**2023/10/28 00:00|Лекция. Перегрузка операторов.**

## Перегрузка операторов

Продолжаем говорить про классы. На сегодняшнем занятии познакомимся с перегрузкой: что это такое и как это применять.

Ранее мы говорили, что **класс** — это наше представление данных, то есть собственный тип данных. У него есть свои характеристики и свои способности. В Python есть встроенные операторы, которые используются по отношению к другим типам. Например, если брать числа, то используем оператор сложения — плюс, используем оператор вычитания — минус, деление, умножение, сравнение больше, меньше, равно, не равно. Их на самом деле огромное количество. Это поведение определено, заложено по отношению к стандартным типам. **Но можно изменить поведение этих операторов по отношению к нашим объектам. Это и называется перегрузкой.**

Допустим, у нас есть тип данных человек(Human). Мы уже использовали один из магических методов len() для того, чтобы определить размер объекта. Представим, что необходимо сравнить больше один человек, чем другой или нет. Как это можно сделать? Например, можно воспользоваться специальным методом lt()(рис.1). Он будет принимать параметры self и other — это те методы, которые позволяют нам перегружать стандартные операторы. Зачастую в операциях у нас участвуют 2 объекта/2 элемента. Например, сравнивать можно одного человека с другим, одно число с другим, то есть в операции сравнения, у нас участвует двое. При вычитании, сложении, умножении у нас точно также 2 участника в этой операции и здесь достаточно просто представить, что self — это один участник операции, второй — это other. Как это выглядит?

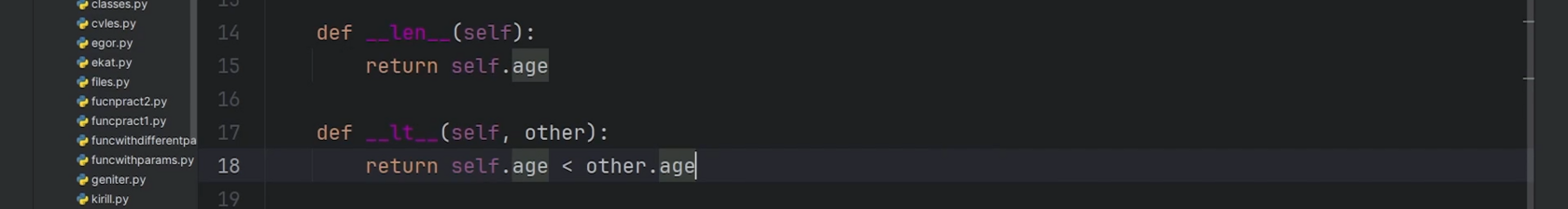


Рис.1

**\_\_Lt\_\_()**— это можно сказать, что Lower than, то есть меньше чем. Данный метод будет переопределять поведение стандартного оператора меньше, то есть перегружать его. Чтобы его перегрузить, давайте будем возвращать self.age меньше, чем other.age. Иначе говоря, берём 1 участника и 2 участника и будем возвращать True, если возраст первого участника меньше, чем возраст второго участника нашей операции.

**Как это будет выглядеть?**Например, у нас есть объект den и объект max. Дену 22 года, Максу 22, но у него было день рождения и ему стало 23. Дальше возьмём Дена и сравним с Максом. Запустим и увидим True(рис.2), потому что Дену в действительности сейчас меньше лет, чем Максу. Если поставим это перед днём рождения, то получим False(рис.3), потому что их возраст, по сути, равен. Мы перегрузили оператор сравнения меньше.

