**Специальные методы классов**

## Специальные методы классов

На этом уроке мы с вами будем говорить про специальные методы и классы.

**Класс**— это, по сути, некое описание, некая инструкция, на основе которой мы можем создавать огромное количество объектов, каждый из которых будет являться уникальным. Плюс ко всему, класс содержит некие характеристики объекта и методы, то есть то, что наш объект умеет делать.

Мы уже познакомились с одним из таких специальных методов — это **метод \_\_init\_\_**с двойным подчеркиванием, который, грубо говоря, начинается в самом начале, как только мы описываем классы. Это своего рода конструктор метод, который срабатывает единожды, когда мы создаём объект класса. Объект класса — это переменная, которая создаётся на основе этого класса.

В нашем случае, если брать код из предыдущего занятия, у нас 2 объекта “den” и “max”(рис.1). Этот конструктор \_\_init\_\_ на самом деле работает не совсем так. Сначала выполняется не \_\_init\_\_, а выполняется метод**\_\_new\_\_**, который тоже является специальным методом. Однако его редко используют в переопределении, то есть мы обычно ограничиваемся обычным \_\_init\_\_(рис.2).

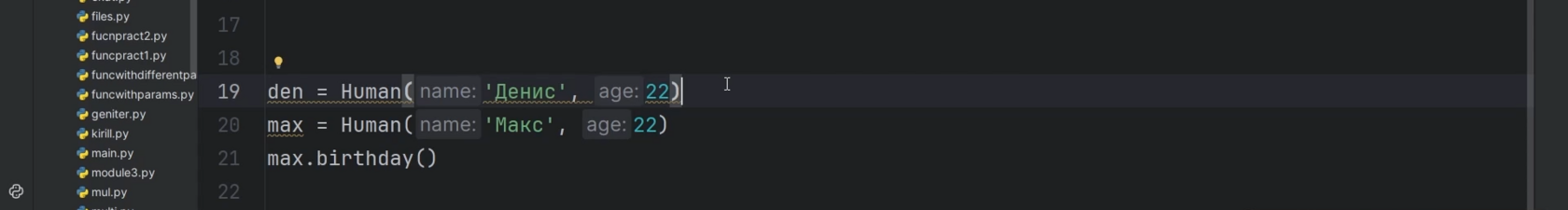


Рис.1

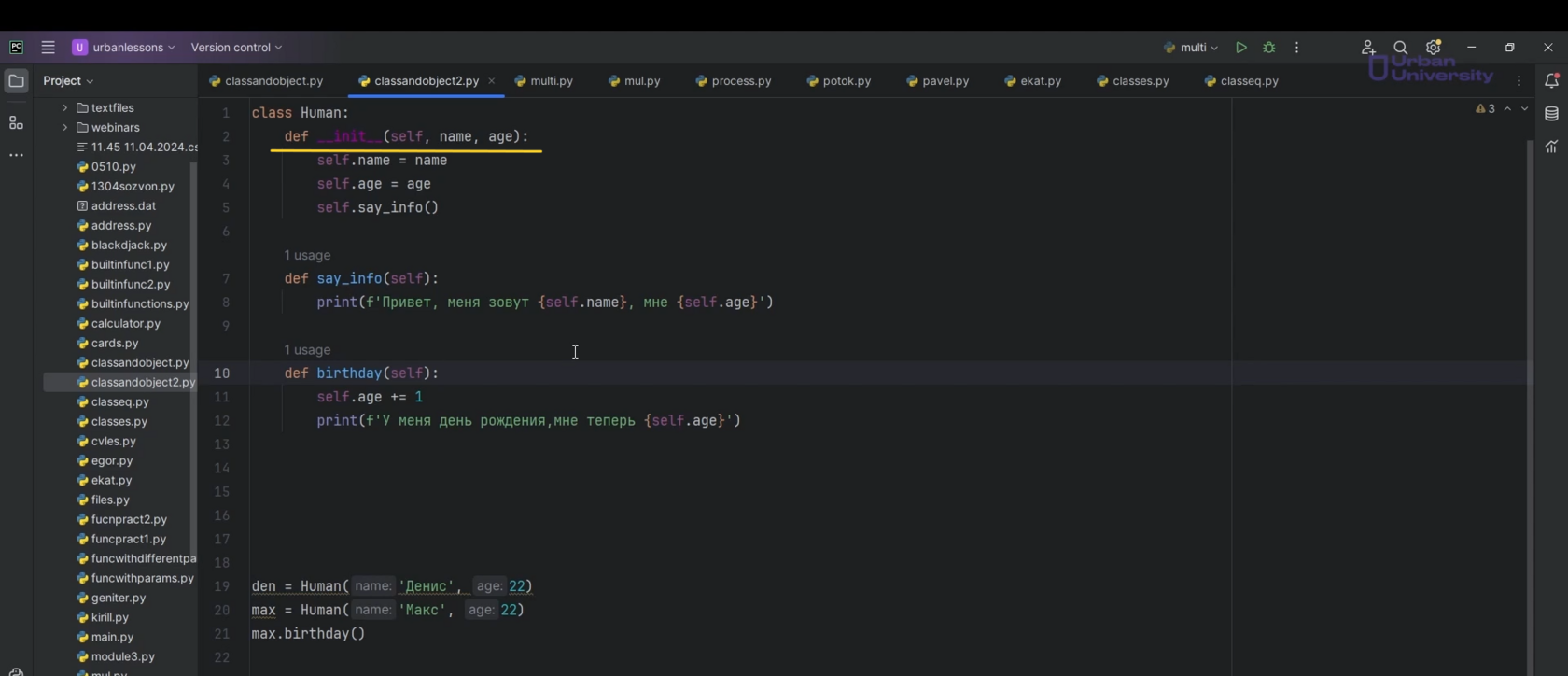


Рис.2

Те методы, которые содержат двойное подчеркивание, называются специальными или магическими методами. Другое название вы можете увидеть, например, dunder- методы, сокращение от слова double under - двойное нижнее подчеркивание, отсюда и соответственно, взялось название.

На этом занятии мы познакомимся с некоторыми из этих магических методов.

#### Начнем с destruct(деструктор). Что такое деструктор?

Мы познакомились ранее с конструктором методом, который срабатывает, когда мы создаём объект класса. Но объекты существуют не вечно, и в какой-то момент они просто исчезают из памяти: это либо когда наш интерпретатор заканчивает свою работу, то есть все строчки кода выполнены, либо же, когда заканчиваются ссылки на этот объект. Мы, например, можем с помощью оператора \_\_del\_\_ удалить, и соответственно, у нас будет срабатывать этот деструктор. Давайте посмотрим, как это выглядит.

Возьмем и создадим метод (с двойным нижним подчёркиванием)**\_\_del\_\_**— это и есть деструктор. Тут определяется логика того, что у нас будет происходить в случае, когда наш объект перестанет существовать. Теперь давайте напишем, чтобы это не было слишком грустно, напишем, что наш человек ушёл( print(f’{self.} Ушёл’))(рис.3).

