**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 №1**

Выполнил: студент группы Иб-321

Комышков Владислав Дмитриевич

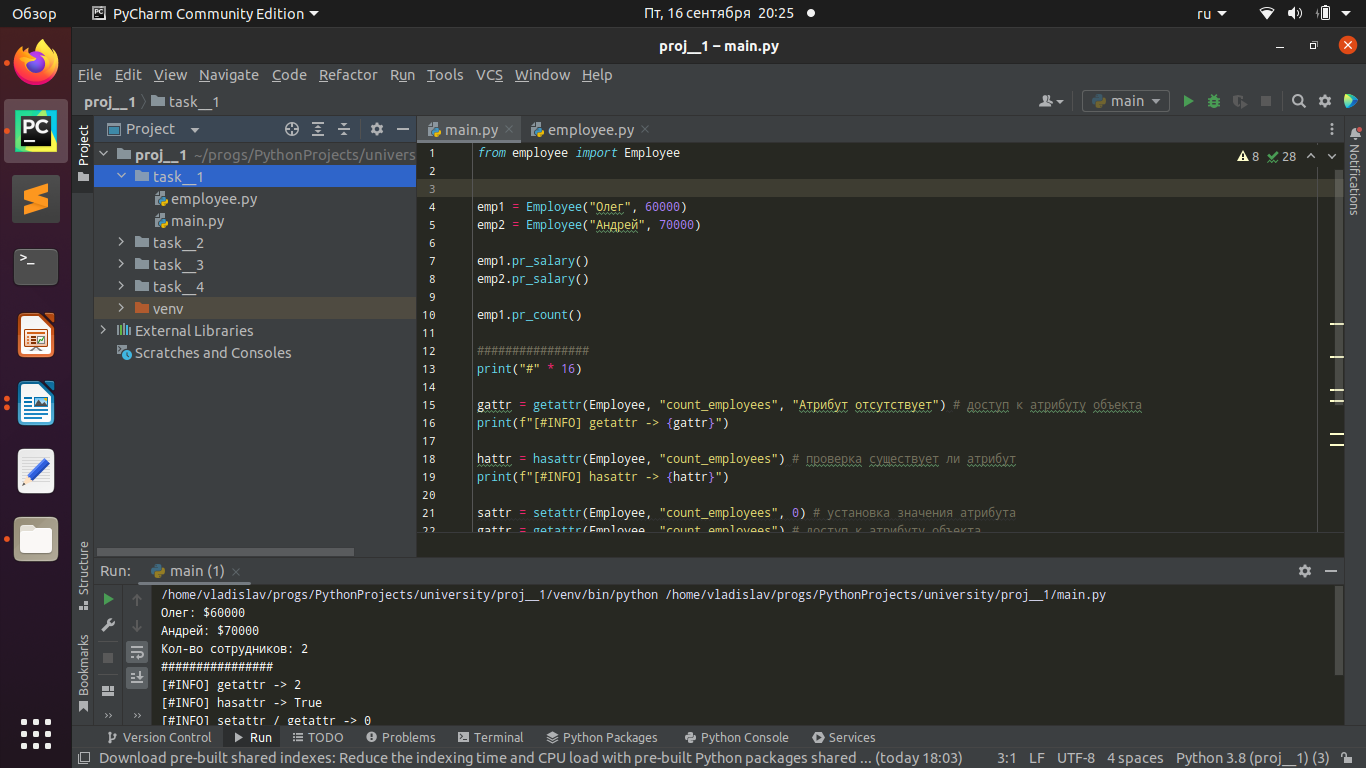
**Задание 1**

**Текст задания**

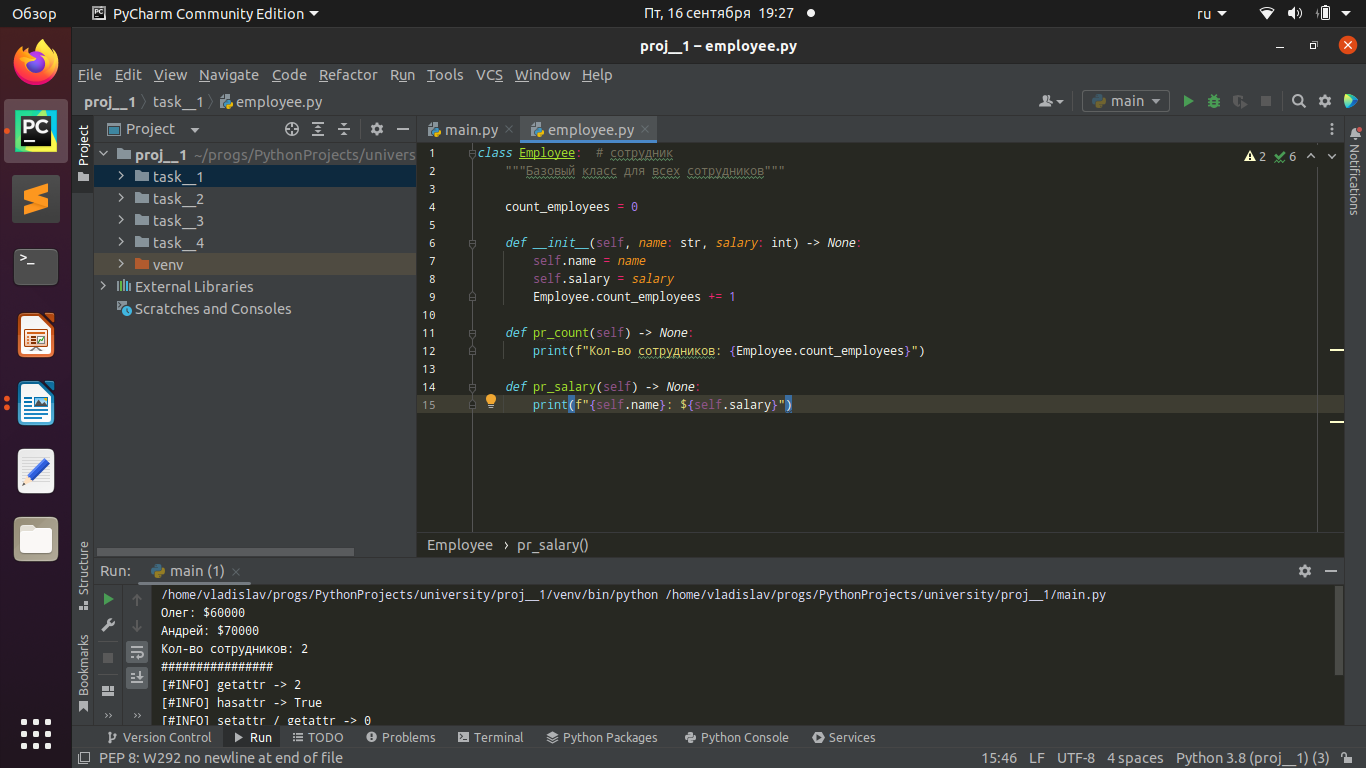
Проверить на практике работу указанных функций по работе с атрибутами на примере класса «Сотрудник»

**Ход работы**

Разделим проект на **2 файла** (Рис 1.1). В первом напишем класс, во втором будем работать с функциями для работы с атрибутами.

*Рис 1.1 — Файлы проекта к заданию 1*

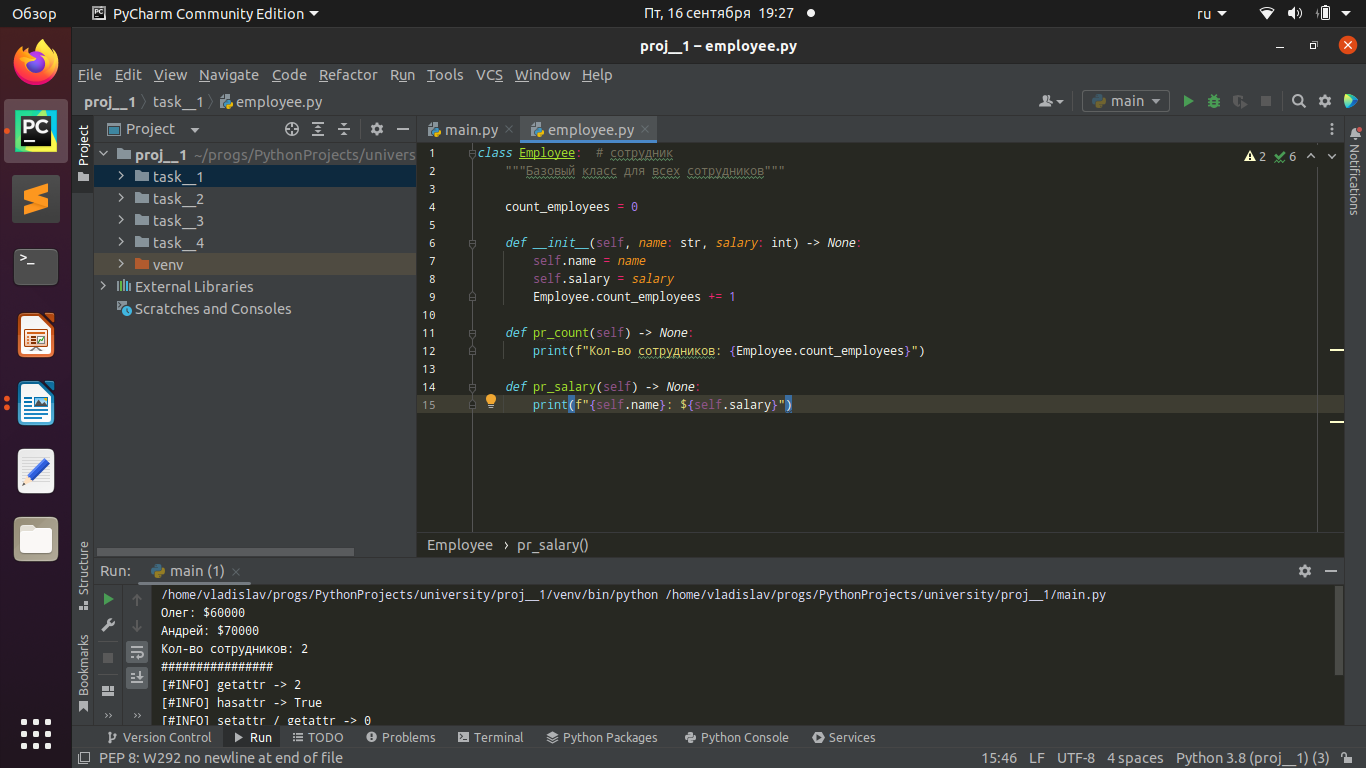
Создаем **класс Employee** (*с англ. «сотрудник»*). Все экземпляры класса будут иметь **атрибут count\_employees** - счетчик сотрудников (Рис. 1.2).

