**2023/10/29 00:00|Лекция. Различие атрибутов класса и экземпляра. Пространство имен класса.**

## Различие атрибутов класса и экземпляра. Пространство имен класса

На предыдущих занятиях мы разобрались, что **класс**— это наш собственный тип данных. Он имеет свои характеристики и свои методы. Научились пользоваться магическими методами, перегружать операторы, создавать объекты на основе этих классов. Однако ранее пробовали также и создавать объекты с характеристиками, которые принадлежат конкретно к объектам.

На этом уроке рассмотрим, как это работает немного подробнее.

В нашем конструкторе init() прописаны характеристики name и age(рис.1). Можно сказать, что это атрибуты нашего класса, но это будет неверно, потому что данные атрибуты будут принадлежать только к объектам. Внутри самого класса их не существует и бывают ситуации, когда необходимо создавать атрибуты и они будут общие для всех экземпляров. Чтобы создавать такие атрибуты, как правило, нам необходимо просто создать переменную. Её можно создать в пределах определённого метода, либо же вначале описания нашего класса.

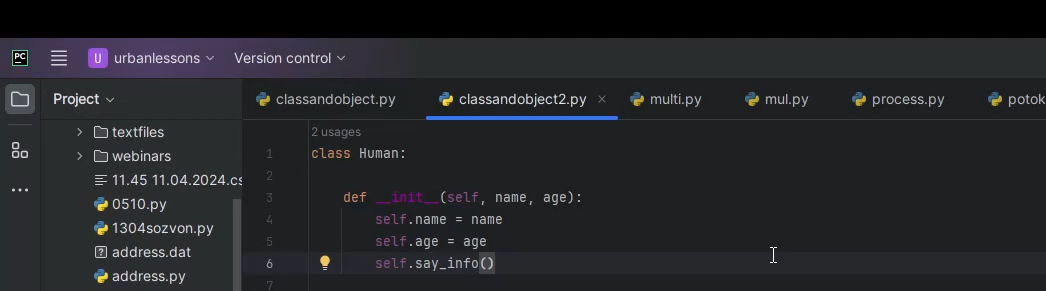


Рис.1

Давайте создадим переменную head и она будет принимать значение True(рис.2), так как у всех людей есть голова. Данный атрибут не будет принадлежать ни одному из наших объектов. Это будет классовый атрибут. Для того чтобы обратиться к классовому атрибуту, можно написать название нашего класса, поставить точку и достать оттуда этот атрибут(рис.3).

