**Федеральное государственное бюджетное образовательное учреждение**

**высшего образования**

**«РОССИЙСКАЯ АКАДЕМИЯ НАРОДНОГО ХОЗЯЙСТВА   
И ГОСУДАРСТВЕННОЙ СЛУЖБЫ**

**ПРИ ПРЕЗИДЕНТЕ РОССИЙСКОЙ ФЕДЕРАЦИИ»**

**НИЖЕГОРОДСКИЙ ИНСТИТУТ УПРАВЛЕНИЯ – филиал РАНХиГС**

**Отчет по программированию №3**

Выполнил: студент группы Иб-321

Комышков Владислав Дмитриевич

Оглавление

[Задание 2 3](#__RefHeading___Toc1410_1638435388)

[Ход работы 3](#__RefHeading___Toc1412_1638435388)

[Итог 7](#__RefHeading___Toc1414_1638435388)

[Задание 3 8](#__RefHeading___Toc1416_1638435388)

[Ход работы 8](#__RefHeading___Toc1418_1638435388)

[Итог 14](#__RefHeading___Toc1420_1638435388)

[Задание 4 16](#__RefHeading___Toc1422_1638435388)

[Ход работы 16](#__RefHeading___Toc1424_1638435388)

[Итог 18](#__RefHeading___Toc1426_1638435388)

[Задание 6 19](#__RefHeading___Toc1428_1638435388)

[Класс 19](#__RefHeading___Toc1430_1638435388)

[Пример работы класса 20](#__RefHeading___Toc1432_1638435388)

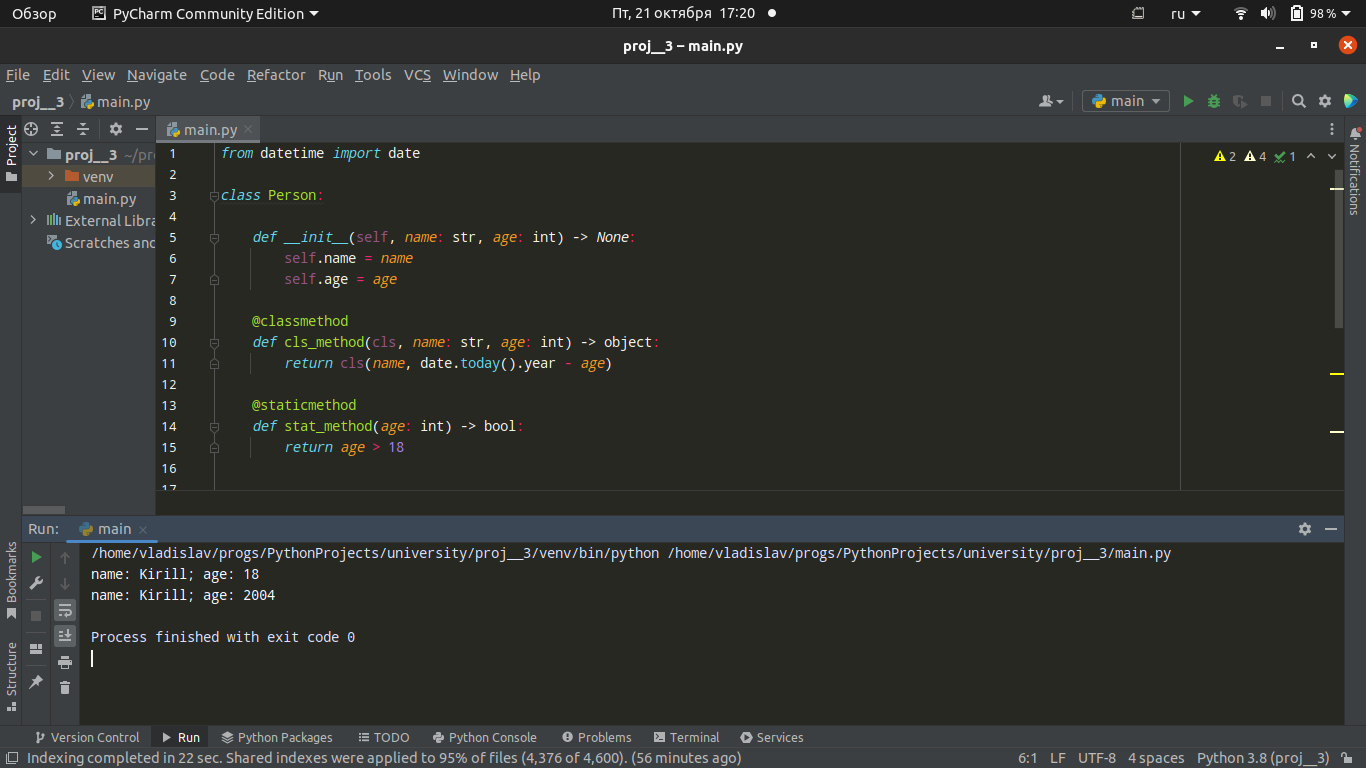
# Задание 2

**Текст задания:**

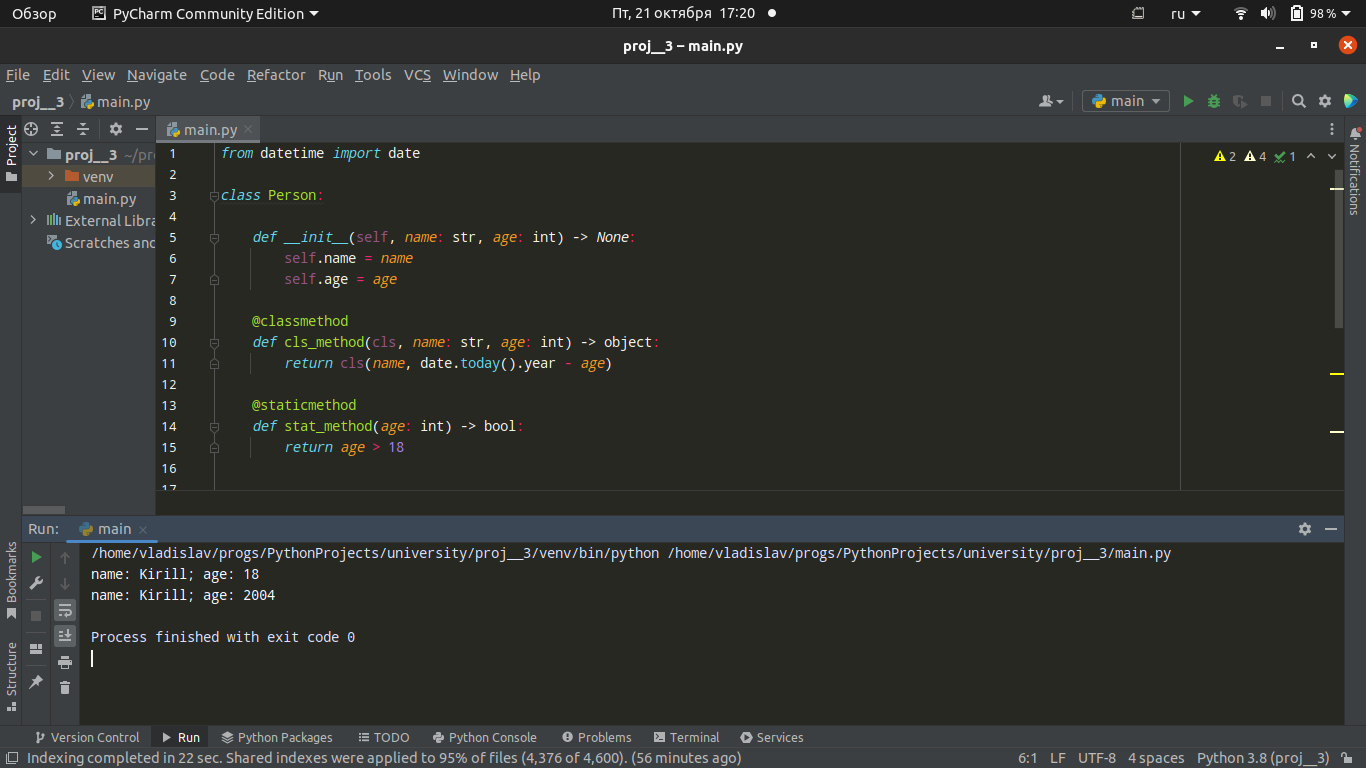
1. Реализовать код, приведённый на рисунке 3.6. Понять и описать как он работает.
2. Переделать программу так, чтобы она реагировала на возраст Николая и Ивана.