*Рис. 1.2 — Создание класса «Сотрудник»*

Конструктор класса будет принимать 2 аргумента:

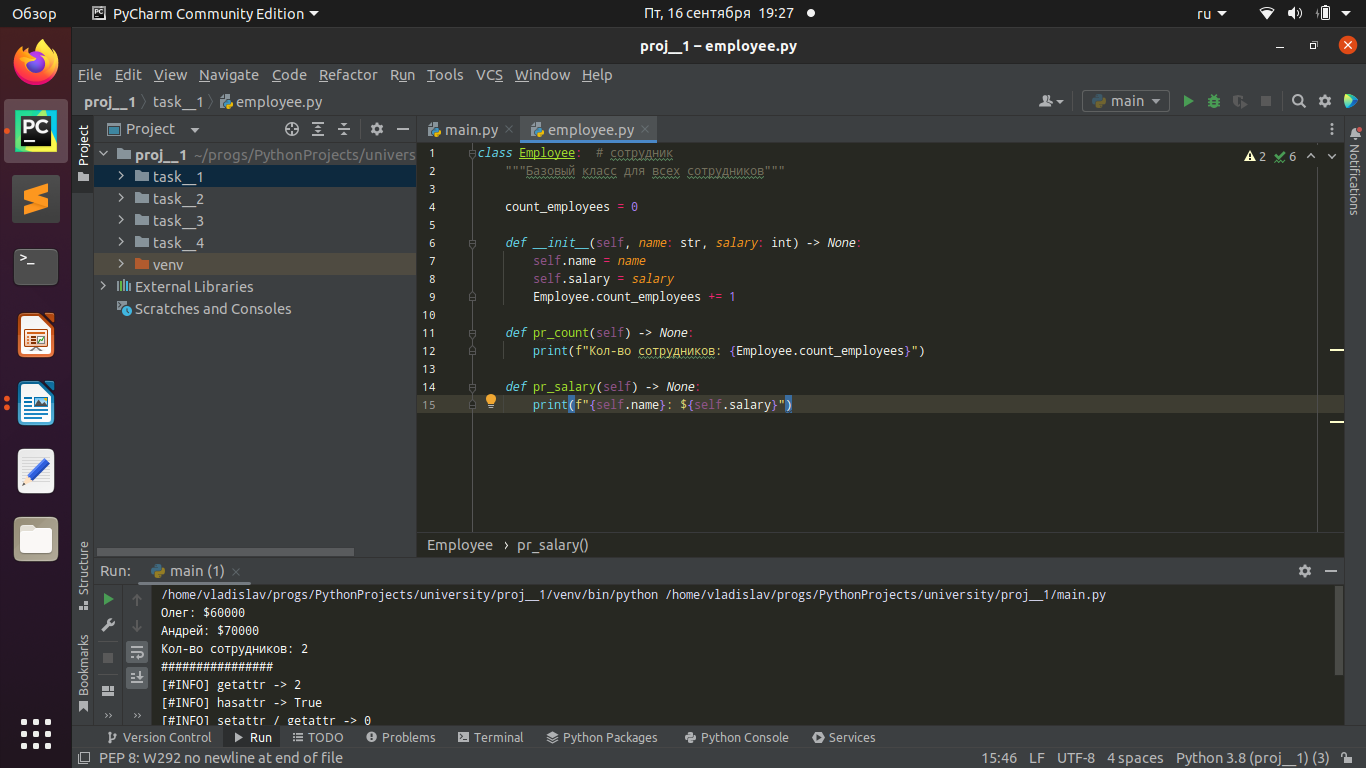
* **name** (*с англ. «имя»*) - тип данных **str**,
* **salary** (*с англ. «зарплата»*) - тип данных **int**.

Он будет передавать значения этих аргументов свойствам объекта, а также увеличивать значение счетчика сотрудников на 1 (Рис. 1.3).

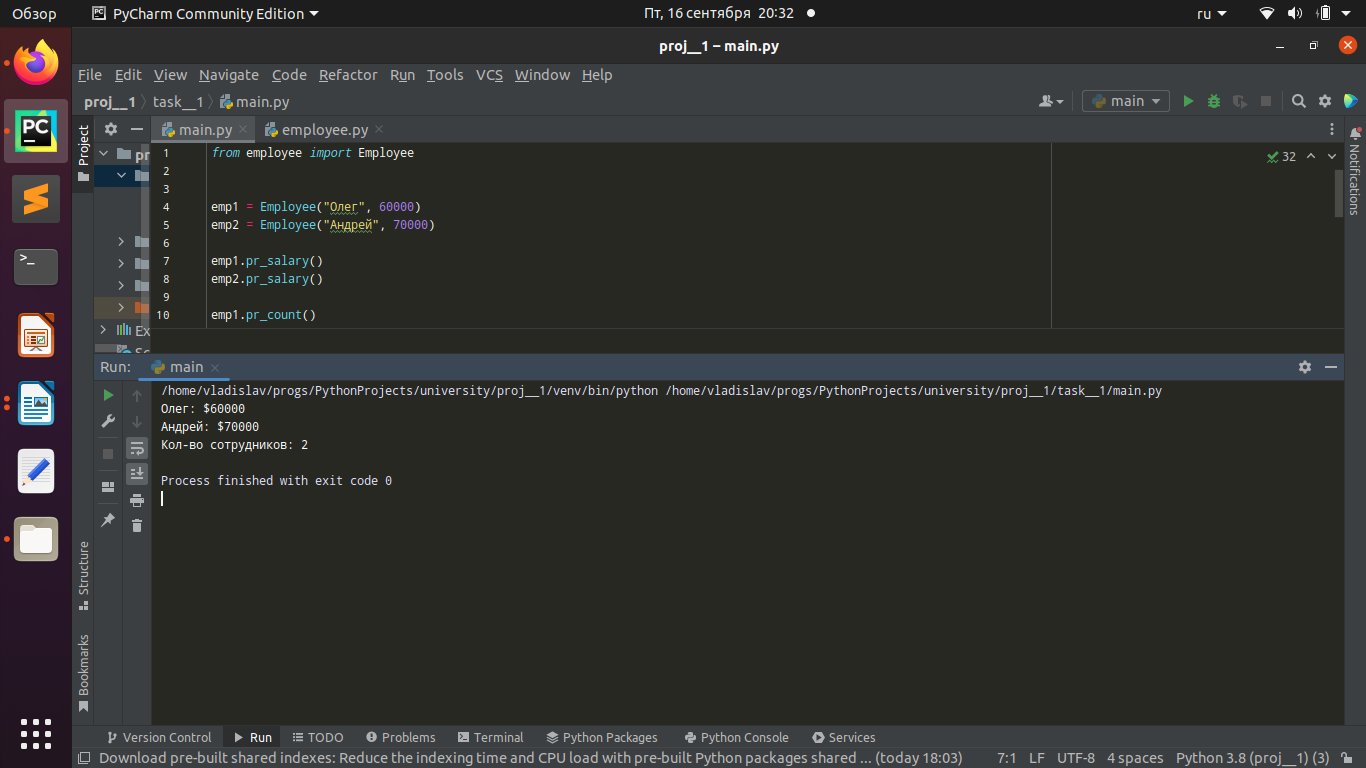
*Рис. 1.3 — Конструктор класса «Сотрудник»*

Класс будет иметь 2 метода (Рис 1.4):

* **pr\_count** — выводит в консоль информацию о кол-ве сотрудников,
* **pr\_salary** — выводит в консоль имя и зарплату сотрудника.

*Рис 1.4 — Методы класса «Сотрудник»*

Импортируем наш класс в основной файл проекта **main.py** и создаем два объекта для **проверки правильности работы класса** и его методов (Рис 1.5).

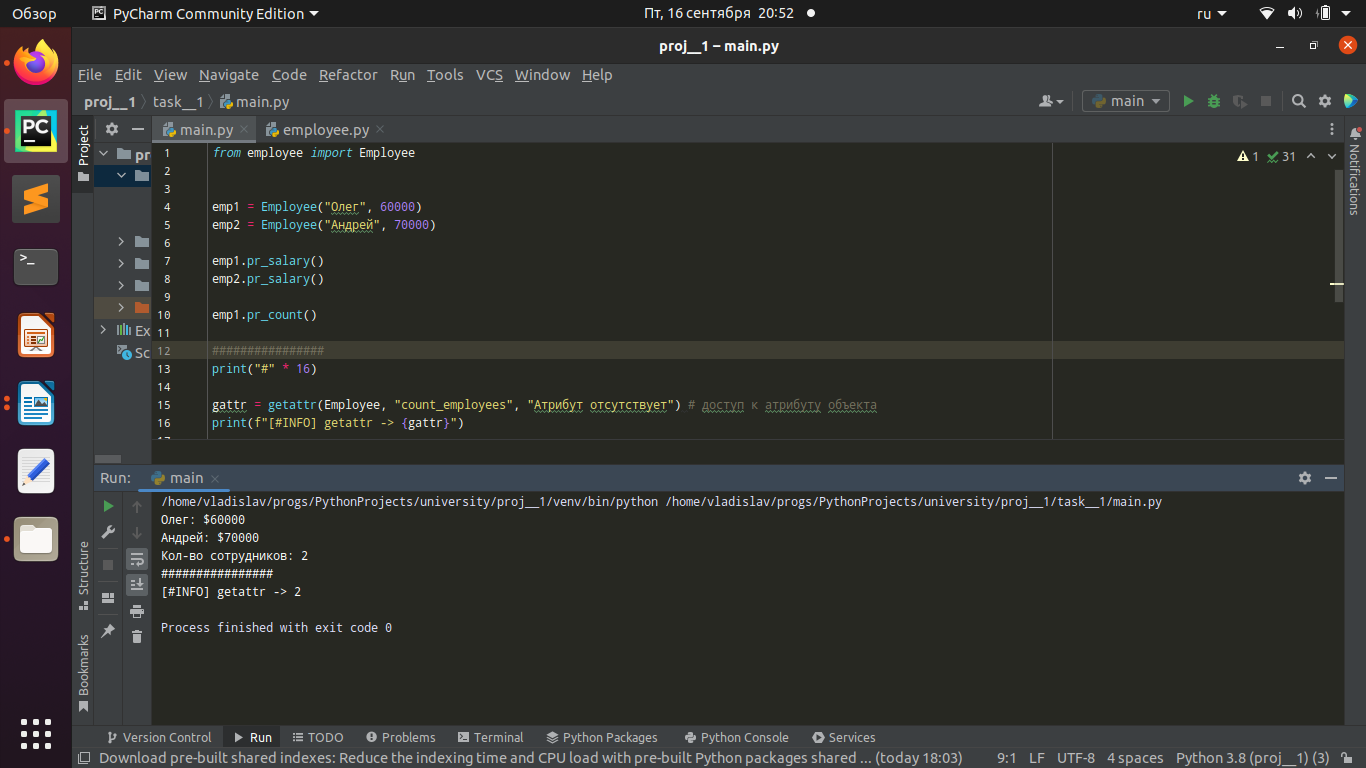
*Рис 1.5 — Проверка работы класса*

Код отработал без ошибок.

Теперь проверим функции для доступа к атрибутам. Мы можем использовать следующие функции:

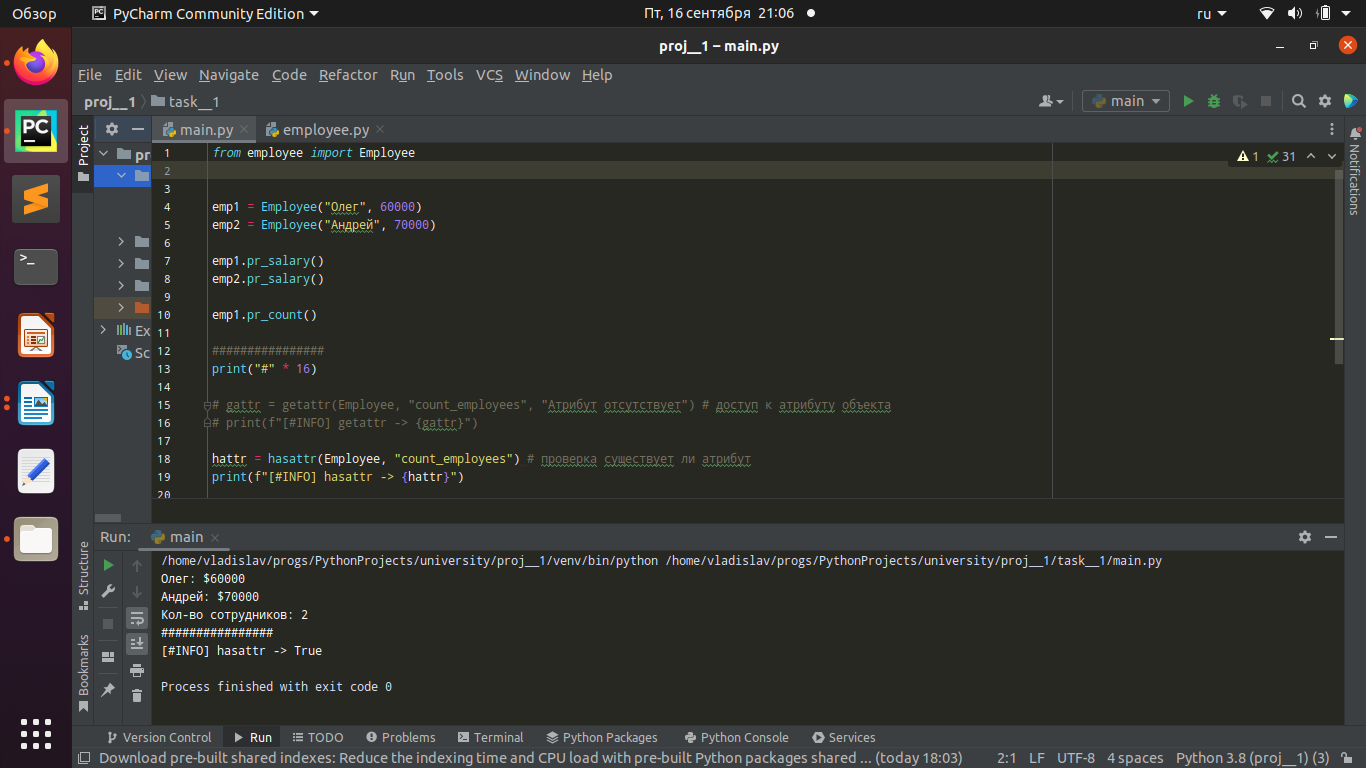
* **getattr(объект, имя [, по умолчанию])** — для доступа к атрибуту объекта.
* **hasattr(объект, имя)** — проверить, если атрибут существует или нет.
* **setattr(объект, имя, значение)** — установить атрибут. Если атрибут не существует, он будет создан.
* **delattr(объект, имя)** — для удаления атрибута.