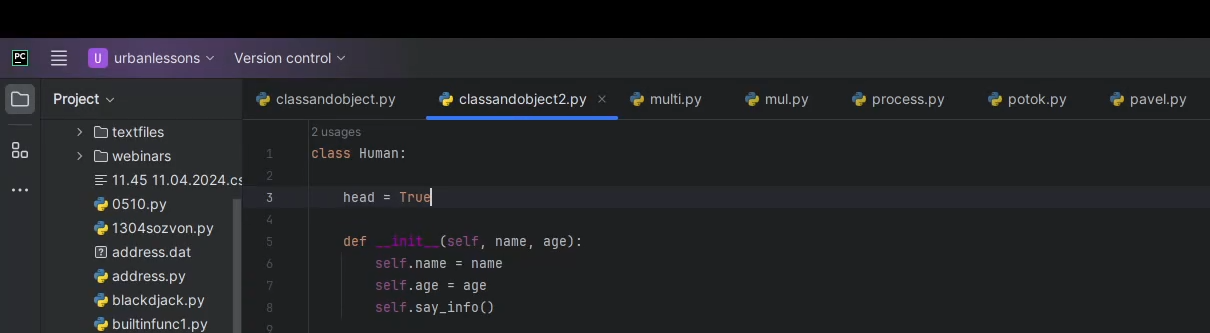


Рис.2

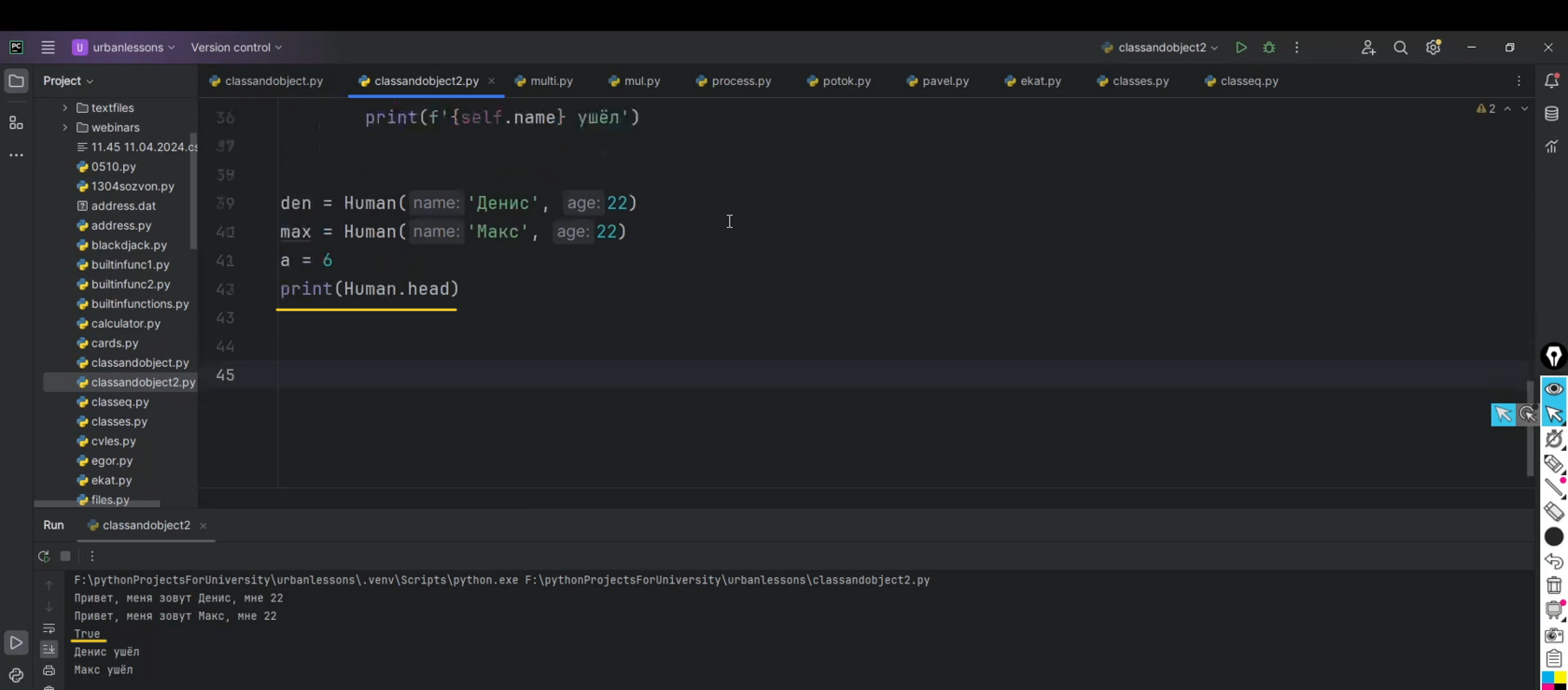


Рис.3

**Что нам даёт это знание?** Давайте запустим этот же самый файл в интерактивном режиме и посмотрим. Если развернуть Human, то увидим, что у нашего класса есть 1 атрибут head(рис.4), но не атрибутов name и age. При этом, если заглянем в наши объекты, то увидим здесь name, age и head(рис.5). **Получается у объектов есть атрибут класса, а у класса нет атрибутов этих объектов.**