## Ход работы

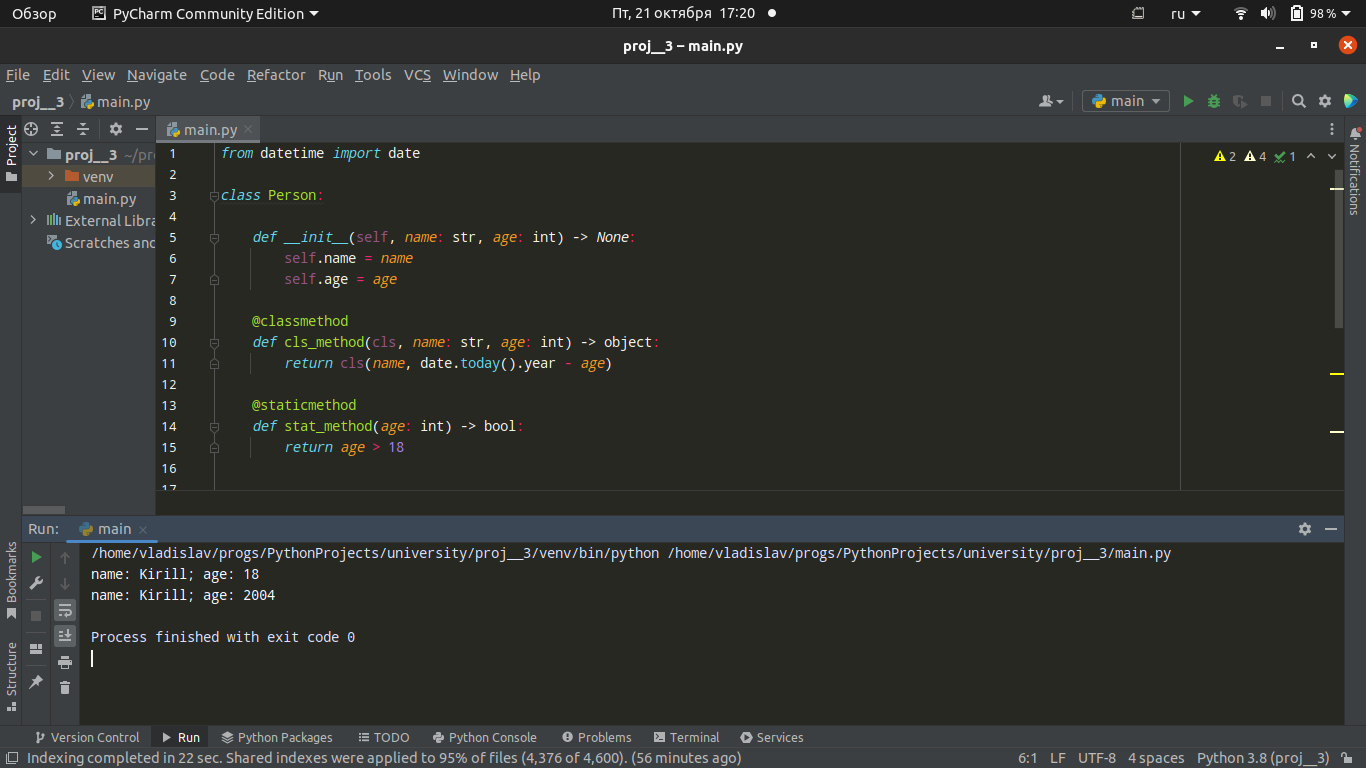
Создаем класс Person (Рис 1.1). Прописываем конструктор, принимающий два значения name(имя) и age(возраст).

Рис 1.1 - класс Person

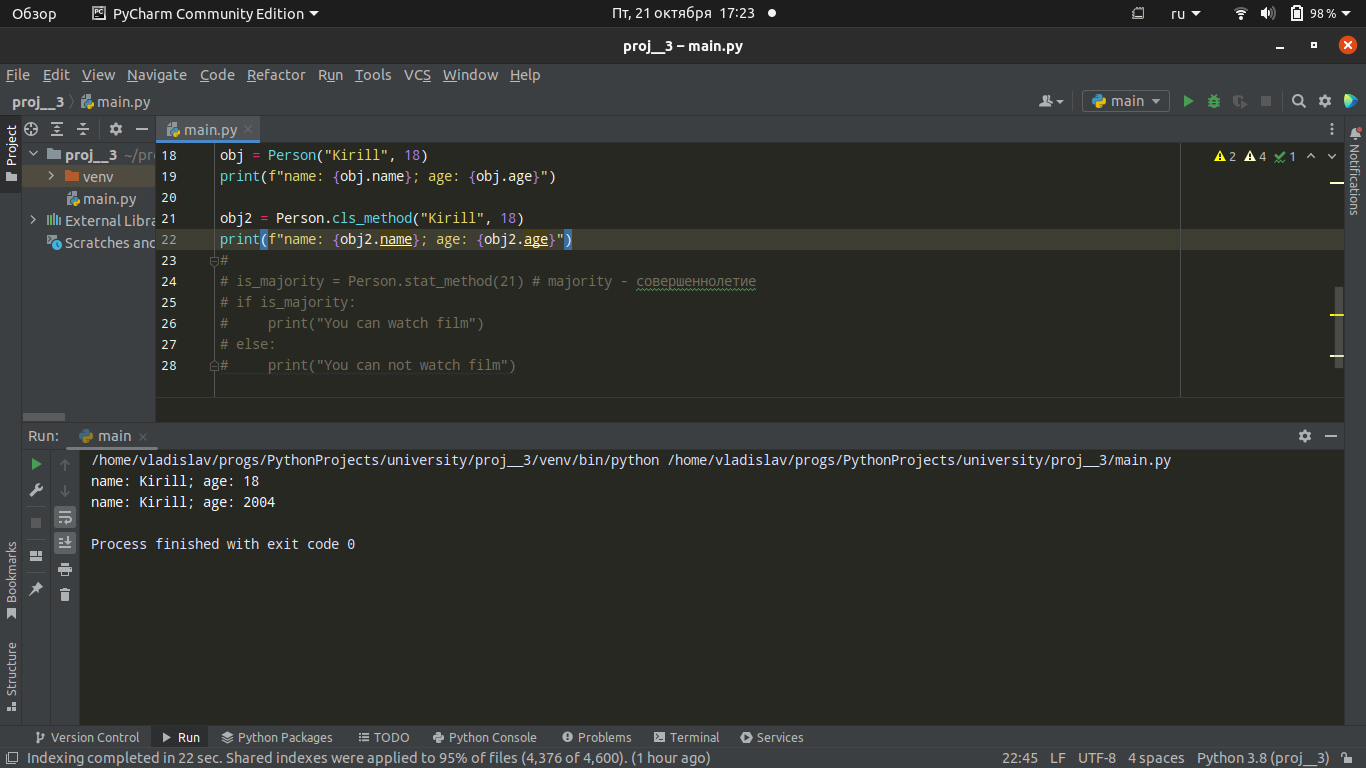
Добавляем метод класса cls\_method (Рис 1.2) и отмечаем его декоратором @classmethod. Скрытый атрибут cls содержит объект класса, соответственно не может влиять на другие экземпляры, но может взаимодействовать с атрибутами класса.

