**2023/11/04 00:00|Лекция. Цели и задачи. Зачем нужно наследование.**

## Цели и задачи. Зачем нужно наследование

На сегодняшнем занятии мы с вами будем говорить про наследование. Ранее на занятиях мы с вами познакомились с таким понятием, как**класс**и определили, что — это некая структура, на основе которой мы можем создавать огромное количество уникальных объектов. Получается что-то похожее на тех паспорт для объекта. Переменные, которые создаются на основе этого класса, называются экземплярами, либо же объектами этого класса.

Вот представим такую ситуацию: у нас есть класс человек, у которого есть один атрибут классовый head(голова), который имеет значение True(рис.1). Есть класс Student, у которого будет метод about, который просто будет выводить строку “Я студент”(рис.2).

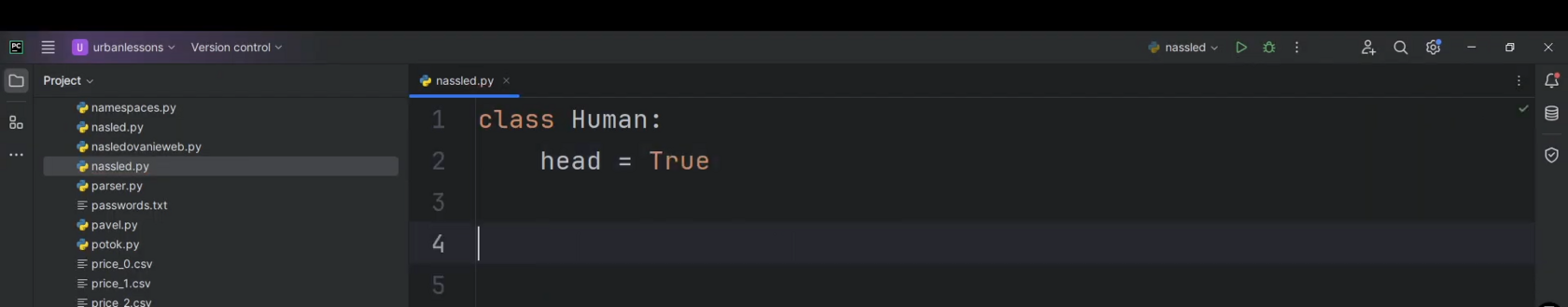


Рис.1

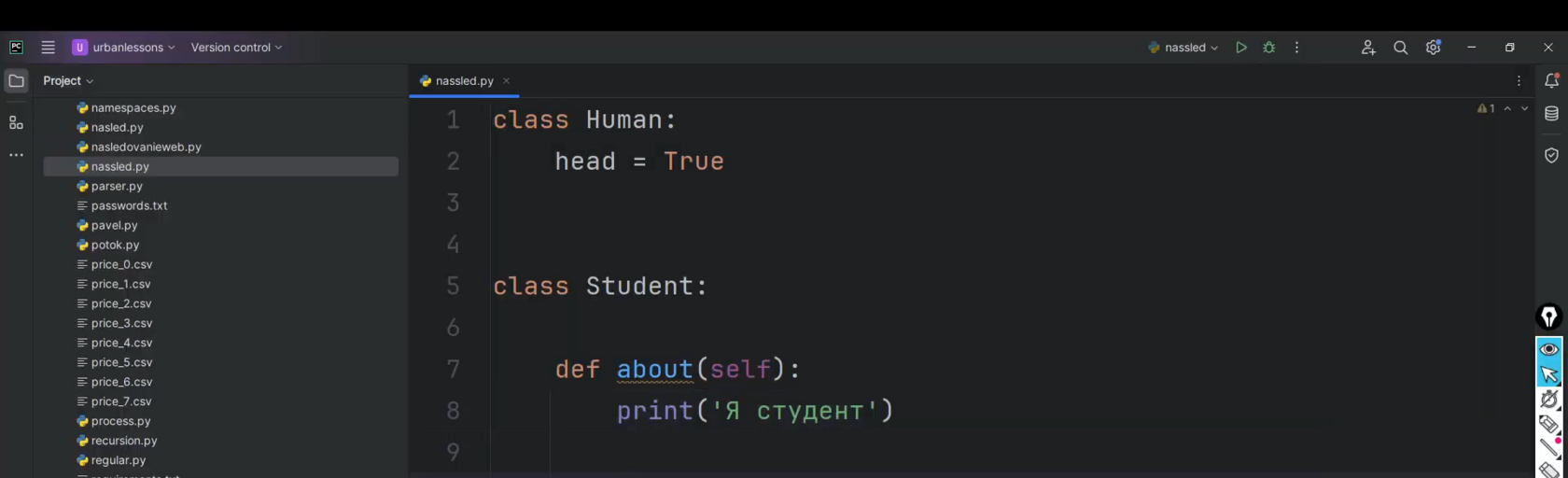


Рис.2

Также давайте создадим экземпляр класса Human (human=Human()) и экземпляр класса Student (student=Student()). Эти 2 класса сейчас не связаны между собой никаким образом. Для human мы можем получить значение атрибута head (human.head). Для student можем вызвать метод about (student.about()). Запуская программу, видим, что сейчас атрибут head – True, и при вызове метода about для student, видим строку “Я студент”(рис.3). Все работает, правильно.

