кафедры «ВТ»

Исхаков И. И.

Ульяновск

2023

**Постановка задачи**

Дан файл data.csv, в котором содержится следующая информация в соответствии с вариантом: счётчик посетителей магазина: номер посетителя, дата и время, признак (bool) входа или выхода, пол, возраст и имя посетителя.

Используя объектно-ориентированный подход и классы, необходимо:

* вывести информацию об объектах, отсортировав их по одному полю (строковому);
* вывести информацию об объектах, отсортировав их по одному полю (числовому);
* вывести информацию, соответствующую какому-либо критерию (пользователи, которые находятся в магазине на данный момент).

К классам предъявляются следующие требования:

* класс должен содержать итератор;
* должна быть реализована перегрузка стандартных операций (repr, например);
* должно быть реализовано наследование;
* запись значений в свойства - только через \_\_setattr\_\_;
* возможность доступа к элементам коллекции по индексу (\_\_getitem\_\_);
* должны быть реализованы статические методы;
* должны быть реализованы генераторы.

Также, программа должна сохранять новые данные обратно в .csv файл.

**Описание реализации**

Для работы с .csv файлами использовалась специальная библиотека импорта и экспорта для электронных таблиц и баз данных csv, а также csv.DictReader и csv.DictWriter для чтения и записи файлов соответственно. Для сортировки списков использовались методы sorted и filter, а также лямбда-функции с условиями сортировки.

В соответствии с требованиями предъявляемыми к классам, было реализовано 2 класса — это родительский класс «Человек» и дочерний ему класс «Пользователь» (посетитель) Интернет-магазина. Таким образом было реализовано наследование. В классе «Пользователь» были реализованы итератор и генератор, которые позволяют пройтись по списку с датами всех посещений пользователем магазина. Перегрузка стандартной операции repr позволяет выводить в консоль только необходимую информацию о том или ином пользователе и делать это в достаточно понятном формате.

Для реализации такой парадигмы ООП как инкапсуляция, переменные в обоих классах были сделаны с модификатором доступа \_\_private, а доступ к ним осуществлён только через соответствующие «геттеры» (getitem) и «сеттеры» (setattr).

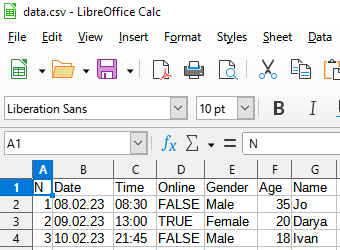
Также, в классе «Человек» был реализован статический метод на проверку совершеннолетия того или иного объекта этого класса.

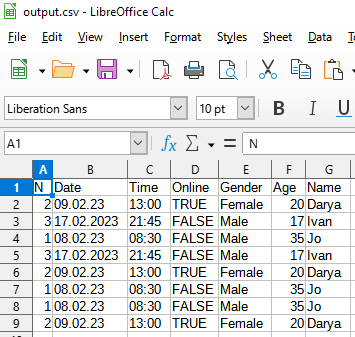
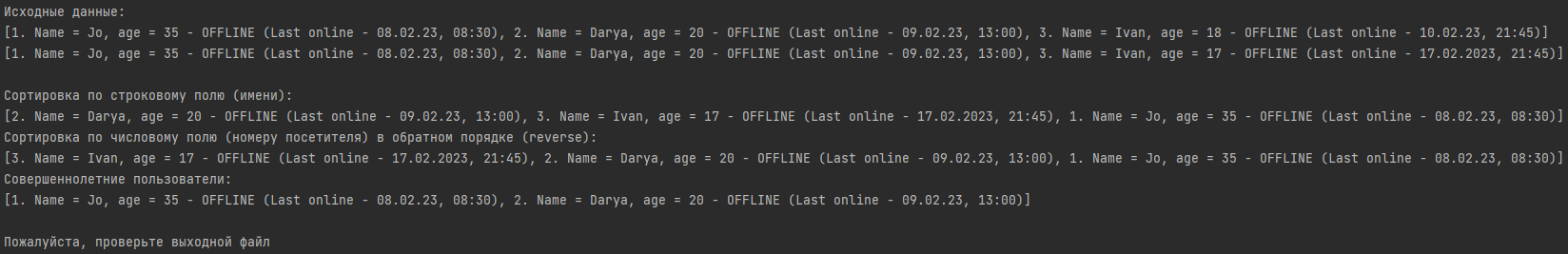
В конце было реализовано сохранение новых данных в выходной файл output.csv.