Рис 1.2 - метод класса cls\_method

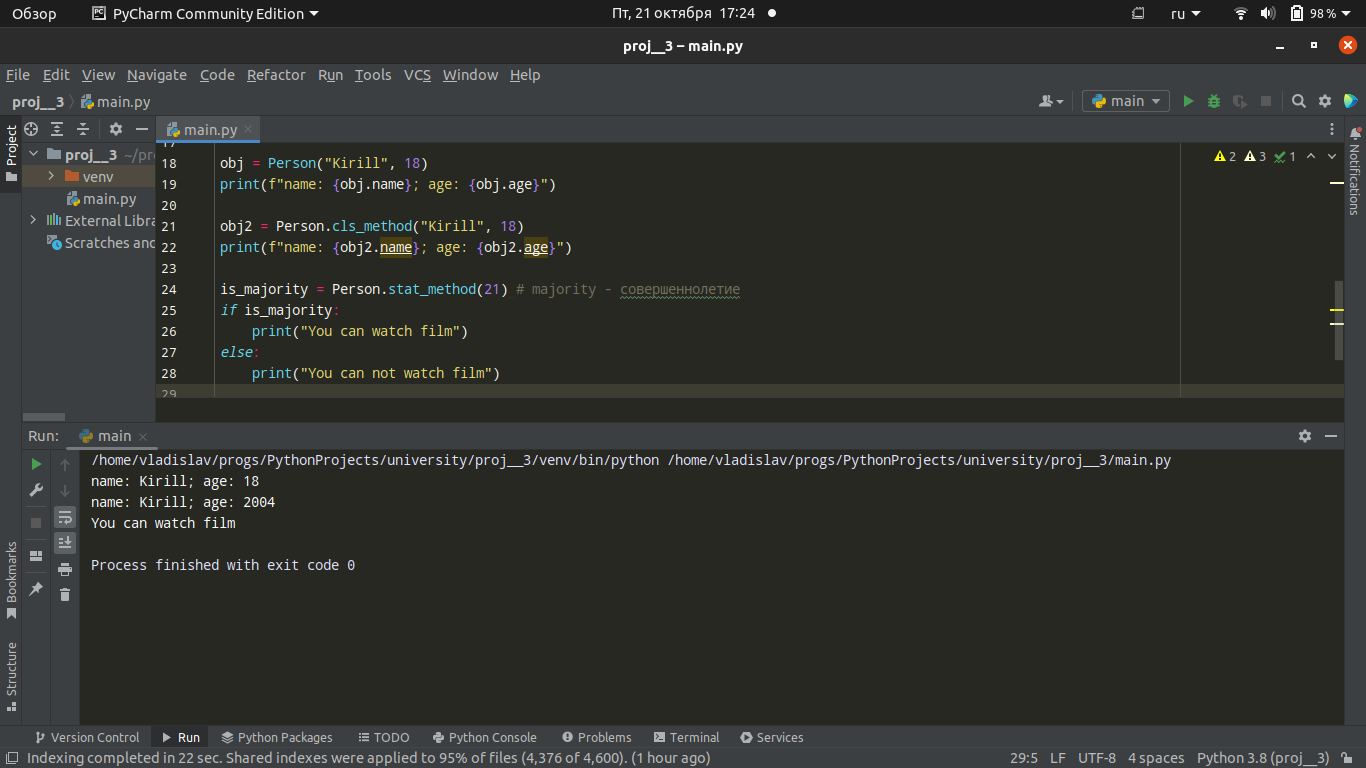
Также добавим статический метод класса stat\_method (Рис 1.3), отметив его декоратором @staticmethod, для проверки возраста. Этот метод не может влиять на экземпляры и не может взаимодействовать с атрибутами класса.

Рис 1.3 - метод класса stat\_method

Создадим два объекта (Рис 1.4), первый будет создан с помощью конструктора, второй через метод класса. Обратимся к их атрибутам и выведем в консоль.

Рис 1.4 — проверка методов класса

Следующим шагом проверим работу статического метода (Рис 1.5). Результат выведем в консоль.

Рис 1.5 — скрипт и резульат работы программы

Перепишем класс так, чтобы при создании объекта, с помощью нашего статического метода stat\_method, сразу проверять возраст (Рис 1.6).

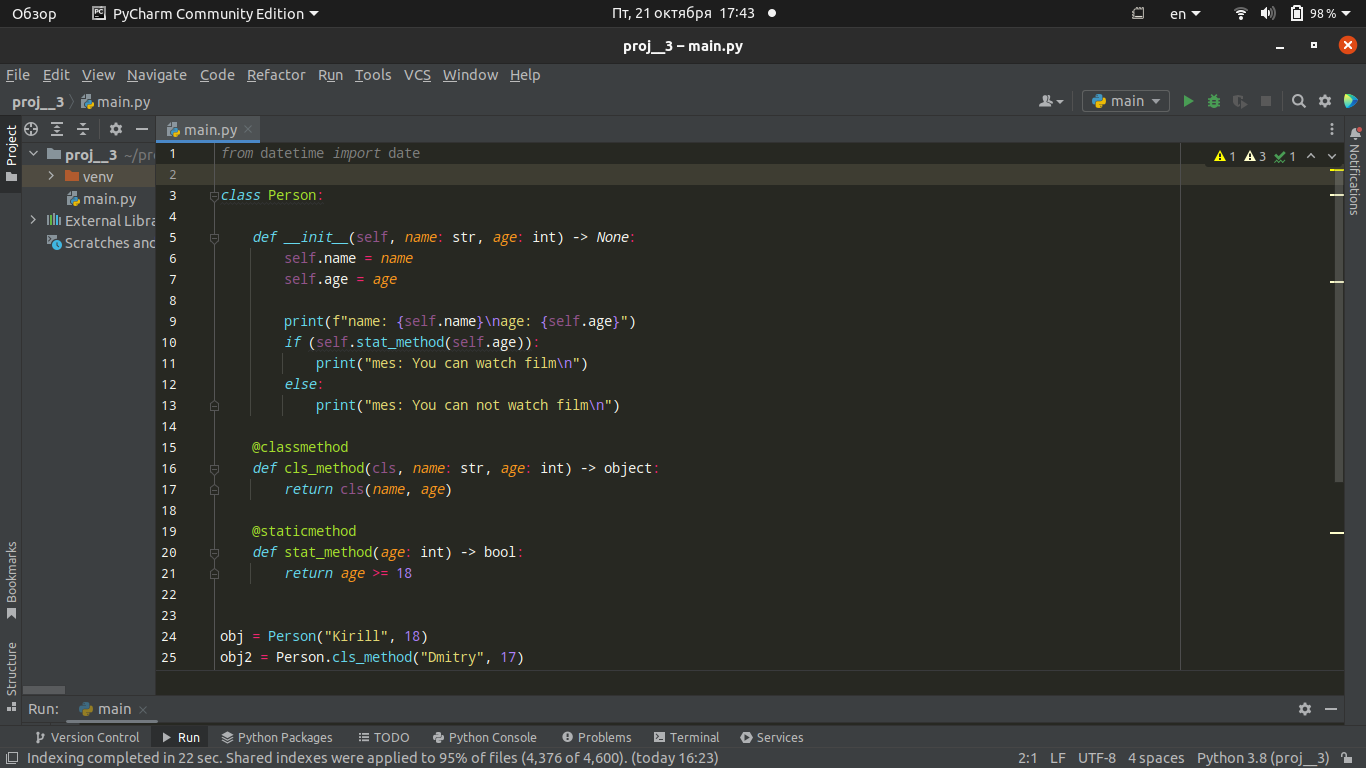
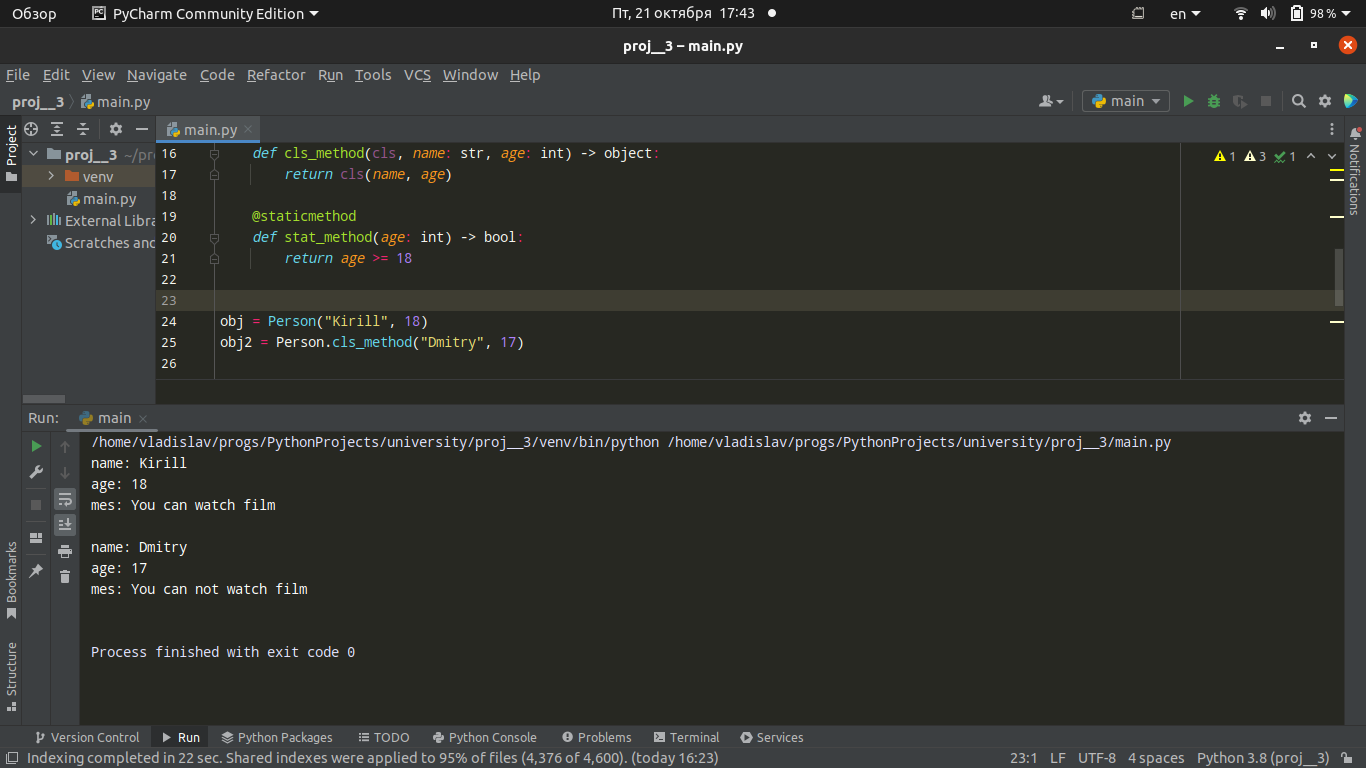


