Министерство науки и высшего образования Российской Федерации

Федеральное государственное бюджетное образовательное учреждение высшего образования «Ульяновский государственный технический университет»

Кафедра «Вычислительная техника»

**Отчет по лабораторной работе №4**

Дисциплина: «Разработка профессиональных приложений»

Классы

Вариант № 17

Выполнил:

студент группы ИВТАПбд-21

Волкова К.В.

Проверил:

преподаватель кафедры

«Вычислительная техника»

Исхаков И.И.

Ульяновск, 2023

**Задание по варианту**

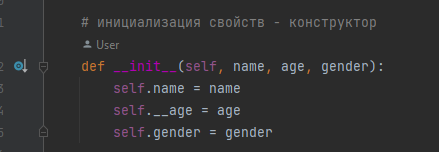
Необходимо переделать лабораторную работу №3 с использованием классов, описывающих предметную область, заданную вариантом, с реализацией следующих особенностей (вполне возможно, что предлагаемое в 3 лабе задание для этого нужно будет расширить):

**Требования к программе:**

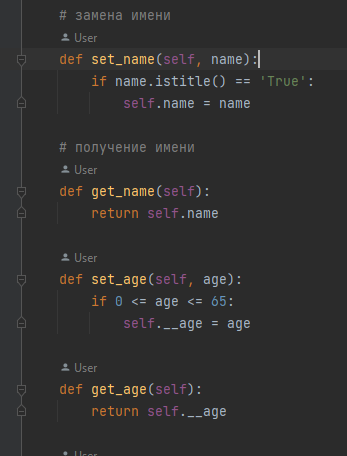
1. Класс должен содержать итератор
2. Должна быть реализована перегрузка стандартных операций (repr, например)
3. Должно быть реализовано наследование
4. Запись значений в свойства - только через \_\_setattr\_\_
5. Возможность доступа к элементам коллекции по индексу (\_\_getitem\_\_)
6. Должны быть реализованы статические методы
7. Должны быть реализованы генераторы

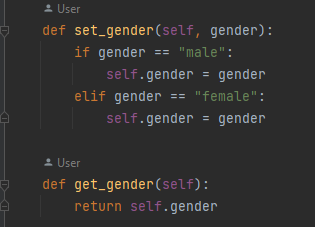
**Описание реализации**

\_\_init\_\_: Инициализирует объект класса Human.

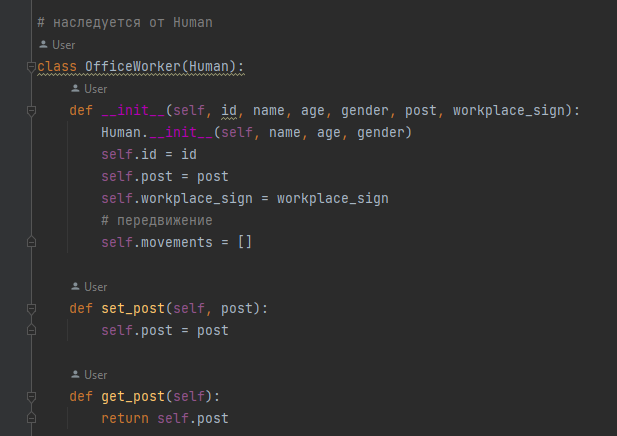


Set/get заменяет и получает данные

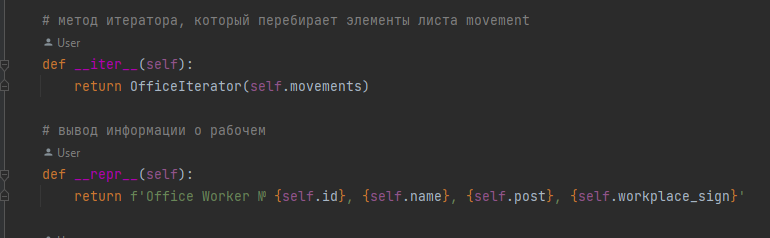




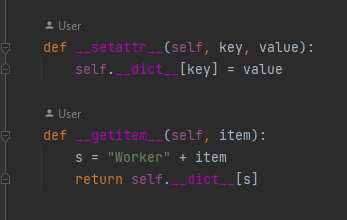
OfficeWorker – наследуемый класс:



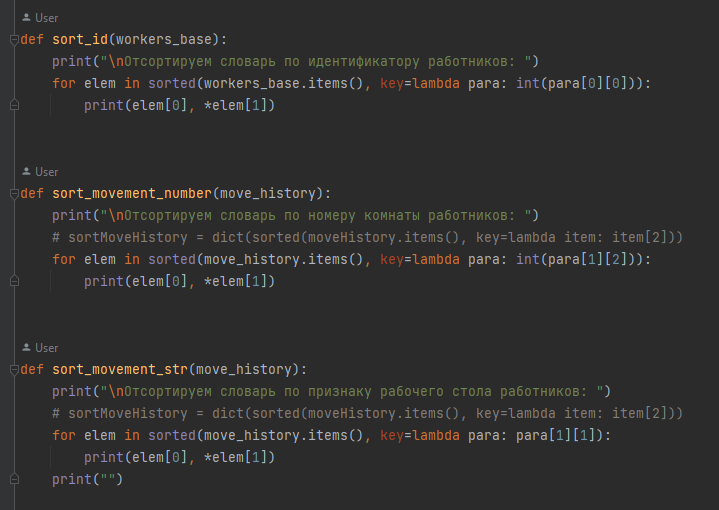
\_\_iter\_\_ метод, который перебирает элементы, а \_\_repr\_\_ выводит информацию о рабочих.



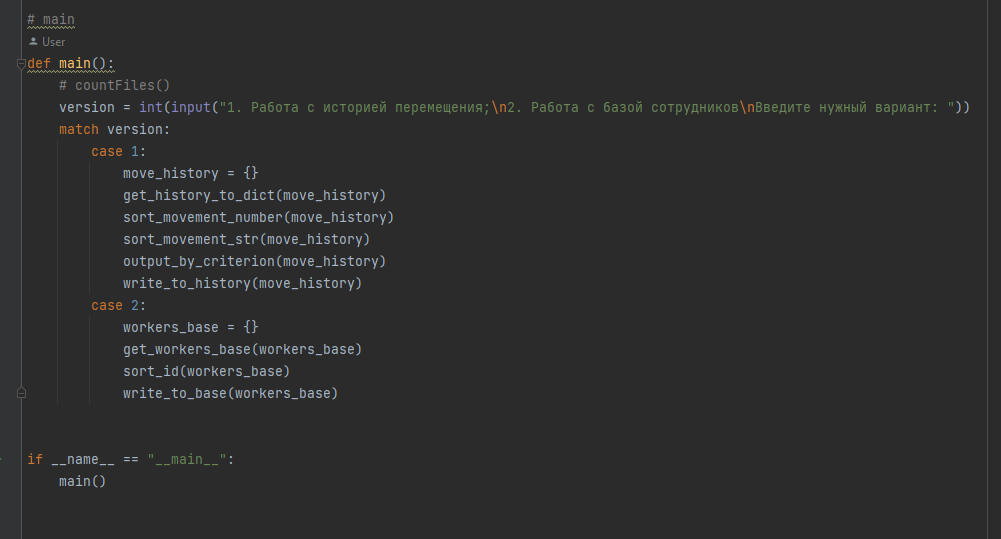
\_\_setattr\_\_ и \_\_getitem\_\_ реализованы так:



Реализация сортировок выглядит так:



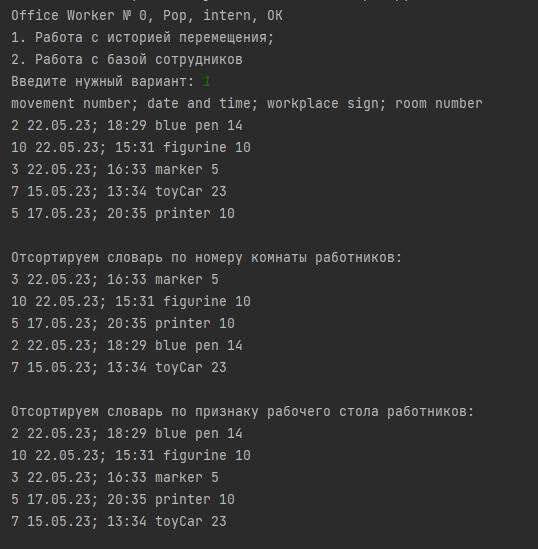
Функция main: Создает экземпляр класса OfficeWorker, вызывает различные методы для выполнения задач: подсчета объектов, чтения данных из файла, вывода данных, сортировки и фильтрации, а также добавления новой записи в файл.