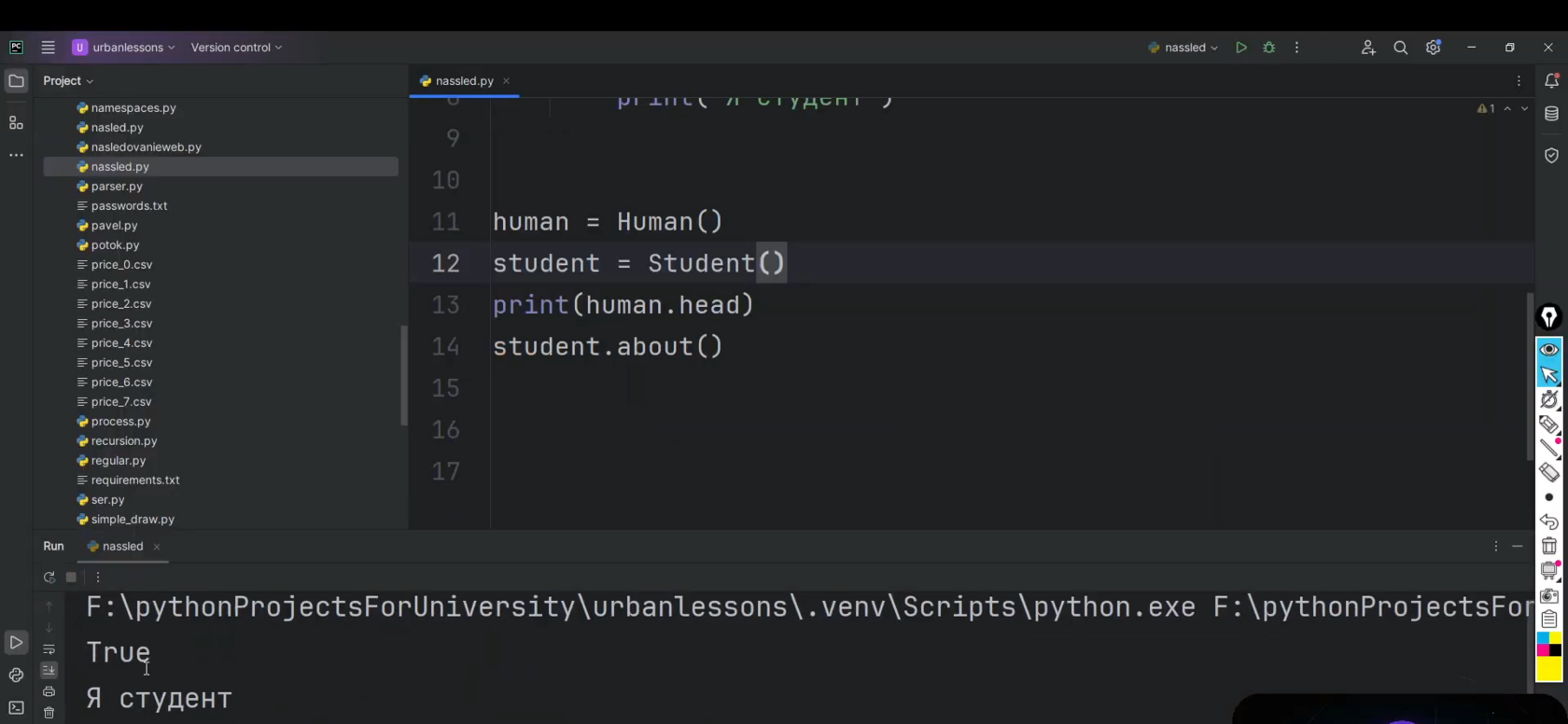


Рис.3

Однако, что, если мы хотим для student получить доступ к атрибуту head (student.head)? При запуске получаем ошибку(рис.4). Ту же самую ошибку мы будем видеть, если для human вызвать метод about. Потому что у human свое пространство имен, у student свое пространство имён, то есть это 2 абсолютно разных пространства имён. Сейчас они не связаны между собой никаким образом. Однако логично предположить, что student все-таки имеет что-то общее с классом Human, так как student тоже человек.