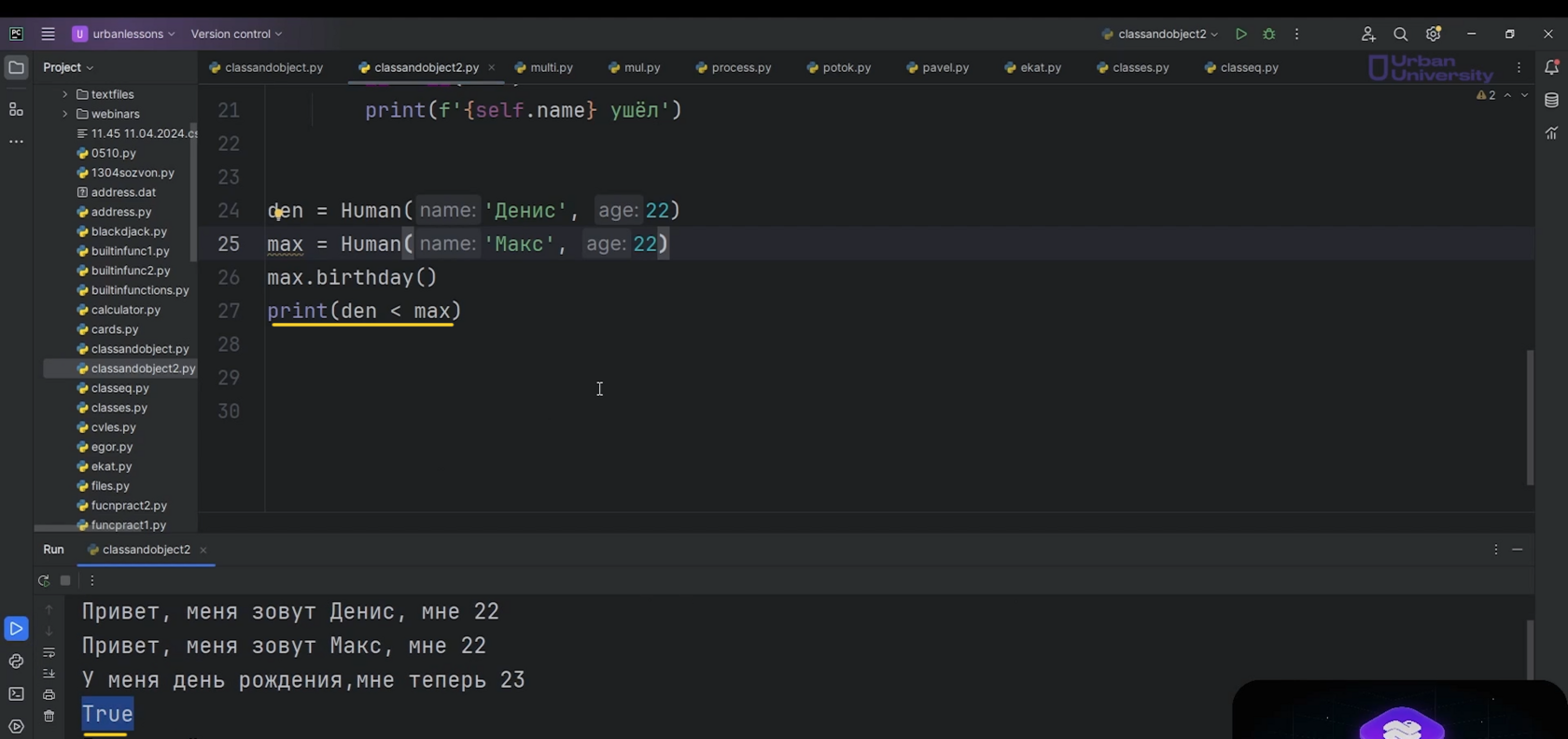


Рис.2



Рис.3

То же самое можно сделать для оператора больше. **\_\_Gt\_\_()** — Greater than, больше чем, когда self.age больше, чем other.age(рис.4). Тут если скажем, что Ден больше Макса, то увидим False(рис.5). При этом, если поменять их местами, скажем Макс больше Дена (это уже после дня рождения Макса), то увидим значение True(рис.6). Мы перегрузили оператор сравнения.