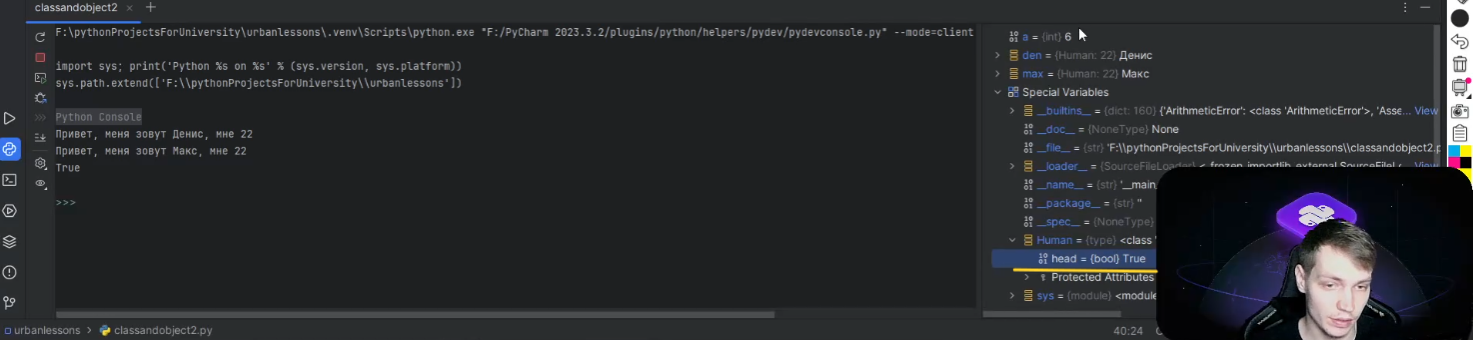


Рис.4

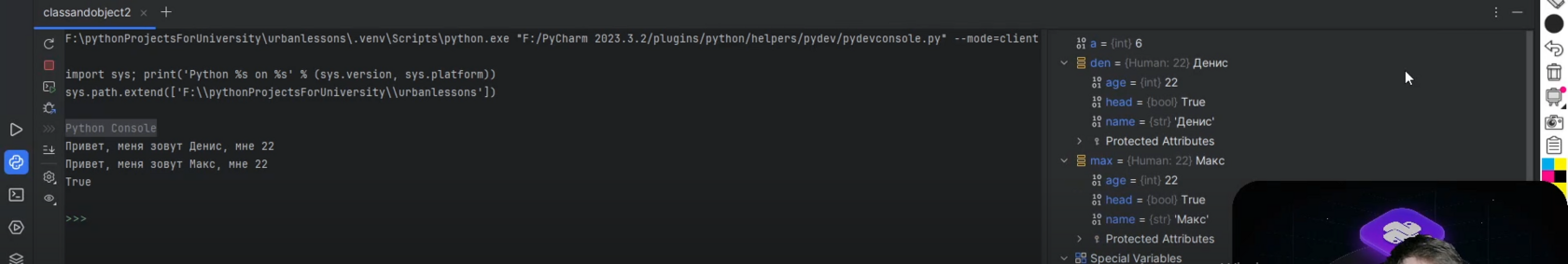


Рис.5

На одном из первых занятий мы говорили, что при создании объектов создается новое пространство имён. На самом деле Рython очень любит пространство имён и одно из правил гласит, что “Чем больше пространств имён, тем лучше”. С точки зрения реализации, даже вот этого head у нас нет внутри наших объектов.

**Давайте посмотрим в интерактивном режиме.**Возьмём наш класс Нuman и вызовем метод dict() для того, чтобы посмотреть, как выглядит наш класс в виде словаря(рис.6). Здесь можно увидеть знакомую конструкцию, однако незнакомые ключи и значения. Но все же кое-что знакомое здесь есть: head=True, то есть у нас ключ head, значение True. Также здесь есть init инициализатор, ряд функций и значения.

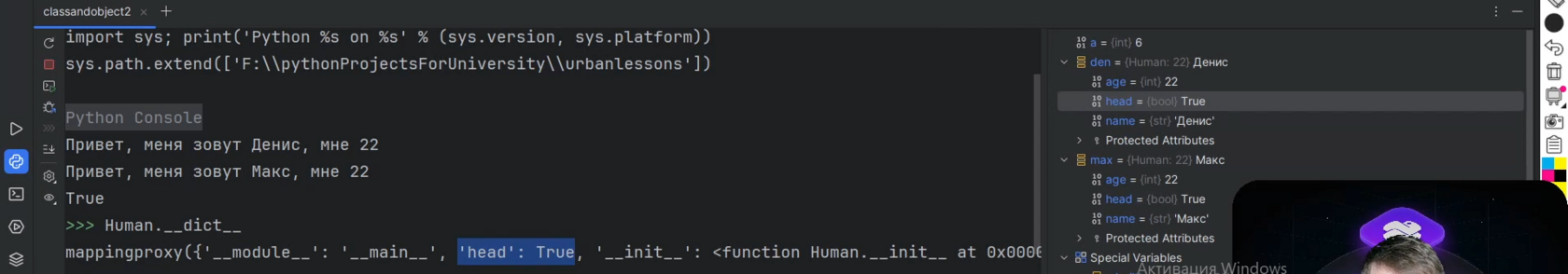


Рис.6

Дальше возьмём den и посмотрим на то, как он выглядит в виде словаря. Здесь можно увидеть только name и age(рис.7). При создании den у нас появляется собственное пространство имён, и внутри этого пространства имён появляется атрибут name и age. Если возьмём наш класс Human, атрибут head и изменим его значение на False, то увидим, что здесь появилось False(рис.8). Иначе говоря, у всех экземпляров, которые наследуются от этого класса, это значение изменилось на False(рис.9).