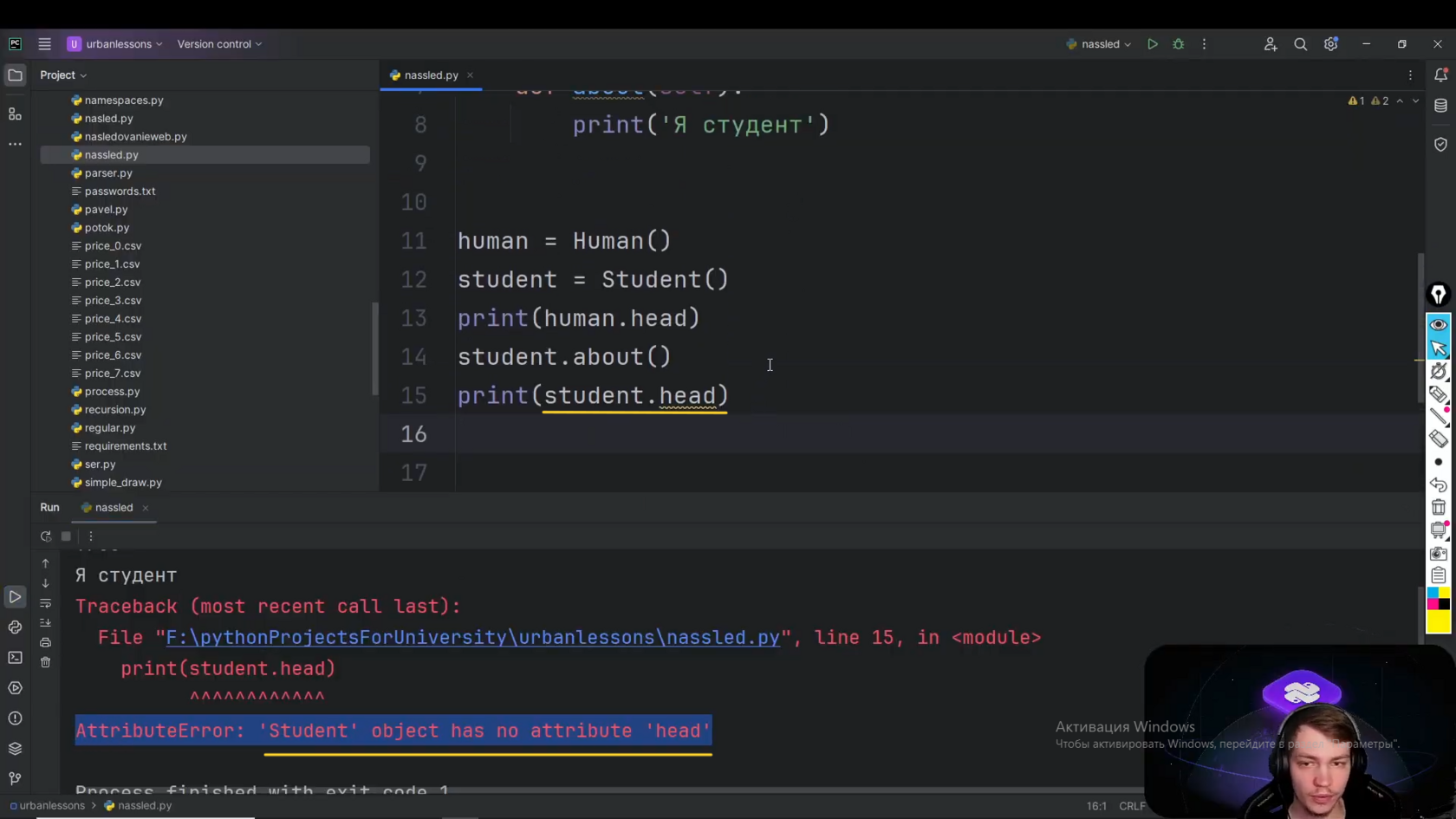


Рис.4

**Как сделать так, чтобы мы могли из студента получить доступ к атрибутам, доступным в классе Human?** Чтобы это сделать, нужно реализовать механизм наследования. Для этого после имени класса необходимо открыть круглые скобки и написать имя класса, от которого будет наследоваться наш класс. Получается, если у студента открыть скобки и написать здесь Human, то класс Student будет наследоваться от класса Human(рис.5). В таком случае класс Human будет являться базовым или родительским классом. Класс Student будет являться дочерним. Если рисовать это как схему, то получается: сначала будет идти Human, а потом Student (стрелочки идут от дочернего к родительскому классу)(рис.5).

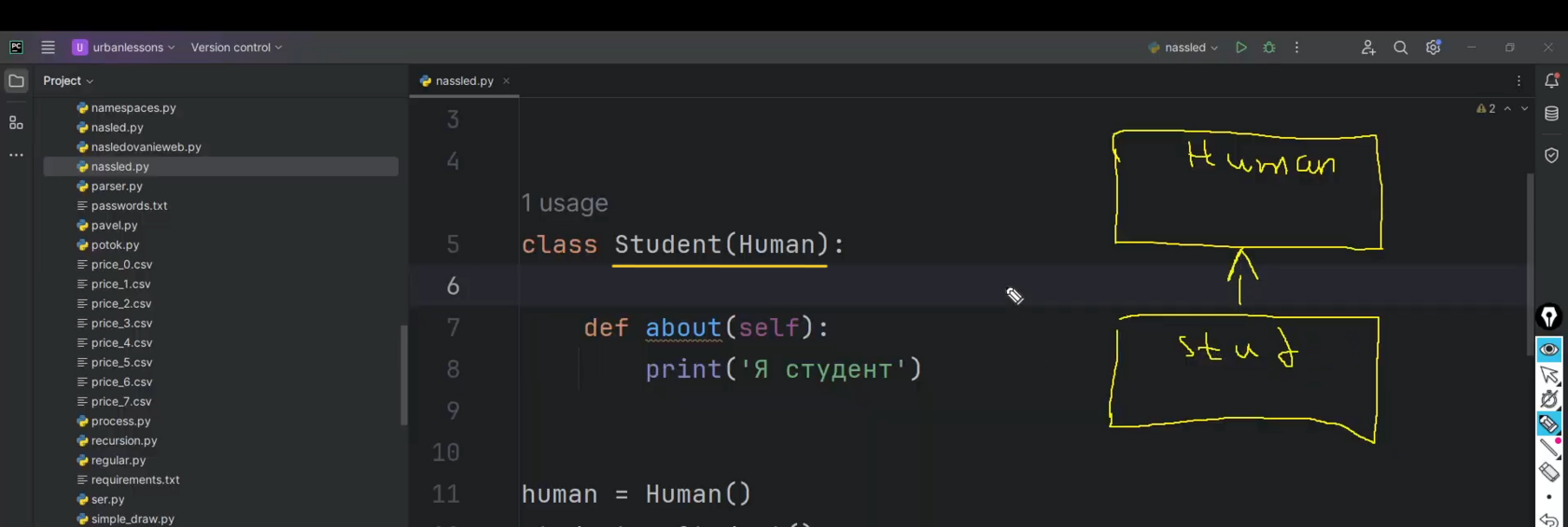


Рис.5

Давайте попробуем теперь в классе Student получить доступ к атрибуту head (student.head). Запустим и видим здесь True(рис.6). В итоге получилось достать атрибут, который находится в классе Human. Однако, если возьмем human и попытаемся вызвать метод about, то получим ошибку(рис.7). То есть из дочерних классов мы можем получить доступ к атрибутам, которые находятся в родительском классе.