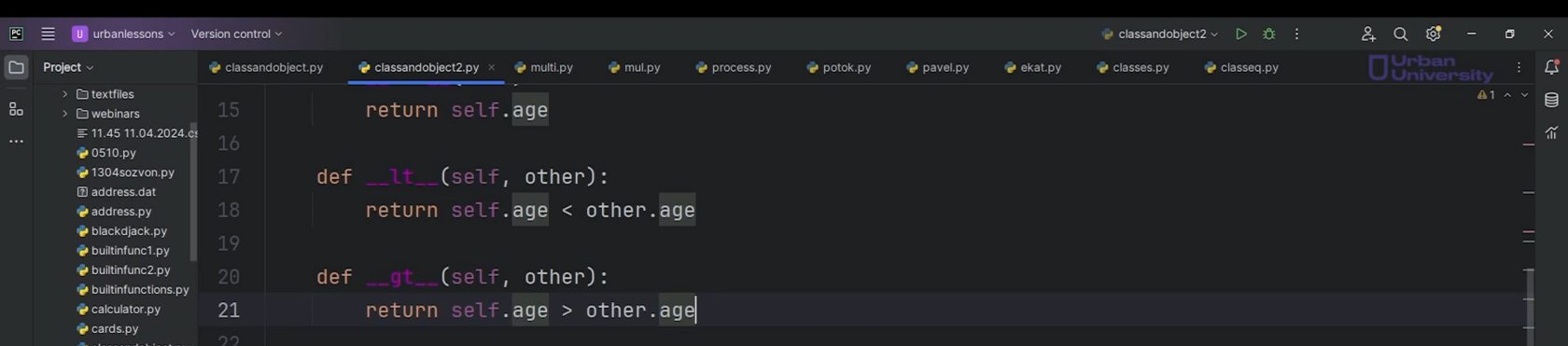


Рис.4

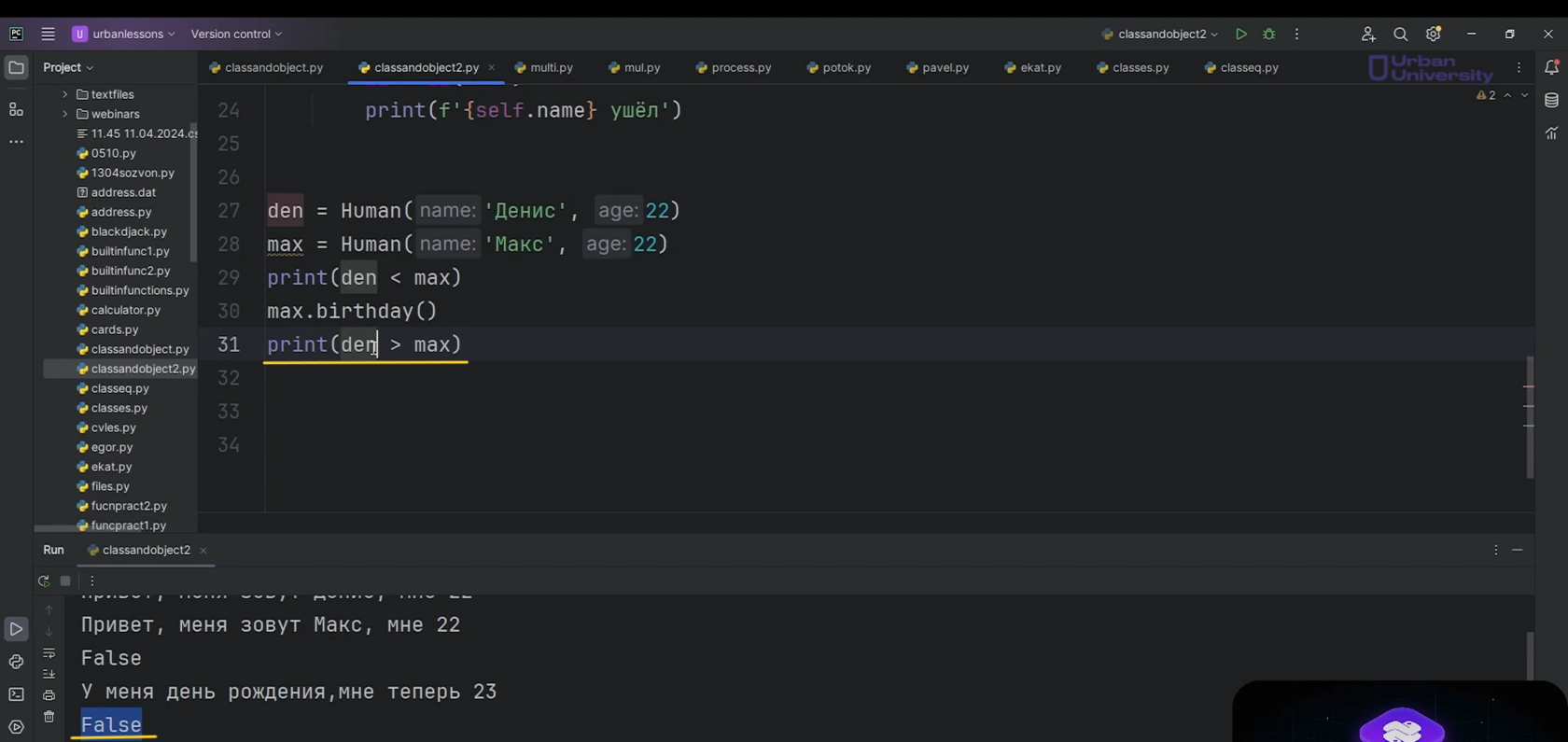


Рис.5

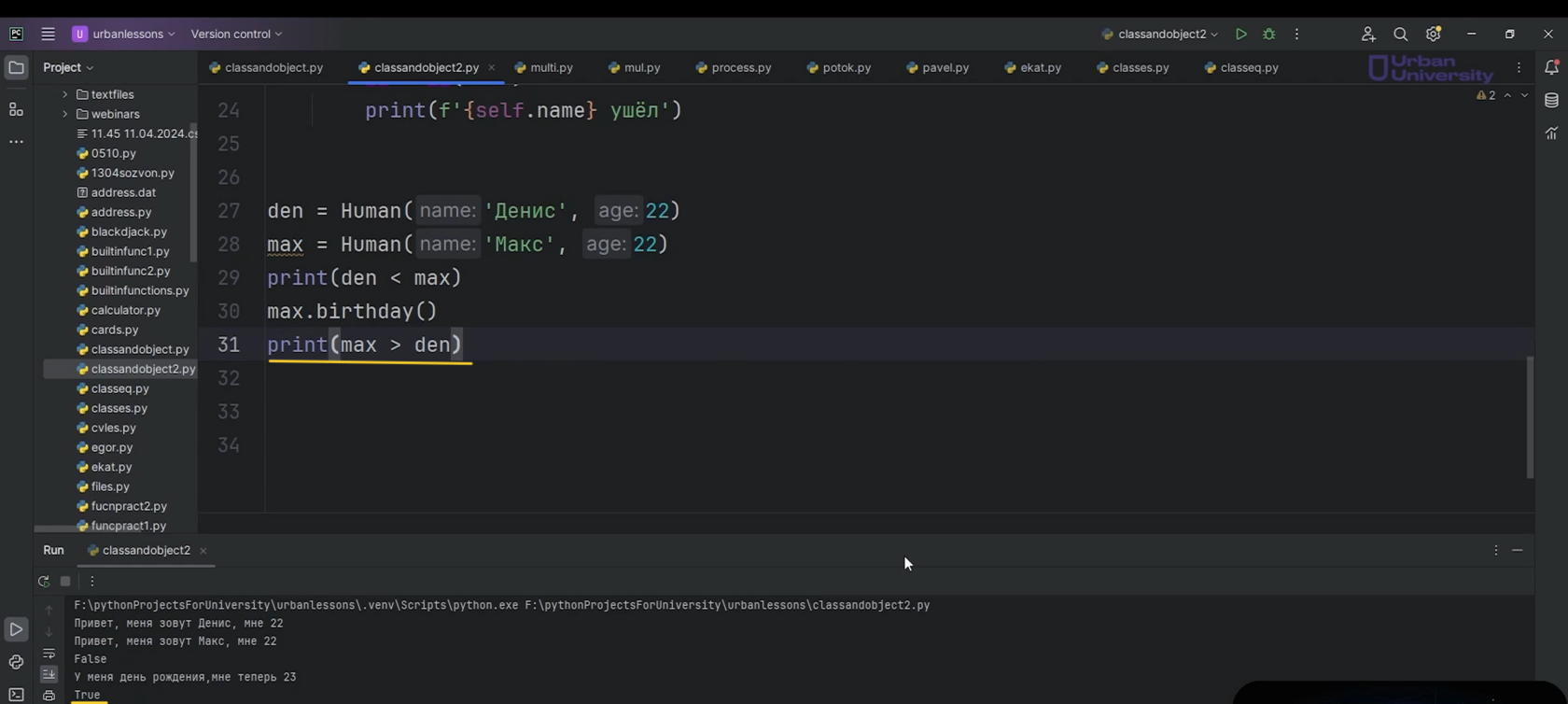


Рис.6

То же самое можно делать для оператора равенство**\_\_eq\_\_()**(рис.7).

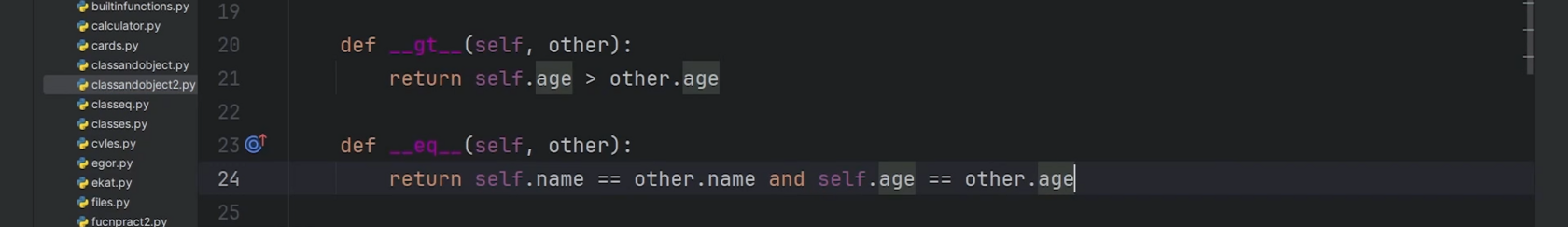


Рис.7

Захотели сравнить Дена с Максом и сказать, что они абсолютно одинаковые. Для этого возьмем сначала имена двух участников операции и возраст. О двух людях можно сказать, что они у нас одинаковые, если их имена и их возраст равен. При попытке сравнить max и den, увидим False, так как имена и возраста двух участников не равны. Даже если сравнить их раньше, все равно получим False(рис.8), потому что у них разные имена. Однако можно взять max, взять его атрибут name, изменить на Денис, в таком случае здесь уже будет True(рис.9). У них теперь одинаковый атрибут name и одинаковый атрибут age. Получается, эти 2 объекта равны.