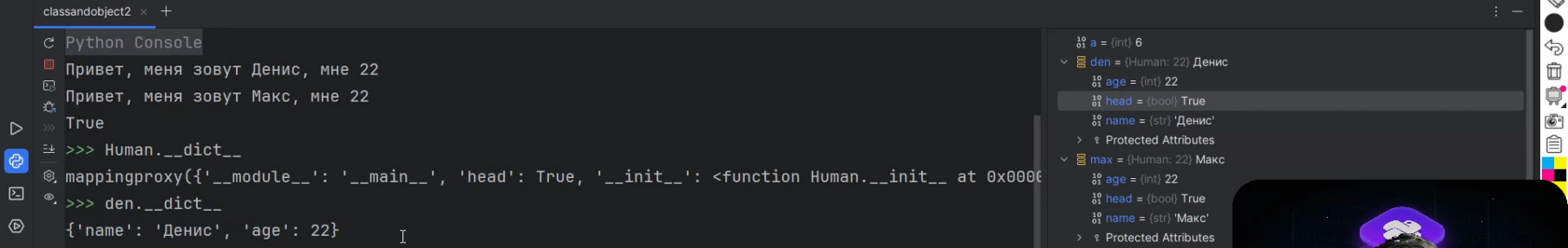


Рис.7

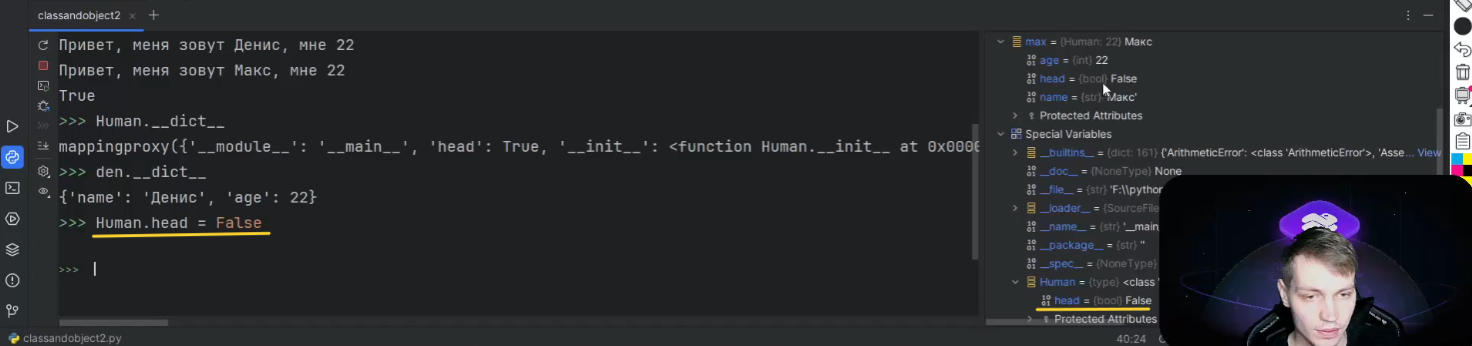


Рис.8

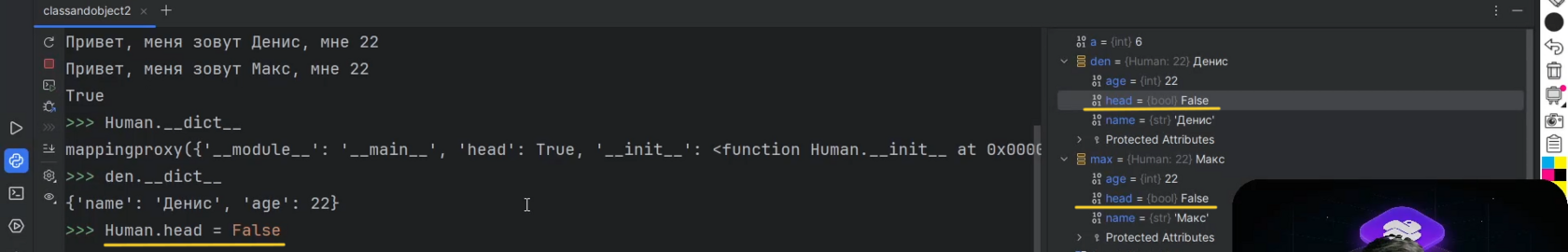


Рис.9

**Представим это в виде схемы.** У нас есть наш класс Human и внутри него есть определённый атрибут head со значением False(рис.10). Когда на основе этого класса создаются объекты, в нашем случае den и max, они наследуются от 1 класса Human. Однако у den и max есть name и age(рис.11). Если вспомнить то, как работают наши функции, когда разбирали локальные, глобальные переменные, то можем вспомнить, что сначала, когда мы обращаемся к чему-то, смотрим в пределах нашего локального пространства имён. Если внутри локального пространства имён у нас не находится соответствующего имени, то начинаем искать в другом месте, в пределах области видимости, в пределах того, куда можем забраться.