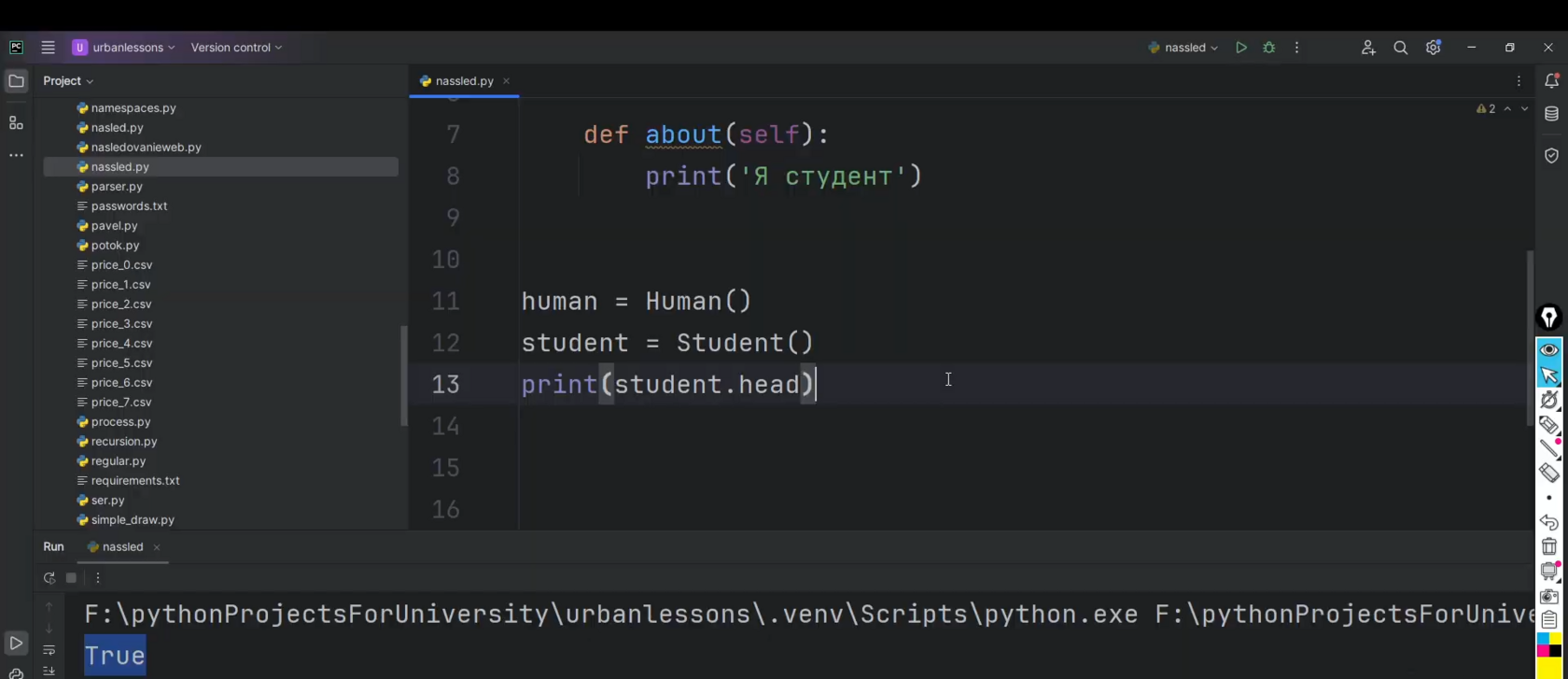


Рис.6



Рис.7

**Происходит это вот таким вот образом:**изначально мы проверяем наличие атрибута или определенного метода внутри дочернего класса. При попытке достать head, мы должны сначала проверить, есть ли этот head внутри класса Student. Только потом, если атрибут или метод с соответствующим именем здесь не был найден, благодаря наследованию, мы получаем доступ к пространству имен класса Human. Мы можем заглянуть в пространство имен класса Human для того, чтобы найти этот атрибут или метод. Получается из дочерних классов мы имеем доступ к атрибутам и методам родительского класса.

**Но есть различные обходные пути.** Давайте теперь у класса Student атрибут head пропишем False(рис.8). Если из Student пытаться достать этот атрибут, то будем получать уже значение False(рис.9).

