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 №9

Выполнил:

студент группы ИВТАПбд-22

Ступин А. О.

Проверил:

преподаватель кафедры

«Вычислительная техника»

Исхаков И.И.

Ульяновск, 2023

**Задание по варианту**

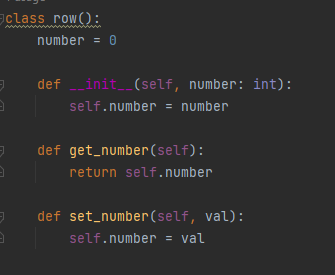
Необходимо переделать лабораторную работу №3 с использованием классов, описывающих предметную область, заданную вариантом, с реализацией следующих особенностей (вполне возможно, что предлагаемое в 3 лабе задание для этого нужно будет расширить):

**Требования к программе:**

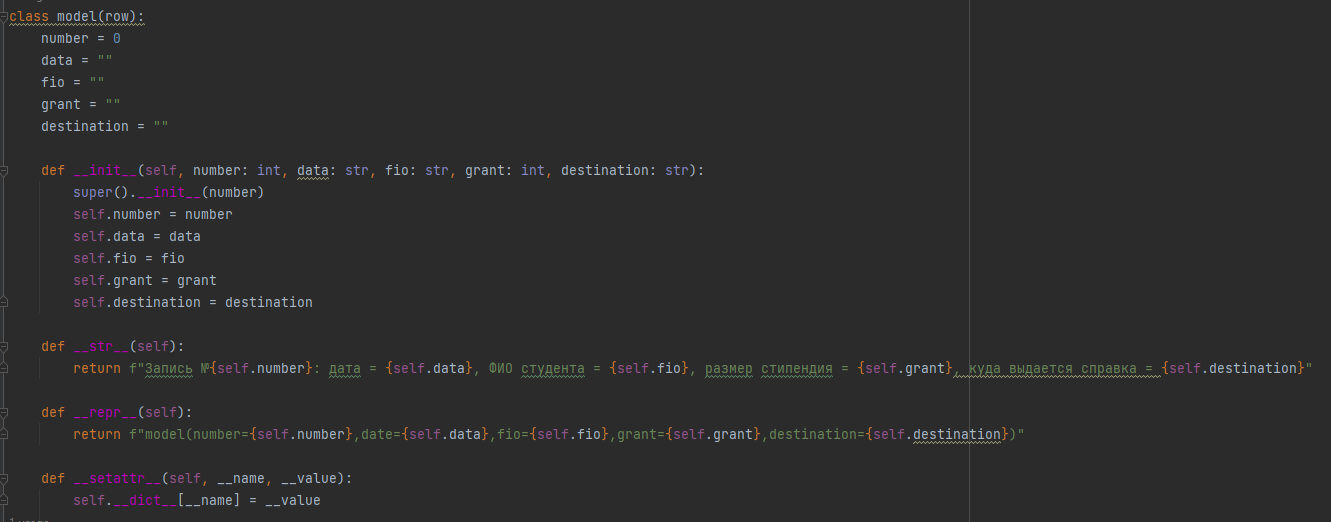
1. Класс должен содержать итератор
2. Должна быть реализована перегрузка стандартных операций (repr, например)
3. Должно быть реализовано наследование
4. Запись значений в свойства - только через \_\_setattr\_\_
5. Возможность доступа к элементам коллекции по индексу (\_\_getitem\_\_)
6. Должны быть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

**Описание реализации**

Класс row содержит атрибут number и методы init, get\_numbe и set\_number. Метод init устанавливает значение атрибута number. Метод get\_number возвращает значение атрибута number. Метод set\_number устанавливает значение атрибута number. Number – это идентификатор в файле data.csv.



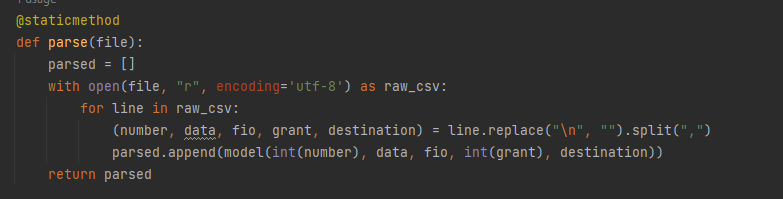
Класс model наследуется от класса row. Метод init устанавливает значения всех атрибутов, включая унаследованный атрибут number, c помощью метода init родительского класса. Метод str возвращает строку, содержащую значения всех атрибутов. Метод repr реализует перегрузку и возвращает строку, представляющую объект класса model в виде кода Python. Метод setattr устанавливает значение атрибута по его имени.



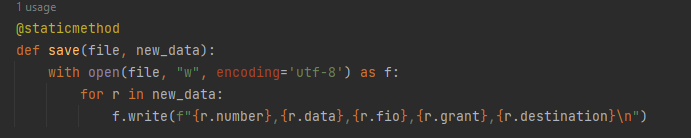
В классе Data атрибут file\_path хранит путь к файлу, data хранит словарь данных, а pointer используется для отслеживания списка данных. Метод init устанавливает file\_path и data объекта класса на основе файла с данными, используя метод parse (парсинг). Метод str позволяет получить строковое представление данных, где каждый элемент данных выводится на новой строке.