Начнем с **getattr**, получим значение счетчика сотрудников и выведем в консоль (Рис 1.6).

*Рис 1.6 — Проверка работы функции getattr*

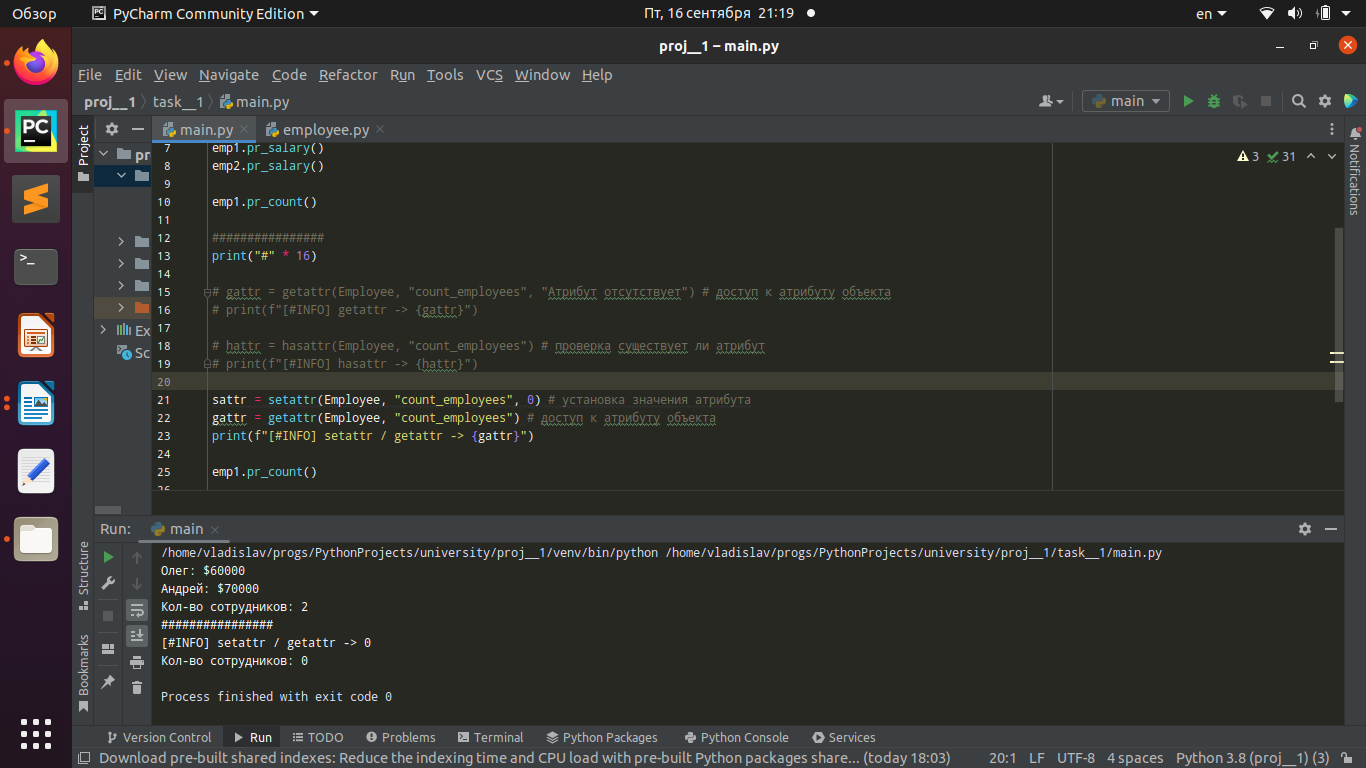
Скрипт отработал корректно, ответ верный — 2.

Следующая функция — **hasattr** (Рис 1.7). Проверим существует ли атрибут **count\_employees**.

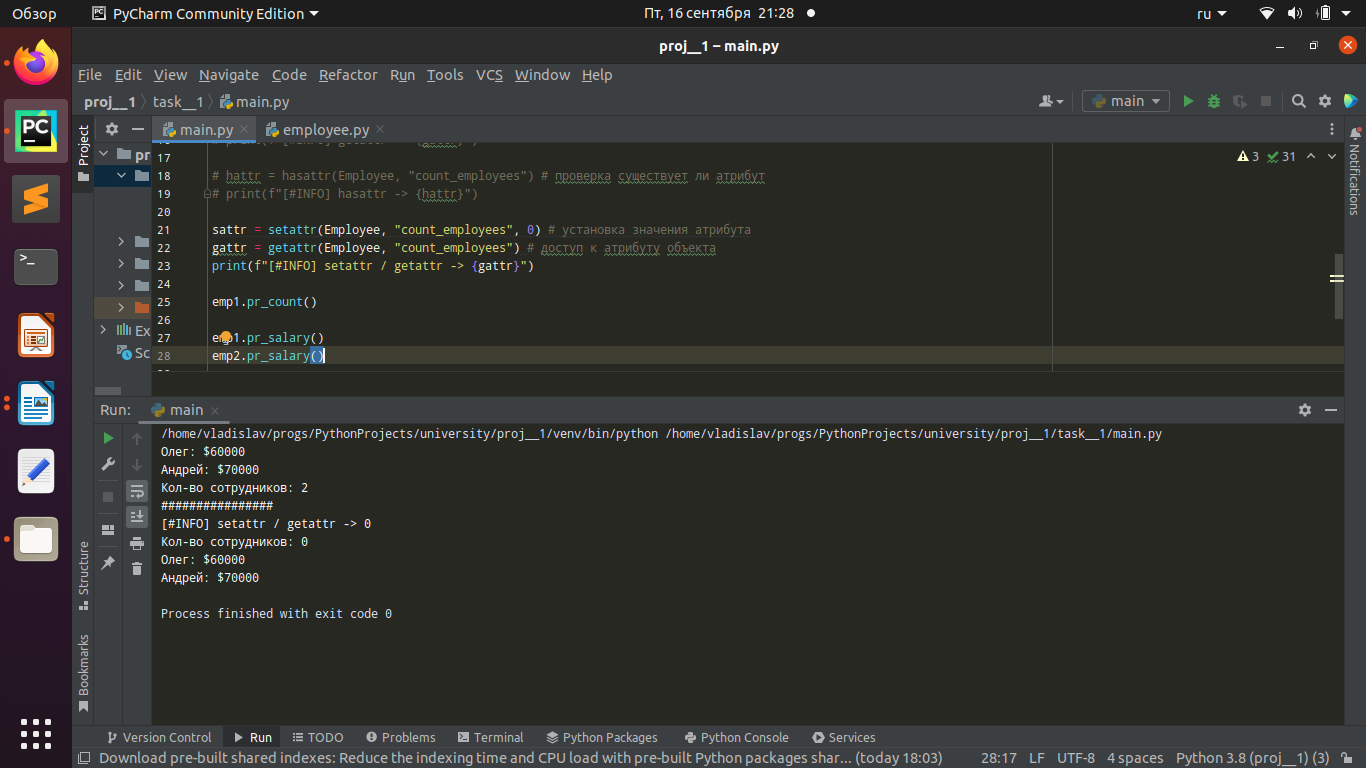
*Рис 1.7 — Проверка работы функции hasattr*

Ответ — True, то есть атрибут существует.

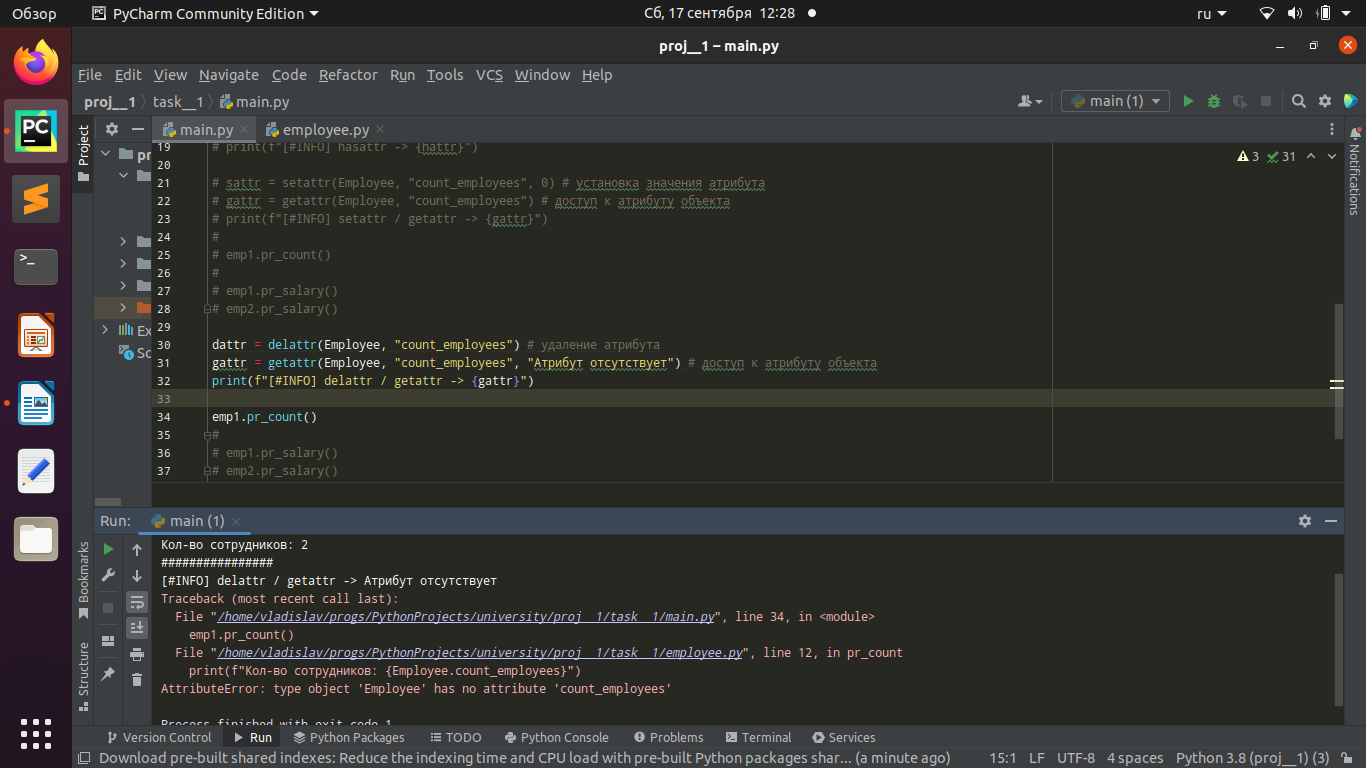
Для проверки функции **setattr** воспользуемся функцией **getattr** и методом класса **pr\_count**, чтобы увидеть изменения атрибута счетчика (Рис 1.8).