Рис 1.6 — класс Person и его методы

Подправим скрипт. Используя те же объекты, запустим программу и посмотрим на результат (Рис 1.7).

Рис 1.7 — скрипт и результат работы программы

### Итог

Мы реализовали код, приведённый в примере. Описали его работу.

Переделали программу так, чтобы она реагировала на возраст (атрибут) объектов.

**Примечание:**

В ходе работы изменены имена переменных, методов, выводимые сообщения в консоль, а также задаваемые в качестве атрибутов имени (name) и возраста (age) значения.

# Задание 3

**Текст задания:**

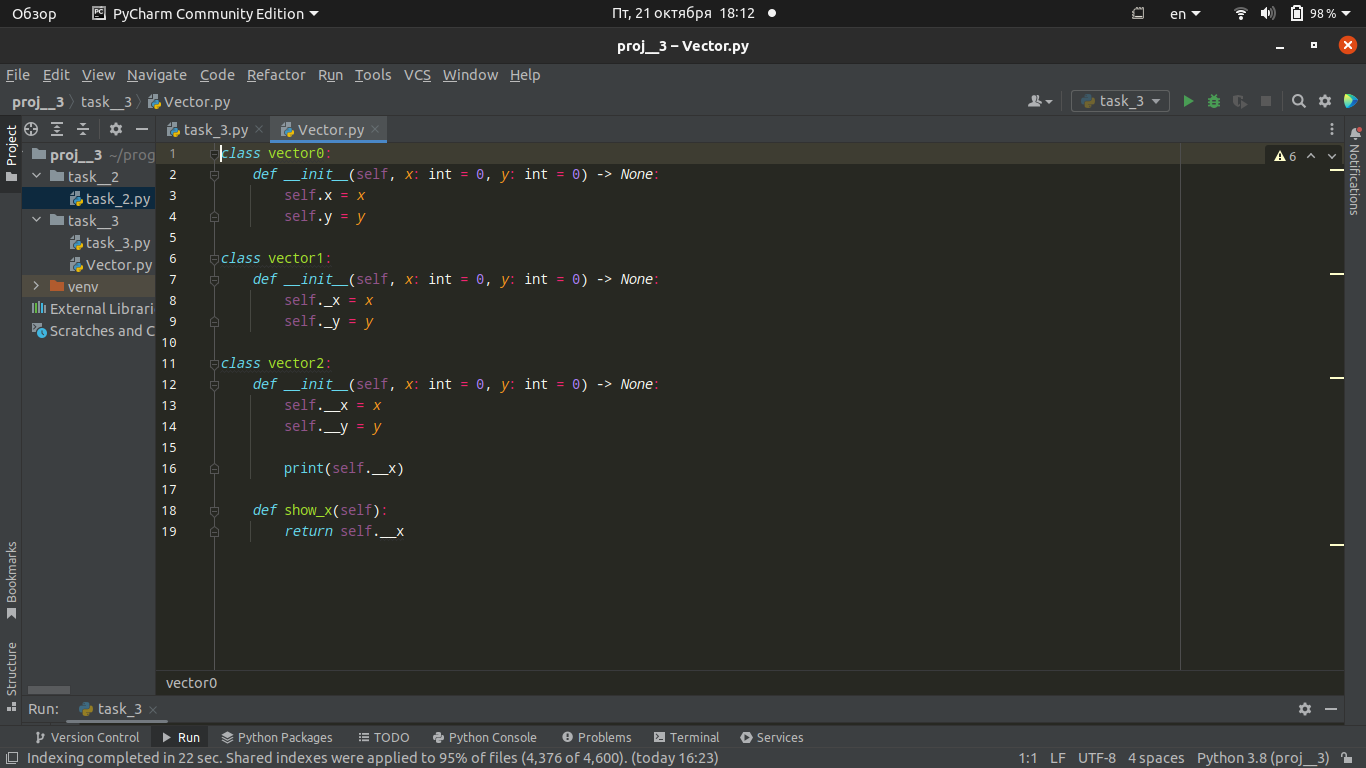
1. Разобраться, как работает код, представленный на рисунке 3.8.

2. Предсказать результат выполнения, в том числе для варианта снятия значка комментария в строке 126.

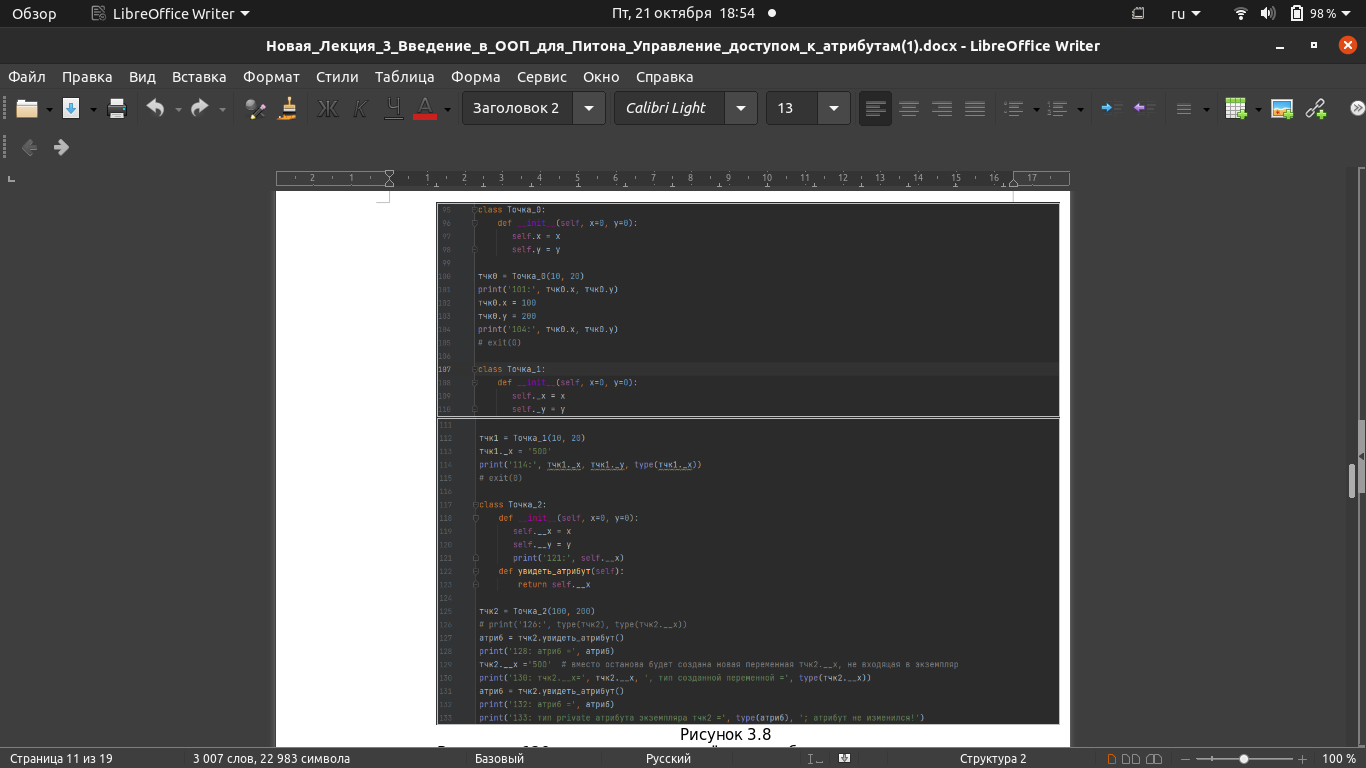
3. Реализовать код и проверить правильность предсказания на практике.

