**2023/10/29 00:00|Лекция. Класс object и метод \_\_new\_\_.**

## Класс object и метод \_\_new\_\_

На этом занятии познакомимся с классом Object и новым для нас магическим методом new().

На самом деле каждый класс, который привыкли использовать, он наследует характеристики и методы из класса Object. Чтобы в этом убедиться, давайте возьмём, к примеру, класс int (целые числа). С ним мы практически на регулярной основе работаем и обратимся к атрибуту **mro**(рис.1). **Он позволяет посмотреть на цепочку наследования для этого класса.** При запуске увидим, что в цепочке наследования присутствует некий класс Object, то есть даже целое число наследуется от какого-то уже существующего класса.

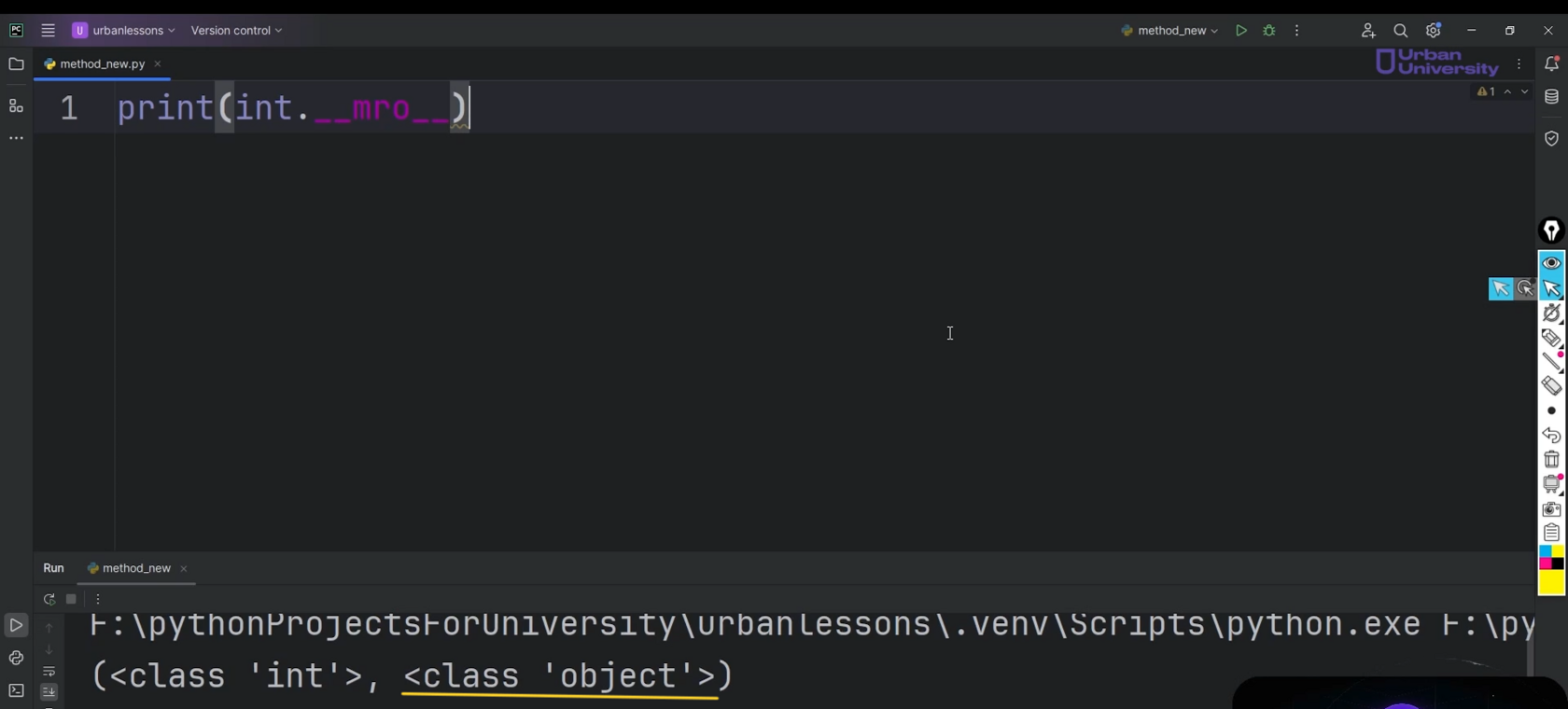


Рис.1

Давайте на него взглянем для этого в print(), выведем object(рис.2) и зажмём ctrl, можно даже не в print(), просто нам нужно имя, через которое можно перейти к его первоисточнику. Тут видим, что **класс Object является базовым классом всей иерархии классов**(рис.3), а внутри можно увидеть какие-то знакомые, какие-то нет методы, например, привычный нам init()(рис.4). На самом деле**init(), срабатывает после того, как создается объект класса.** Именно он позволяет нам присваивать какие-то уникальные характеристики для каждого объекта. А вот **метод new() срабатывает перед созданием объекта класса.**