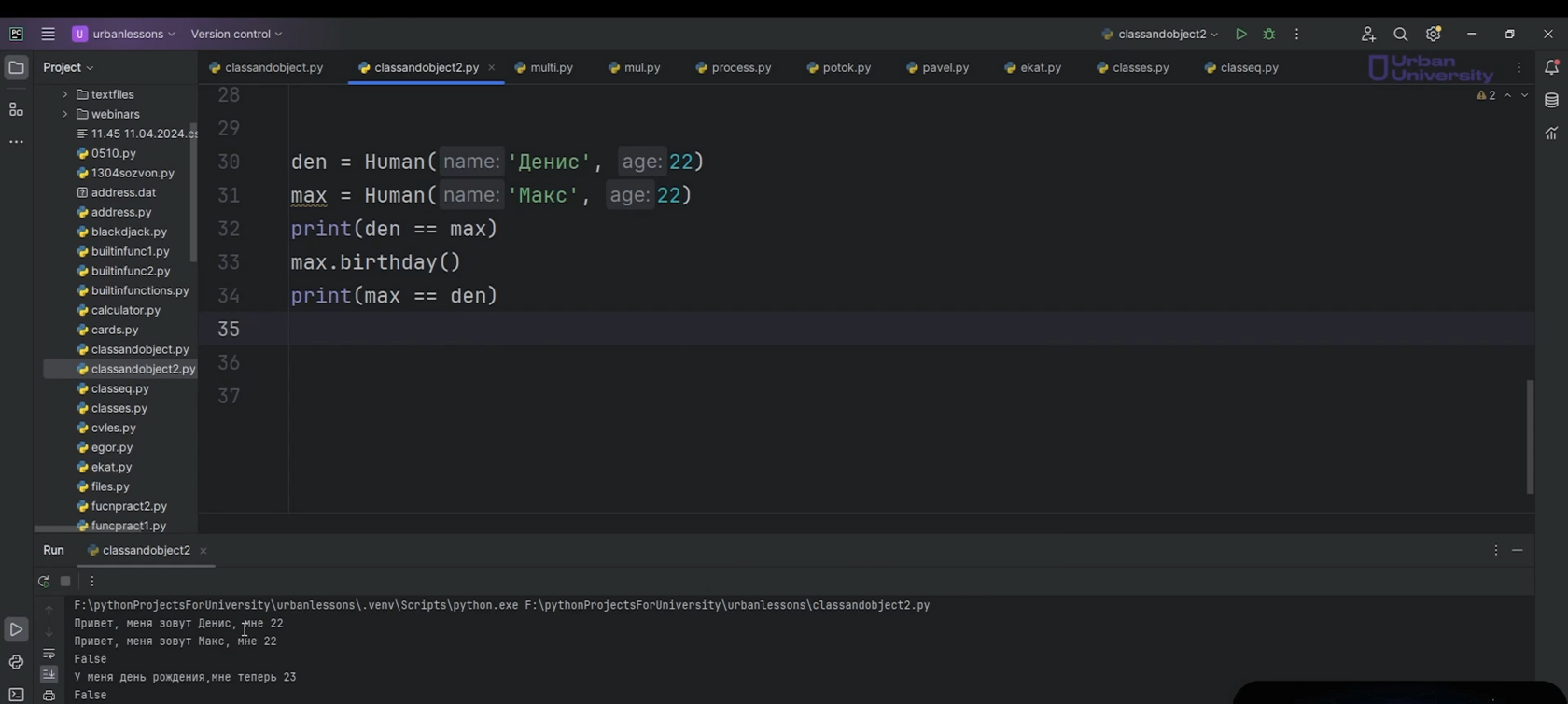


Рис.8

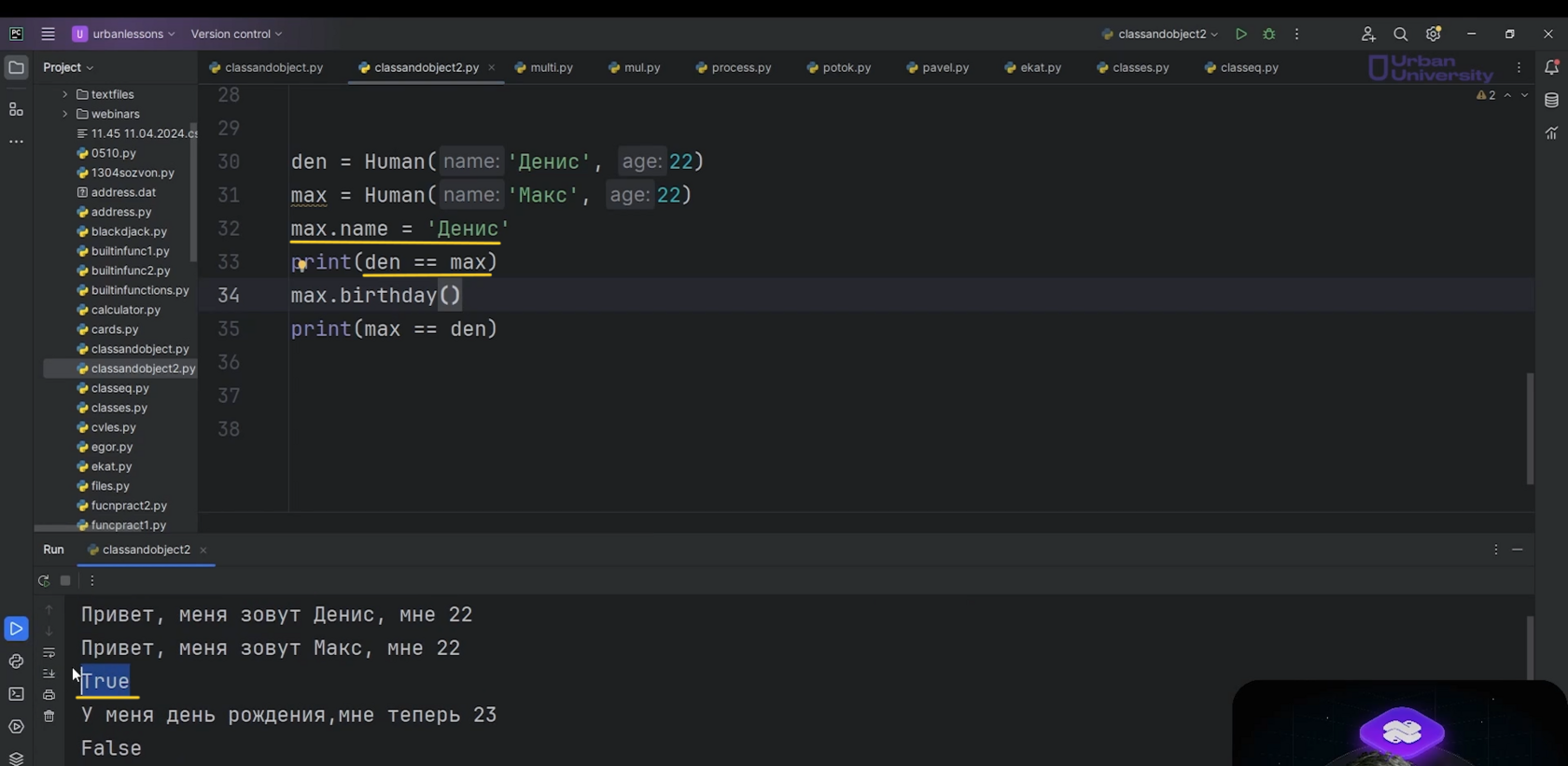


Рис.9

Также мы можем говорить, что у нас будет происходить в случае, если захотим проверить истинно или неистинно. В таком случае можем воспользоваться методом **\_\_bool\_\_()**(рис.10). Здесь будем возвращать результат работы функции bool() по отношению к нашему возрасту это неявное преобразование, у нас любое число отличное от нуля будет давать True. Если у Дена или у Макса возраст не равен нулю, то можем применять это как истину для этих объектов. Это может быть разный контекст. Все зависит от того, как вам необходимо это реализовать. Ну, например, если den существует, то скажем ему рассказать о себе, при запуске увидим “Привет, меня зовут Денис” вызывается 2 раза(рис.11), потому что возраст у Дениса 22. Если нам нужно получить какую-то логическую интерпретацию нашего объекта, можно переопределить метод bool(). Однако это относится скорее не к перегрузке, а к магическим методам в целом. К перегрузке относятся больше стандартные операторы, в которых участвуют два объекта при выполнении операции. Это тоже очень полезная штука.