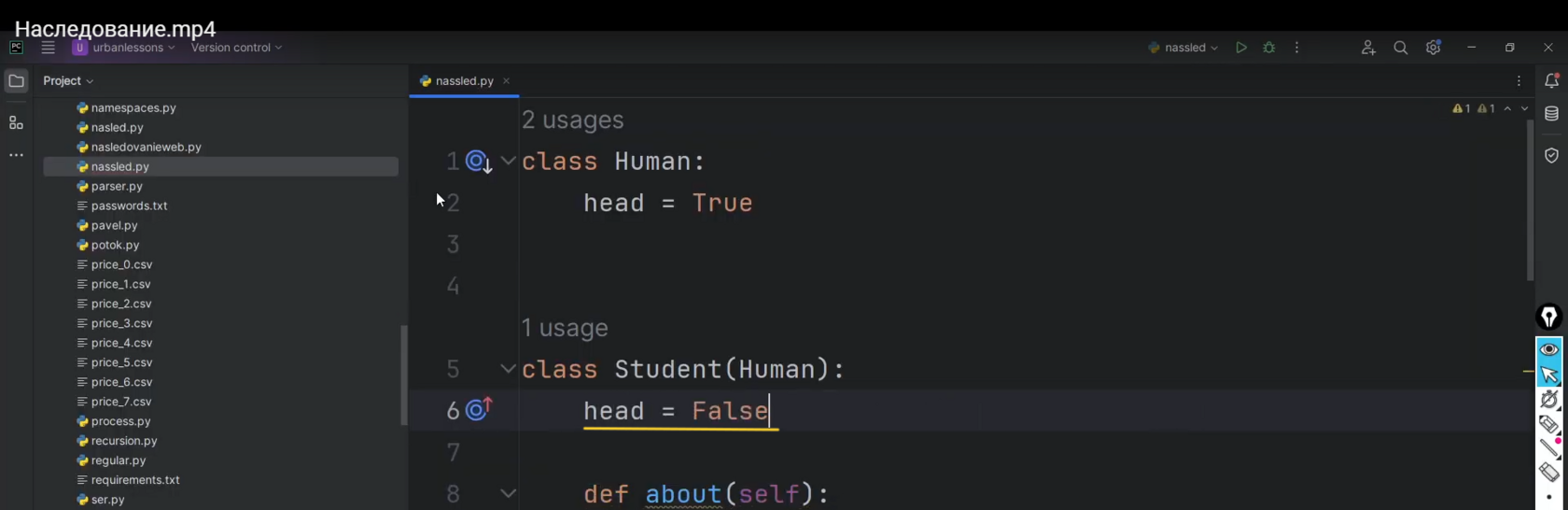


Рис.8

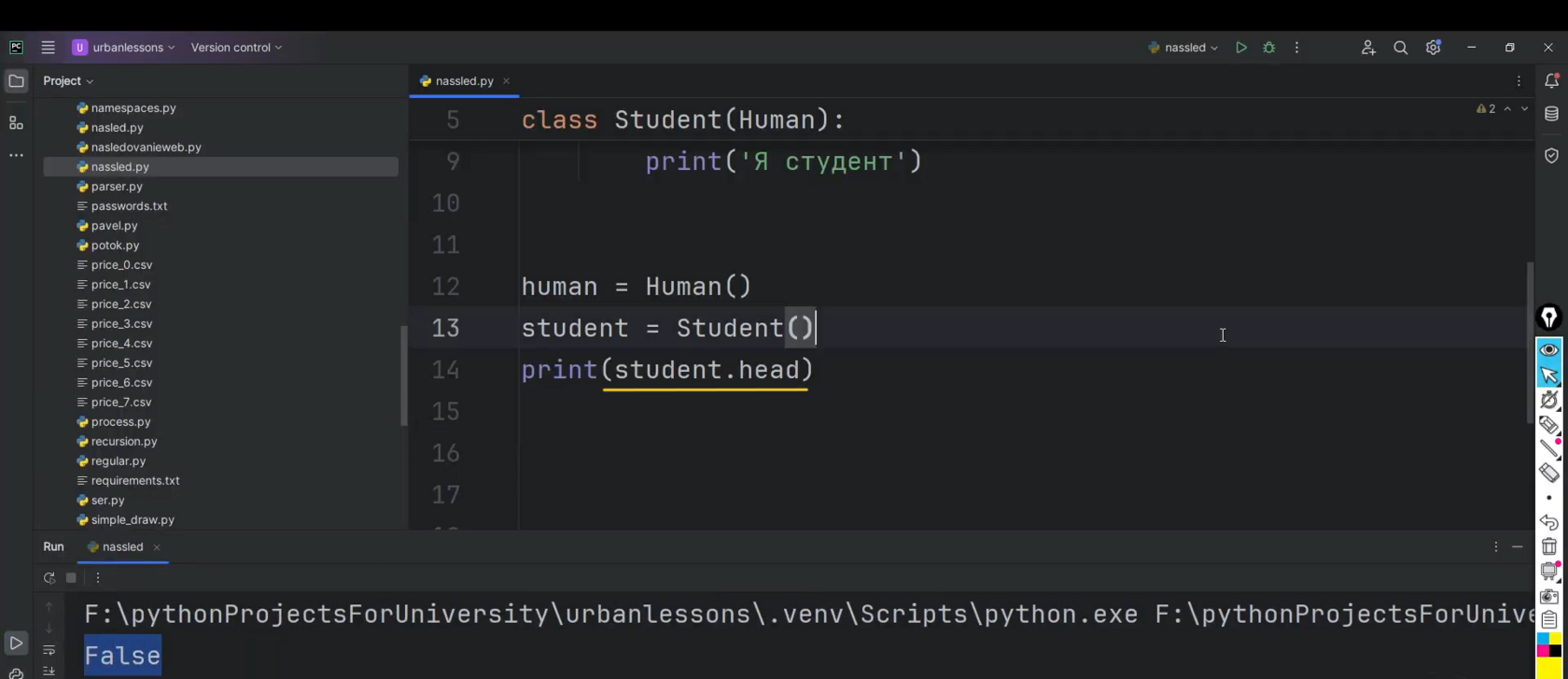


Рис.9

**Как в родительском классе вызвать, например, about?**

Создадим здесь конструктор init и вспоминаем, что — это метод, который срабатывает в момент, когда создаётся объект этого класса. Тут вызовем метод about для self(рис.10). Получается, если мы оставим строчку human=Human(), то получим ошибку. Давайте пока ее закомментируем. Строчка student=Student() сработает так, как нужно. При запуске создается экземпляр класса Student, однако видим здесь сообщение “Я студент”(рис.11), хотя мы не вызывали этот about напрямую(рис.12).