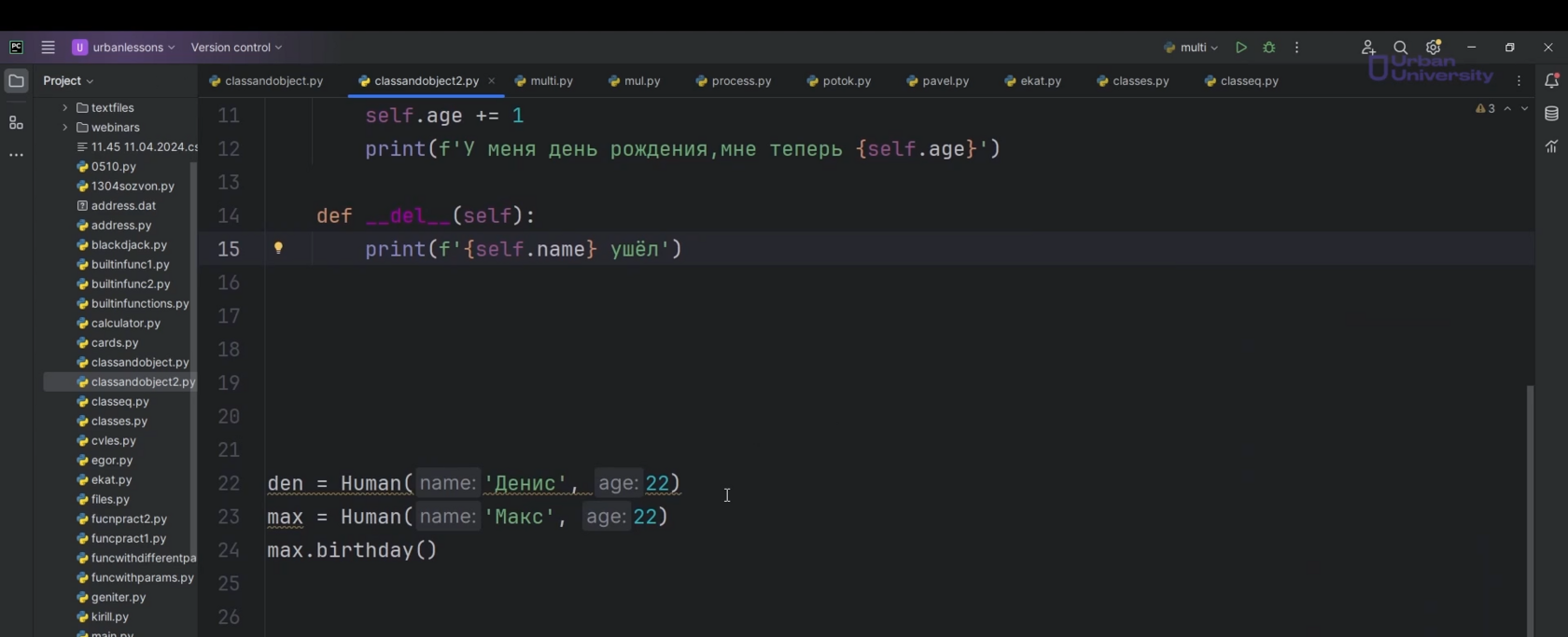


Рис.3

Получается так, что, когда мы запускаем, мы все равно видим код, где “Денис ушел” и “Макс ушел”(рис.4). Потому что на момент выполнения программы выделяется определенная память у компьютера, то есть мы обрабатываем каждую строчку кода, и как только у нас последняя строчка кода выполняется, наш интерпретатора завершает работу, соответственно, освобождает всю занятую до этого память. Также у нас удаляется объект max. Получается, что у них срабатывает этот самый деструктор.