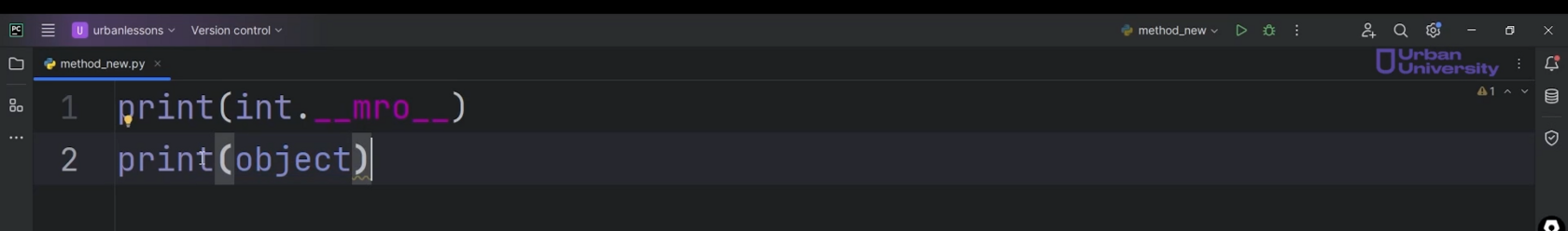


Рис.2

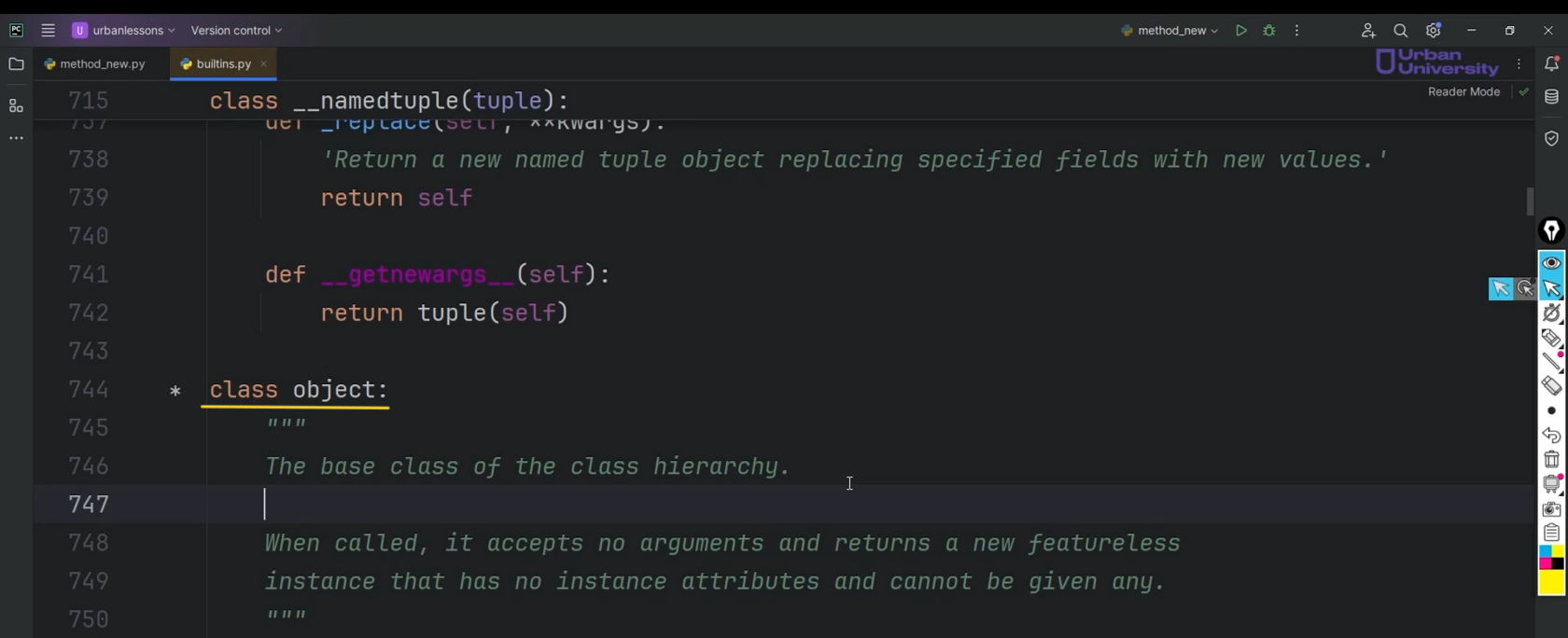


Рис.3

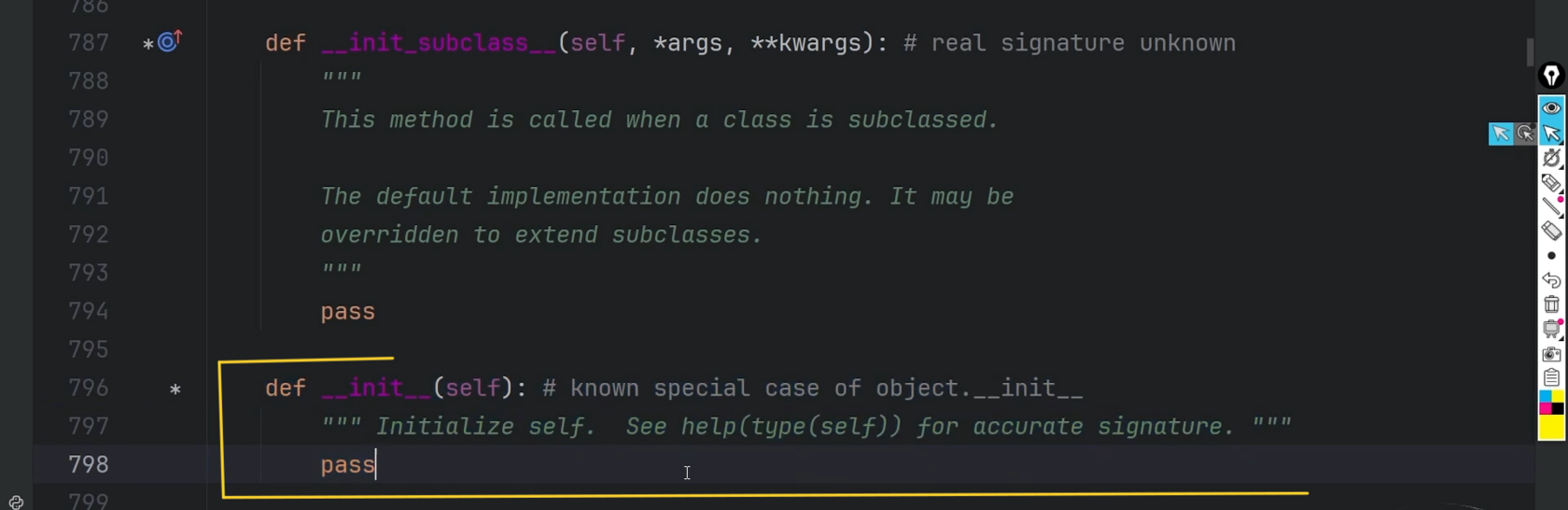


Рис.4

**Давайте посмотрим, как это работает.**Возьмём, к примеру, создадим пользовательский класс User и сделаем у него метод init(). Он будет просто выводить какую-то строку по типу “Я в ините”(рис.5). Получается, сейчас при создании объекта класса User будем видеть строку print(‘Я в ините’). Но давайте взглянем на цепочку наследования для нашего класса User — это пользовательский класс, мы его создавали сами. Запускаем и видим, что и здесь, в цепочке наследования есть класс Object(рис.6).

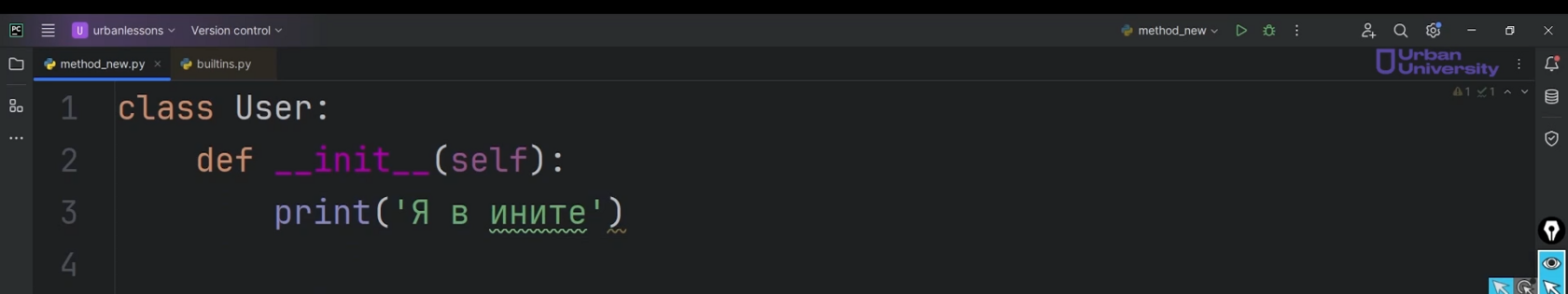


Рис.5

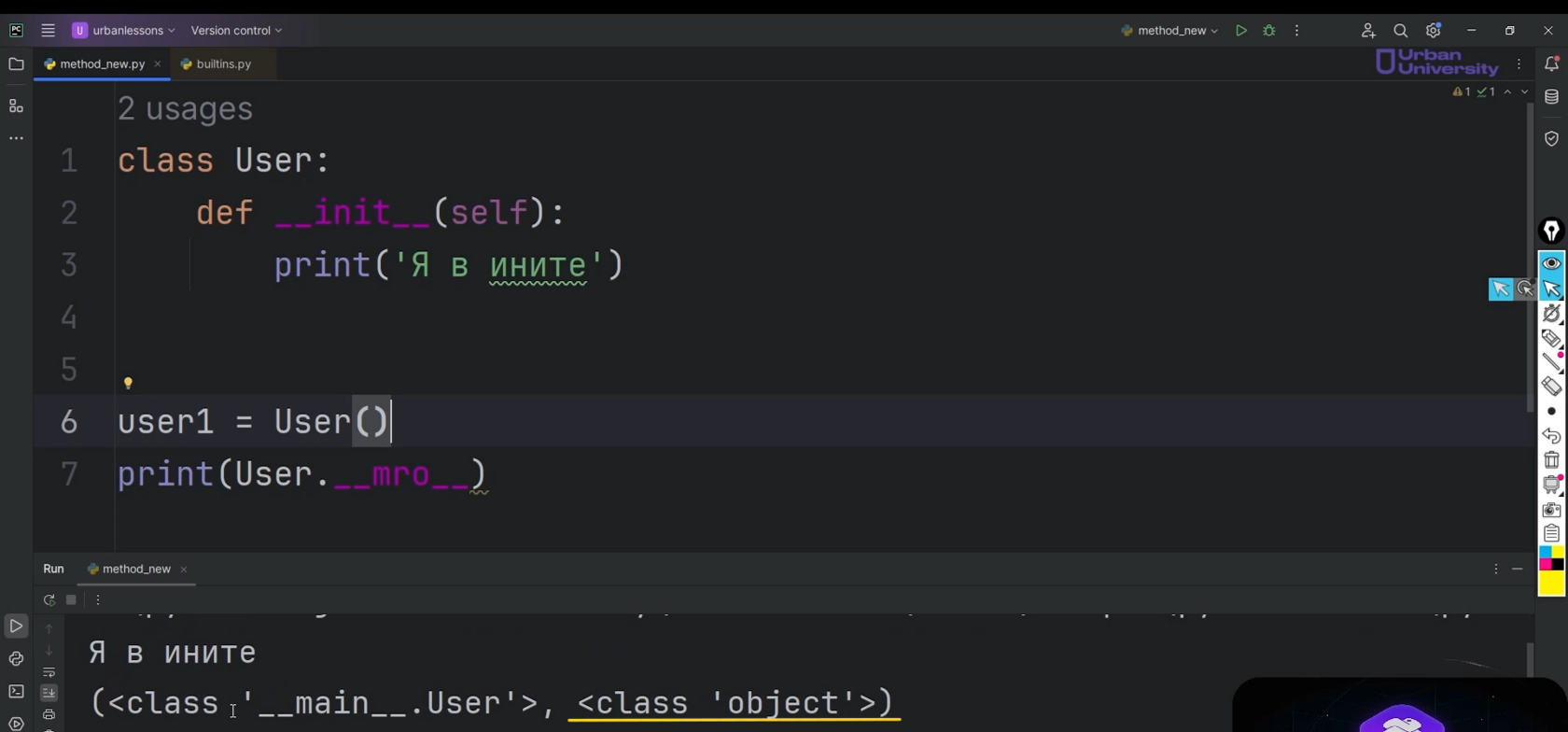


Рис.6

На самом деле каждый объект перенимает вот эти методы, такие как init(), new() и ряд ещё остальных, которые там присутствуют, находящиеся внутри класса Object. По сути, он уже есть внутри нашего класса User.