Рис.10

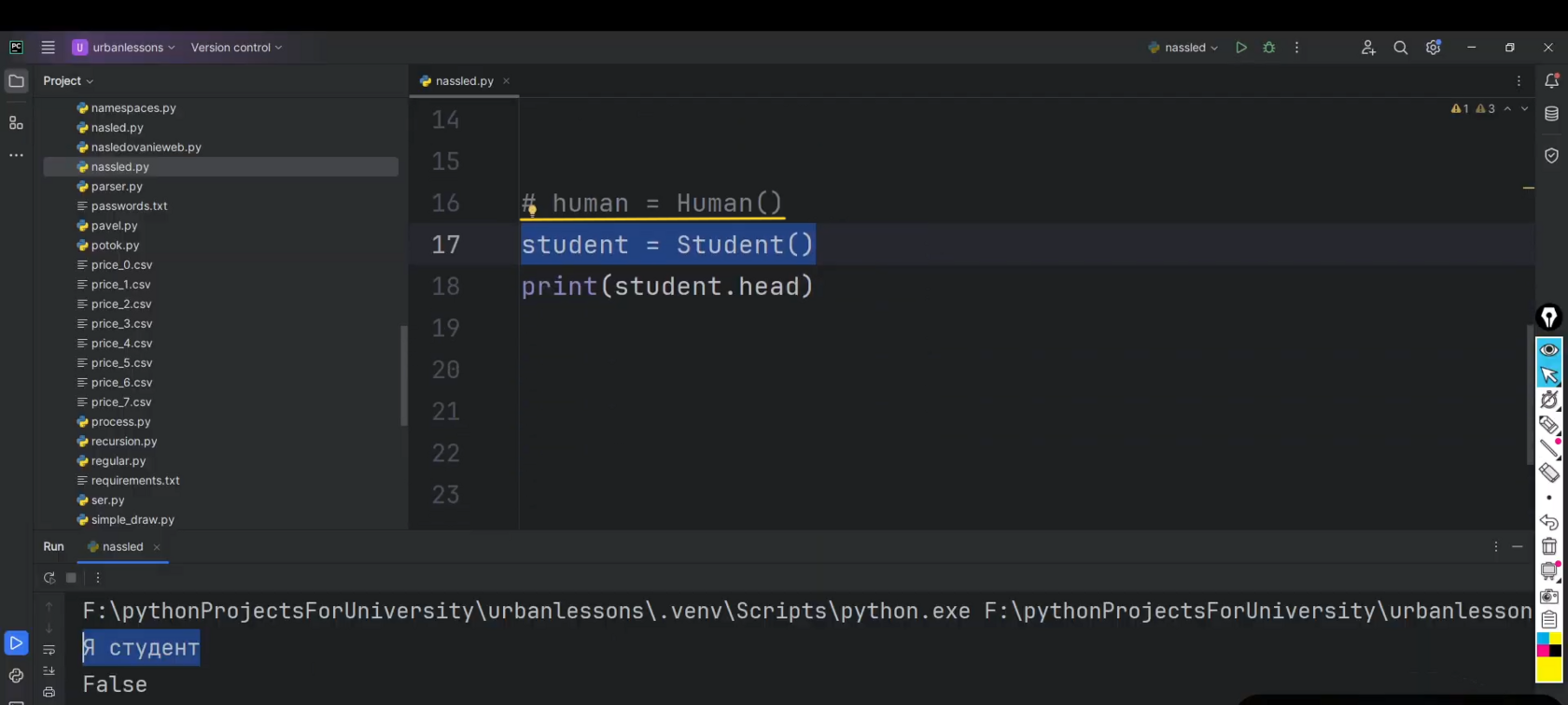


Рис.11

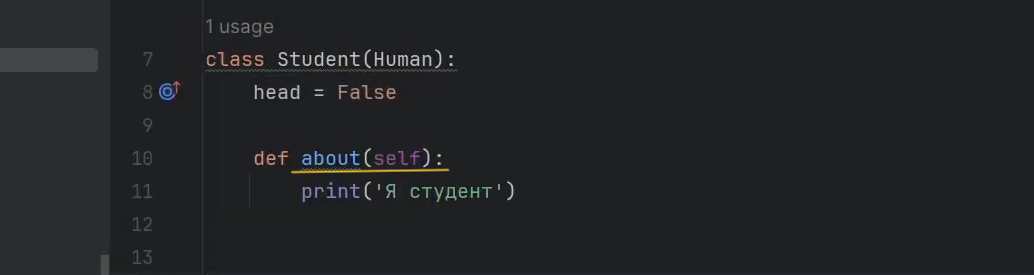


Рис.12

Дело в том, что, когда мы создаем экземпляр класса Student, экземпляр дочернего класса, у нас также идёт обращение к родительскому классу. Получается, что этот init сработал и по отношению к student. Мы увидели здесь информацию “Я студент”, потому что self(рис.13) принял значение student(рис.14), то есть получается, что self сработал не для класса Human, а принял вот этот student, ссылку на объект класса student. А ссылка на объект класса Student имеет доступ к методу about(рис.15). Немного запутано, но со временем принципе вы привыкнете. Наследование — это очень крутой механизм, который позволяет нам избавляться от лишнего кода.

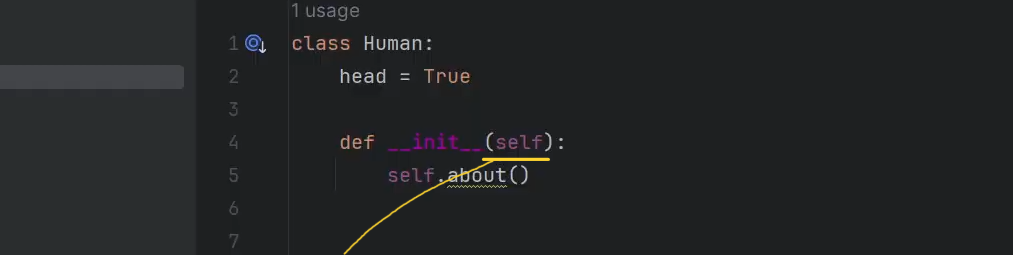


Рис.13

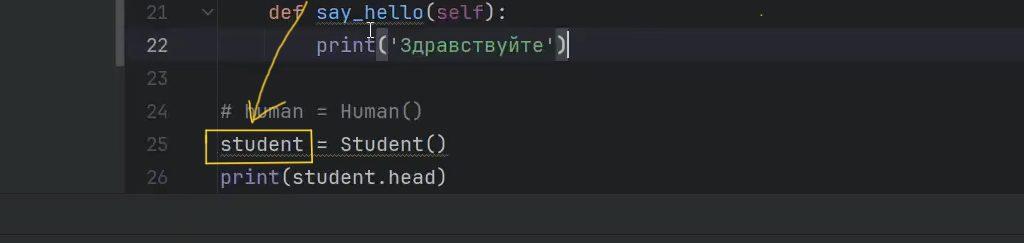


Рис.14

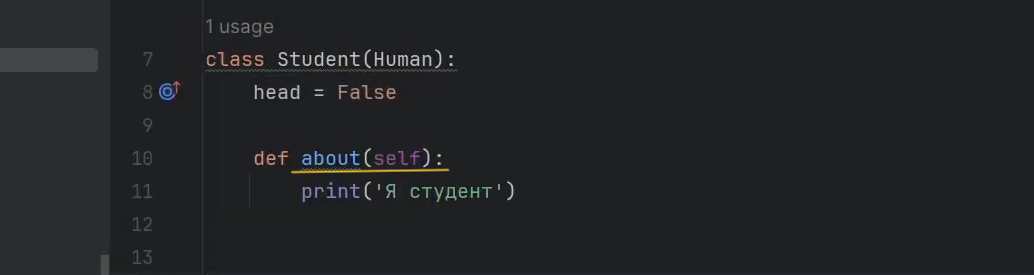


Рис.15

**Как это работает?** Например, есть student, он наследуется от класса Human. Студент может называть свое имя или просто приветствует кого-то. Давайте сделаем say\_hello. Получается, здесь наш студент имеет возможность поздороваться(рис.16). Представим, что есть еще класс Teacher(учитель), который точно так же может здороваться. Смотрите, учитель и студент имеют абсолютно одинаковый метод(рис.17). Как мы с вами помним, на одном из предыдущих занятий уже упоминался принцип, который называется DRY (Don’t repeat yourself), что переводится как — не повторяйся. Мы по возможности должны избегать лишнего дублирования кода. Раз учитель и студент — это дочерние классы, подклассы Human, то мы будем иметь доступ к пространству имен этого класса Human.