Метод iter возвращает объект, который может быть использован для итерации по данным. Метод next возвращает следующий элемент данных в каждой итерации и, когда достигает конца списка, вызывает исключение StopIteration. Метод getitem получает элемент данных из определенной позиции в списке, если указанный индекс целое число и в диапазоне допустимых значений, иначе вызывает соответствующие исключения.

Статическим методом открывается файл data.csv на чтение.



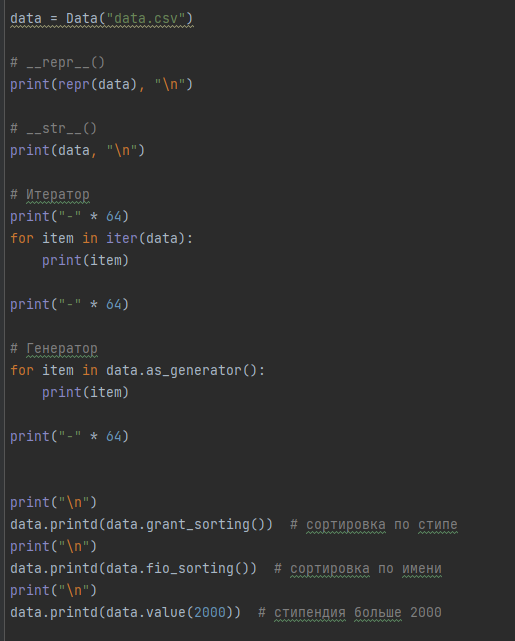
Ещё один – на запись (и сохранение).



Сортировки из Лабораторной работы №3.

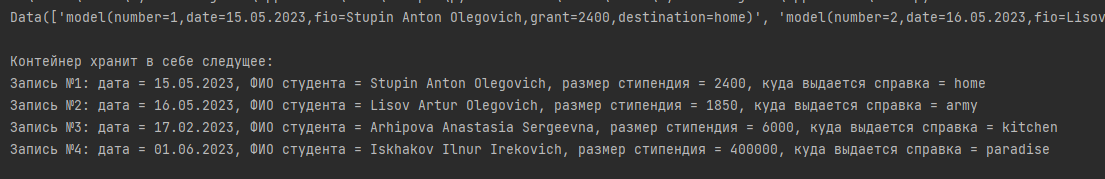


Наконец, происходит вывод строковых представлений словаря, выводы через итератор и генератор (в выводе выделены «-----------------») и выводы с сортировками.

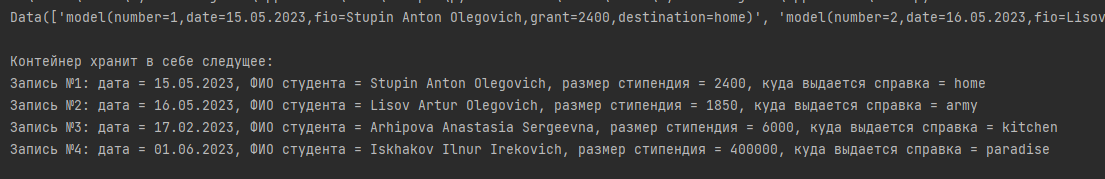


**Результаты работы программы**

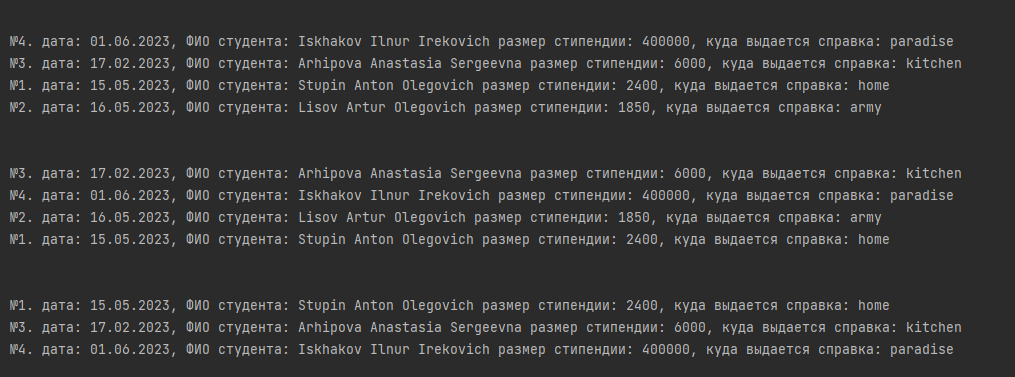
Выводы строкового представления через repr и содержимого словаря через str:



Итератор и генератор:



Сортировки (стипендия, фио, стипендия > 2000):