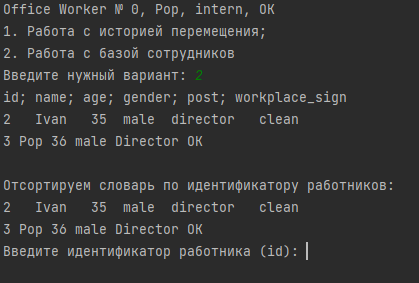


**Описание возникших затруднений**

Затруднений не возникло

**Результаты работы программы**





**Код программы**

# Лабораторная работа №4  
  
import os  
from pathlib import Path  
import csv  
  
  
class Human(object):  
 *"Базовый класс для определения человека"* # инициализация свойств - конструктор  
 def \_\_init\_\_(self, name, age, gender):  
 self.name = name  
 self.\_\_age = age  
 self.gender = gender  
  
 # замена имени  
 def set\_name(self, name):  
 if name.istitle() == 'True':  
 self.name = name  
  
 # получение имени  
 def get\_name(self):  
 return self.name  
  
 def set\_age(self, age):  
 if 0 <= age <= 65:  
 self.\_\_age = age  
  
 def get\_age(self):  
 return self.\_\_age  
  
 @staticmethod  
 def is\_adult(age):  
 if age >= 18:  
 return age  
  
 def set\_gender(self, gender):  
 if gender == "male":  
 self.gender = gender  
 elif gender == "female":  
 self.gender = gender  
  
 def get\_gender(self):  
 return self.gender  
  
# наследуется от Human  
class OfficeWorker(Human):  
 def \_\_init\_\_(self, id, name, age, gender, post, workplace\_sign):  
 Human.\_\_init\_\_(self, name, age, gender)  
 self.id = id  
 self.post = post  
 self.workplace\_sign = workplace\_sign  
 # передвижение  
 self.movements = []  
  
 def set\_post(self, post):  
 self.post = post  
  
 def get\_post(self):  
 return self.post  
  
 # дата передвижения  
 def movement(self, date):  
 self.movements.append(date)  
  
 # метод итератора, который перебирает элементы листа movement  
 def \_\_iter\_\_(self):  
 return OfficeIterator(self.movements)  
  
 # вывод информации о рабочем  
 def \_\_repr\_\_(self):  
 return f'Office Worker № {self.id}, {self.name}, {self.post}, {self.workplace\_sign}'  
  
 def generator(self, i):  
 while i < len(self.movements):  
 yield self.movements[i]  
 i += 1  
 else:  
 yield 0  
  
 def \_\_setattr\_\_(self, key, value):  
 self.\_\_dict\_\_[key] = value  
  
 def \_\_getitem\_\_(self, item):  
 s = "Worker" + item  
 return self.\_\_dict\_\_[s]  
  
  
class OfficeIterator:  
 def \_\_init\_\_(self, movements, start=0):  
 OfficeWorker.\_\_init\_\_(self, movements)  
 self.movements = movements  
 self.movements\_count = start  
  
 def \_\_next\_\_(self):  
 if self.movements\_count < len(self.movements):  
 date = self.movements[self.movements\_count]  
 self.movements\_count += 1  
 return date  
 else:  
 return 0  
  
 def \_\_iter\_\_(self):  
 return self  
  
  
office\_worker1 = OfficeWorker(0, "Pop", "20", "male", "intern", "OK")  
print(office\_worker1.\_\_repr\_\_())  
  