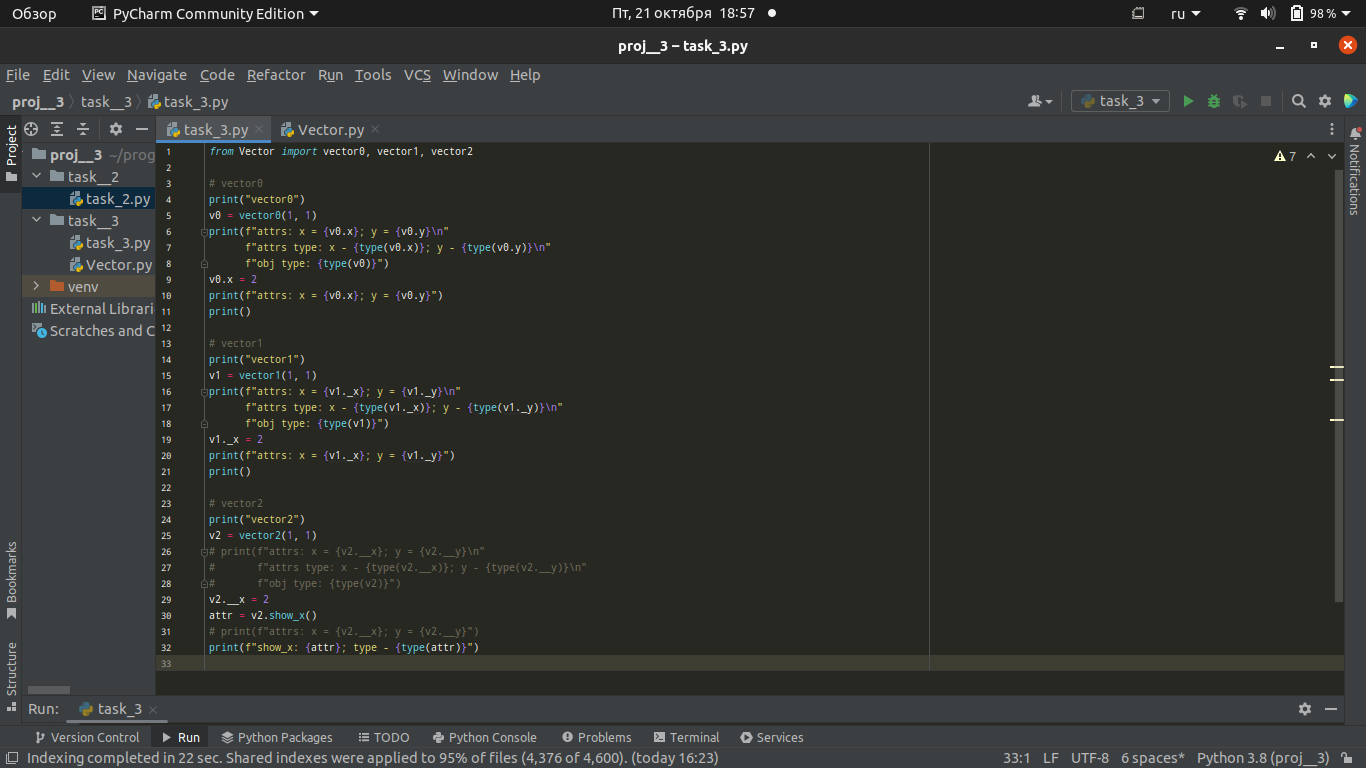
## Ход работы

На рисунке 2.1 представлены классы vector0, vector1, vector2 и их методы.

Рис 2.1 — классы vector0, vector1, vector2 и их методы

На рисунках 2.2.1 и 2.2.2 представлены код программы из задания и реализованный нами код

Рис 2.2.1 — код программы из задания

Рис 2.2.2 — код нашей программы

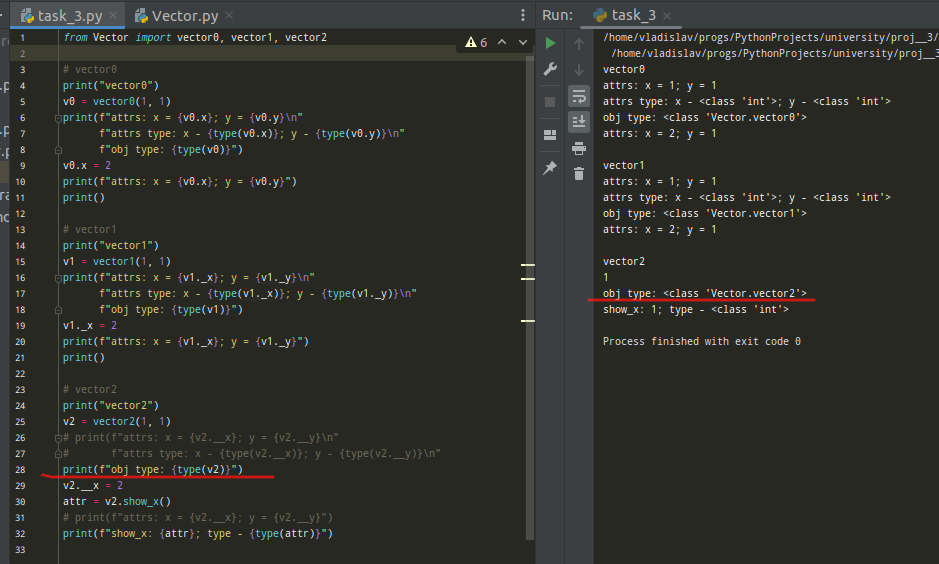
Код из примера (Рис 2.2.1) выполнится с 95 по 129 строчку, а далее выйдет ошибка, так как мы не можем выводит значения приватных переменных обращаясь к ним через класс. Для вывода в строке 127 используется метод класса, так мы можем увидеть значение.

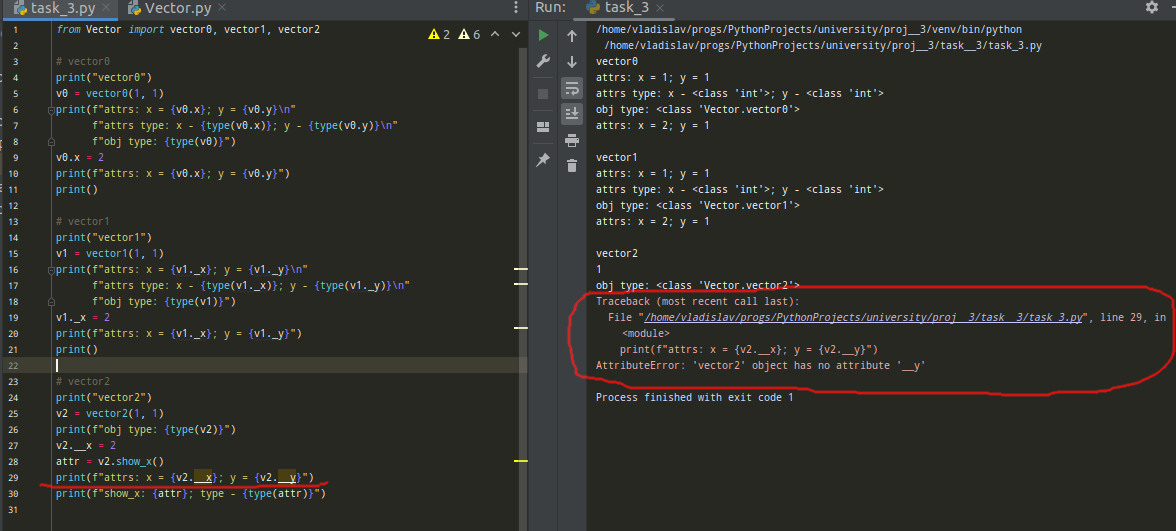
Также мы не можем менять значения приватных переменных, поэтому строчка 129 не сработает, но и не выдаст ошибку.