*Рис 1.8 — Проверка работы функции setattr*

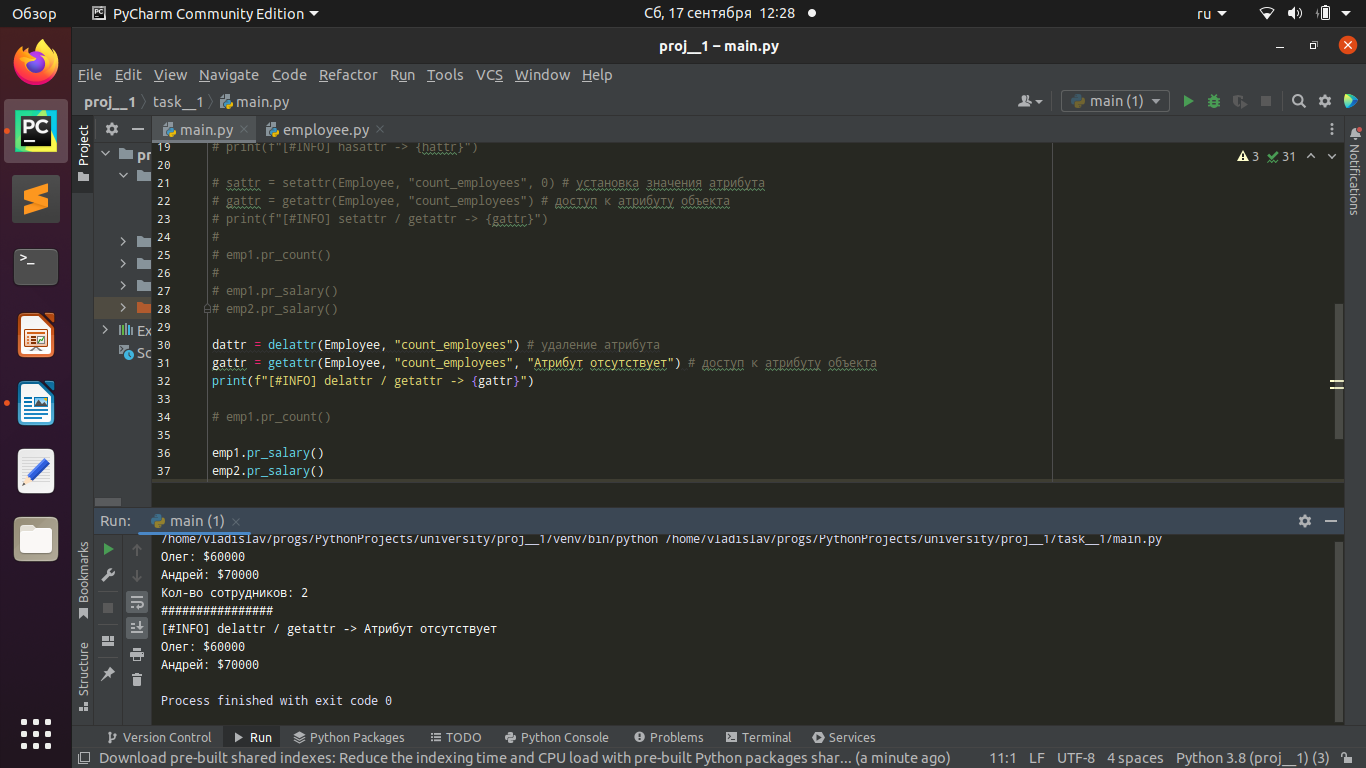
В результате мы обнулили счетчик. На другие **свойства объектов** это никак не повлияло (Рис 1.9).

*Рис 1.9 — Свойства объектов*

Теперь попробуем удалить счетчик с помощью функции **delattr** и обратиться к нему через функцию **getattr** и метод класса **pr\_count** (Рис 1.10).

*Рис 1.10 — Проверка работы функции delattr*

В итоге при обращении через функцию **getatter** мы получаем ответ - «Атрибут отсутствует». Если же мы попытаемся обратиться через метод класса, то получим ошибку, которая говорит о том что такого атрибута нет, но как и в прошлом случае это не влияет на другие **свойства объектов** (Рис 1.11).

*Рис 1.11 — Свойства объектов*

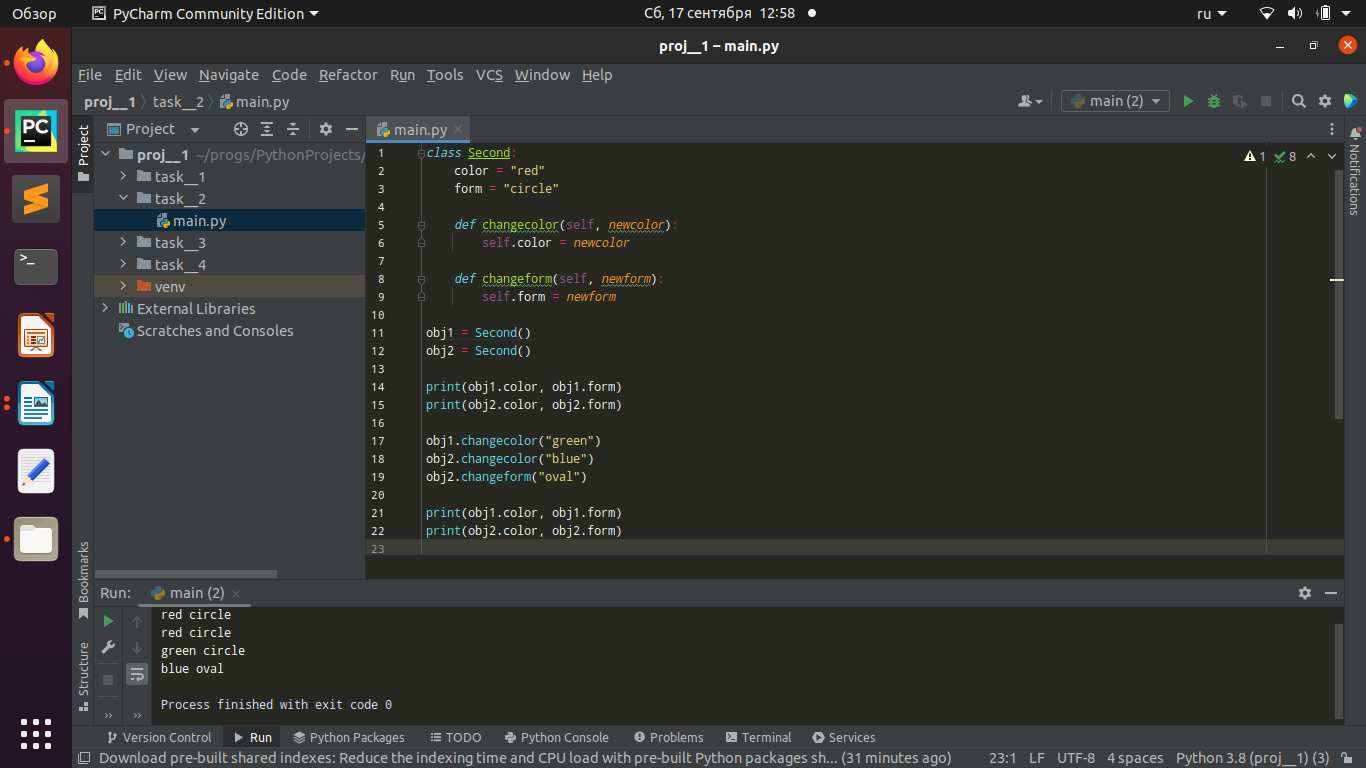
**Итог**

На примере класса «Сотрудник» мы научились пользоваться и проверили на практике функции по работе с атрибутами.

**Задание 2**

**Текст задания**

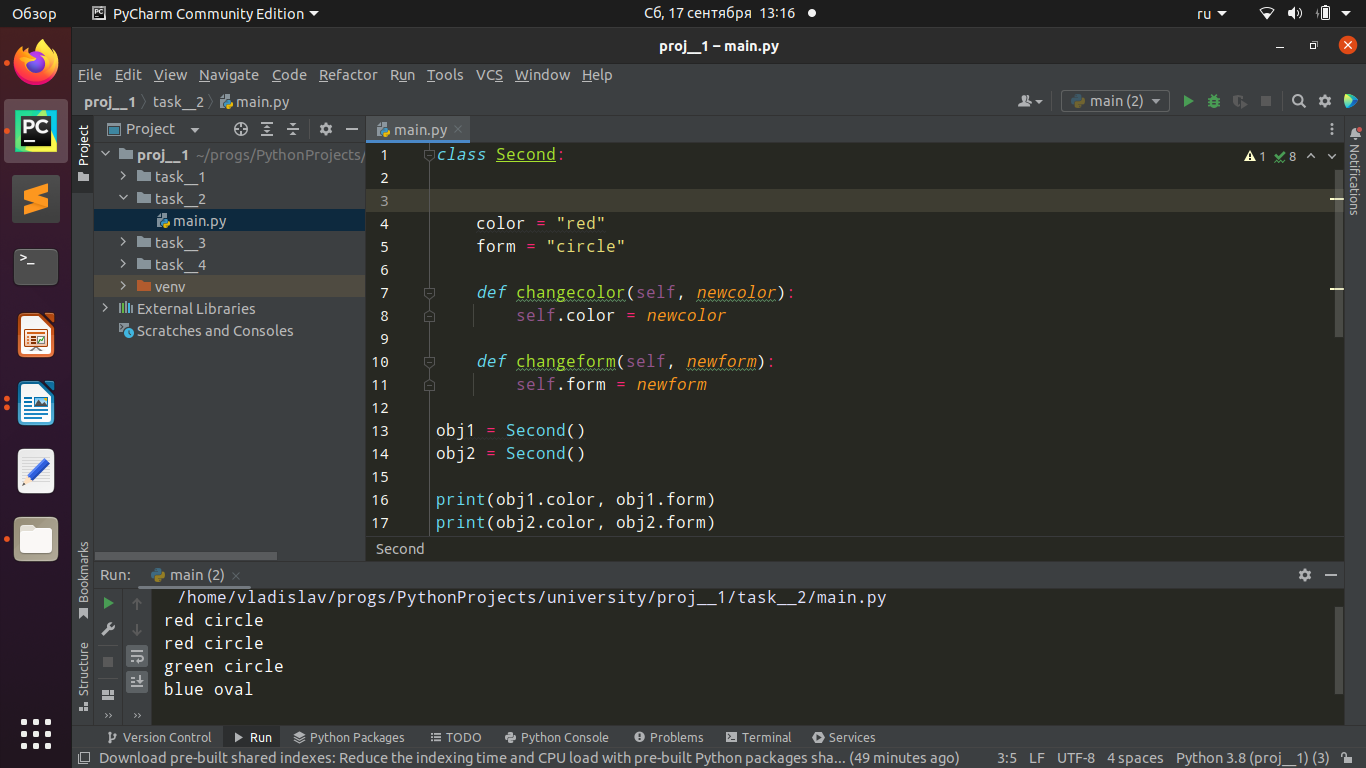
Имеется **текст программы** (Рис 2.1).

*Рис 2.1 — Текст программы к заданию 2*

До реализации в PyCharm повести анализ, выделив класс, объекты и экземпляры класса. После чего предсказать результаты программы. После чего реализовать программу PyCharm и сравнить результаты с ранее предсказанными.