Для того чтобы его переопределить создадим метод new() и выведем строку “Я в нью”. Обратите внимание, что метод new() принимает cls в качестве первого параметра(рис.7), а не self. На предыдущих занятиях выяснили, что **self является указателем на объект класса, а вот этот cls будет являться указателем на класс.**Получается, сейчас мы переопределили метод new().



Рис.7

Запускаем нашу программу и видим, что вывелась только одна строка и init() не сработал(рис.8). Если вывести наш объект User1, то увидим None(рис.9), то есть наш объект не был создан, а раз объект не был создан, то и init() не сработал. Мы не увидели строчку print(‘Я в ините’). Это все происходит из-за того, что **метод new() должен возвращать ссылку на класс**, а здесь наш метод new() не возвращает ничего, поэтому объект не может создаться.

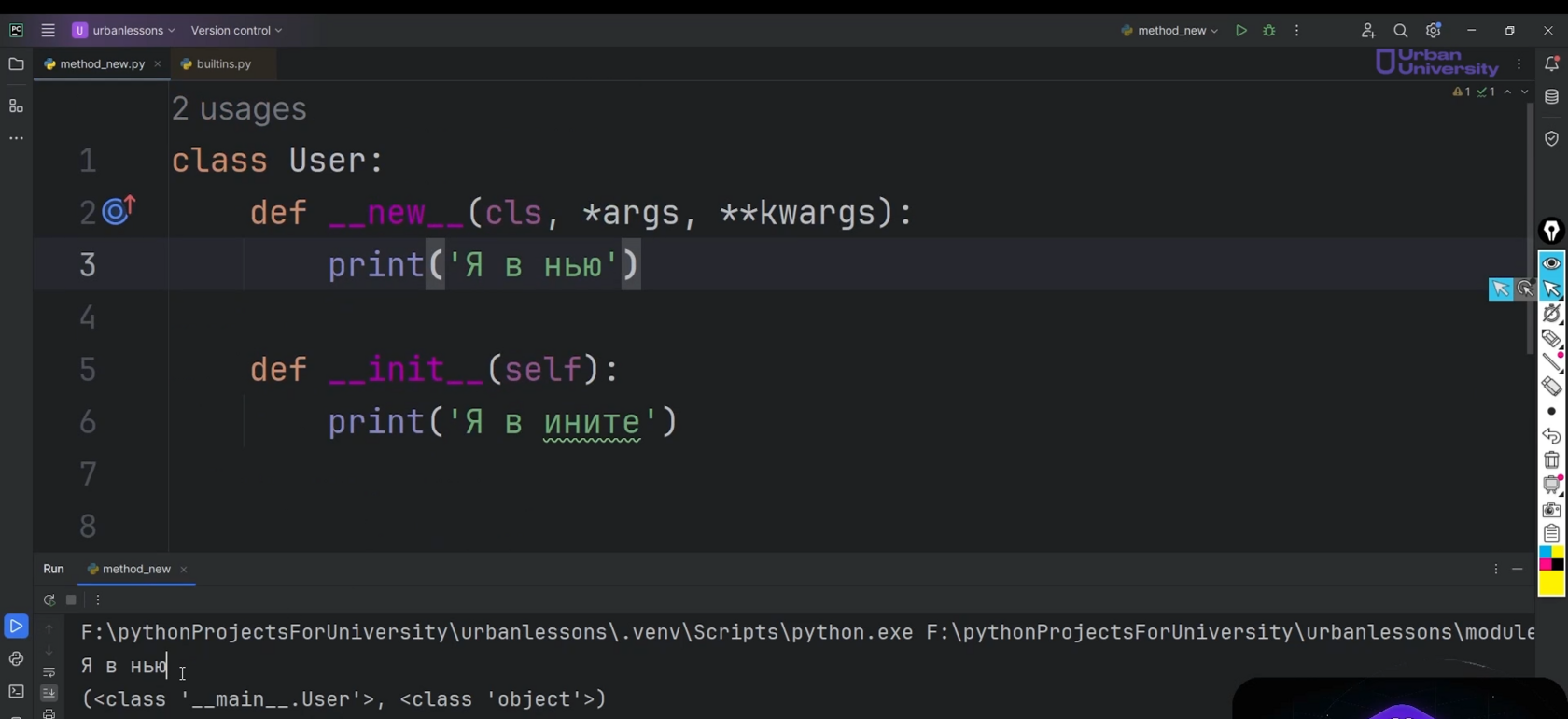


Рис.8



Рис.9

#### В каких же случаях может пригодиться переопределение метода?

На самом деле ситуации бывают разные. Одна из популярных ситуаций — это, например, такой **паттерн Singleton**, когда нам нужно, чтобы в нашей программе объект класса создавался единожды, чтобы избежать повторной его инициализации.  
Посмотрим реализацию Singleton в учебном варианте. Например, вы создаёте уникального пользователя. Нам нужно вернуть (return) ссылку на этот класс User. Чтобы это сделать воспользуемся методом Super() и вызовем метод new(), передав туда ссылку на класс (cls)(рис.10) и получится так, что из класса object готовый метод применим в нашем собственном.

Super() - это функция, которая позволяет вызывать методы родительского класса в дочернем классе. Она используется, когда нужно вызвать методы родительского класса в дочернем классе, чтобы избежать дублирования кода и улучшить его читаемость.



Рис.10

Получается сейчас, если запустим код, то видим, что наш объект создался успешно, и увидим “Я в нью” и “Я в ините”(рис.11). Однако если будем создавать повторно наших пользователей, видим False(рис.12), то есть они ведут в абсолютно разные места в памяти. В этом можно даже удостовериться, воспользовавшись функцией id(), сюда передать нашего User1 и вызвать id() для User2. Видим абсолютно разные результаты(рис.13).