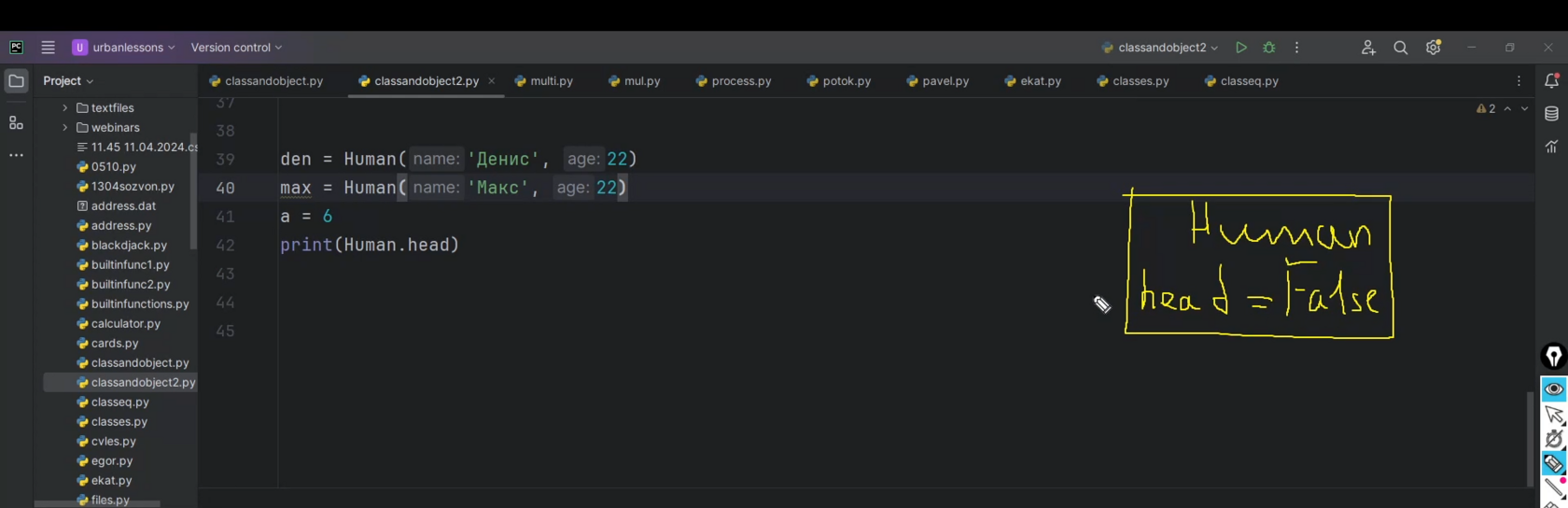


Рис.10

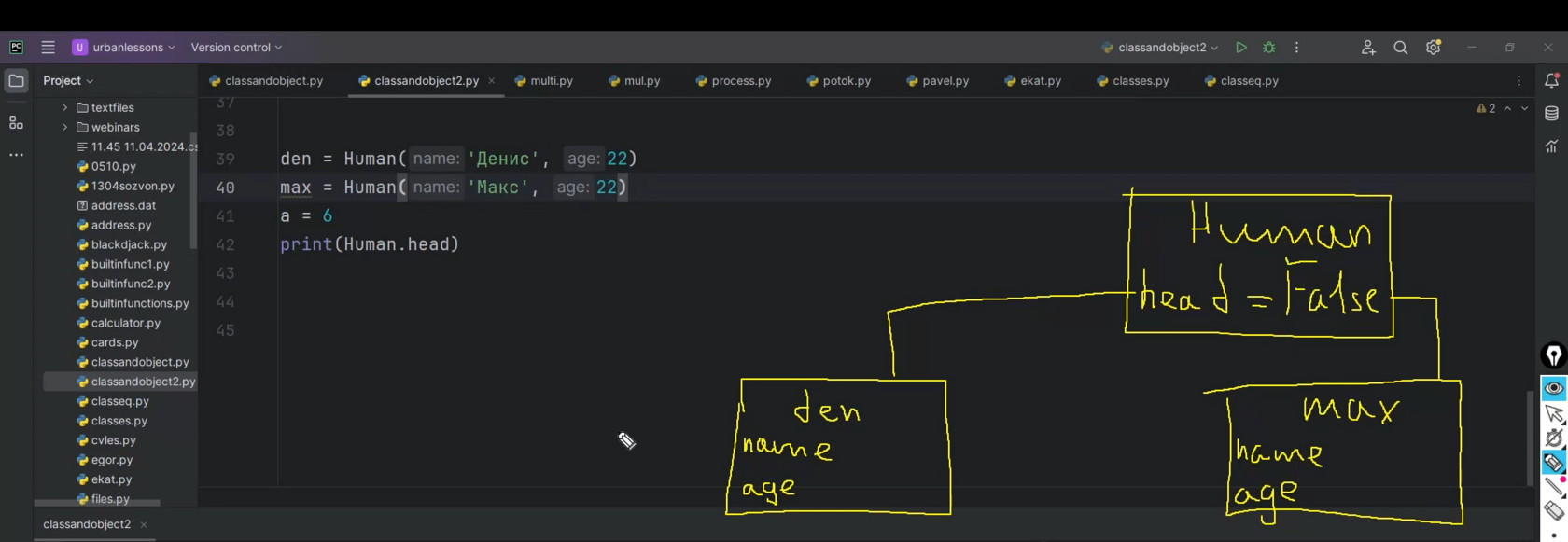


Рис.11

Здесь, когда мы пытаемся обратиться к den и достать оттуда атрибут head, будем получать то значение, которое находится внутри нашего класса, потому что у экземпляров класса есть доступ к атрибутам самого класса. Также будем получать это значение, находящееся внутри класса Human. Соответственно, при изменении значения в нашем классе, у нас это же значение будет меняться внутри каждого экземпляра(объекта), но фактически у нас этого значения нет ни у den, ни у max. Однако если сейчас возьмём, например, dеn, достанем у него атрибут head и изменим его на true(рис.12), то здесь, с точки зрения нашей памяти, у нас появилось у den значение True, а у Макса и у Human осталось False. **Почему так произошло?**