**Исходный код реализации**

import csv # Библиотека импорта и экспорта для электронных таблиц и баз данных  
import datetime  
  
  
# Родительский класс "Человек"  
class Person:  
 # Метод инициализации экземпляров класса после их создания  
 def \_\_init\_\_(self, name, age, gender):  
 self.\_\_name = name  
 self.\_\_age(age)  
 self.\_\_gender = gender  
  
 def \_\_age(self, value):  
 if value <= 120:  
 self.\_\_age = value  
  
 # Статические метод проверки на совершеннолетие  
 @staticmethod  
 def is\_adult(age):  
 return age >= 18  
  
 # Метод (сеттер) для получения имени человека (пользователя)  
 def setName(self, newName):  
 self.\_\_name = newName  
  
 # Метод (сеттер) для получения возраста человека (пользователя)  
 def setAge(self, newAge):  
 if newAge <= 120:  
 self.\_\_age = newAge  
  
 # Метод (сеттер) для получения пола человека (пользователя)  
 def setGender(self, newGender):  
 if (newGender == "Male") or (newGender == "Female"):  
 self.\_\_gender = newGender  
  
 # Метод (геттер) для изменения имени человека (пользователя)  
 def getName(self):  
 return self.\_\_name  
  
 # Метод (геттер) для изменения возраста человека (пользователя)  
 def getAge(self):  
 return self.\_\_age  
  
 # Метод (геттер) для изменения пола человека (пользователя)  
 def getGender(self):  
 return self.\_\_gender  
  
  
# Дочерний класс "Пользователь", наследованный от родительского класса "Человек"  
class User(Person):  
 def \_\_init\_\_(self, name, age, gender, id, date, time, online):  
 Person.\_\_init\_\_(self, name, age, gender)  
 self.\_\_id = id  
 self.\_\_visits = []  
 self.login(date)  
 self.\_\_time = time  
 self.\_\_online = online  
  
 # Метод, который вызывается при новом посещении пользователем Интернет-магазина  
 def login(self, date):  
 self.\_\_visits.append(date)  
  
 # Кастомный итератор класса (описание ниже)  
 def \_\_iter\_\_(self):  
 return UserDateIterator(self.\_\_visits)  
  
 # Переопределённый метод вывода информации об объекте класса "Пользователь"  
 def \_\_repr\_\_(self):  
 # Если сегодня пользователь заходил на сайт  
 if (datetime.datetime.now().strftime("%d.%m.%Y") == self.\_\_visits[len(self.\_\_visits) - 1]):  
 # Значит он онлайн (время не учитывается)  
 return f"{self.\_\_id}. Name = {self.getName()}, age = {self.getAge()} - ONLINE (Log in at {self.\_\_time})"  
 else:  
 # Иначе он оффлайн (выводим, когда пользователь заходил на сайт крайний раз)  
 return f"{self.\_\_id}. Name = {self.getName()}, age = {self.getAge()} - OFFLINE (Last online - {self.\_\_visits[len(self.\_\_visits) - 1]}, {self.\_\_time})"  
  
 # Генератор для перебора списка посещений пользователем Интернет-магазина  
 def generator(self, i):  
 while i < len(self.\_\_visits):  
 yield self.\_\_visits[i] # Ключевое слово yield возвращает нас обратно в этот метод, запоминая состояние выхода  
 i += 1  
 else:  
 yield 0  
  
 # Метод присвоения значения value какому-либо атрибуту key объекта класса "Пользователь"  
 def \_\_setattr\_\_(self, key, value):  
 self.\_\_dict\_\_[key] = value  
  
 # Метод позволяет получить значение какого-либо атрибута объекта класса "Пользователь" по его индексу  
 def \_\_getitem\_\_(self, item):  
 s = "\_User\_\_" + item  
 return self.\_\_dict\_\_[s]  
  
  
# Класс кастомного итератора для класса "Пользователь"  
class UserDateIterator():  
 # При создании итератора, получаем список посещений пользователем Интернет-магазина и инициализируем счётчик для его обхода  
 def \_\_init\_\_(self, visits):  
 self.visits = visits  
 self.i = 0  
  
 # Метод возвращает следующий элемент из списка посещений  
 def \_\_next\_\_(self):  
 if self.i < len(self.visits):  
 date = self.visits[self.i]  
 self.i += 1  
 return date  
 else:  
 return 0  
  
  
fI = open("C:\\Users\\Admin\\PycharmProjects\\DPA\\lab4\\data.csv", 'r') # Открываем файл для чтения (read)  
# Класс DictReader модуля csv создаёт объект, который работает как обычный reader(), но отображает информацию о каждой строке в качестве словаря dict  
reader = csv.DictReader(fI, fieldnames = None, restkey = None, restval = None, dialect = "excel")  
  
l = []  
  
for row in reader:  
 # Создаём объект класса пользователя и добавляем его в список  
 a = User(row["Name"], int(row["Age"]), row["Gender"], int(row['N']), row["Date"], row["Time"], row["Online"])  
 l.append(a)  
  
print("Исходные данные: ")  
print(l)  
  
l[2].setAge(17)  
  
l[2].login("17.02.2023")  
  
print(l)  
print()  
  