Рис.11

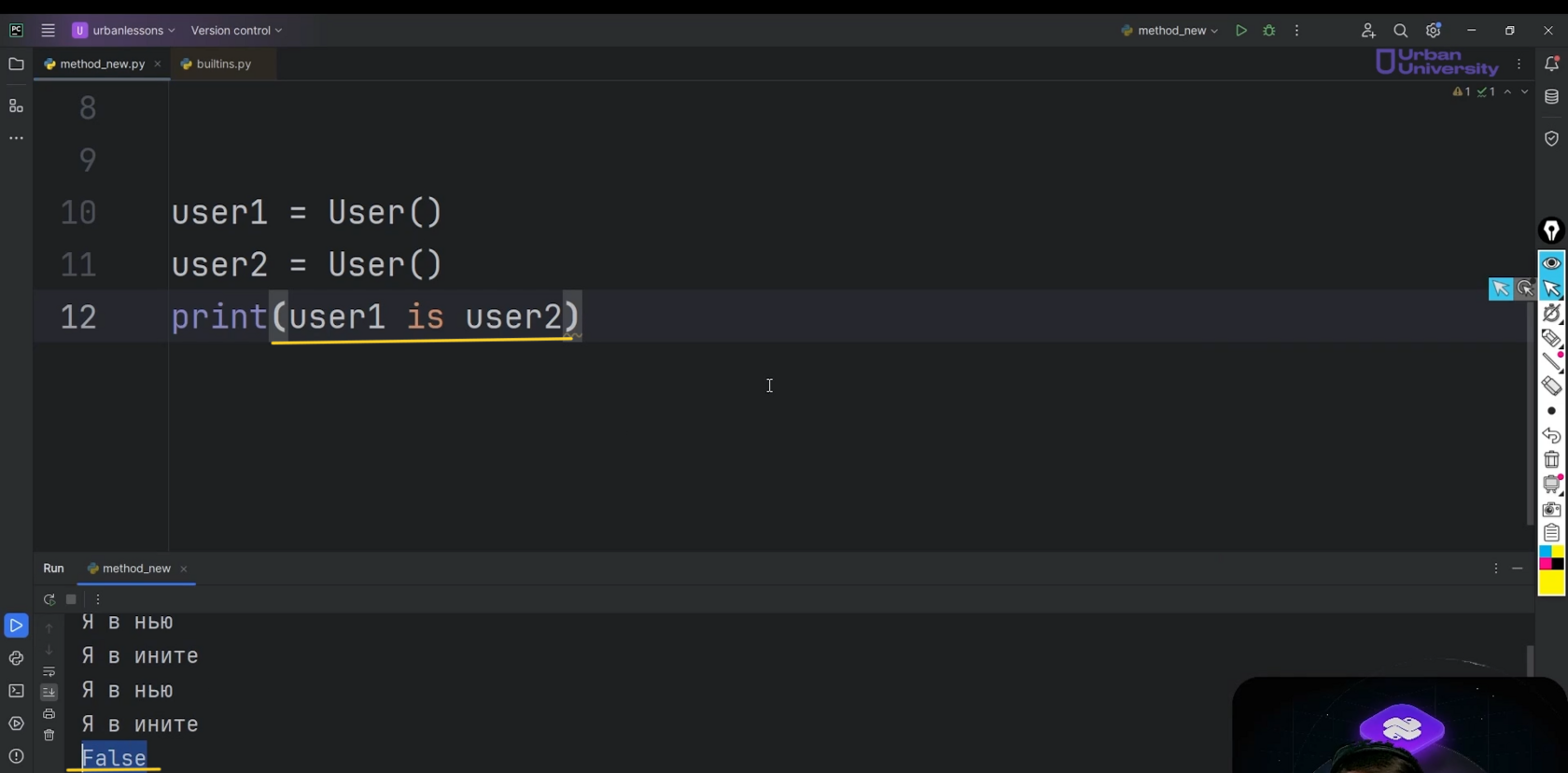


Рис.12

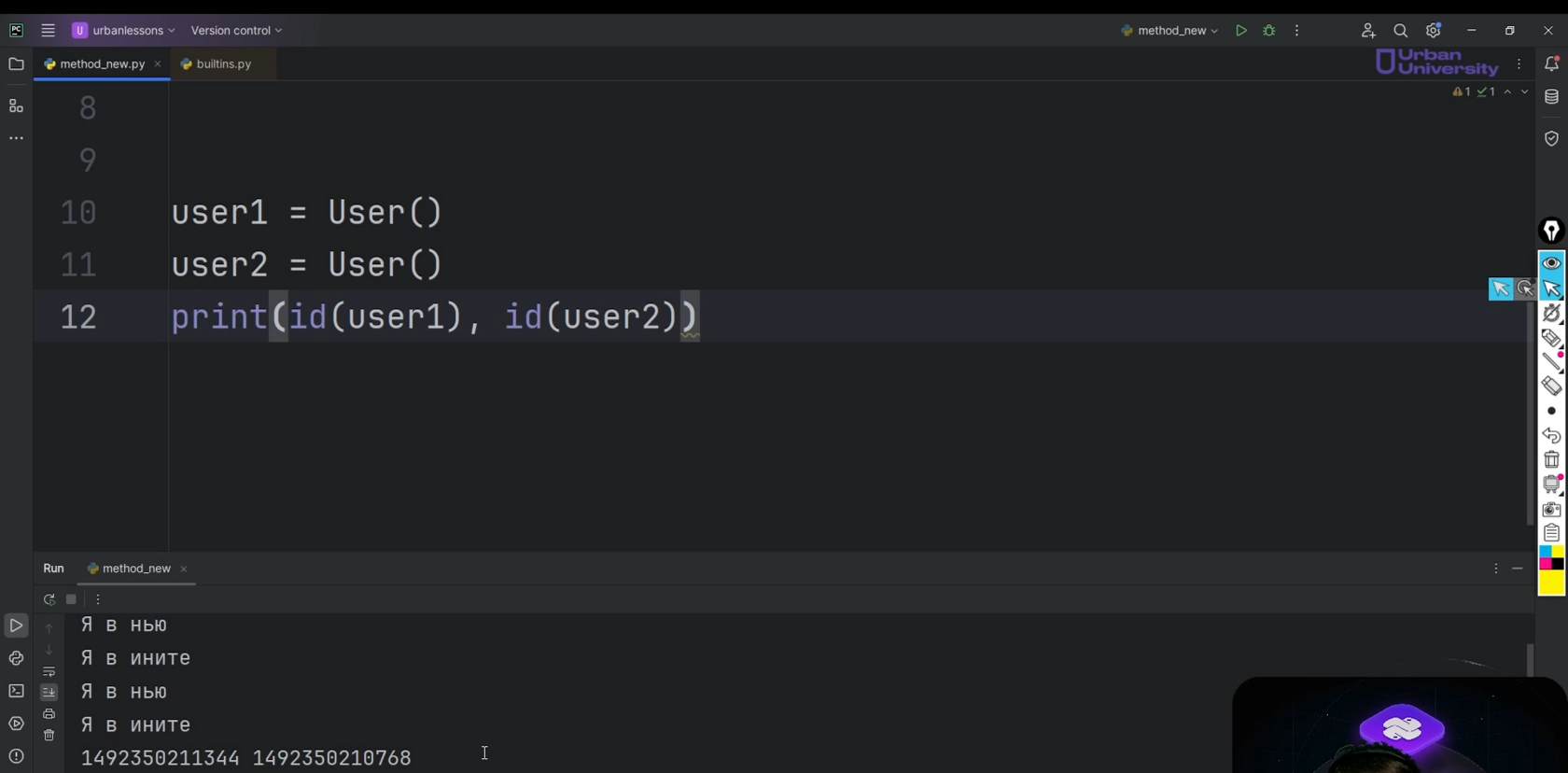


Рис.13

**Рассмотрим реализацию Singleton.** Создадим какой-нибудь классовый атрибут \_\_instance = None и внутри метода new() добавим условие, что если вот этот атрибут \_\_instance внутри нашего класса = None, то есть ещё не был создан ни 1 объект этого класса, то в этот атрибут запишем ссылку на наш класс. Метод new() будет возвращать эту самую ссылку(рис.14). Таким образом видим, что при такой реализации, несмотря на то, что создаются новые объекты, они все ведут к одному адресу в памяти(рис.15). Если взять User1 и воспользоваться оператором is, то в этот раз получим результат True(рис.16). В будущем это вам может очень сильно пригодиться. Это и есть метод new().