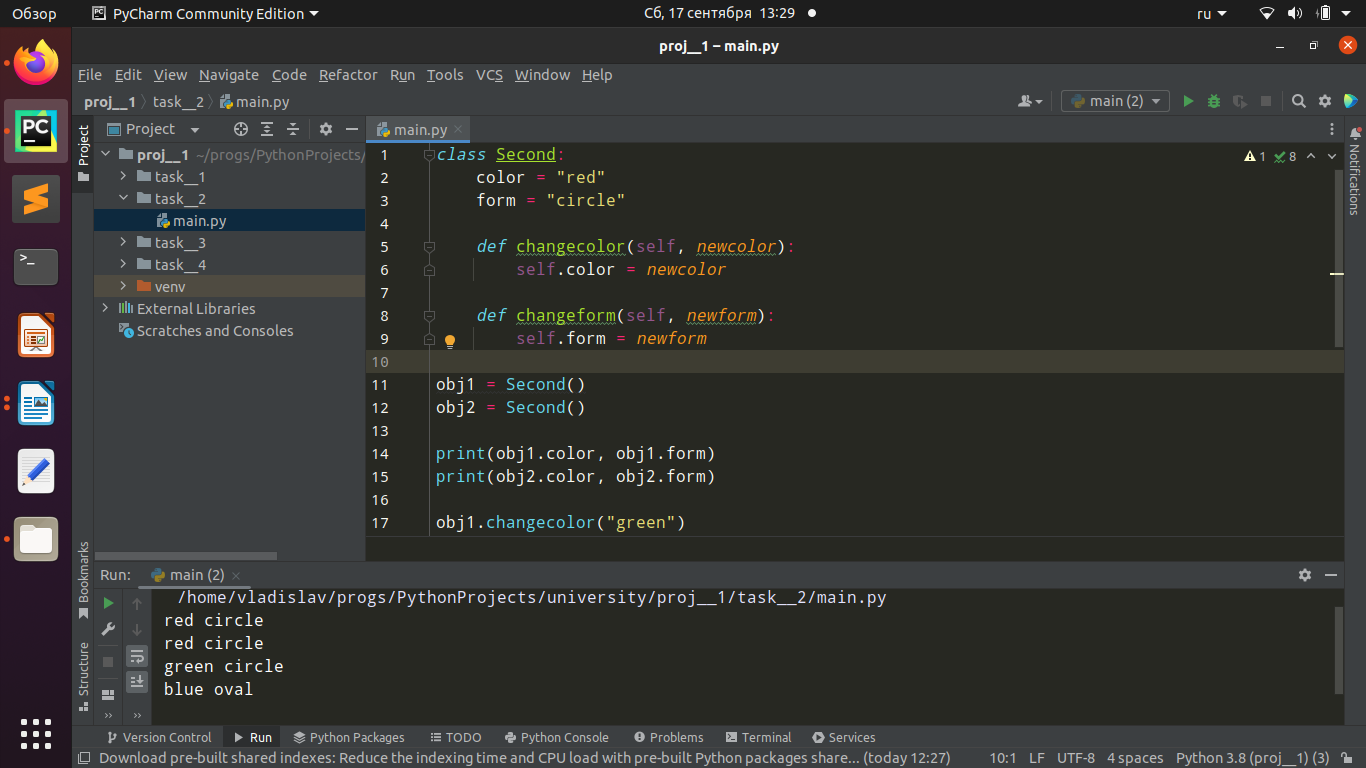
**Ход работы**

Сначала определим класс, для этого в python используется оператор **class**, а затем пишется имя класса, в нашем случае - **Second** (Рис 2.2).

*Рис 2.2 — Класс Second*

Затем рассмотрим атрибуты класса, их 2 (Рис 2.3):

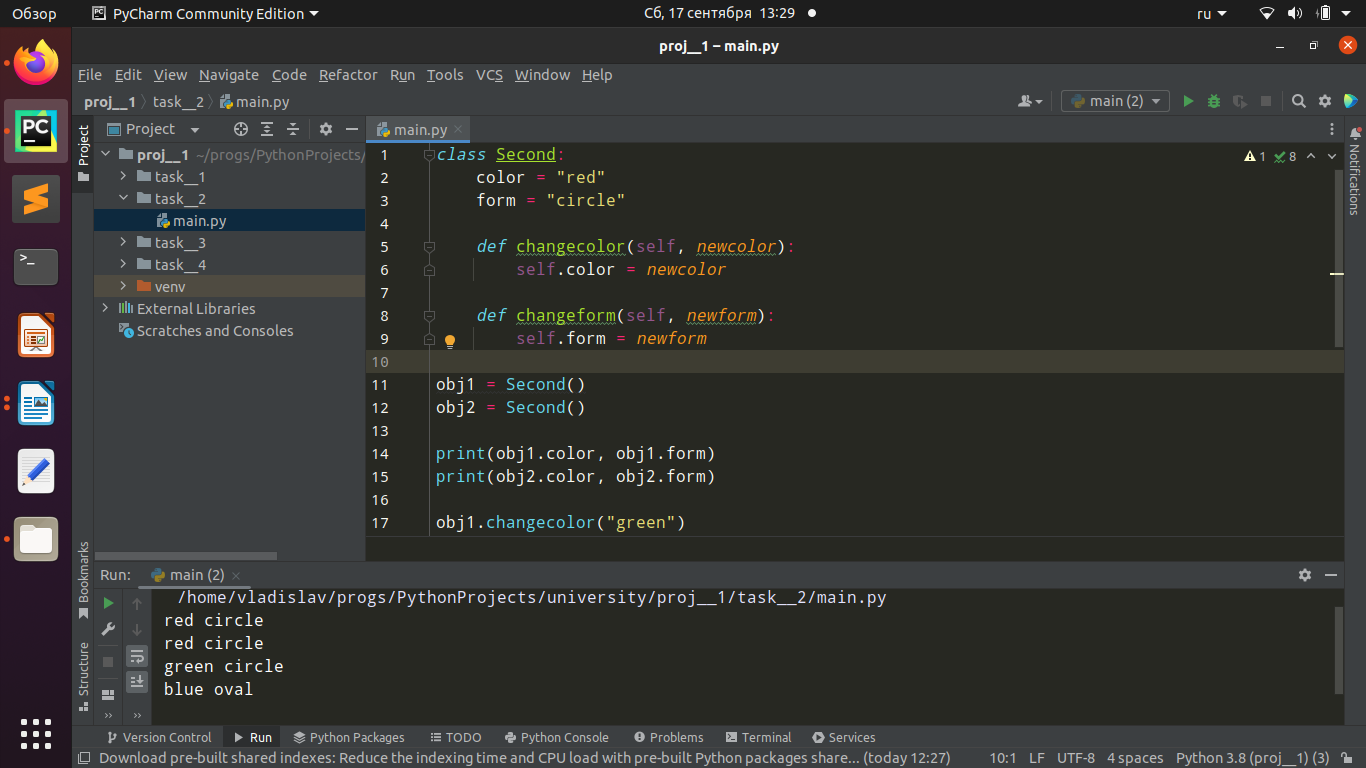
* **color**, то есть цвет, с начальным значением - «red»;
* **form** — форма, с начальным значением - «circle».

*Рис 2.3 — Атрибуты класса Second*

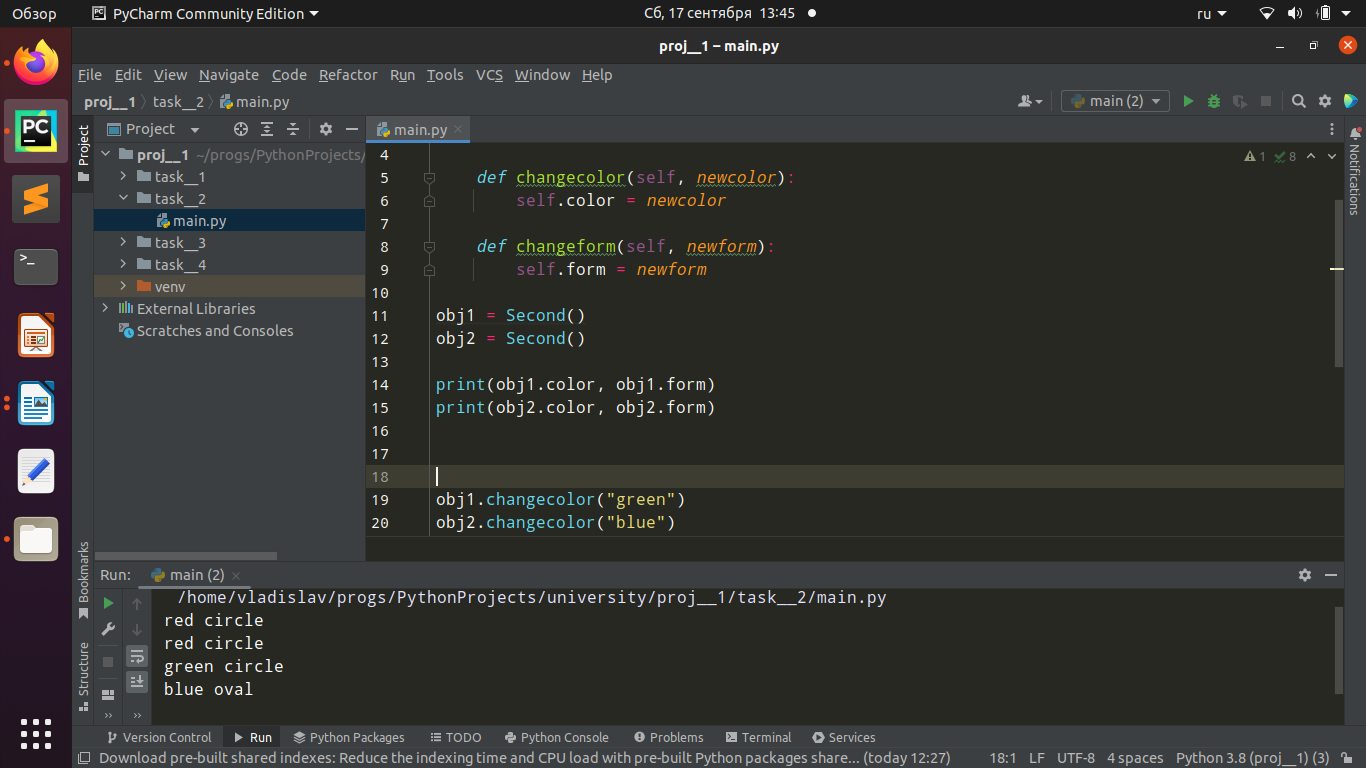
Также этот класс имеет 2 метода (Рис 2.4):

* **changecolor** — для изменения атрибута **color**;
* **changeform** — для изменения атрибута **form**.

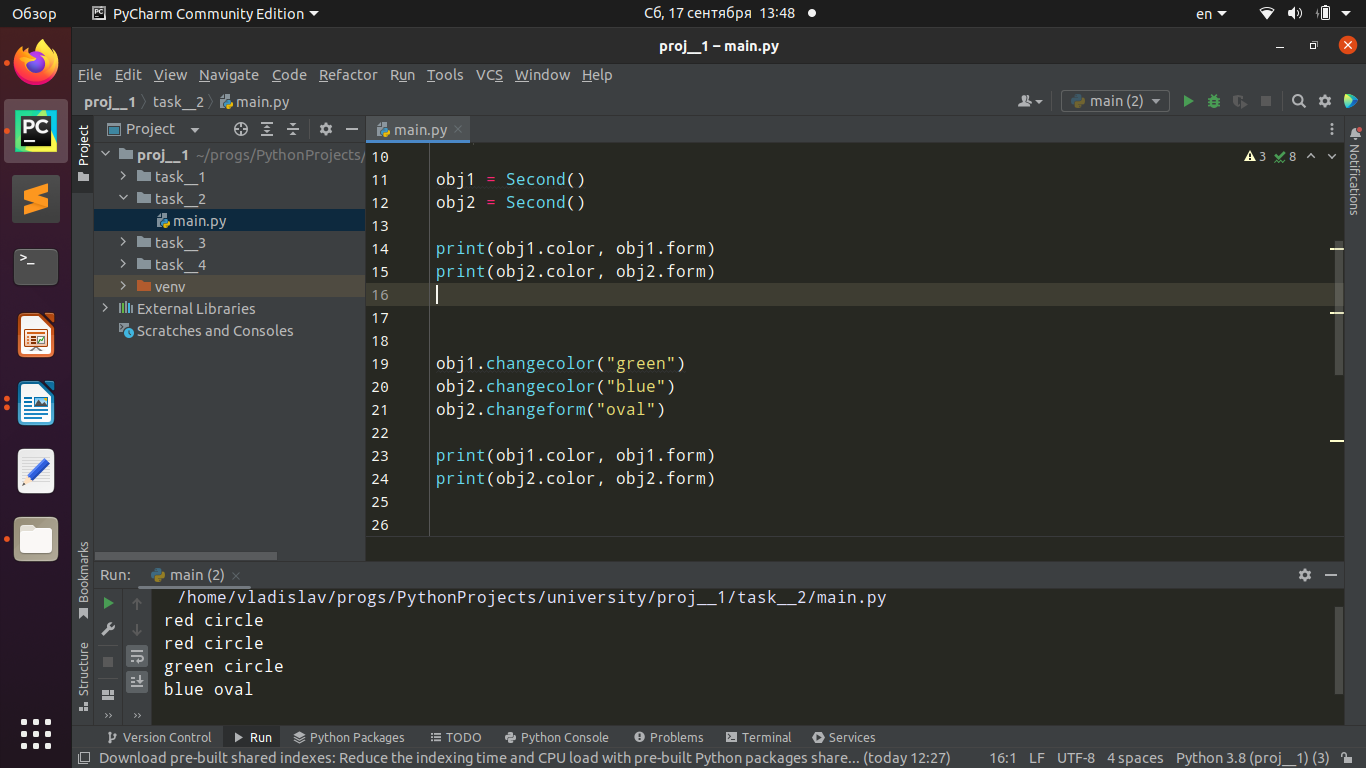
Изменения атрибутов производятся с помощью аргумента **self**. Методы в python объявляются с помощью оператора **def**.