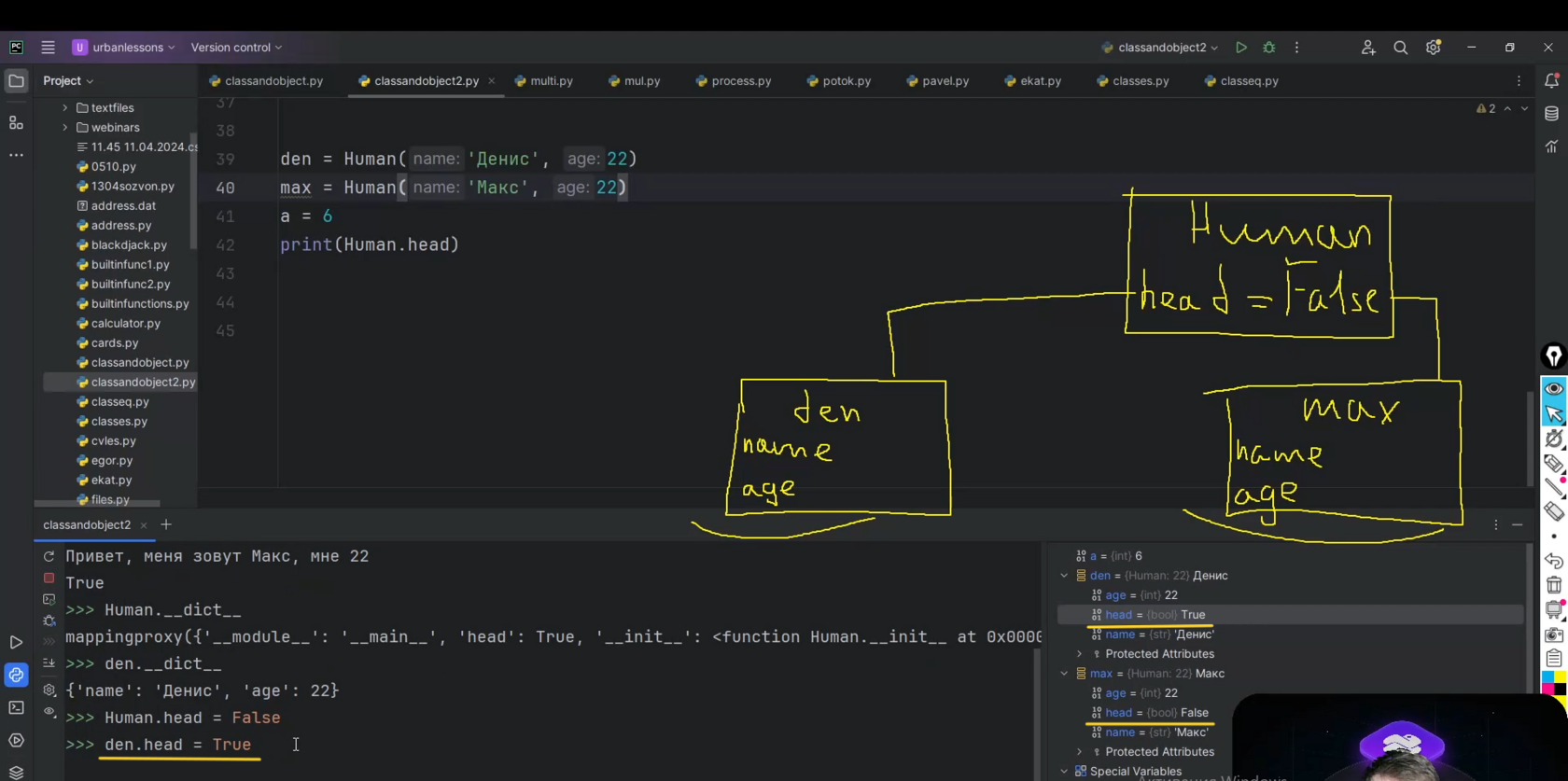


Рис.12

Для того чтобы понять это, давайте сейчас возьмём den и попробуем посмотреть на него как на словарь(рис.13). Получается, что у нас этот атрибут head создался внутри нашего экземпляра(рис.14). Теперь при обращении к этому атрибуту, мы в первую очередь будем смотреть на атрибут, содержащийся внутри экземпляра, а не внутри класса.