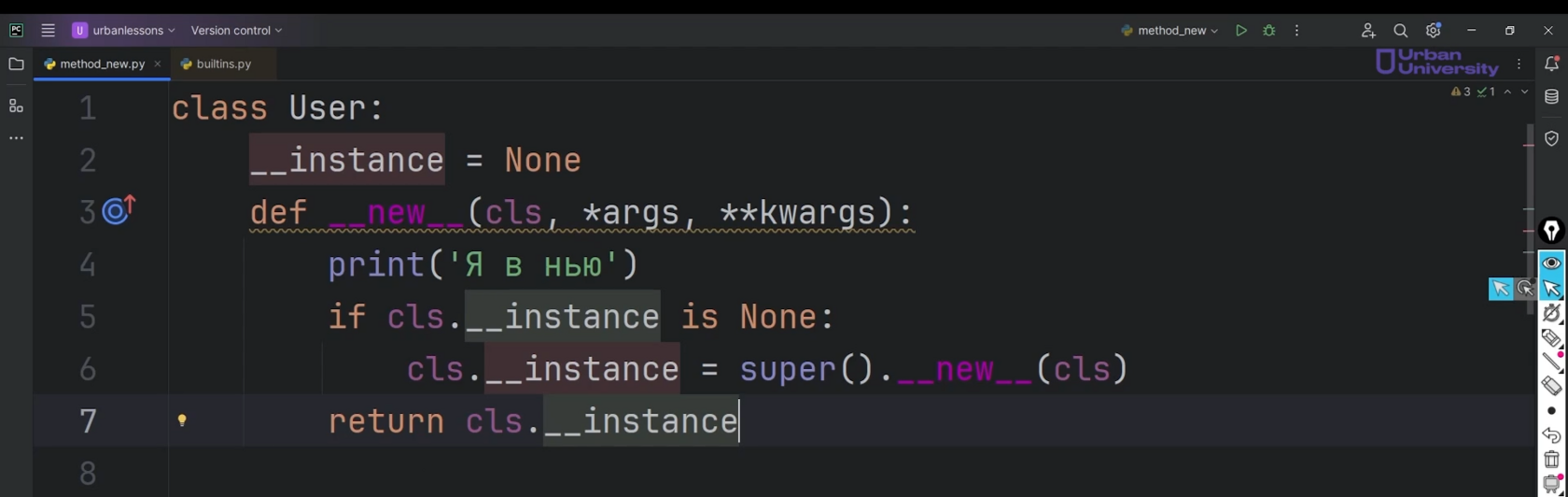


Рис.14

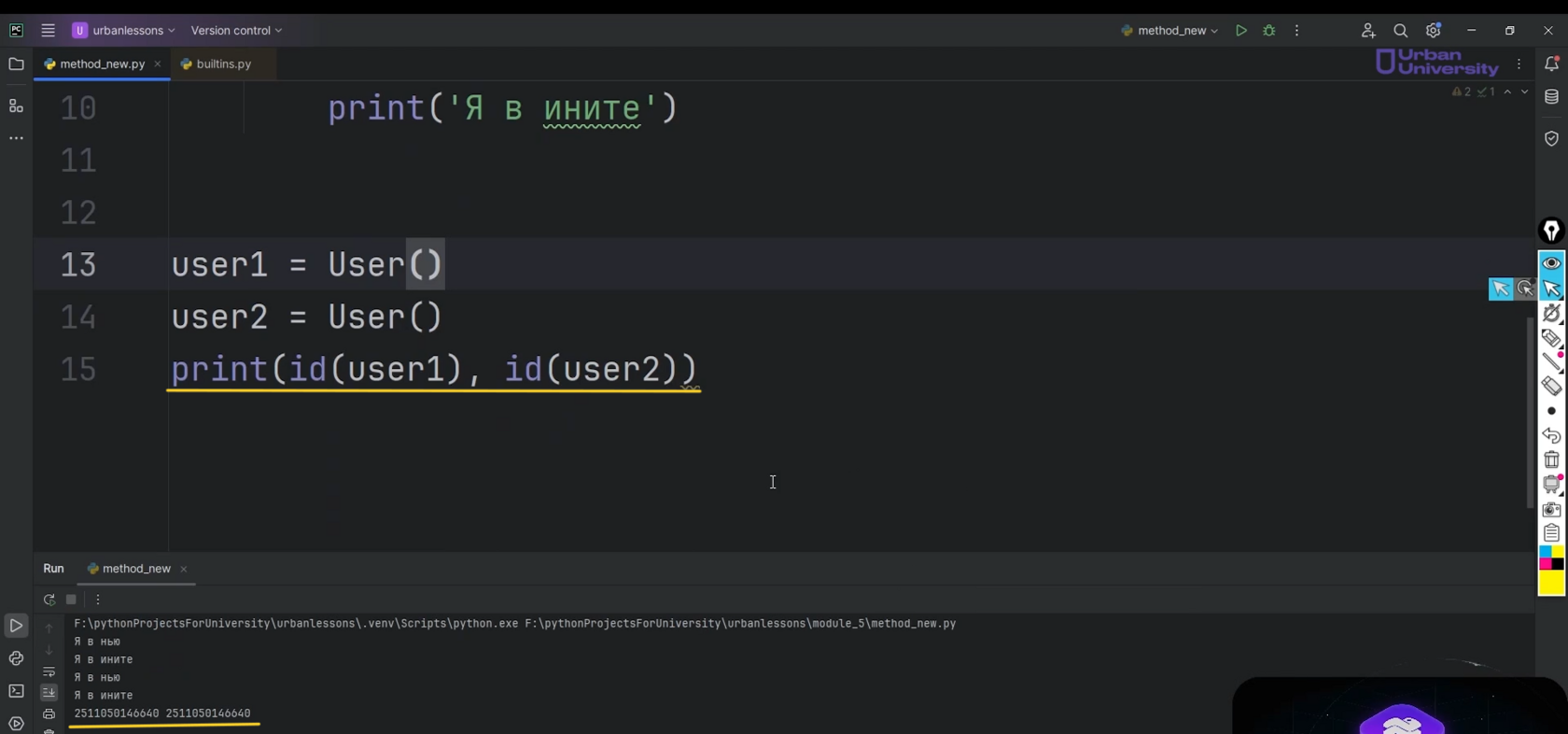


Рис.15

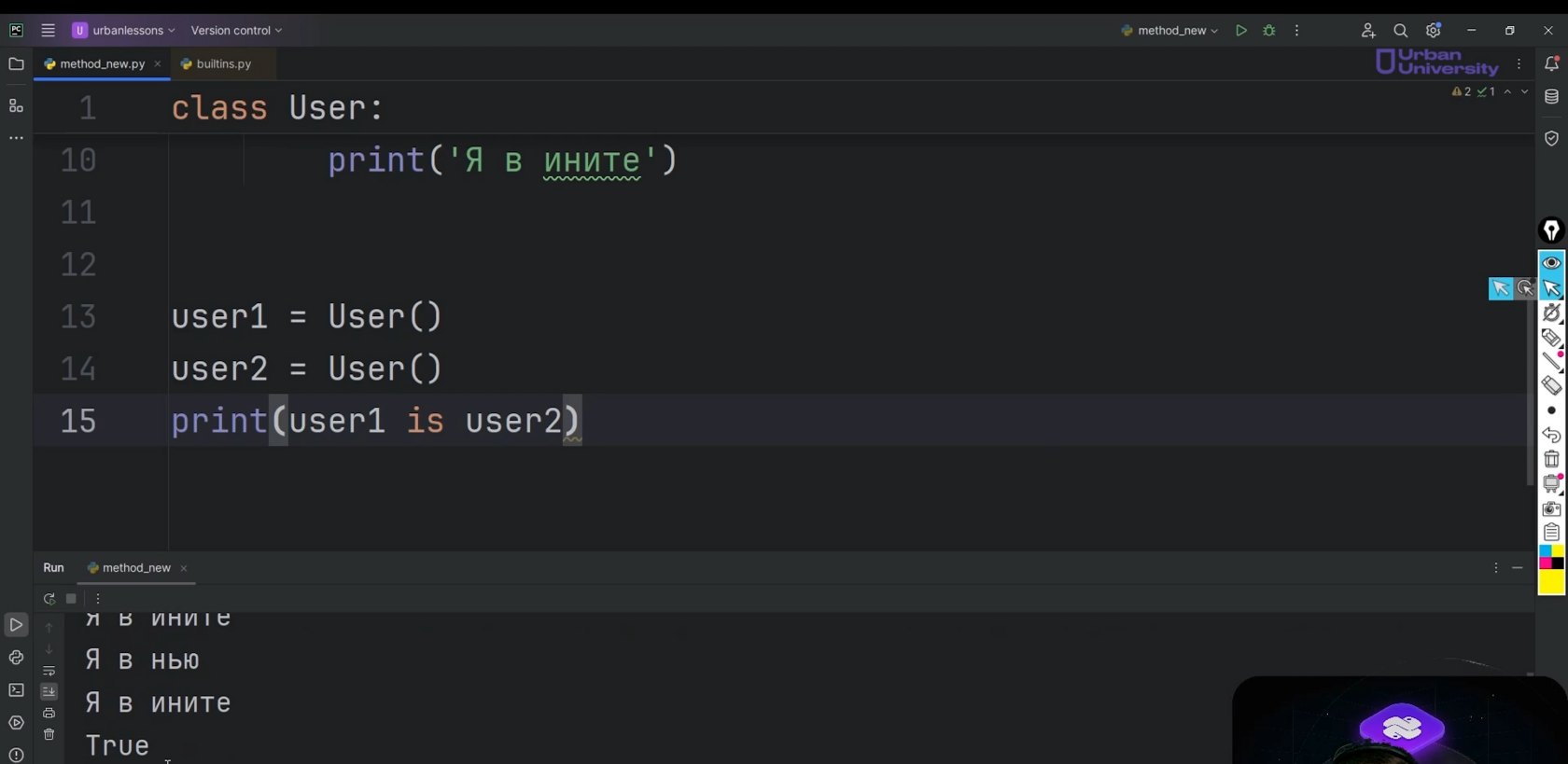


Рис.16

Последнее, что рассмотрим, может пригодиться в работе с вашими классами — это **переменное количество параметров внутри, например, того же init()**. Допустим, вы пишите класс и не знаете, сколько параметров будет принимать объект этого класса. Ранее вы уже изучали функции и здесь принцип на самом деле похож.

Давайте подготовим небольшие тестовые данные. Пускай у нас будет other — это список, содержащий в себе будущие \*args, и user для \*\*kwargs, ключи и значения(рис.17). В init() напишем \*args, \*\*kwargs(рис.18).