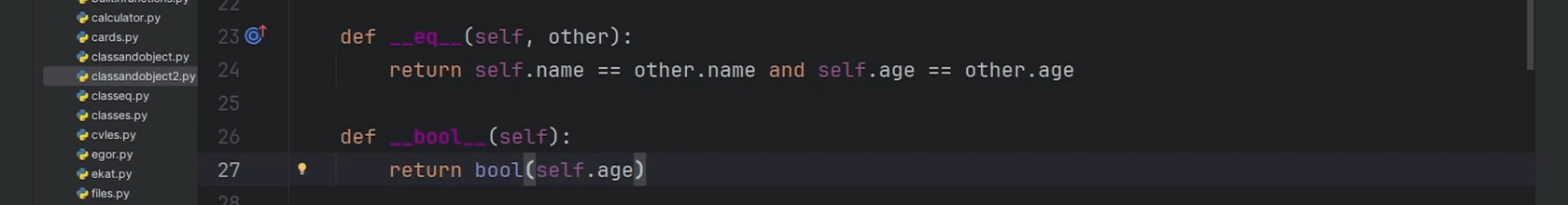


Рис.10

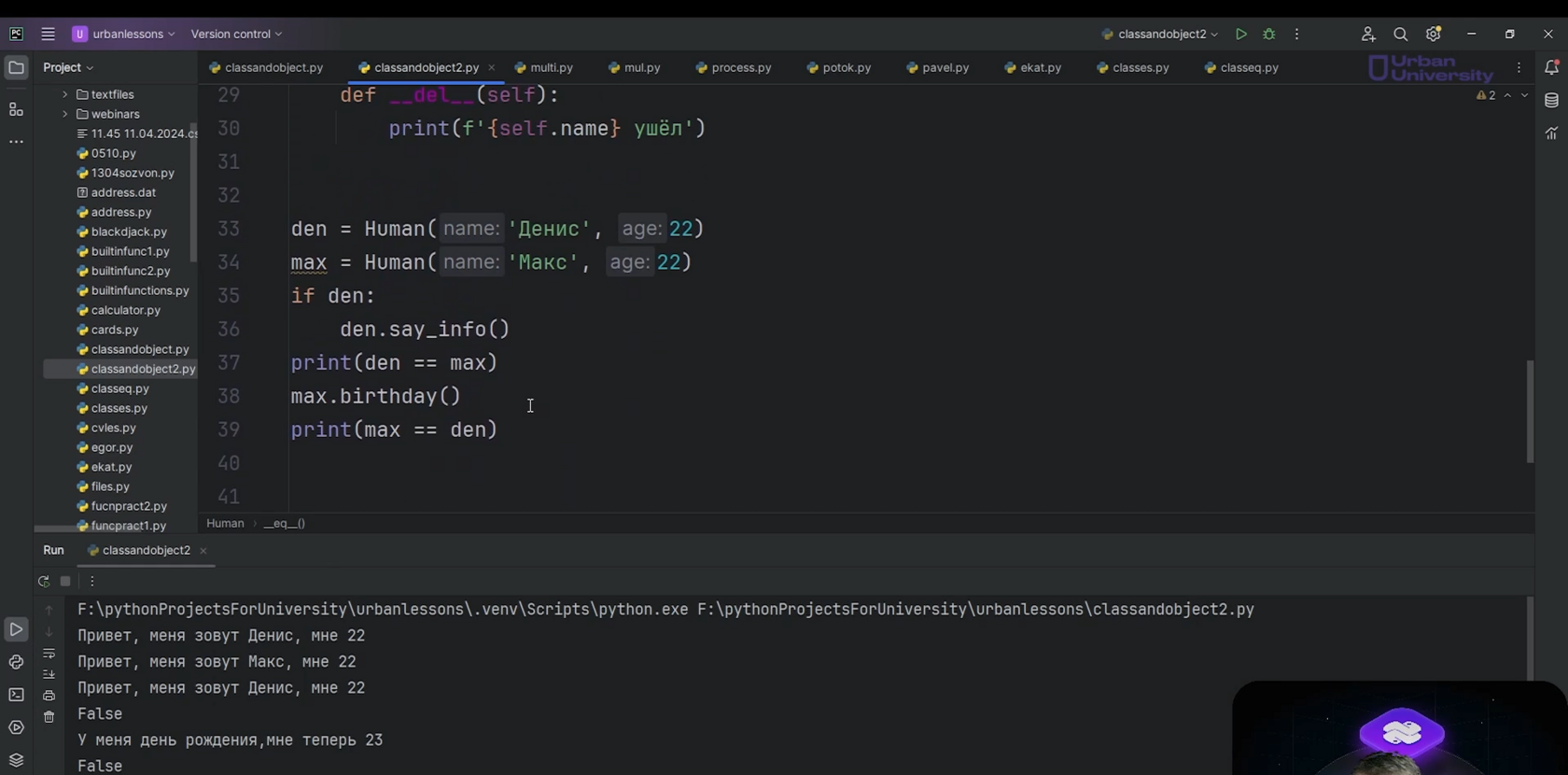


Рис.11

Плюс ко всему, если выводим какое-то число, допустим создадим переменную ‘a’, то при обращении к этой переменной, получим ее число(рис.12). Если обратимся к объекту den, получим непонятное описание “Human object и адрес памяти”(рис.13). Выглядит не очень презентабельно. Однако это можно исправить. Для этого давайте создадим метод **\_\_str\_\_()**(рис.14). Он будет определять строковое представление нашего объекта и здесь будем возвращать имя.