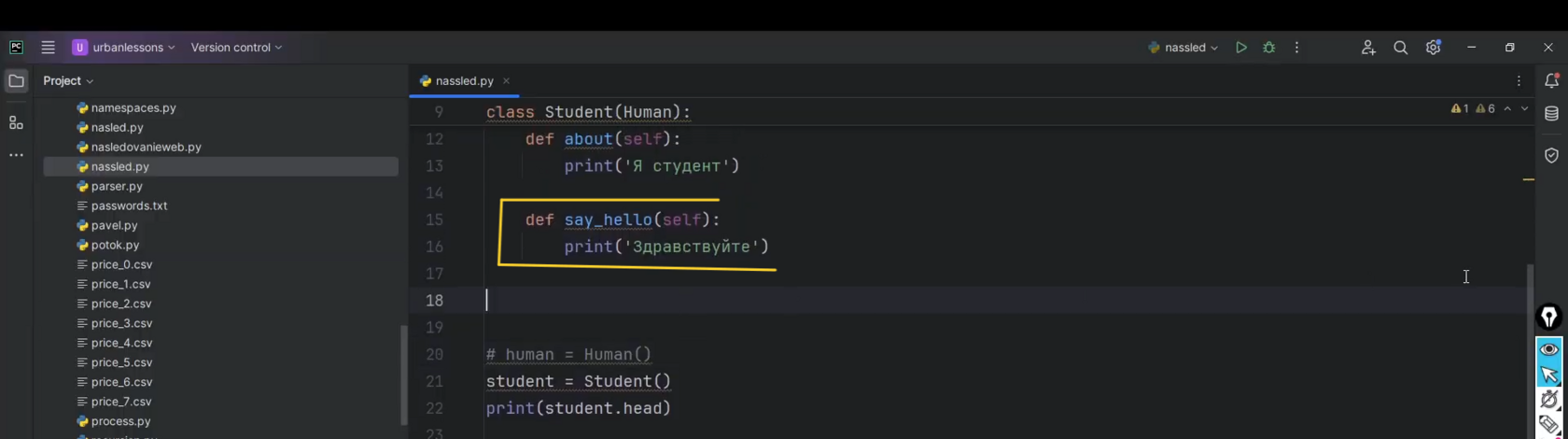


Рис.16

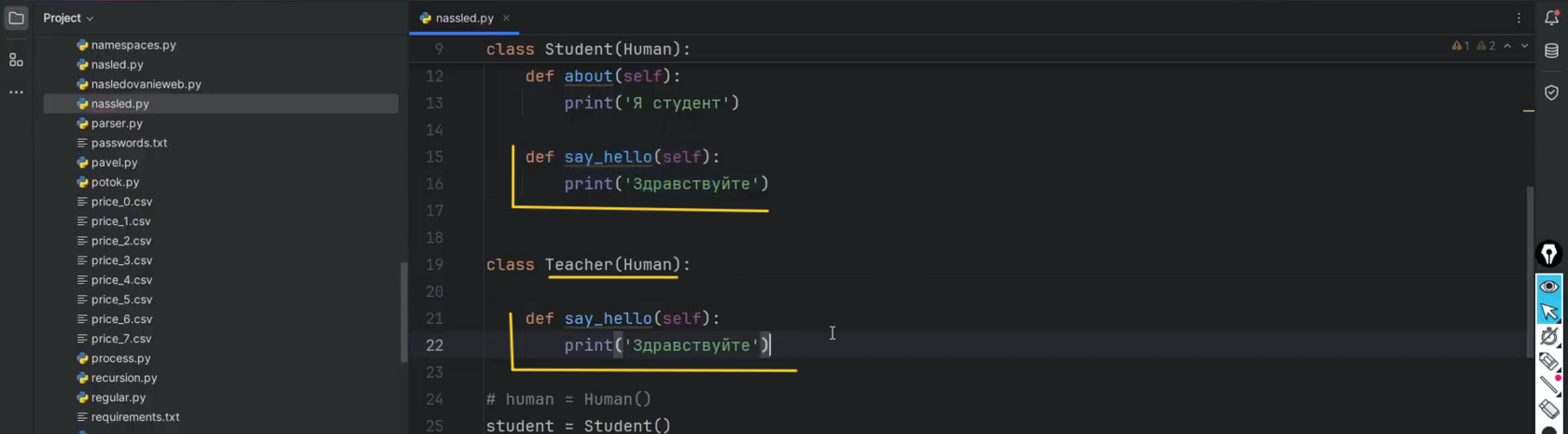


Рис.17

Логично, что мы с вами можем этот метод вынести в родительский класс(рис.18). Даже если в Teacher ничего не будует, просто обычный pass заглушка(рис.19) будет стоять, получается, мы можем и для студента вызывать этот метод и для учителя, и при запуске всё это будет работать(рис.20). Потому что в принципе логика такая: каждый человек может поздороваться. Студент является человеком, учитель тоже является человеком. Однако в дочерних классах эти атрибуты могут отличаться. Механизм наследования используют для того, чтобы расширить функционал базового класса, либо как-то изменить.