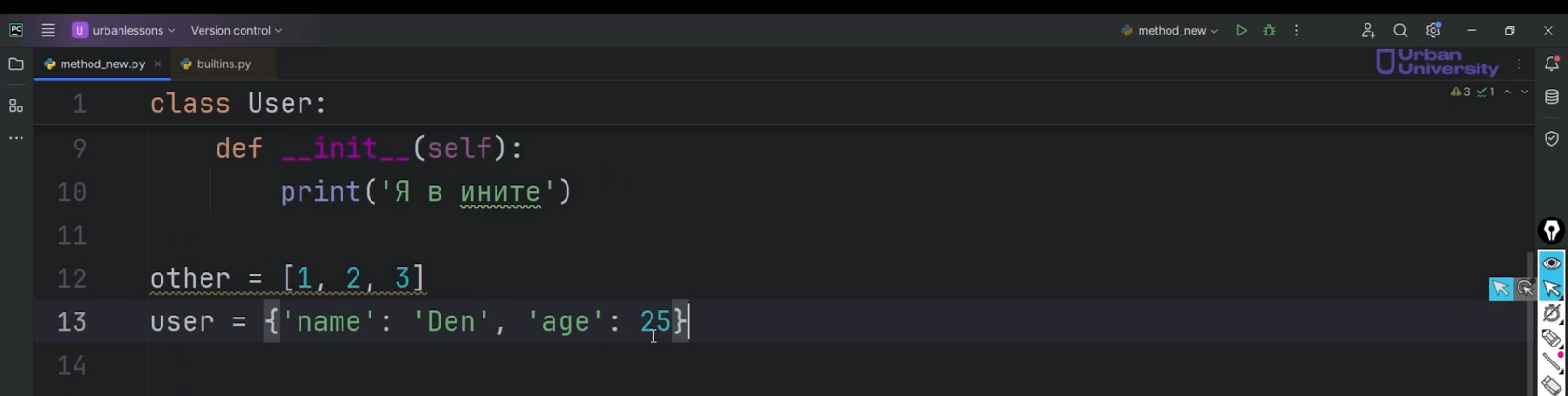


Рис.17

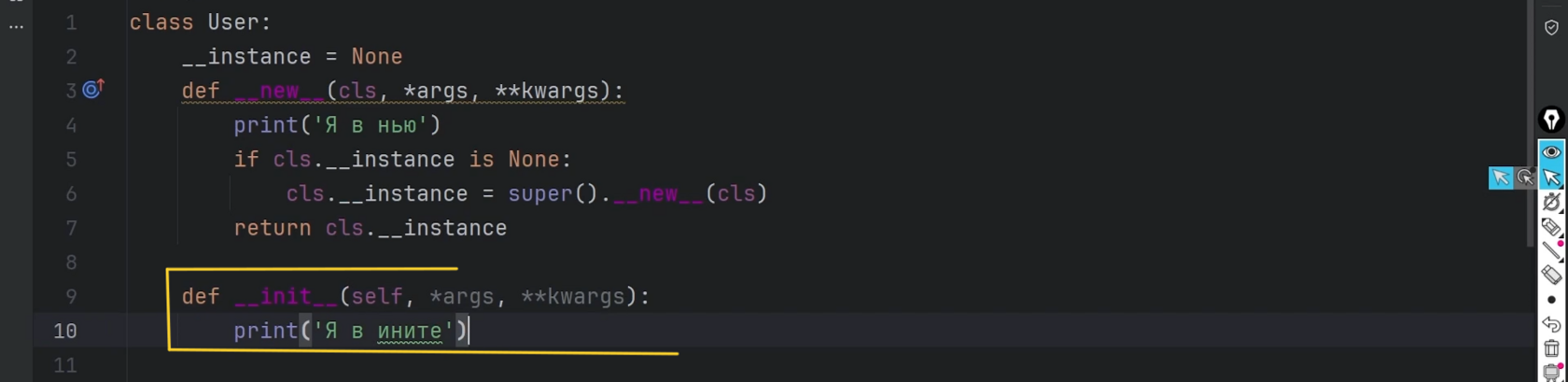


Рис.18

**Как же теперь их определить?** Мы не знаем, как их сохранить, какие конкретно они будут. Можем сделать, например, вот так: self.args = args, self.kwargs = kwargs(рис.19) и в user1 = User() передать other и user, User2 можно убрать. Выведем user1.args и user1.kwargs(рис.20).

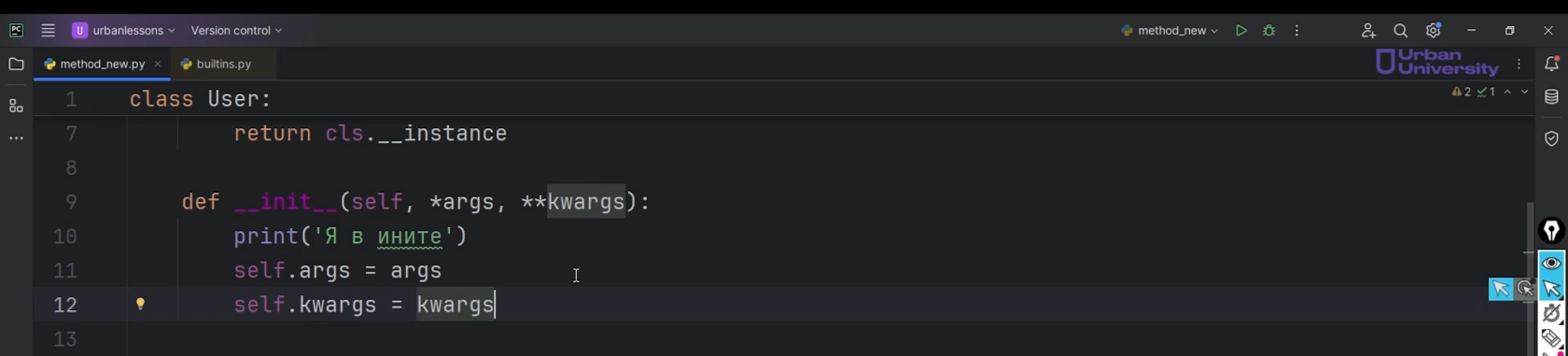


Рис.19

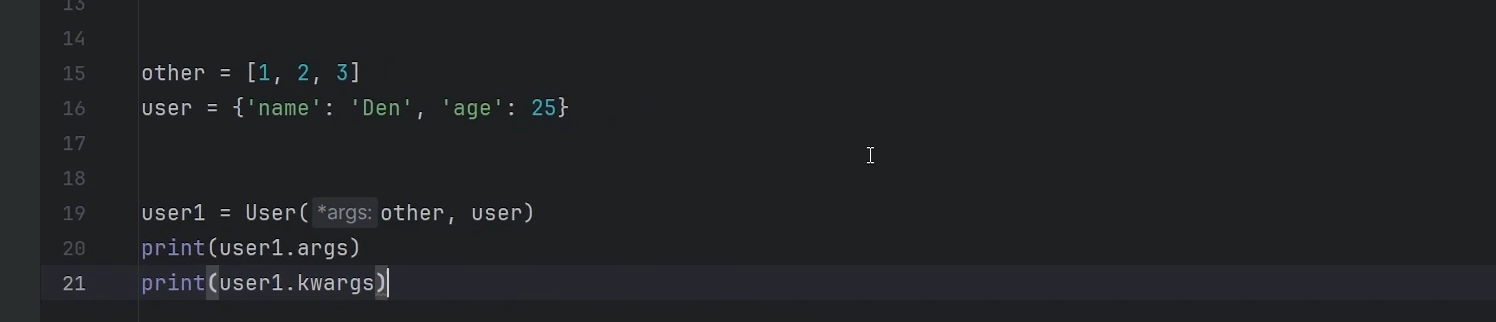


Рис.20

Запускаем и видим, что в args у нас попали список и словарь, в kwargs не попало ничего(рис.21), потому что other и user у нас пришлись на позиционные параметры. Если сделать, например, name = den, то сюда в kwargs уже прилетит именованный параметр(рис.22).

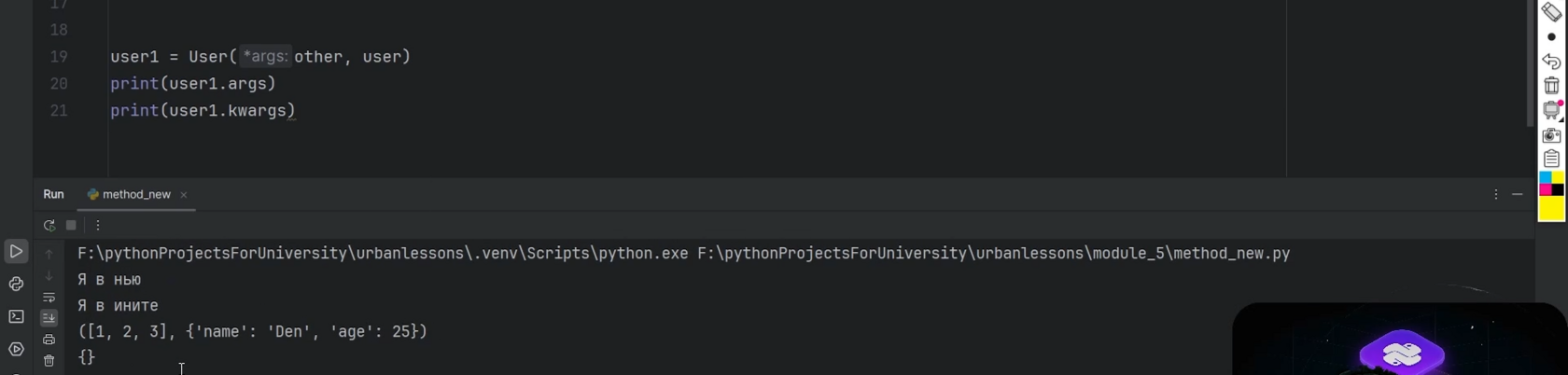


Рис.21

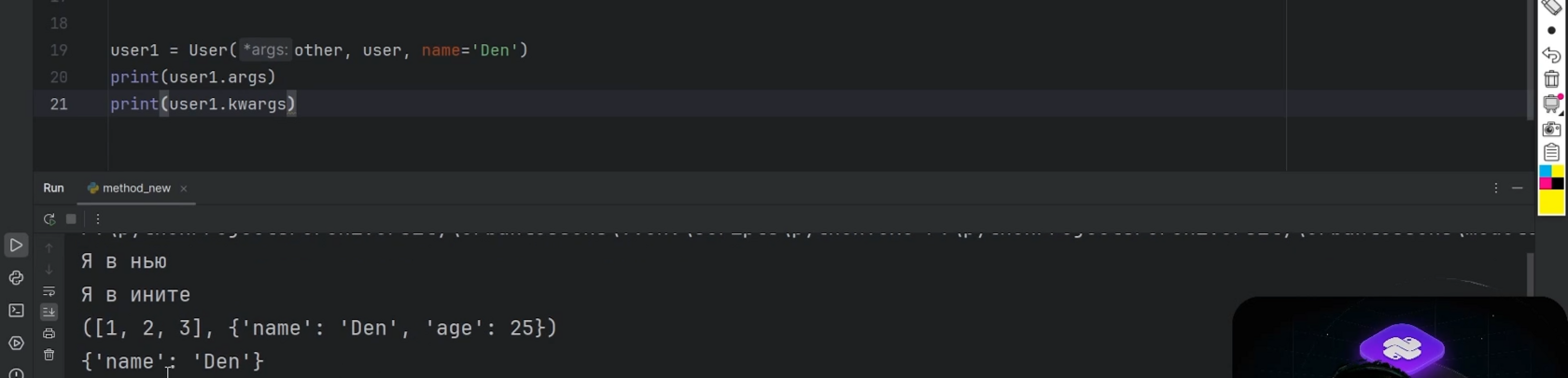


Рис.22

**Однако представим, что хотим распаковать наши тестовые данные.** Вспоминаем, как это делать. Берём, ставим звёздочку для args (\*other) и 2 звёздочки для kwargs(\*\*user). Получается, теперь в первый параметр (self.args = args) у нас прилетит кортеж, содержащий 3 значения, и во 2 параметр (self.kwargs = kwargs) прилетит словарик(рис.23).