  
  
def get\_history\_to\_dict(move\_history):  
 print("movement number; date and time; workplace sign; room number")  
 with open("movement\_history.csv", "r", encoding='utf-8') as f1:  
 reader = csv.reader(f1)  
 for row in reader:  
 print(\*row)  
 move\_history[row[0]] = row[1:]  
  
  
def get\_workers\_base(workers\_base):  
 print("id; name; age; gender; post; workplace\_sign")  
 with open('Office\_workers.csv', 'r', encoding='utf-8') as f2:  
 reader = csv.reader(f2)  
 for row in reader:  
 print(\*row)  
 workers\_base[row[0]] = row[1:]  
  
  
def sort\_id(workers\_base):  
 print("\nОтсортируем словарь по идентификатору работников: ")  
 for elem in sorted(workers\_base.items(), key=lambda para: int(para[0][0])):  
 print(elem[0], \*elem[1])  
  
  
def sort\_movement\_number(move\_history):  
 print("\nОтсортируем словарь по номеру комнаты работников: ")  
 # sortMoveHistory = dict(sorted(moveHistory.items(), key=lambda item: item[2]))  
 for elem in sorted(move\_history.items(), key=lambda para: int(para[1][2])):  
 print(elem[0], \*elem[1])  
  
  
def sort\_movement\_str(move\_history):  
 print("\nОтсортируем словарь по признаку рабочего стола работников: ")  
 # sortMoveHistory = dict(sorted(moveHistory.items(), key=lambda item: item[2]))  
 for elem in sorted(move\_history.items(), key=lambda para: para[1][1]):  
 print(elem[0], \*elem[1])  
 print("")  
  
  
def output\_by\_criterion(move\_history):  
 criterion = input("Введите признак рабочего стола: \n")  
 print(f'\nСтроки, в которых признак рабочего стола "{criterion}": ')  
 for elem in move\_history:  
 if move\_history[elem][1] == criterion:  
 print(elem, \*move\_history[elem])  
  
  
def write\_to\_history(move\_history):  
 newKey = input("Введите номер перемещения (ключ): ")  
 newDate = input("Введите дату и время через точку с запятой: ")  
 newWorkplaceSign = input("Введите признак рабочего стола: ")  
 newRoomNumber = input("Введите номер комнаты: ")  
 move\_history[newKey] = [newDate, newWorkplaceSign, newRoomNumber]  
 with open('movement\_history.csv', 'w', encoding='utf-8') as f:  
 for elem in move\_history:  
 f.write(elem + ',' + move\_history[elem][0] + ',' + move\_history[elem][1] + ',' + move\_history[elem][2] + '\n')  
  
  
def write\_to\_base(workers\_base):  
 new\_id = input("Введите идентификатор работника (id): ")  
 new\_name = input("Введите имя сотрудника: ")  
 new\_age = input("Введите возраст сотрудника: ")  
 new\_gender = input("Введите пол сотрудника: ")  
 new\_post = input("Введите должность сотрудника: ")  
 new\_workplace\_sign = input("Введите признак рабочего стола: ")  
 workers\_base[new\_id] = [new\_name, new\_age, new\_gender, new\_post, new\_workplace\_sign]  
 with open('Office\_workers.csv', "w", encoding='utf-8') as f2:  
 for elem in workers\_base:  
 f2.write(elem + ',' + workers\_base[elem][0] + ',' + workers\_base[elem][1] + ',' + workers\_base[elem][2]  
 + ',' + workers\_base[elem][3] + ',' + workers\_base[elem][4] + '\n')  
  
  
  
  
# main  
def main():  
 # countFiles()  
 version = int(input("1. Работа с историей перемещения;\n2. Работа с базой сотрудников\nВведите нужный вариант: "))  
 match version:  
 case 1:  
 move\_history = {}  
 get\_history\_to\_dict(move\_history)  
 sort\_movement\_number(move\_history)  
 sort\_movement\_str(move\_history)  
 output\_by\_criterion(move\_history)  
 write\_to\_history(move\_history)  
 case 2:  
 workers\_base = {}  
 get\_workers\_base(workers\_base)  
 sort\_id(workers\_base)  
 write\_to\_base(workers\_base)  
  
  
if \_\_name\_\_ == "\_\_main\_\_":  
 main()