Приложение 1. Исходный код программы

import os  
  
  
def processing():  
 DIR = 'test\_data'  
 count\_of\_files =len([name for name in os.listdir(DIR) if os.path.isfile(os.path.join(DIR, name))])  
 print(f"Количество файлов в директории: {count\_of\_files}")  
  
class row():  
 number = 0  
  
 def \_\_init\_\_(self, number: int):  
 self.number = number  
  
 def get\_number(self):  
 return self.number  
  
 def set\_number(self, val):  
 self.number = val  
  
class model(row):  
 number = 0  
 data = ""  
 fio = ""  
 grant = ""  
 destination = ""  
  
 def \_\_init\_\_(self, number: int, data: str, fio: str, grant: int, destination: str):  
 super().\_\_init\_\_(number)  
 self.number = number  
 self.data = data  
 self.fio = fio  
 self.grant = grant  
 self.destination = destination  
  
 def \_\_str\_\_(self):  
 return f"Запись №{self.number}: дата = {self.data}, ФИО студента = {self.fio}, размер стипендия = {self.grant}, куда выдается справка = {self.destination}"  
  
 def \_\_repr\_\_(self):  
 return f"model(number={self.number},date={self.data},fio={self.fio},grant={self.grant},destination={self.destination})"  
  
 def \_\_setattr\_\_(self, \_\_name, \_\_value):  
 self.\_\_dict\_\_[\_\_name] = \_\_value  
class Data():  
 file\_path = ""  
 data = {}  
 pointer = 0  
  
 def \_\_init\_\_(self, file):  
 self.file\_path = file  
 self.data = self.parse(file)  
  
 def \_\_str\_\_(self):  
 d\_str = '\n'.join([str(rm) for rm in self.data])  
 return f"Содержимое:\n{d\_str}"  
  
 def \_\_repr\_\_(self):  
 return f"Data({[repr(rm) for rm in self.data]})"  
  
 def \_\_iter\_\_(self):  
 return self  
  
 def \_\_next\_\_(self):  
 if self.pointer >= len(self.data):  
 self.pointer = 0  
 raise StopIteration  
 else:  
 self.pointer += 1  
 return self.data[self.pointer - 1]  
  
 def \_\_getitem\_\_(self, item):  
 if not isinstance(item, int):  
 raise TypeError("Индекс должен быть целым числом.")  
 if 0 <= item < len(self.data):  
 return self.data[item]  
 else:  
 raise IndexError("Неверный индекс.")  
  
 def as\_generator(self):  
 self.pointer = 0  
 while self.pointer < len(self.data):  
 yield self.data[self.pointer]  
 self.pointer += 1  
  
 @staticmethod  
 def parse(file):  
 parsed = []  
 with open(file, "r", encoding='utf-8') as raw\_csv:  
 for line in raw\_csv:  
 (number, data, fio, grant, destination) = line.replace("\n", "").split(",")  
 parsed.append(model(int(number), data, fio, int(grant), destination))  
 return parsed  
  
 def fio\_sorting(self):  
 return sorted(self.data, key=lambda f: f.fio)  
  
 def grant\_sorting(self):  
 return sorted(self.data, key=lambda f: f.grant, reverse=True)  
  
 def value(self, value):  
 r = []  
 for d in self.data:  
 if (d.grant > value):  
 r.append(d)  
 return r  
  
 def add\_new(self, data, fio, grant, destination):  
 self.data.append(model(len(self.data) + 1, data, fio, grant, destination))  
 self.save(self.file\_path, self.data)  
  
 @staticmethod  
 def save(file, new\_data):  
 with open(file, "w", encoding='utf-8') as f:  
 for r in new\_data:  
 f.write(f"{r.number},{r.data},{r.fio},{r.grant},{r.destination}\n")  
  
 def print(self):  
 for r in self.data:  
 print(f"№{r.number}. дата: {r.data}, ФИО студента: {r.fio}, размер стипендии: {r.grant}, куда выдается справка: {r.destination}")  
  
 def printd(self, d):  
 for r in d:  
 print(f"№{r.number}. дата: {r.data}, ФИО студента: {r.fio} размер стипендии: {r.grant}, куда выдается справка: {r.destination}")  
  
def get\_files\_count\_in\_directory(file):  
 (loc, dirs, files) = next(os.walk(file))  
 return len(files)  
  
data = Data("data.csv")  
  
# \_\_repr\_\_()  
print(repr(data), "\n")  
  
# \_\_str\_\_()  
print(data, "\n")  
  
# Итератор  
print("-" \* 64)  
for item in iter(data):  
 print(item)  
  
print("-" \* 64)  
  
# Генератор  
for item in data.as\_generator():  
 print(item)  
  
print("-" \* 64)  
  
  
print("\n")  
data.printd(data.grant\_sorting()) # сортировка по стипе  
print("\n")  
data.printd(data.fio\_sorting()) # сортировка по имени  
print("\n")  
data.printd(data.value(2000)) # стипендия больше 2000