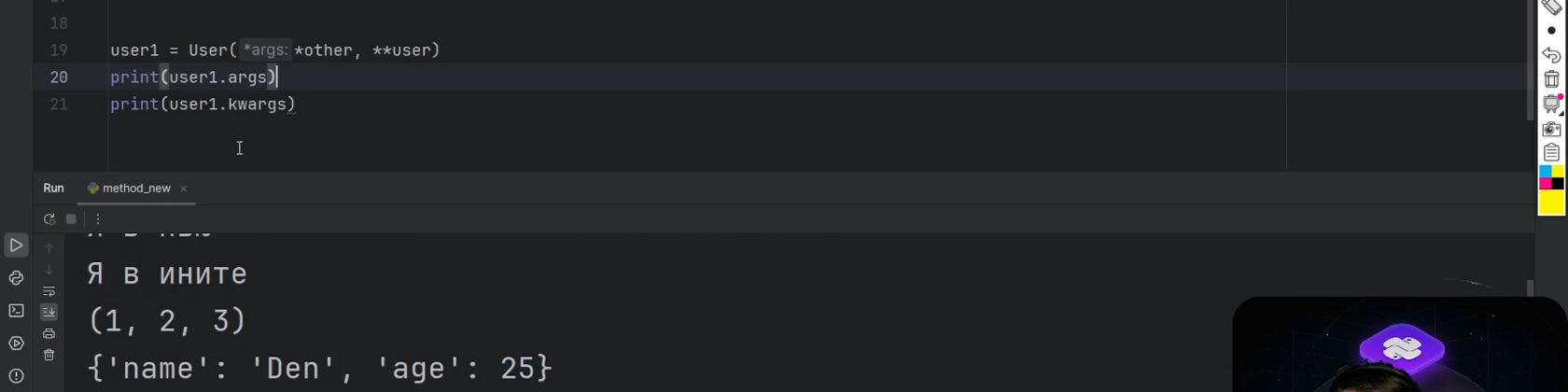


Рис.23

**Как же достать из строки user = {‘name’: ‘Den’, ‘age’: 25} name и age?**Чтобы можно было достать конкретно name и age, можно воспользоваться, например, таким вот способом: можем взять, создать характеристику name и из kwargs достать по ключу name значение и сохранить его в этот атрибут name в эту характеристику(рис.24). То же самое можно сделать с возрастом. Если соответствующий ключ не будет найден, то просто будем записывать туда None. При таком раскладе можно очень удобно получать какие-то отдельные характеристики(рис.25).

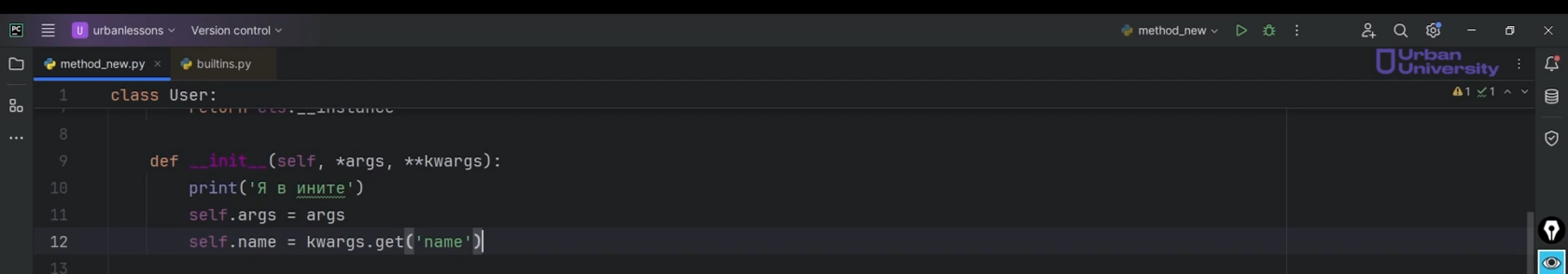


Рис.24

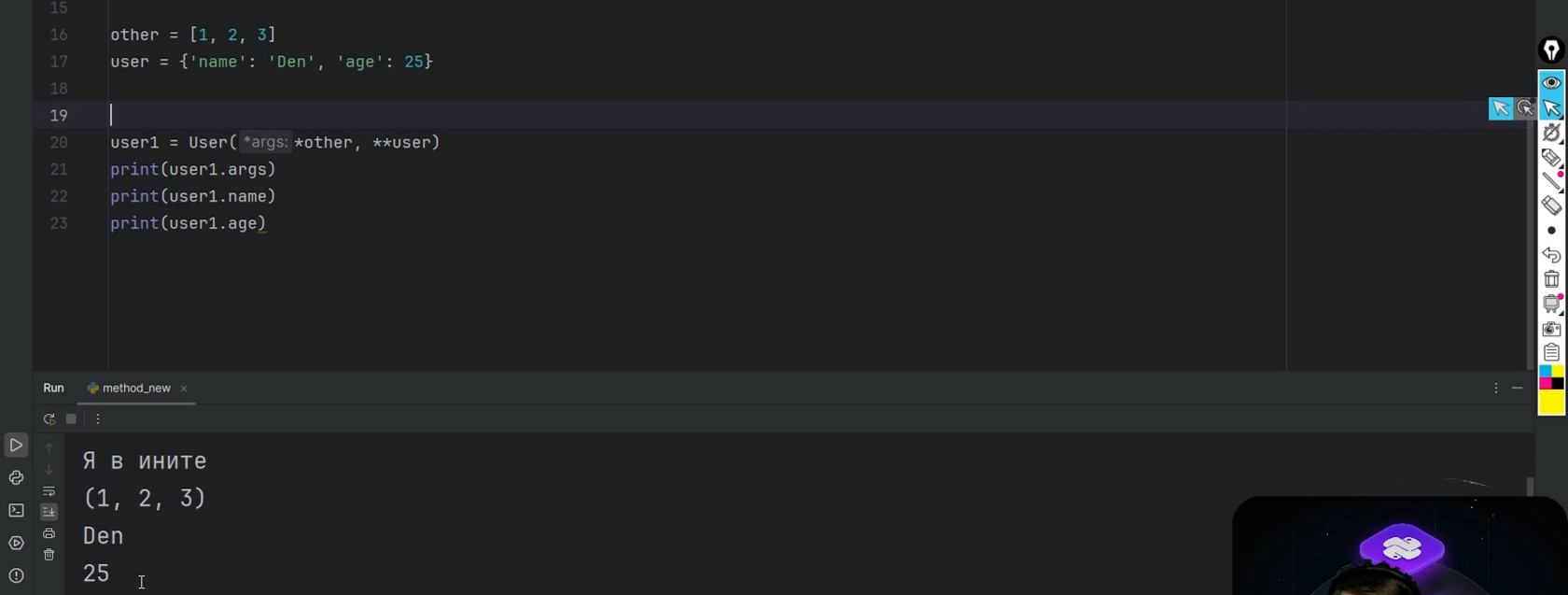


Рис.25

При этом если их будет больше, например, сделаем еще ‘gender’: ‘male’, при получении еще и его получим ошибку(рис.26), так как у нас нет такого атрибута. Заранее же мы не можем знать, какие данные у нас будут. Вот для таких случаев есть неплохой способ: создать цикл for, пробежаться по ключам и значениям в этих именованных параметрах, воспользовавшись методом items() и с помощью метода setattr для каждого объекта класса задавать соответствующий ключ, сохранять в него значения self, key, values(рис.27). В результате работы такого кода получим удовлетворительный для нас результат(рис.28). То есть будем пробегаться по всем этим именованным параметрам и создавать для нашего объекта класса соответствующую характеристику с определённым значением. Это очень удобно. То же самое на самом деле можно делать и с args, только там нет необходимости использовать метод items(), хватит и одного значения в принципе.