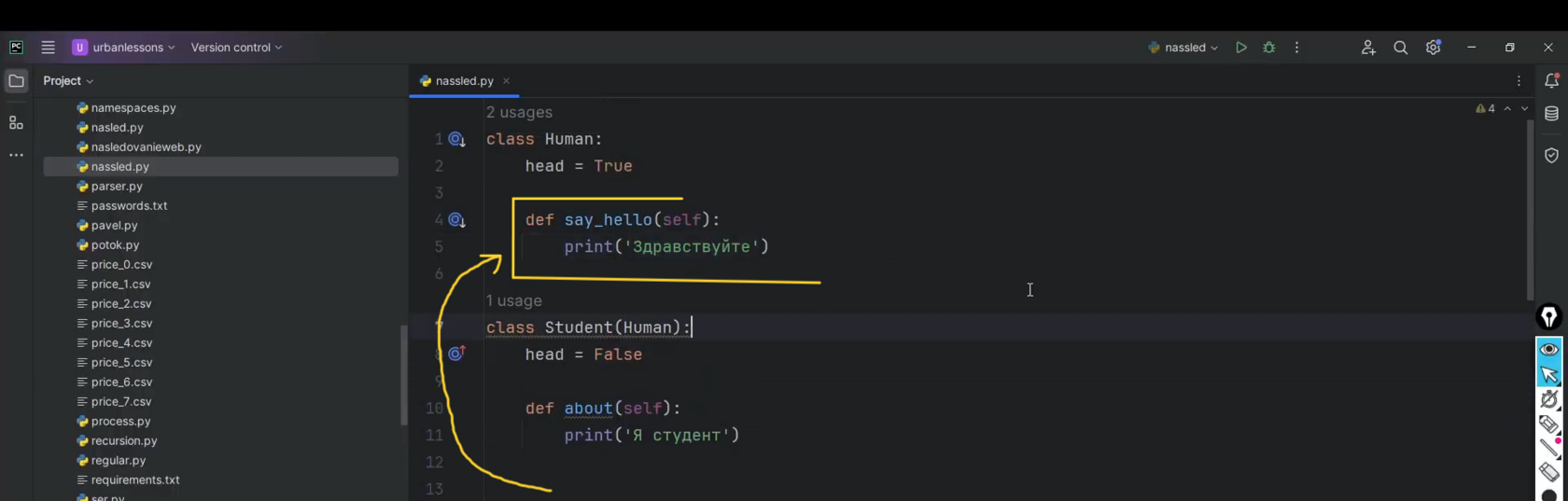


Рис.18

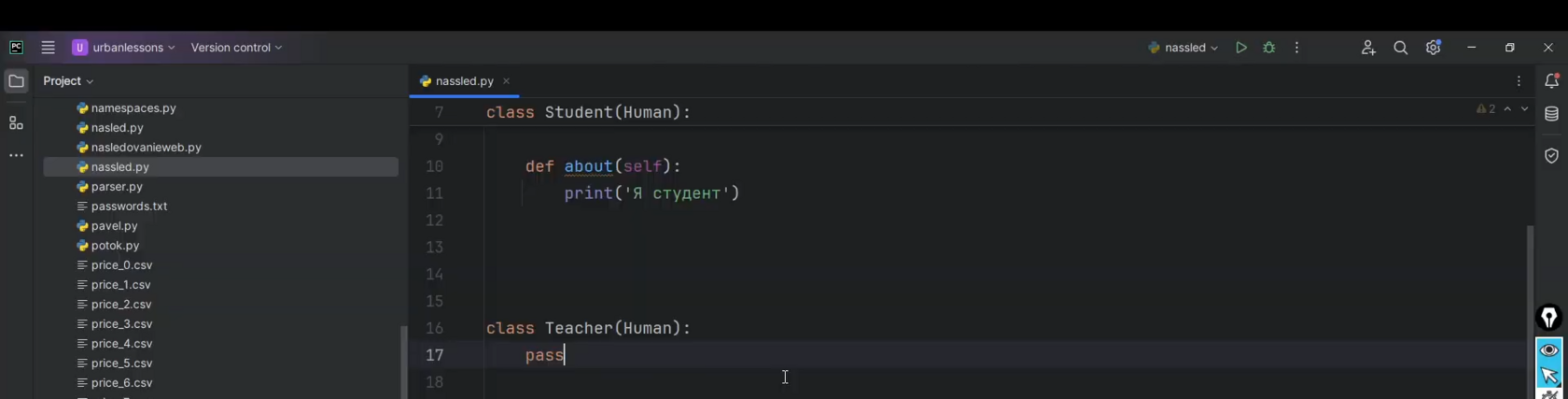


Рис.19

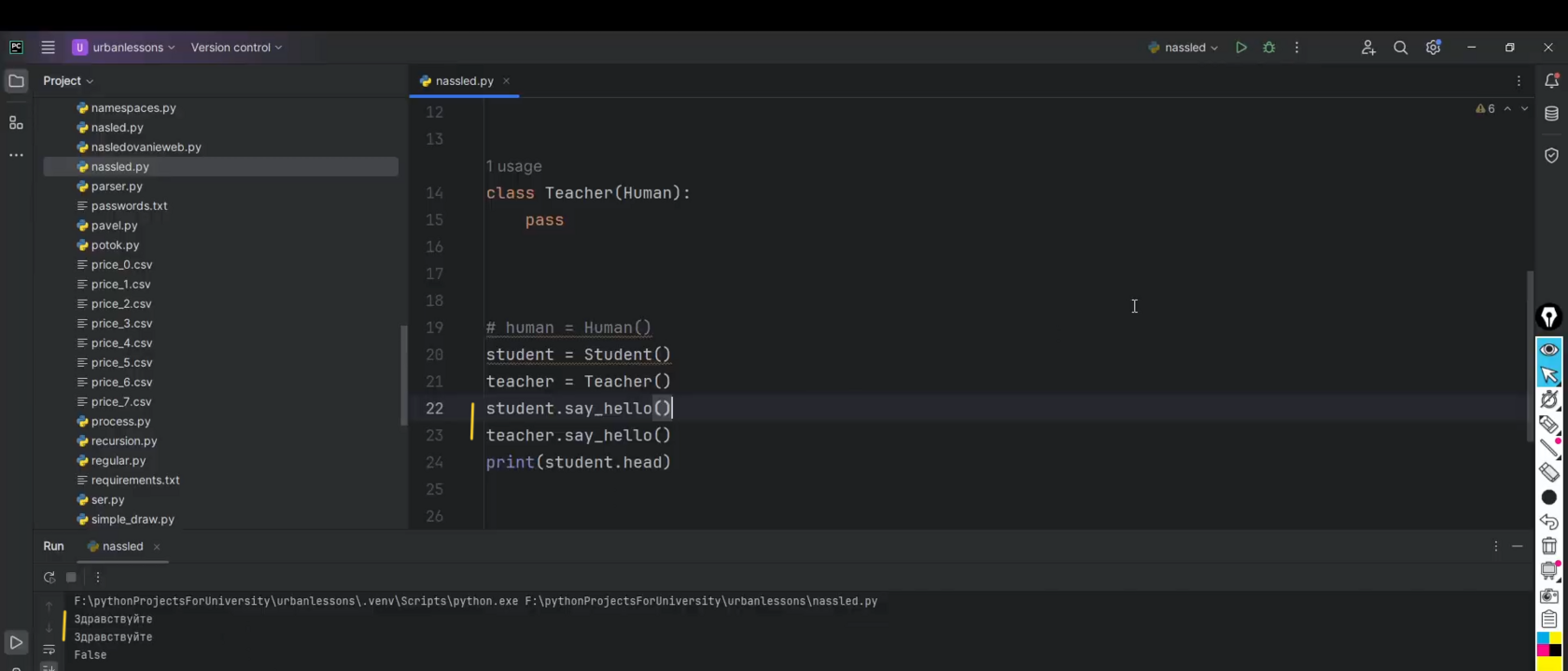


Рис.20

Изначально мы ссылаемся на пространство имен, которые находятся внутри класса, по отношению с которым, мы работаем. Если мы для студента пытаемся вызвать атрибут или метод, мы смотрим сначала внутрь студента. Однако, если внутри студента не найден соответствующий метод либо атрибут, мы смотрим наверх, уже в родительский класс и пытаемся найти это в его пространстве имён. Если не находим, соответственно, получаем AttributeError.