*Рис 2.4 — Методы класса Second*

Далее создаются 2 экземпляра класса: obj1 и obj2. В консоль выводятся значения их атрибутов (Рис 2.5).

*Рис 2.5 — Экземпляры класса Second*

Затем с помощью методов класса меняются значения их атрибутов и снова выводятся в консоль (Рис 2.6).

*Рис 2.6 — Изменение значений атрибутов экземпляров класса Second*

В результате мы должны получить ответ:

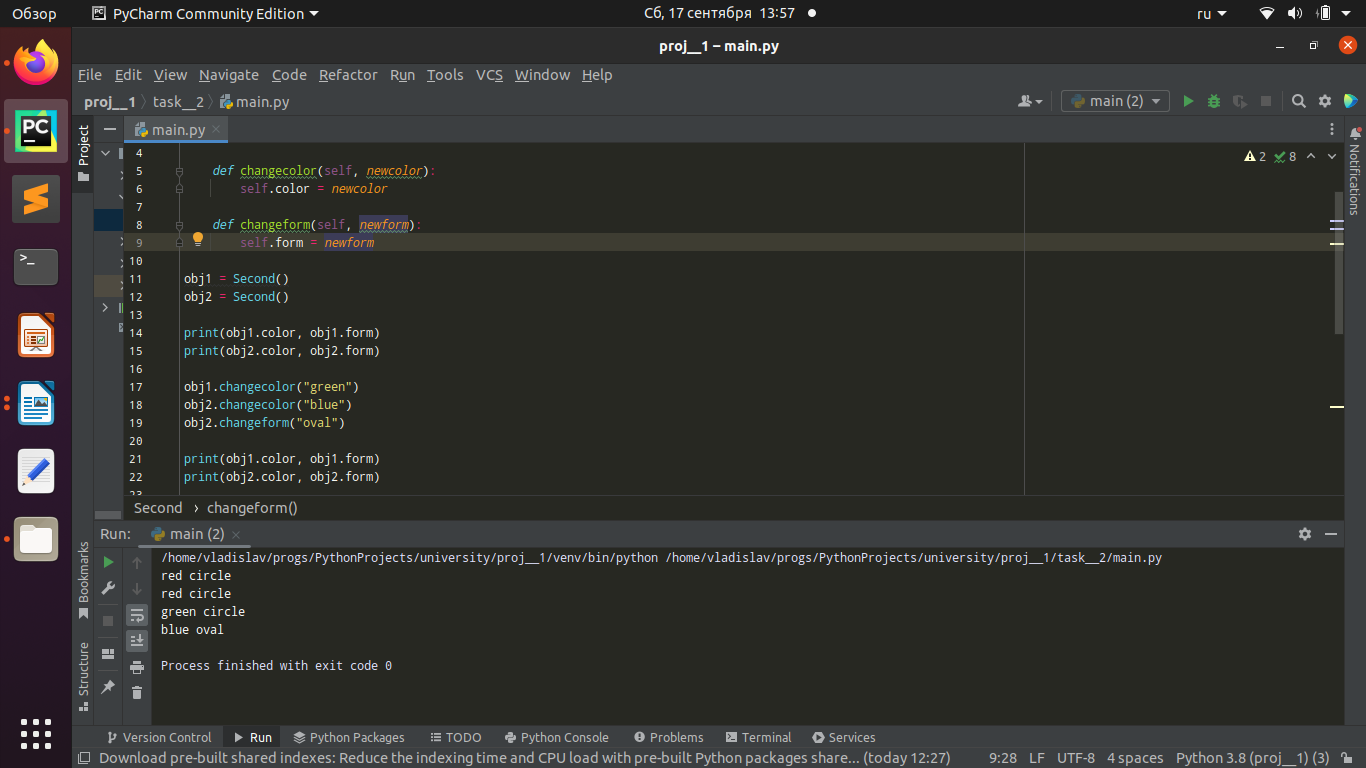
**red circle**

**red circle**

**green circle**

**blue oval**

Запустим программу и сравним (Рис 2.7).

*Рис 2.**7 — Результат работы программы*

Результат оказался верным, как мы и предполагали.

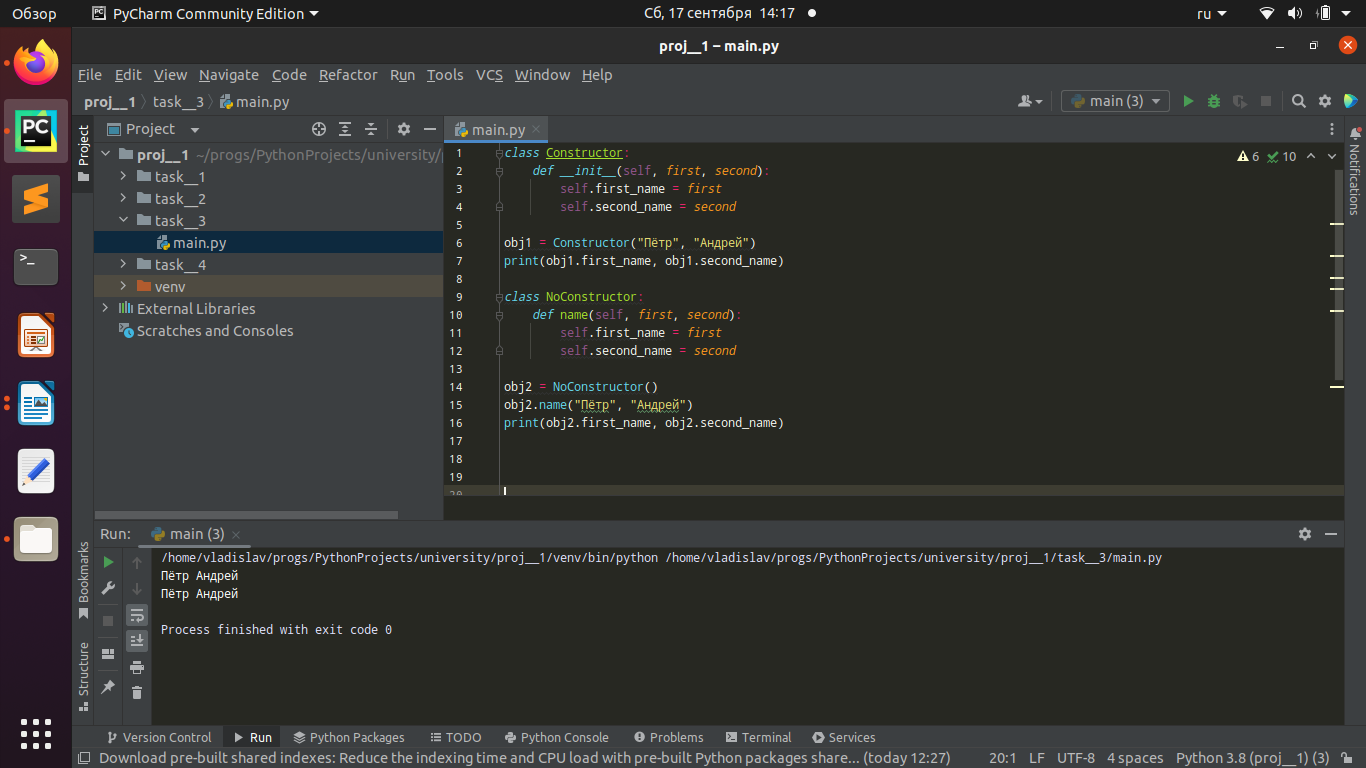
**Итог**

Проанализировав текст программы мы определили класс, объекты и экземпляры класса. Также верно предсказали результат программы, реализовав ее в PyCharm и сравнив результаты с ранее предсказанными.

**Задание 3**

**Текст задания**

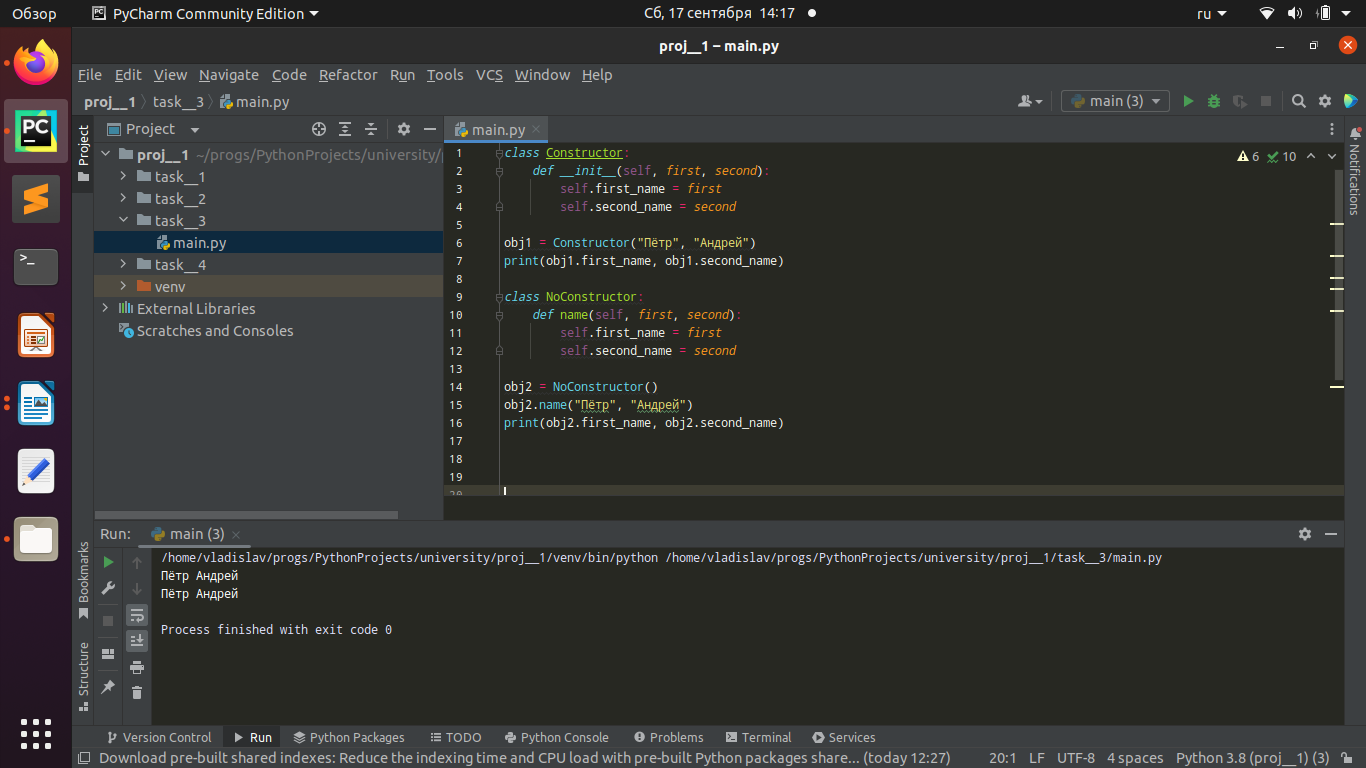
Создать два простых похожих по функционалу класса (Рис 3.1).