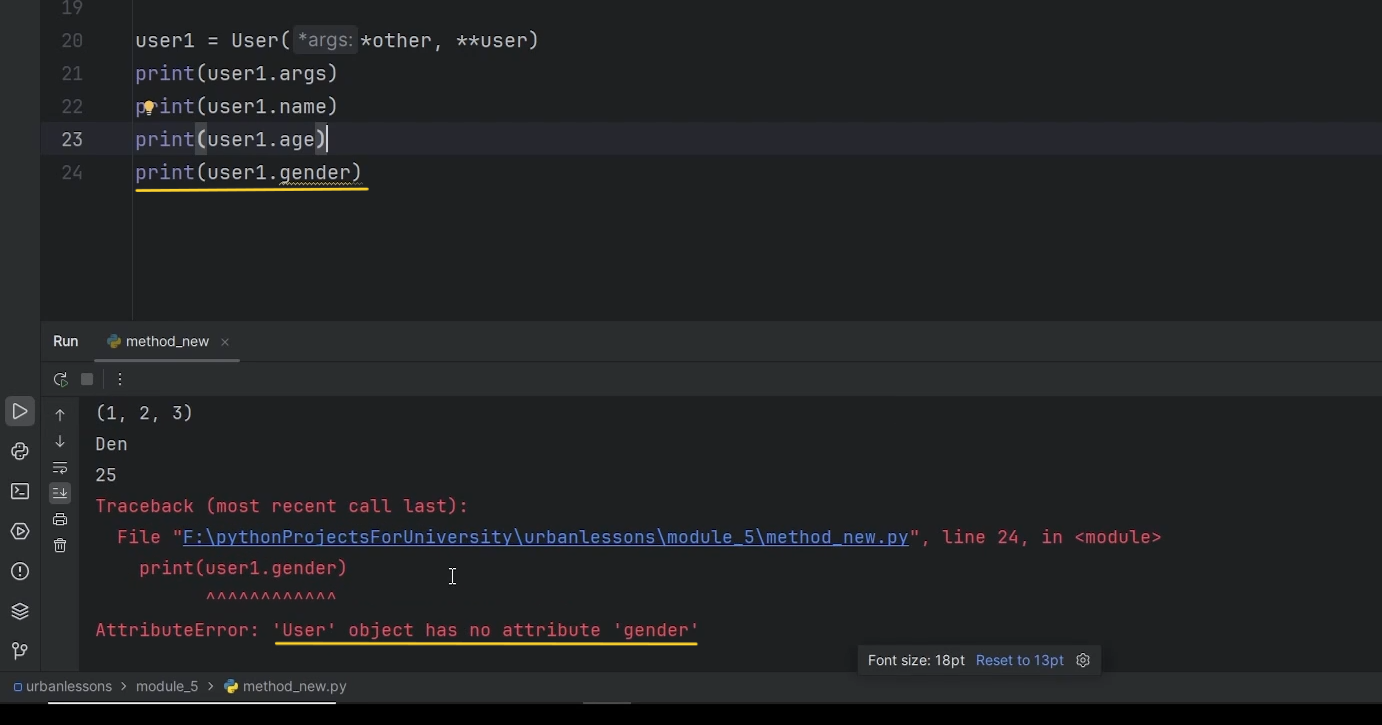


Рис.26

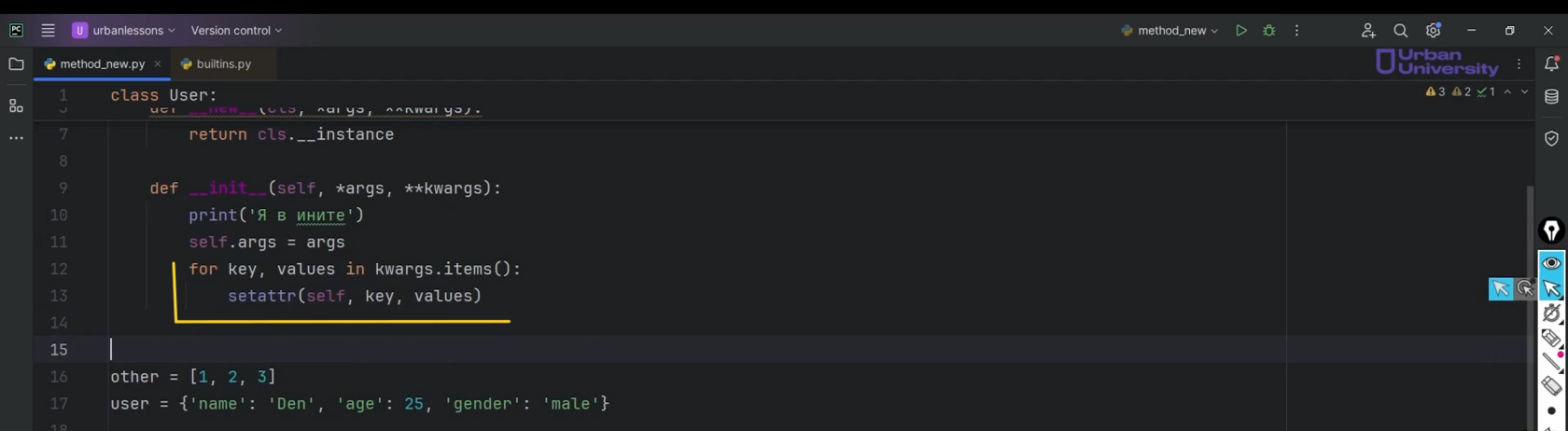


Рис.27

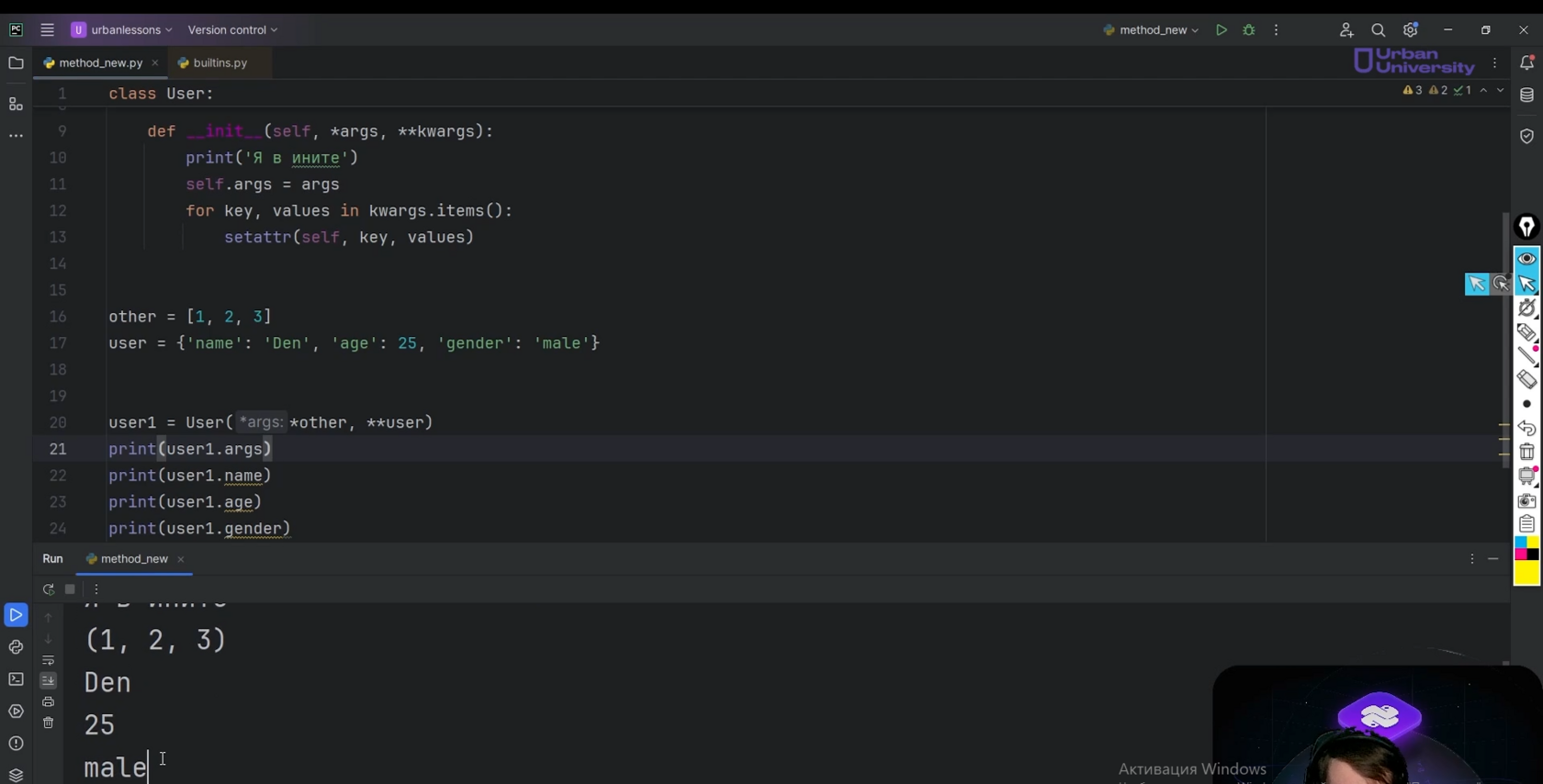


Рис.28