Если раскомментировать 126 строчку, то код сработает также по 129 строчку.

Наш код имеет схожий принцип работы, протестируем его несколько раз, меняя строчки 26-28 и 31.

Результаты представлены на рисунках 2.3.1-2.3.2.

Рис 2.3.1 — тип приватной переменной

Рис 2.3.2 — ошибка обращения к приватной переменной через класс

### Итог

Мы разобрались, как работает код, представленный в задании.

Предсказали результат выполнения, в том числе для варианта снятия значка комментария в строке 126.

Реализовали свой код и проверить правильность предсказания на практике.

**Примечание:**

В ходе работы изменены имена переменных, методов, выводимые сообщения в консоль. Для удобства классы были реализованы в отдельном файле.

# Задание 4

**Текст задания:**

1.Практически реализовать скрипты, представленные на рисунках 3.14 и 3.15 скрипты.

2.Что в этих скриптах можно отнести к методам- сеттерам?

3.Проанализировать их работу и вывести правило использования (технологию – то есть очерёдность действий) установления и использования свойства (связки декораторов свойства, геттера и сеттера).

4. Написать программу, запрашивающую имя, фамилию и должность сотрудника, и в ответ на это выдающую результат вида «должность – имя, фамилия». Например, «Секретарь – Лена Иванова».

## Ход работы

Реализация скриптов из задания (Рис 3.1 — 3.2).

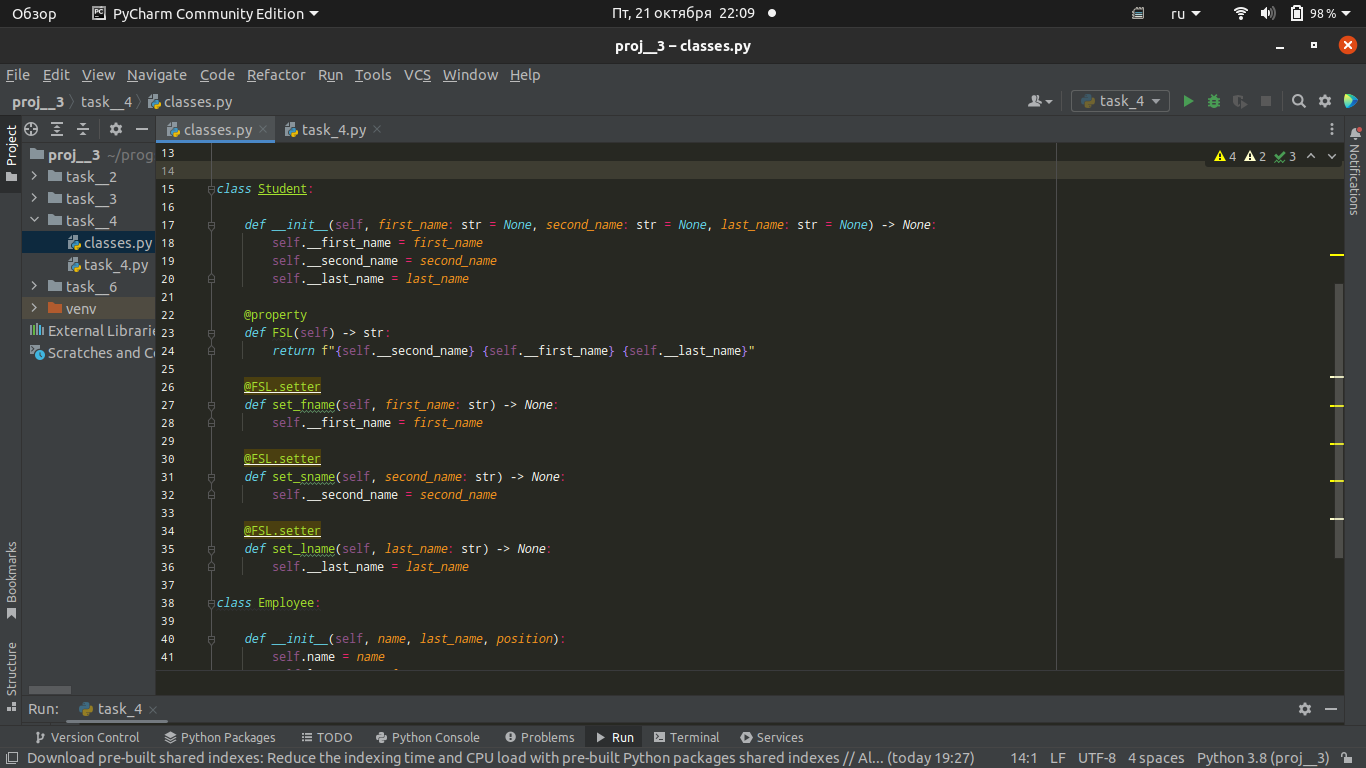
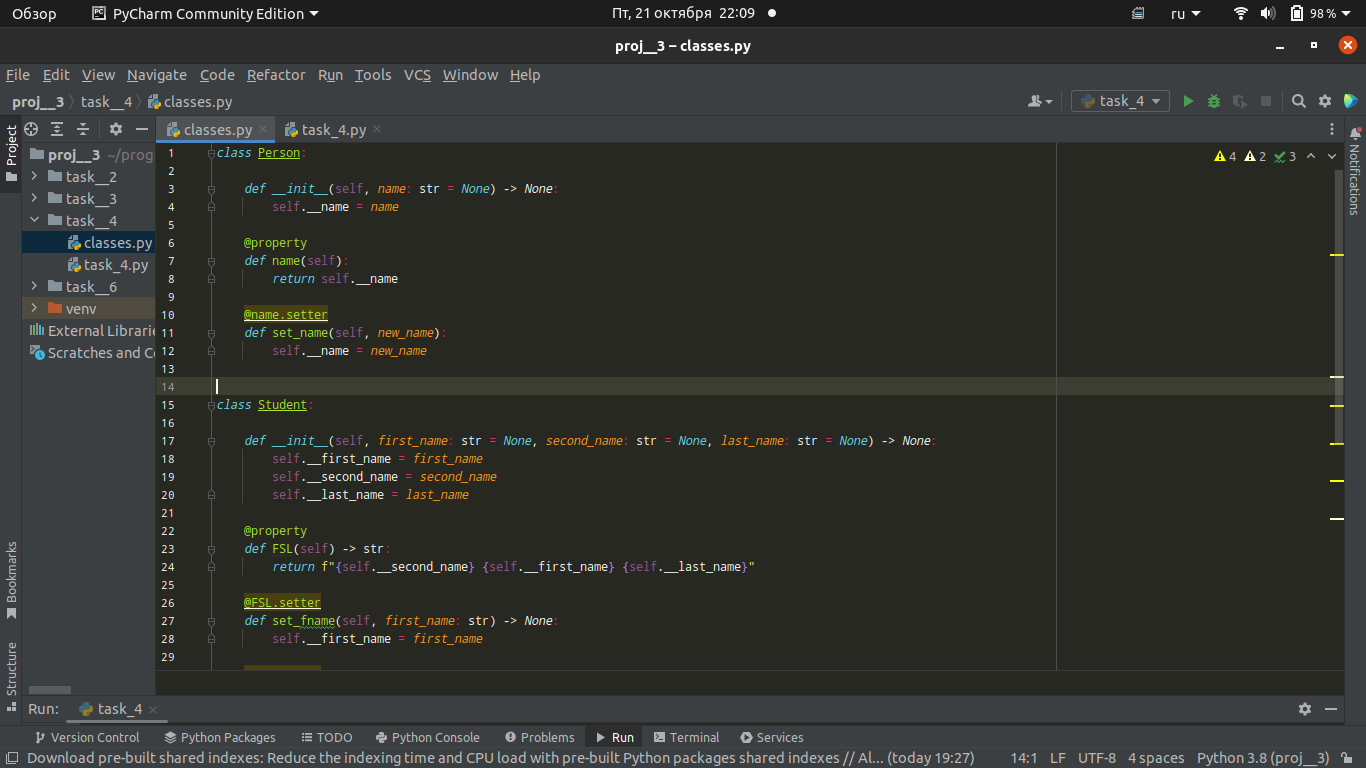