*Рис 3.1 — Текст программы к заданию 3*

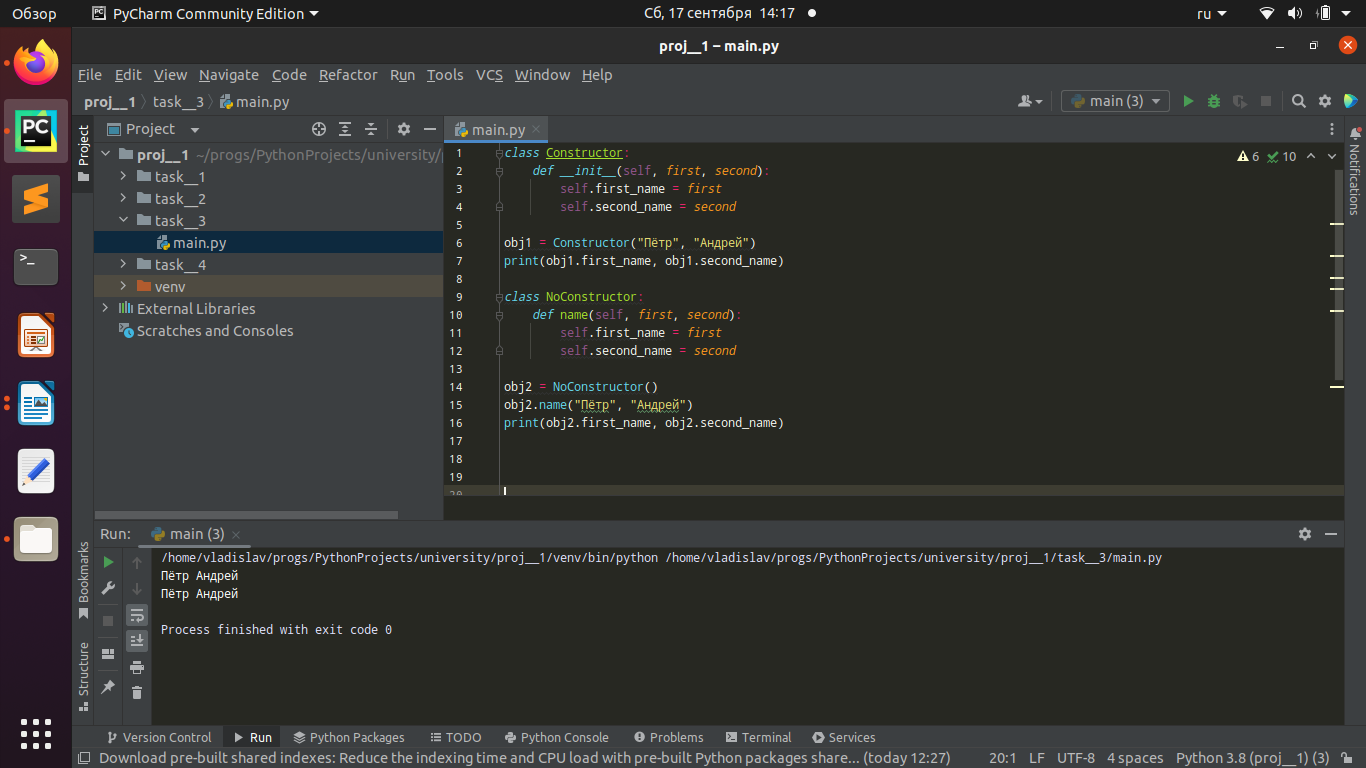
Объяснить и предсказать результат их работы.

**Ход работы**

Сперва сравним классы и заметим, что оба имеют методы, но первый класс имеет особый метод **\_\_init\_\_** - конструктор (Рис 3.2). Это означает что мы должны при создании объекта сразу передать ему некоторые параметры, а именно: **first** и **second**. Которые в свою очередь присваиваются атрибутам **first\_name** и **second\_name**.

*Рис 3.2 — Конструктор класса Constructor*

Второй класс имеет обычный метод **name** (Рис 3.3), с теми же параметрами и функционалом, но для того чтобы передать значения параметров атрибутам класса, его нужно вызывать отдельно.

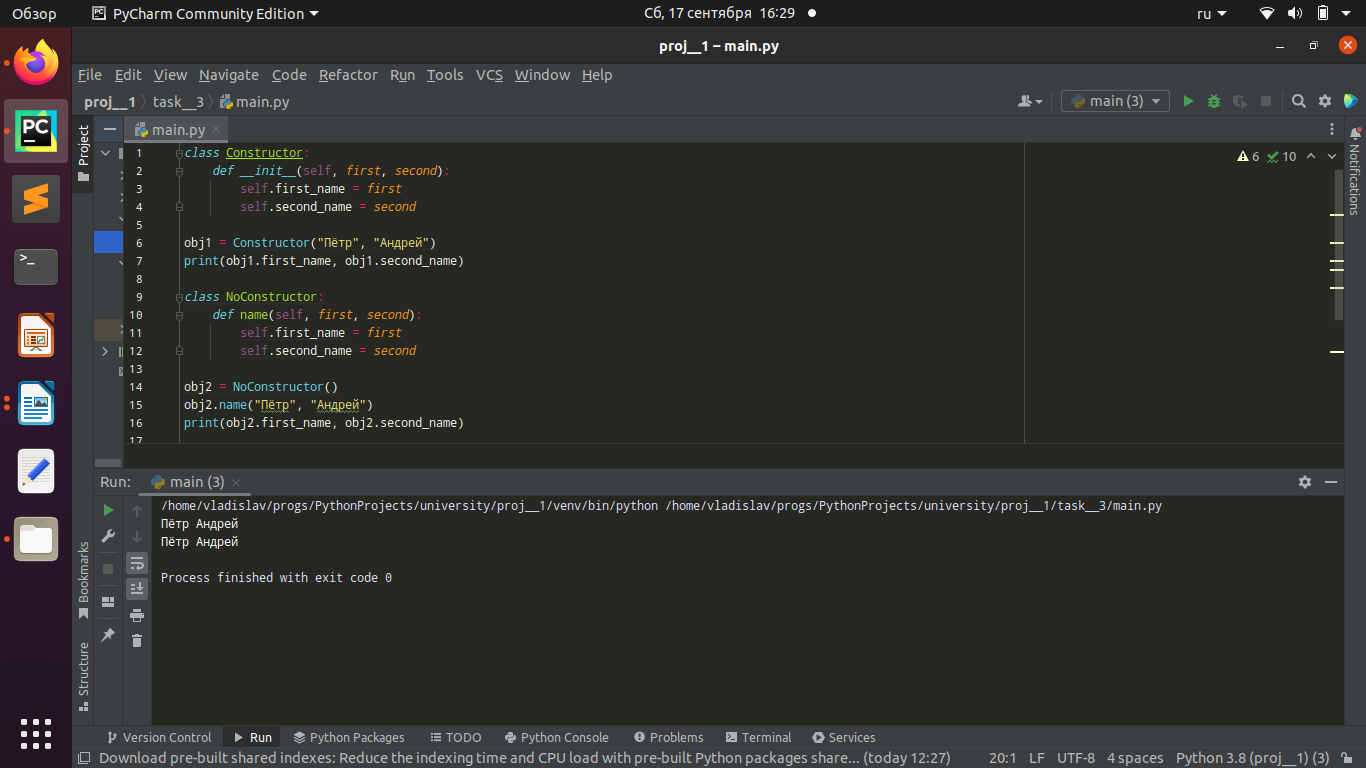
*Рис 3.3 — Метод класса NoConstructor*

В представленном выше **тексте программы** (Рис 3.1) мы видим по одному экземпляру каждого класса. Их атрибуты должны распечататься в консоли. В результате мы должны получить:

**Петр Андрей**

**Петр Андрей**

Запустим программу и сравним (Рис 3.4).

*Рис 3.4 — Результат работы программы*

Результат оказался верным, как мы и предполагали.

**Итог**

Мы создали два простых похожих по функционалу класса, изучили и объяснили их работу. Также верно предположили и проверили результат работы программы.

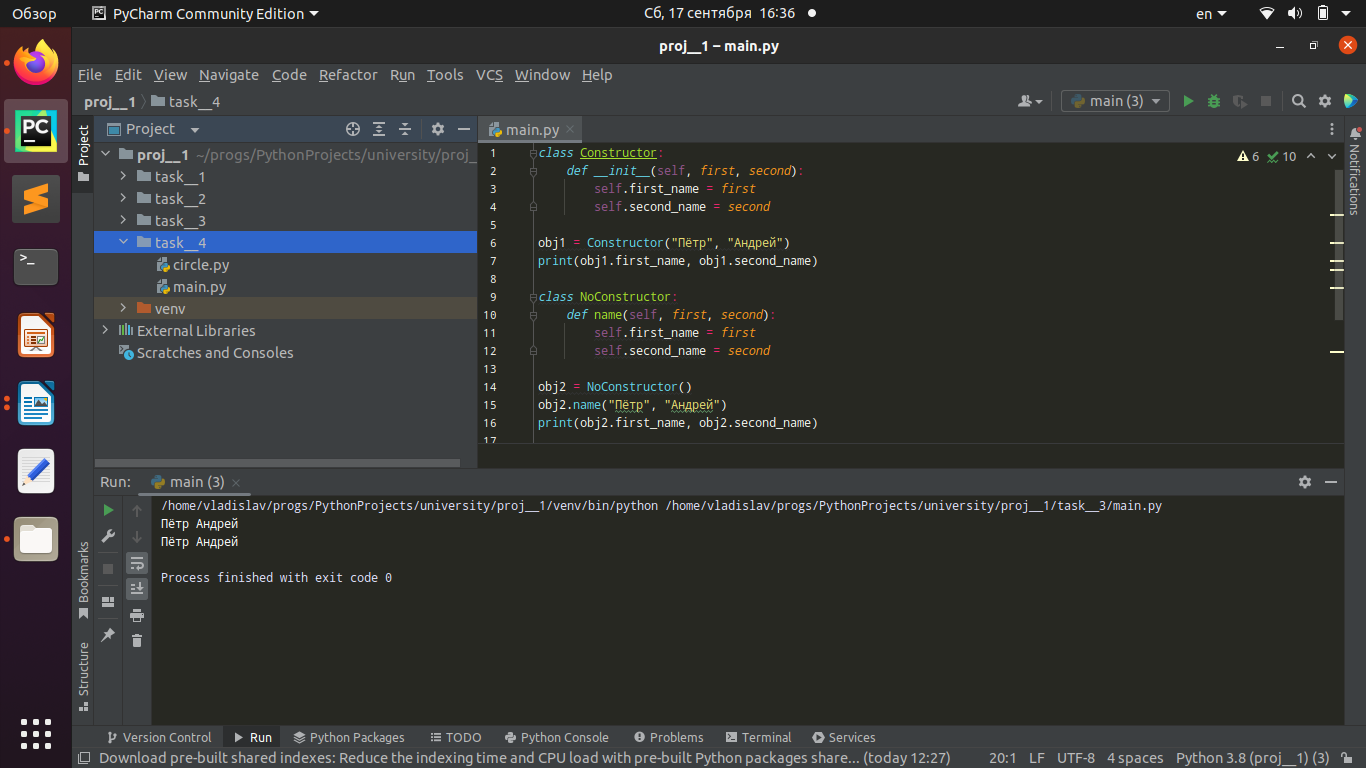
**Задание 4**

**Текст задания**

Создать класс для создания объекта «Круг» с конструктором, устанавливающего начальные значения радиуса и площади, вычисляемой по формуле. Создать два объекта с разными параметрами, распечатать их, а затем в программе поменять их значения.