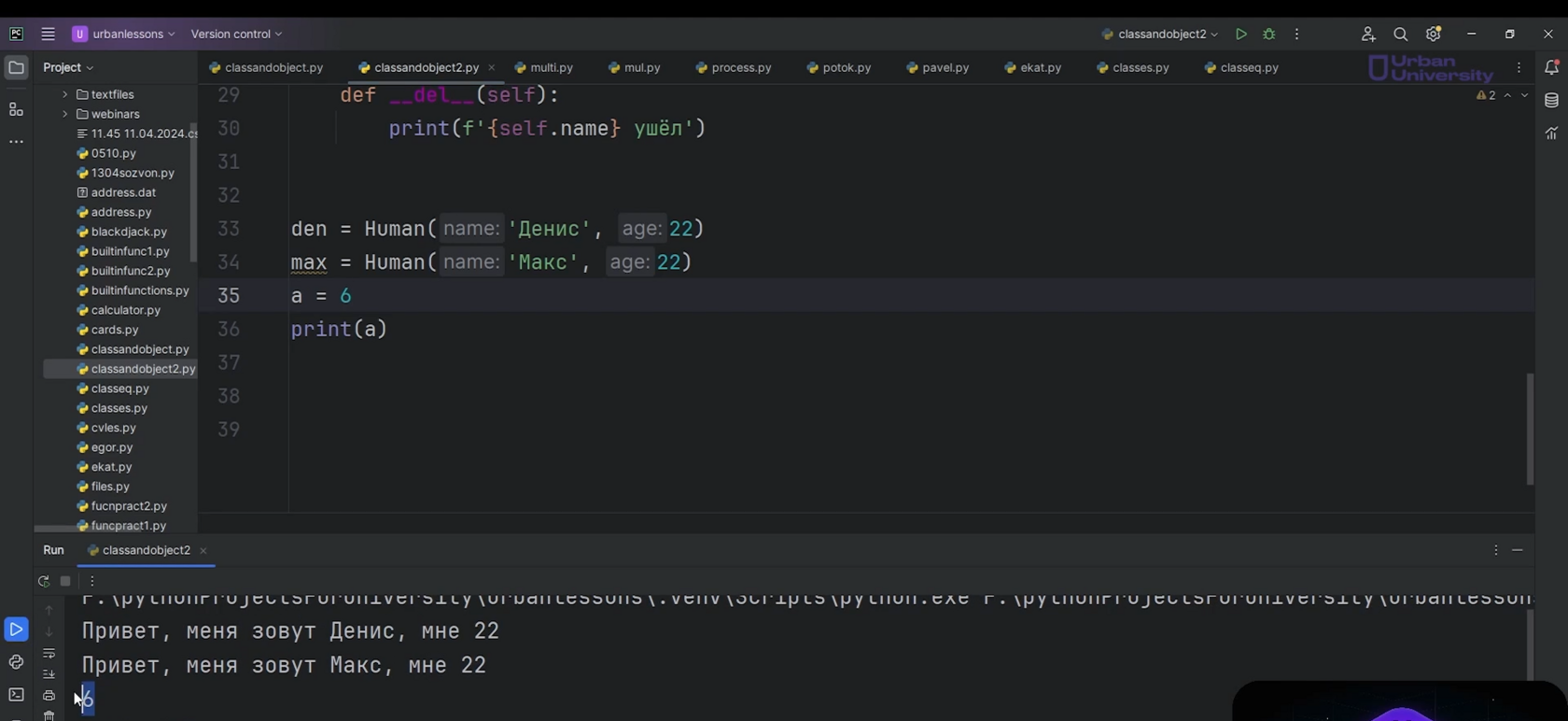


Рис.12

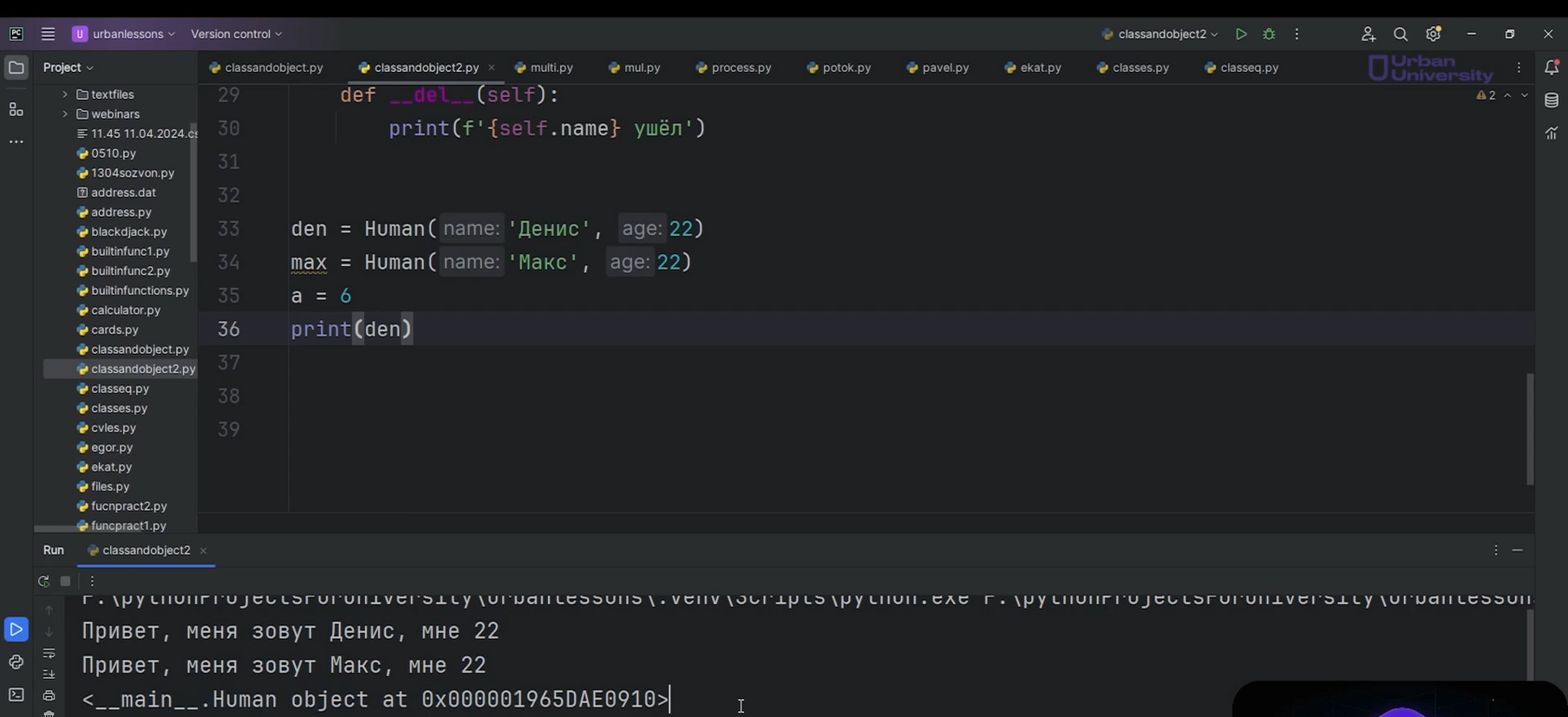


Рис.13

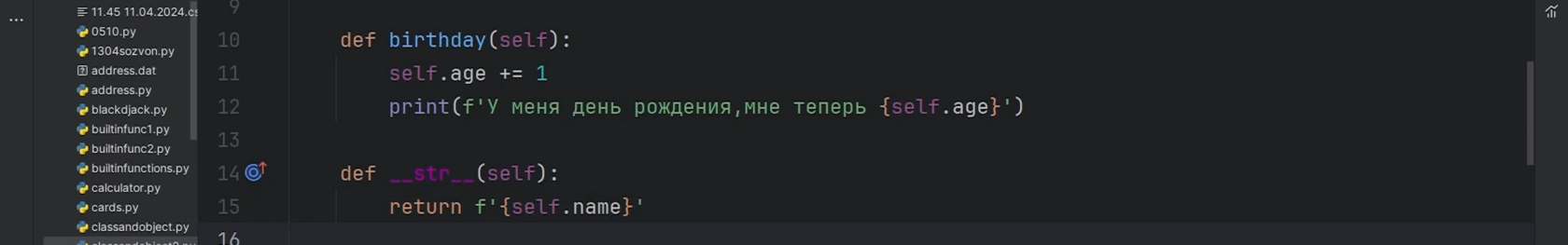


Рис.14

**Как это теперь будет выглядеть?**При выводе den, получаем Денис(рис.15), выводим max, получаем Макс(рис.16).

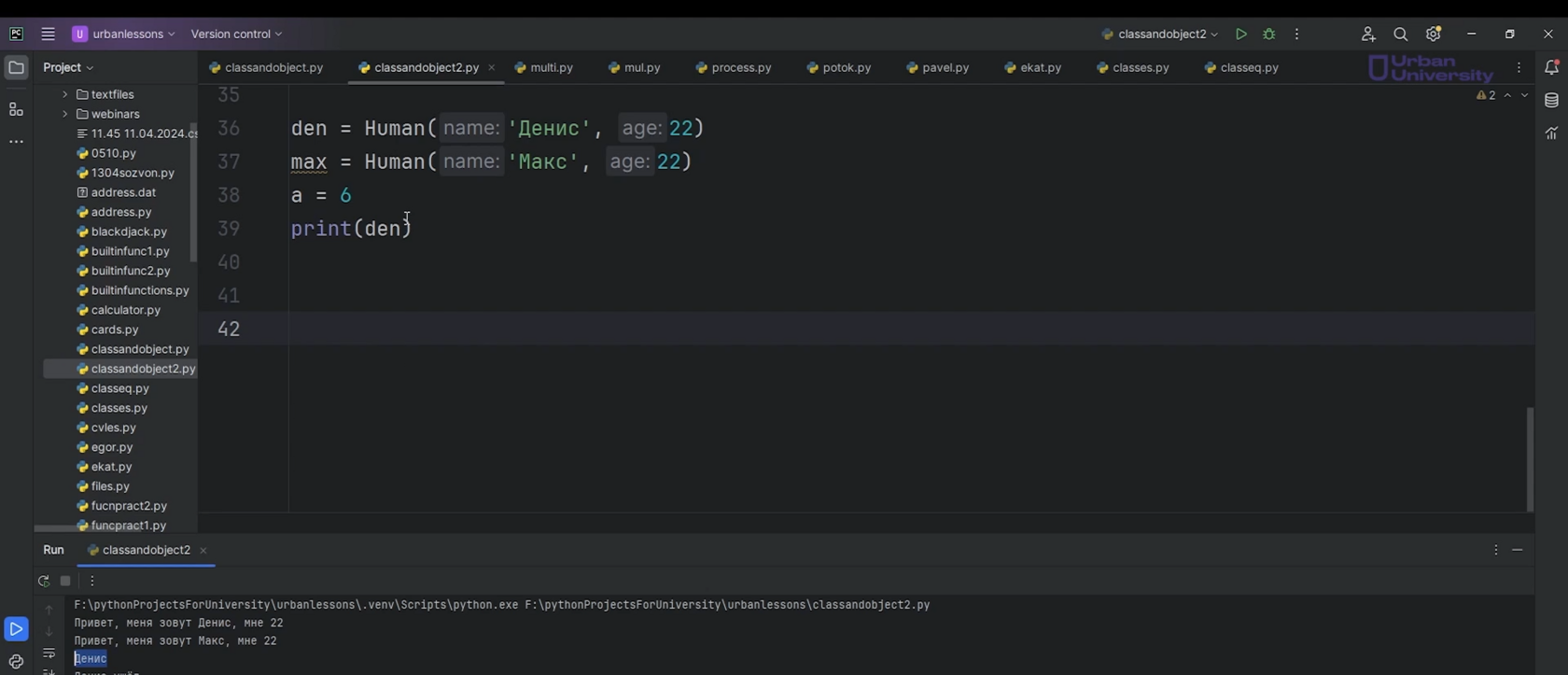


Рис.15

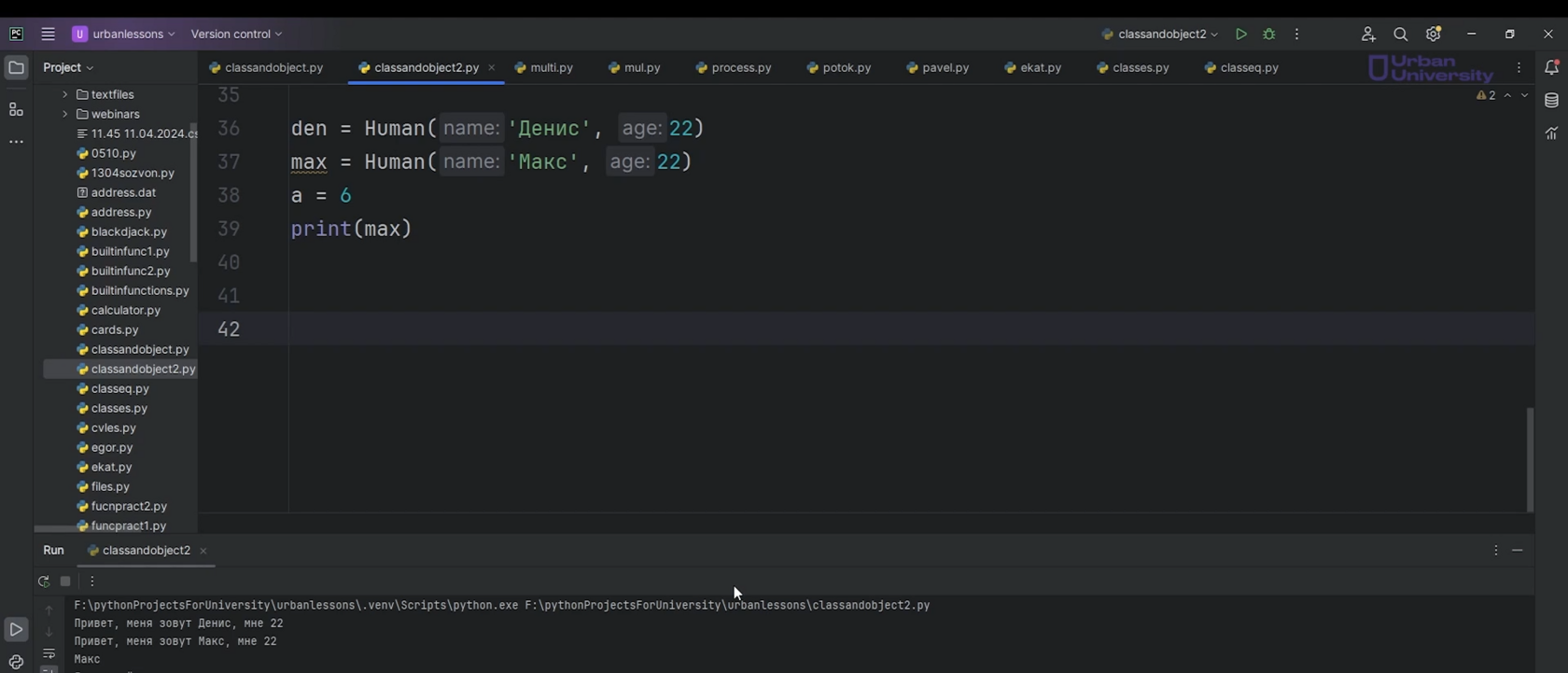


Рис.16

**Классы позволяют очень гибко работать с логикой в нашей программе.** Можно определять, как они будут вести себя, объекты нашего собственного типа данных, при сложении, вычитании, делении, умножении при переводе их в число, при попытке найти их модуль, при попытке сравнить или проверить является ли этот объект истинным или ложным. Вы никогда заранее не можете знать, что вам пригодится при написании вашей программы.

К сожалению, не все описано уже в языке Python. Зачастую приходится использовать собственную сущность, создавать собственные представления. Изначально их тоже не было в Python, но библиотеки используют собственные сущности, определения своих типов данных. Они завязаны на использовании в какой-то мере каких-то стандартных элементов нашего языка. Однако все-таки они тоже являются собственным созданным типом данных.

Никто не может заранее сказать, какие технологии вам будет необходимо, скажем так, применять при реализации вашей какой-то идеи. Поэтому классы достаточно гибкий способ для того, чтобы организовать собственную структуру данных и очень доступно можно работать с этой структурой, определять поведение, как она себя будет вести в программе. Делать это таким образом, что если кто-то будет использовать вашу структуру данных, он тоже сможет взять и ознакомиться с содержимым. Человек, который смотрит, в принципе, поймёт, как работает ваш собственный тип данных и что с ним можно делать. Поэтому классы очень важны и нужно просто тренироваться.