Рис 3.1 — классы персона и ученик по примеру задания

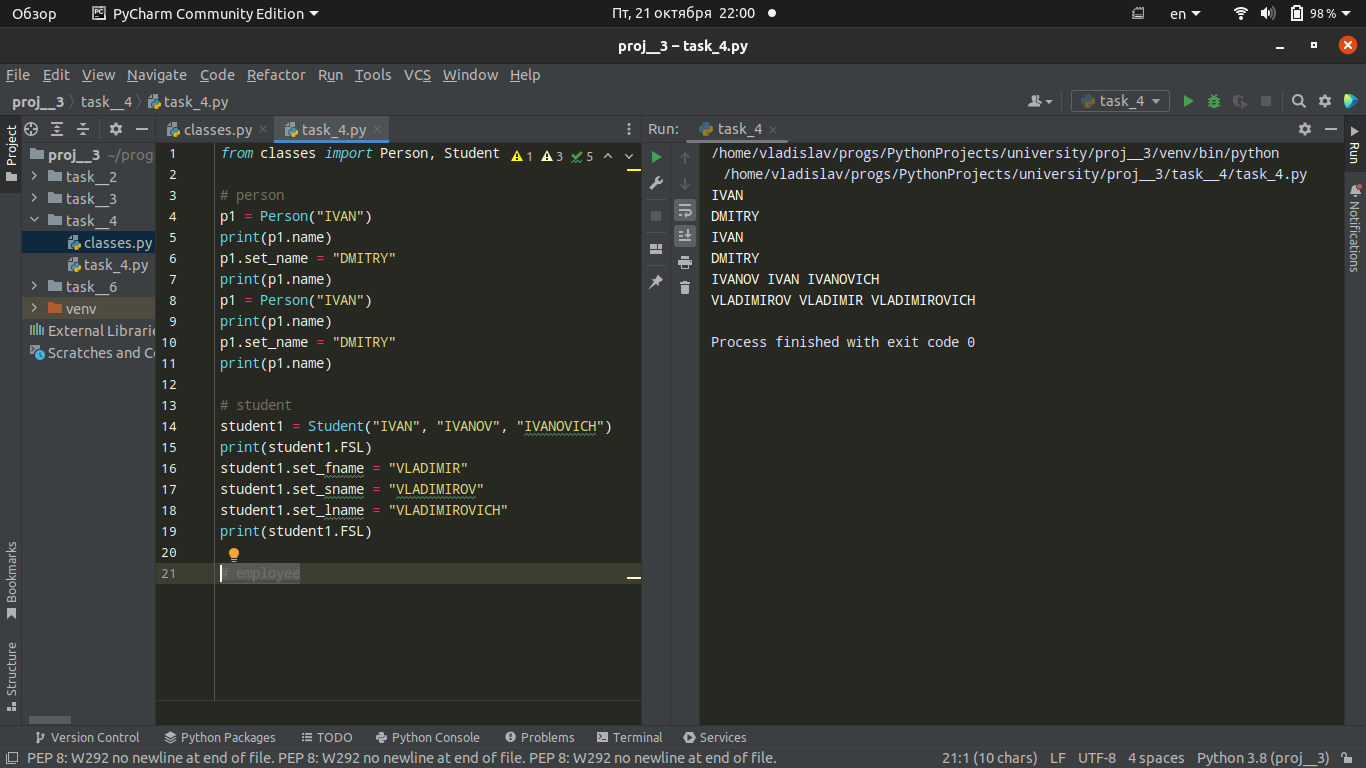


Рис 3.2 — результат работы скрипта по примеру задания

К сеттерам можно отнести все методы имеющие декоратор setter.

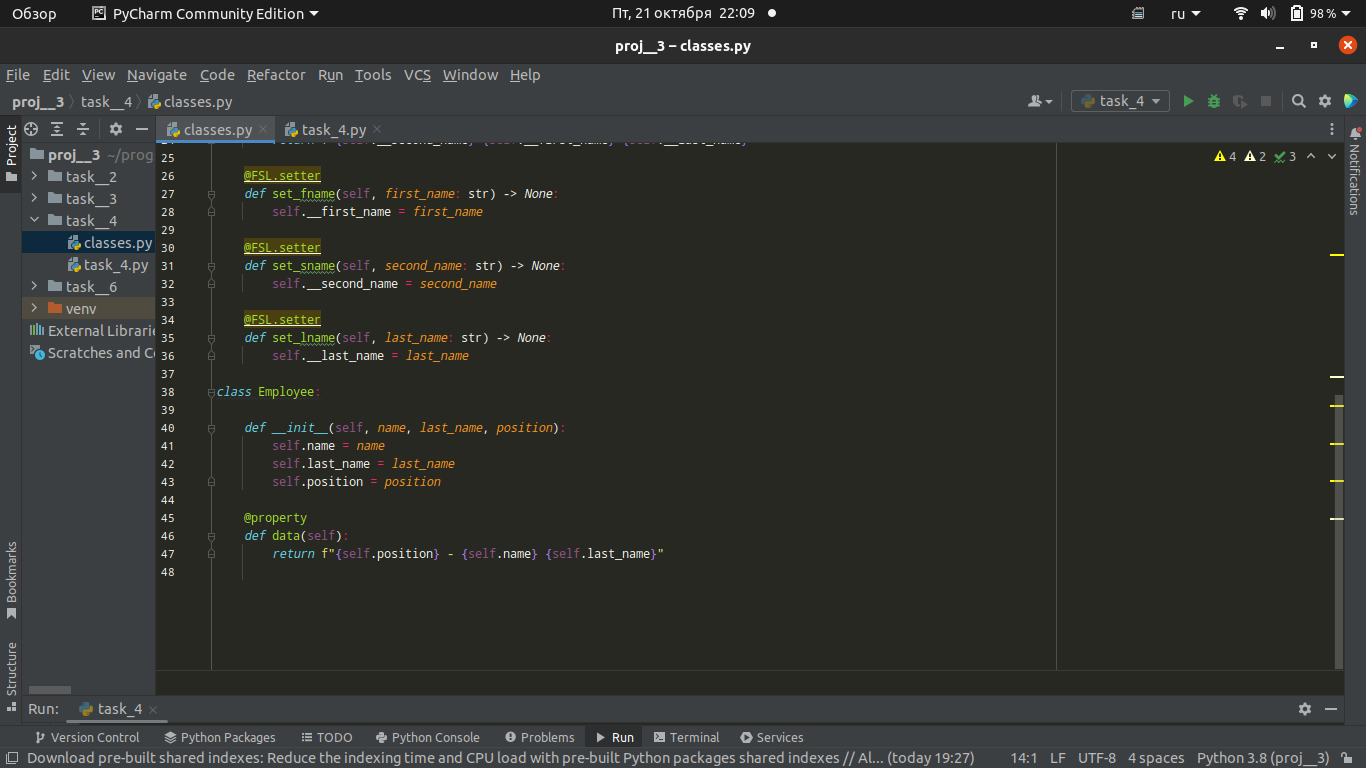
Для того чтобы объявить **геттер** или **сеттер** мы используем такой декторатор как **property**.