# Сортируем список при помощи лямбда-функции по столбцу с именем из таблицы  
sSL = sorted(l, key = lambda d: d.getName())  
# Сортируем список при помощи лямбда-функции по столбцу с номером клиента из таблицы и переворачиваем список (reverse)  
nSL = sorted(l, key = lambda d: d['id'], reverse = True)  
  
print("Сортировка по строковому полю (имени): ")  
print(sSL)  
print("Сортировка по числовому полю (номеру посетителя) в обратном порядке (reverse): ")  
print(nSL)  
  
newL = []  
  
# Отбор посетителей по совершеннолетию (только те, кому больше 17 лет)  
newL = list(filter(lambda x: x.is\_adult(x.getAge()), l))  
print("Совершеннолетние пользователи: ")  
print(newL)  
  
# Создаём файл для записи  
with open("C:\\Users\\Admin\\PycharmProjects\\DPA\\lab4\\output.csv", 'w', newline = '') as csvfile:  
 writer = csv.DictWriter(csvfile, fieldnames = reader.fieldnames) # Класс writer служит для записи данных в файл  
 writer.writeheader() # Записываем заголовок таблицы  
  
 for row in sSL:  
 d = {}  
 d['N'] = row['id'] # Доступ к элементам коллекции по индексу осуществляется при помощи метода \_\_getitem\_\_  
 d['Date'] = row['visits'][-1]  
 d['Time'] = row['time']  
 d['Online'] = row['online']  
 d['Gender'] = row.getGender()  
 d['Age'] = row.getAge()  
 d['Name'] = row.getName()  
  
 # Метод writerow записывает словарь со значениями в файл  
 writer.writerow(d)  
  
 for row in nSL:  
 d = {}  
 d['N'] = row['id']  
  
 # Запись даты через итератор класса  
 iterator = iter(row)  
 b = next(iterator)  
 d['Date'] = str(b)  
  
 while (b):  
 b = next(iterator)  
  
 if (b != 0):  
 d['Date'] = str(b)  
 #  
  
 d['Time'] = row['time']  
 d['Online'] = row['online']  
 d['Gender'] = row.getGender()  
 d['Age'] = row.getAge()  
 d['Name'] = row.getName()  
  
 writer.writerow(d)  
  
 for row in newL:  
 d = {}  
 d['N'] = row['id']  
  
 # Запись даты через генератор класса  
 i = 0  
  
 generator = row.generator(i)  
 b = next(generator)  
 d['Date'] = str(b)  
  
 while (b):  
 b = next(generator)  
  
 if (b != 0):  
 d['Date'] = str(b)  
 #  
  
 d['Time'] = row['time']  
 d['Online'] = row['online']  
 d['Gender'] = row.getGender()  
 d['Age'] = row.getAge()  
 d['Name'] = row.getName()  
  
 writer.writerow(d)  
  
print()  
print("Пожалуйста, проверьте выходной файл")

**Пример работы программы**





**Описание возникших затруднений**

Основные затруднения при выполнении данной лабораторной работы были связаны с реализацией инкапсуляции, так как я до сих пор не уверен, что реализовал её правильным образом.

Я не понял как можно применить переопределение методов setattr и getitem к приватным переменным, так как по сути, у приватных переменных просто изменяется название (\_\_Название класса\_имя переменной), а доступ к ним можно получить всё также извне, но только по новому имени. При этом обратиться к той или иной переменной через встроенный словарь класса \_\_dict\_\_ у меня всё-таки не получилось, поэтому я реализовал «сеттеры» и «геттеры» обычными публичными методами по типу setName, getAge и так далее.

**Описание альтернативных способов решения**

Альтернативным способом решения данной задачи является 3 лабораторная работа с её словарями и списками словарей. Они гораздо проще и понятнее в работе, чем классы, но это и их главный недостаток, так как дополнительно структурировать и масштабировать информацию, хранящуюся в подобных структурах данных достаточно проблематично и вряд ли вообще возможно. Объектно-ориентированный подход же значительно упрощает эту задачу и гораздо ближе человеческому восприятию.

**Вывод**

В ходе выполнения данной лабораторной работы мной были получены теоретические знания об устройстве объектно-ориентированного программирования в Python (до этого момента я вообще думал, что ООП в Питоне нет) и практические навыки работы с классами в данном языке программирования. Также, я узнал, что такое итераторы и генераторы классов