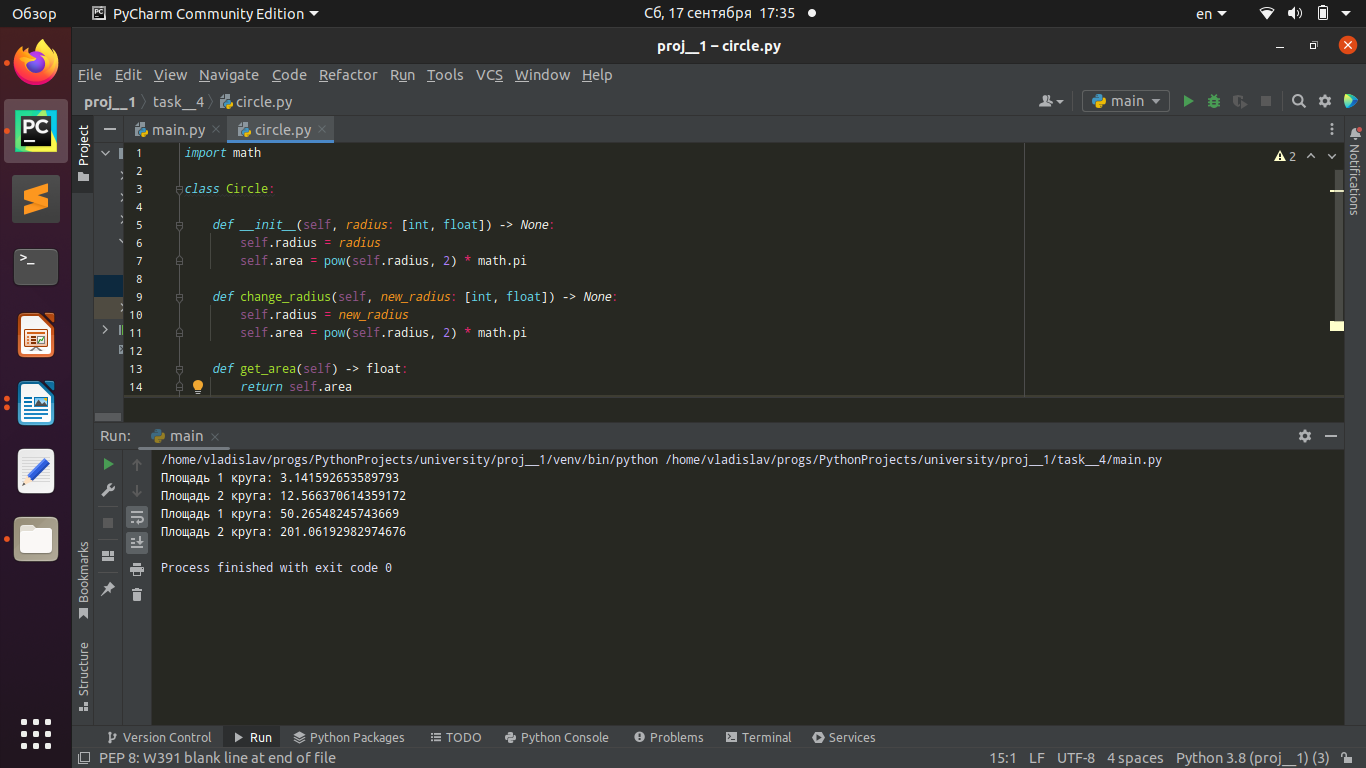
**Ход работы**

Сперва раздели программу на **2 файла** (Рис 4.1). В первом напишем класс, во втором создадим два объекта с разными параметрами, распечатаем их, а затем поменяем их значения.

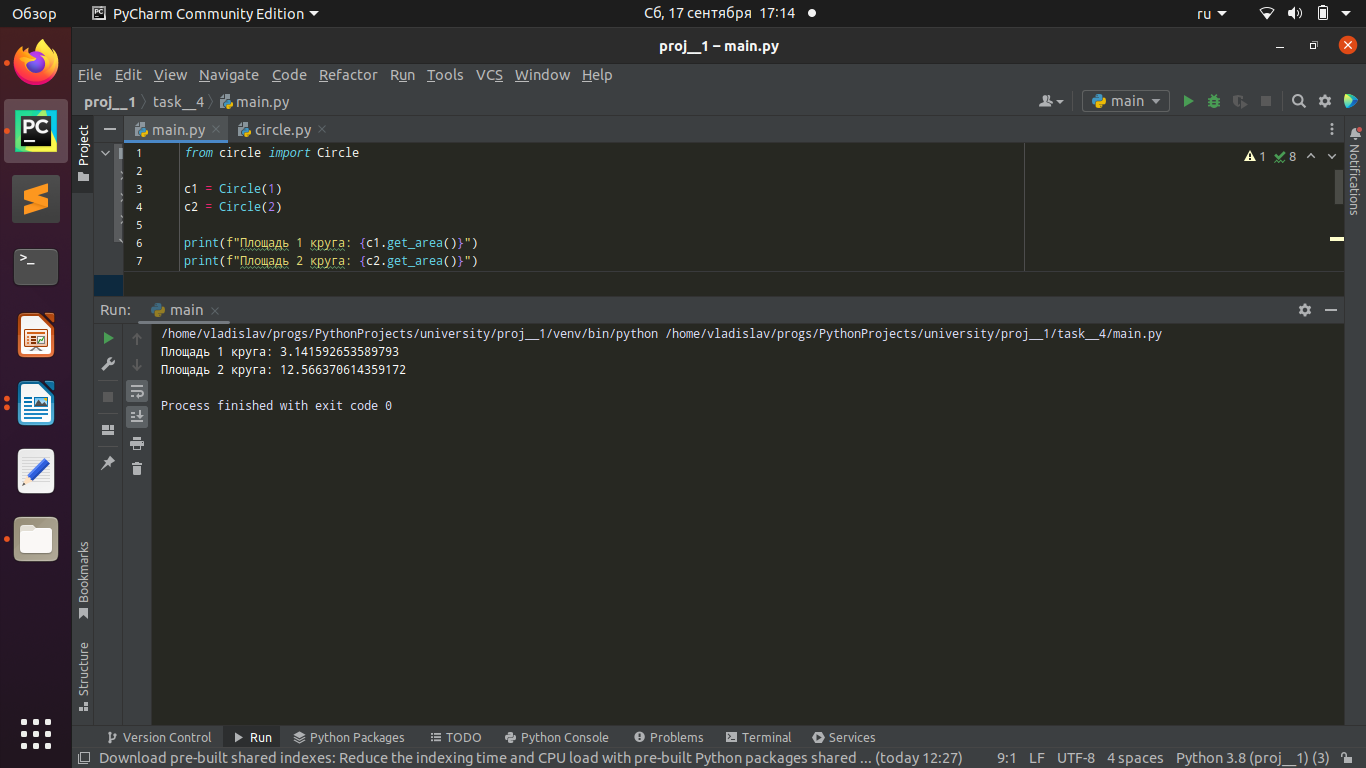
*Рис 4.1 — Файлы проекта к заданию 1*

Назовем **класс Circle** (*с англ. «круг»*). Напишем для него 3 метода (Рис 4.2):

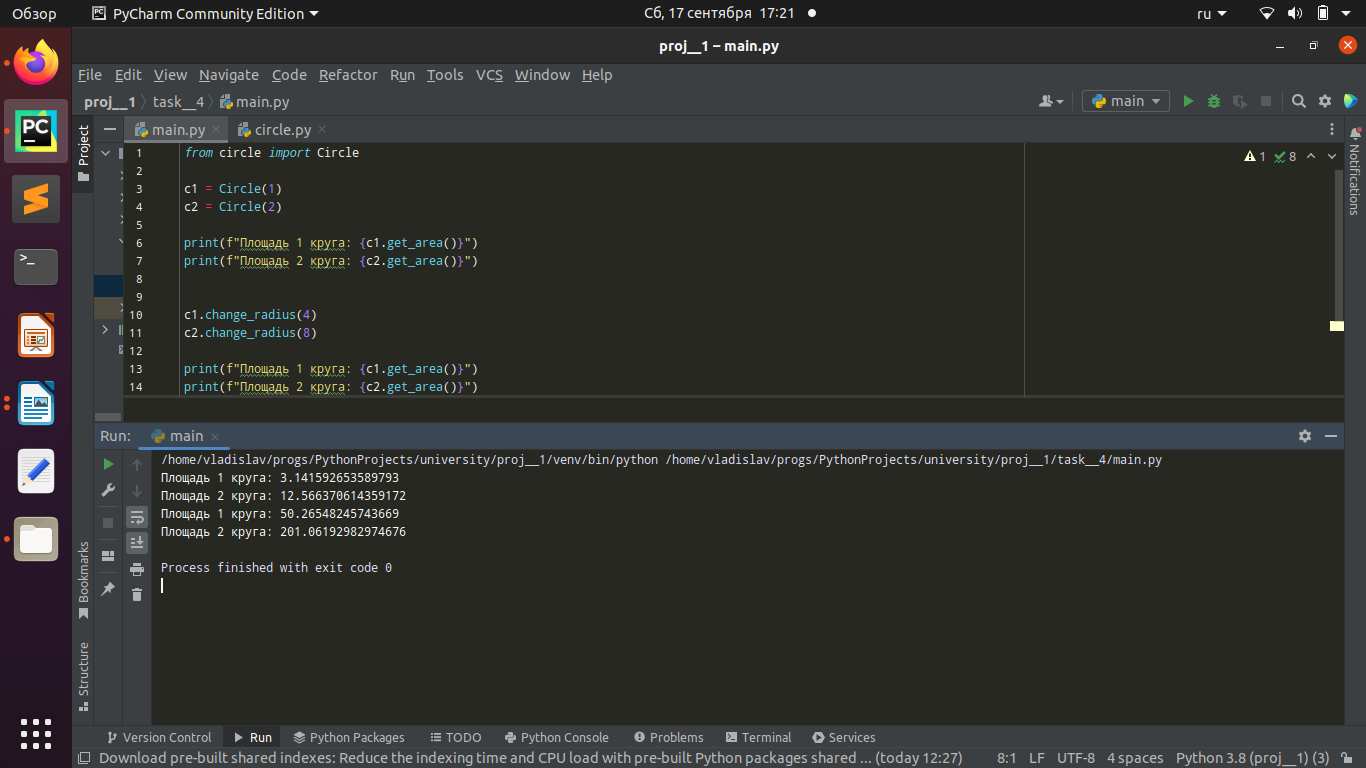
* **\_\_init\_\_** — конструктор. Будет принимать параметр **radius**, с типом данных **int** или **float**, а также вычислять площадь, по формуле, и передавать результат в атрибут **area**. Для расчетов импортируем модуль **math**, из него будем брать число **pi**;
* **change\_radius** — с помощью этого метода будем менять атрибут **radius**. Этот метод будет принимать параметр **new\_radius**, с типом данных **int** или **float**, передавать его в атрибут **radius** и пересчитывать площадь;
* **get\_area** — будет возвращать атрибут **area**, с типом данных **float**.

*Рис 4.2 — Класс Circle*

Импортируем в файл **main.py** наш класс и создадим 2 объекта **c1** и **c2** с начальными параметрами 1 и 2. Обратимся к методу **get\_area** и выведем полученные значения (Рис 4.3).

*Рис 4.3 — Результат работы программы*

Далее при помощи метода **change\_radius** поменяем атрибут **radius**. Снова получим площадь и выведем результат в консоль (Рис 4.4).

*Рис 4.4 — Результат работы программы*

Мы получили новые значения, программа отработала корректно.

**Итог**

Мы создали **класс Circle** для создания объекта с конструктором, устанавливающего начальные значения радиуса и площади, вычисляемой по формуле. Затем создали два объекта с разными параметрами, распечатали их, поменяли значения и снова распечатали. Значения были получены без ошибок.