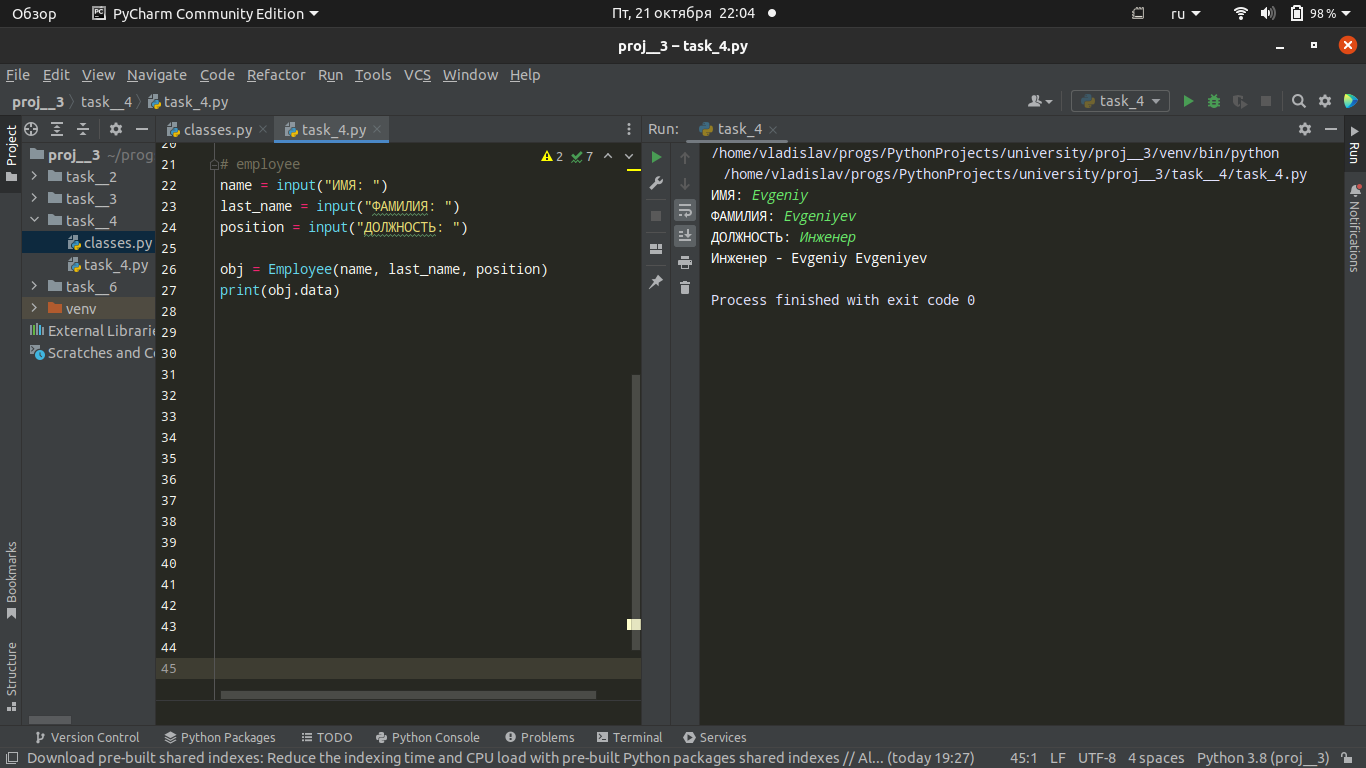
Пишем под **property** функцию возвращающую какие-то атрибуты — это уже будет **getter**.

Но мы можем создать и отдельный **геттер** — создаем декторатор с таким же именем как у функции с декоратором property, затем ставим точку и пишем **getter**, далее объявляем функцию которая будет принимать атрибут, который в свою очередь мы будем передавать в переменную класса.

Если нам нужен **сеттер** мы действуем также, только ключевым словом будет **setter**.

Реализация программы по заданию 4, пункт 4 на рисунках 3.3 — 3.4.

Рис 3.3 — класс сотркудник к заданию

Рис 3.4 — резульат работы программы по заданию

### Итог

Мы на практике проверили работу скриптов данных к заданию.

Проанализировали их работу и вывели правило использования установления и использования геттера и сеттера.

Написали программу по заданию.

**Примечание:**

В ходе работы изменены имена переменных, методов, выводимые сообщения в консоль. Для удобства классы были реализованы в отдельном файле.

# Задание 6

**Текст задания:**

Написать скрипт, решающий такую задачу:

• Разработать класс, представляющий сотовый телефон.

• Атрибутами сотового телефона должны быть:

◦ Имя производителя телефона будет присвоено атрибуту \_ manufact;

◦ Модели (точнее номеру модели) телефона будет присвоен атрибуту \_model

◦ Розничная цена телефона будет присвоена атрибуту \_retail\_price

Кроме того, этот класс будет иметь приведенные ниже методы:

1. Конструктор, который принимает аргументы для производителя, номера модели и розничной цены;

2. Метод, определяющий принимает аргумент для производителя и в случае необходимости позволит изменять значение атрибута\_ manufact после создания объекта;

3. метод set \_ model () принимает аргумент для модели и в случае необходимости позволит изменять значение атрибута \_ model после создания объекта;

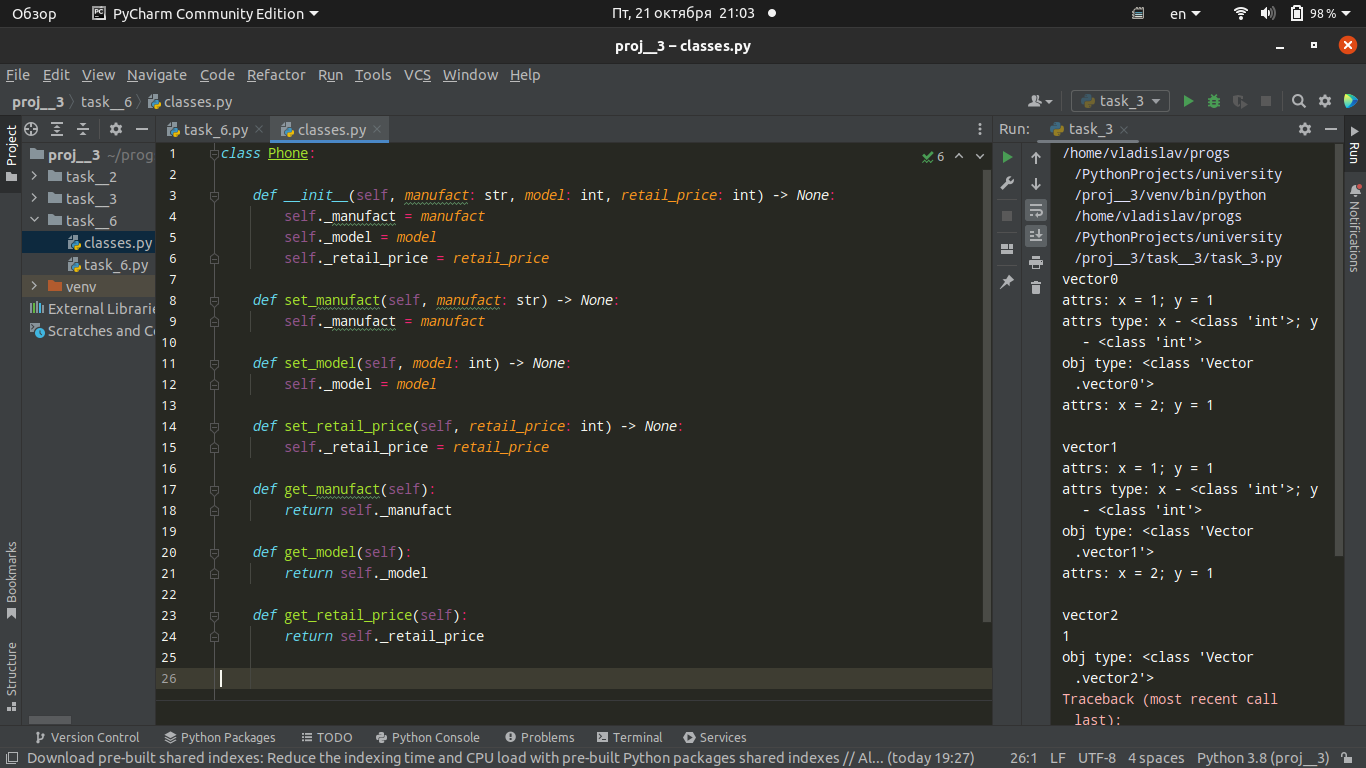
4. метод set \_ retail \_price () принимает аргумент для розничной цены и в случае необходимости позволит изменять значение атрибута retail\_price после создания объекта;

5. метод get manufact () возвращает название производителя телефона;

6. метод get \_ model () возвращает номер модели телефона;

метод get \_ retai l \_price () возвращает розничную цену телефона

### Класс



### Пример работы класса