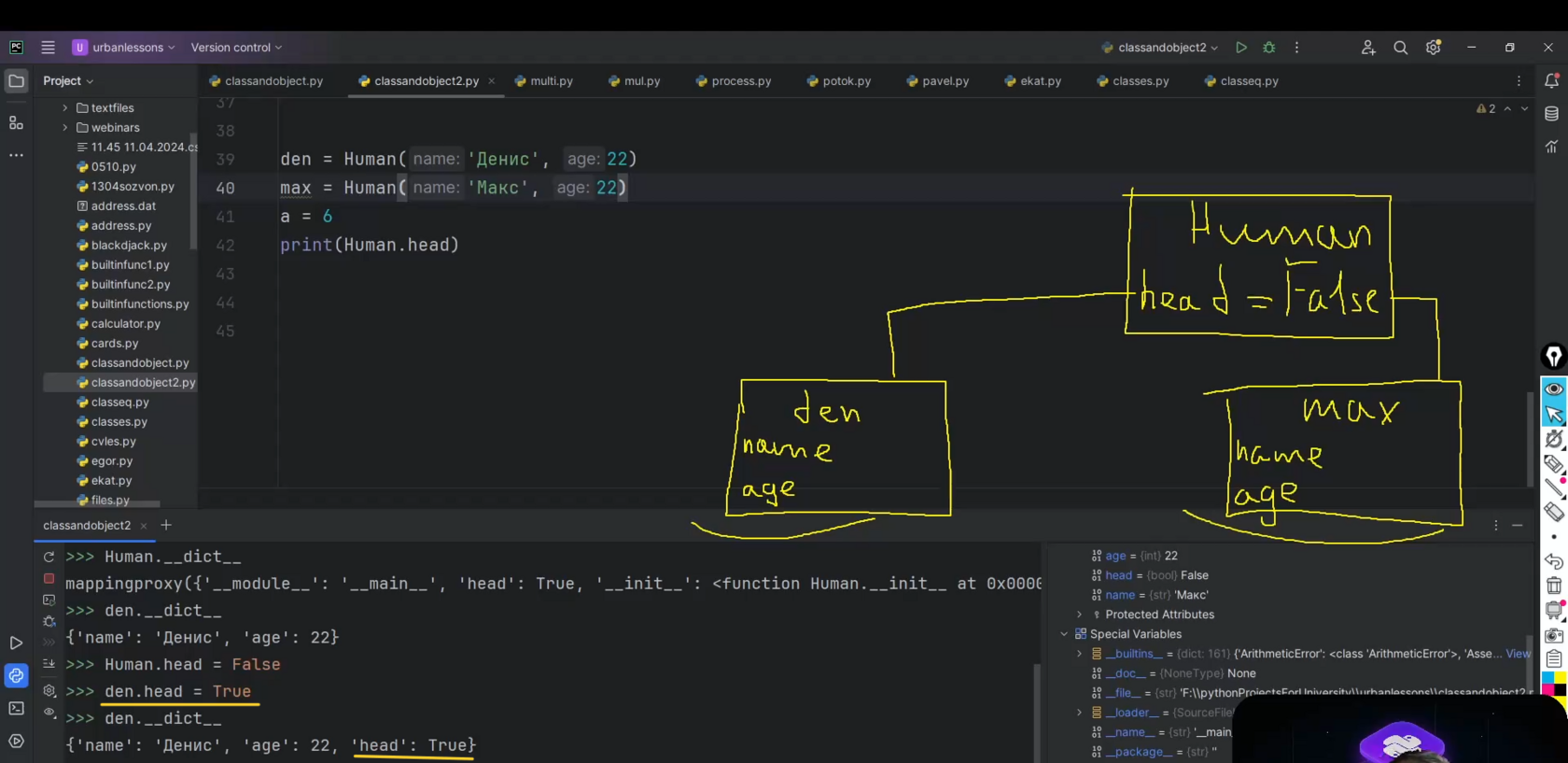


Рис.13

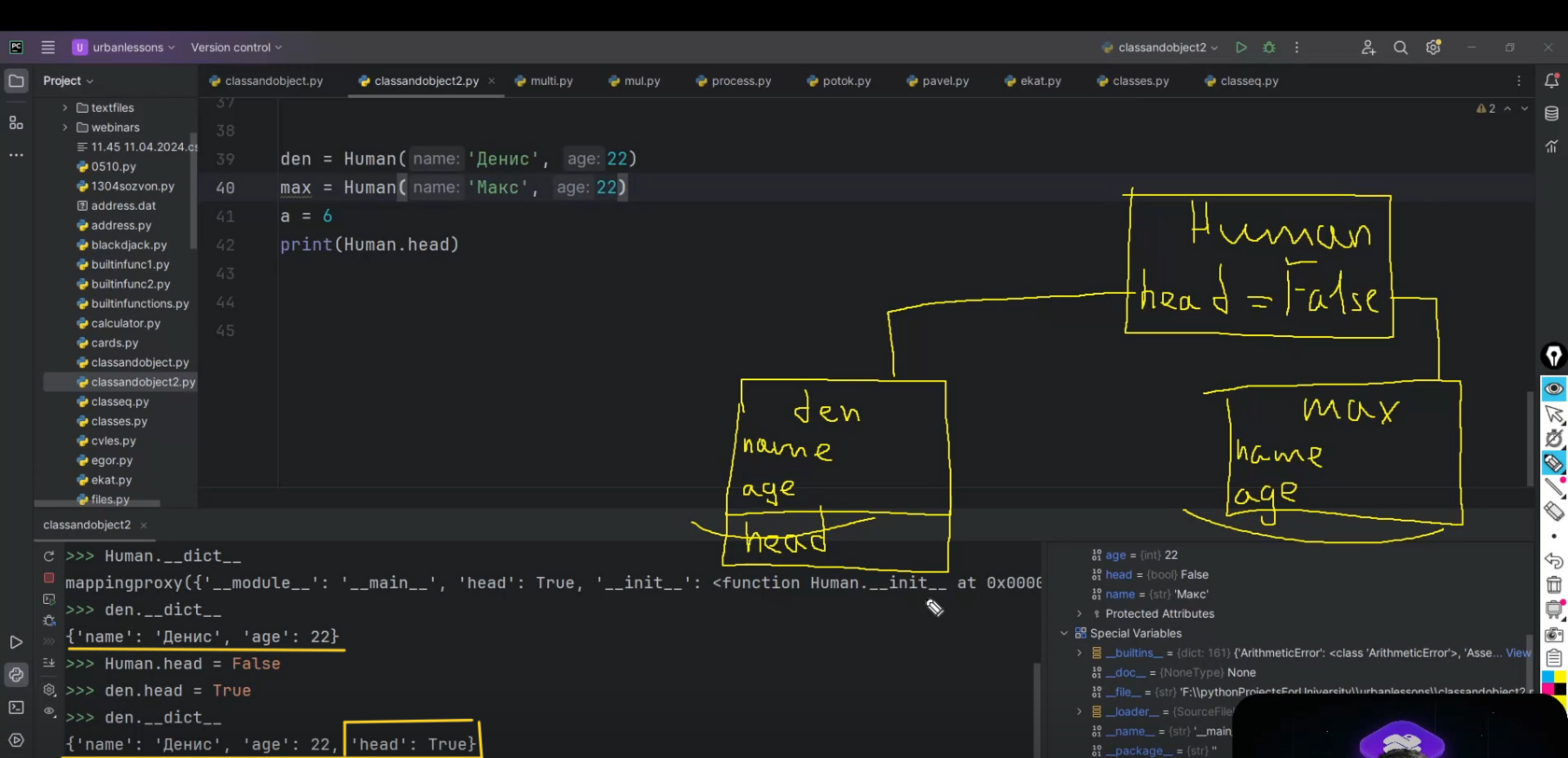


Рис.14

Вот эти различия, важно понимать, важно просто осознавать, как это работает. Как правило, классовые атрибуты используются реже, тем не менее они имеют место быть, потому что для большинства экземпляров все равно есть какие-то атрибуты, которые будут общими. В них будет необходимость при изменении, допустим, этого атрибута внутри класса, чтобы это значение передавалось ко всем экземплярам. То есть, если бы мы не создавали атрибут head внутри den, при изменении этого атрибута внутри класса, изменения бы проходили в каждом экземпляре. Когда нам нужна такая логика, необходимо пользоваться именно атрибутами класса, не атрибутами экземпляра. Это и есть особенность классовых атрибутов и атрибутов экземпляров. Напоминаем, что **экземпляр** — это тот объект, который создаётся на основе класса, то есть переменная, созданная на основе этого класса.