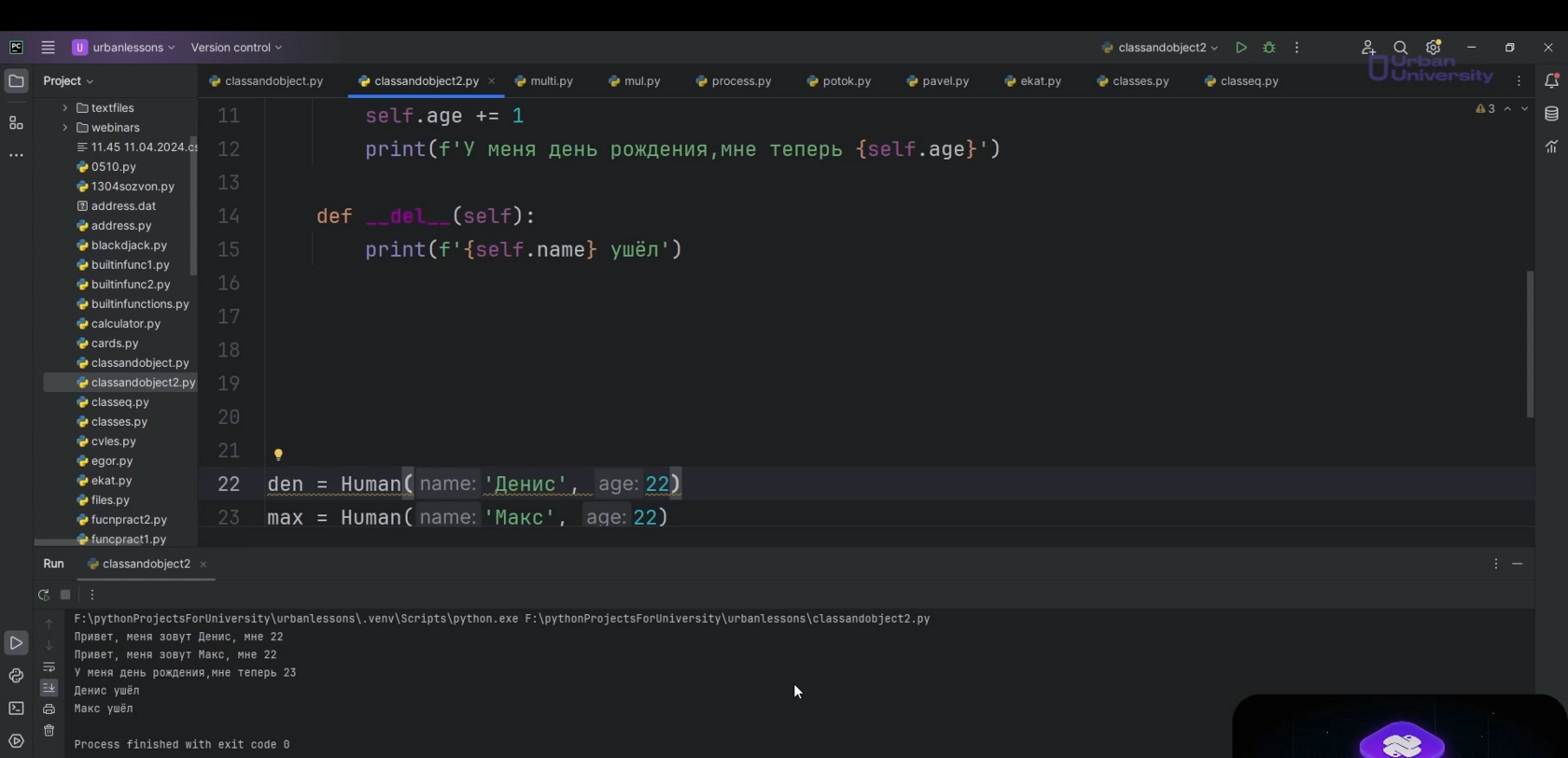


Рис.4

Однако давайте в самом конце просто вызовем функцию input()(рис.5) для того, чтобы запросить пользовательский ввод, чтобы они у нас без предупреждения не удалялись.

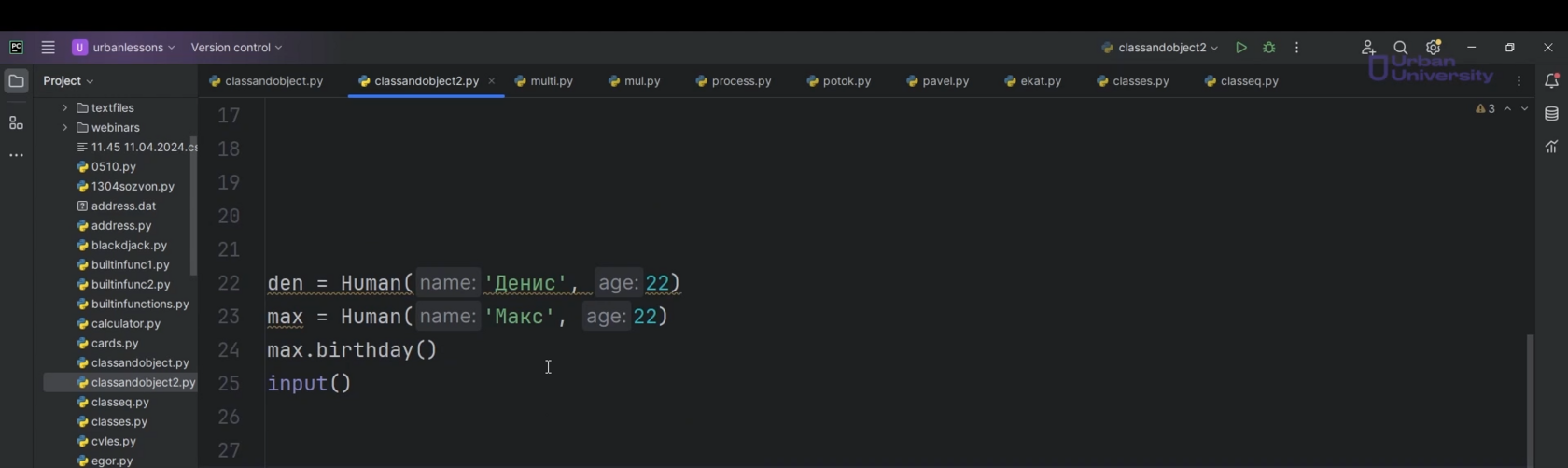


Рис.5

После объектов, например, в ручную с помощью del, удалим нашего den(рис.6). Запускаем. Видим, что ссылка, то есть имя, которое ведет к нашему объекту, было удалено. Ссылки закончились, потому что на наш объект класса Human вело лишь одно имя - den, и когда мы его удалили, сработал деструктор(рис.6). Больше мы не сможем получить доступ к данным.

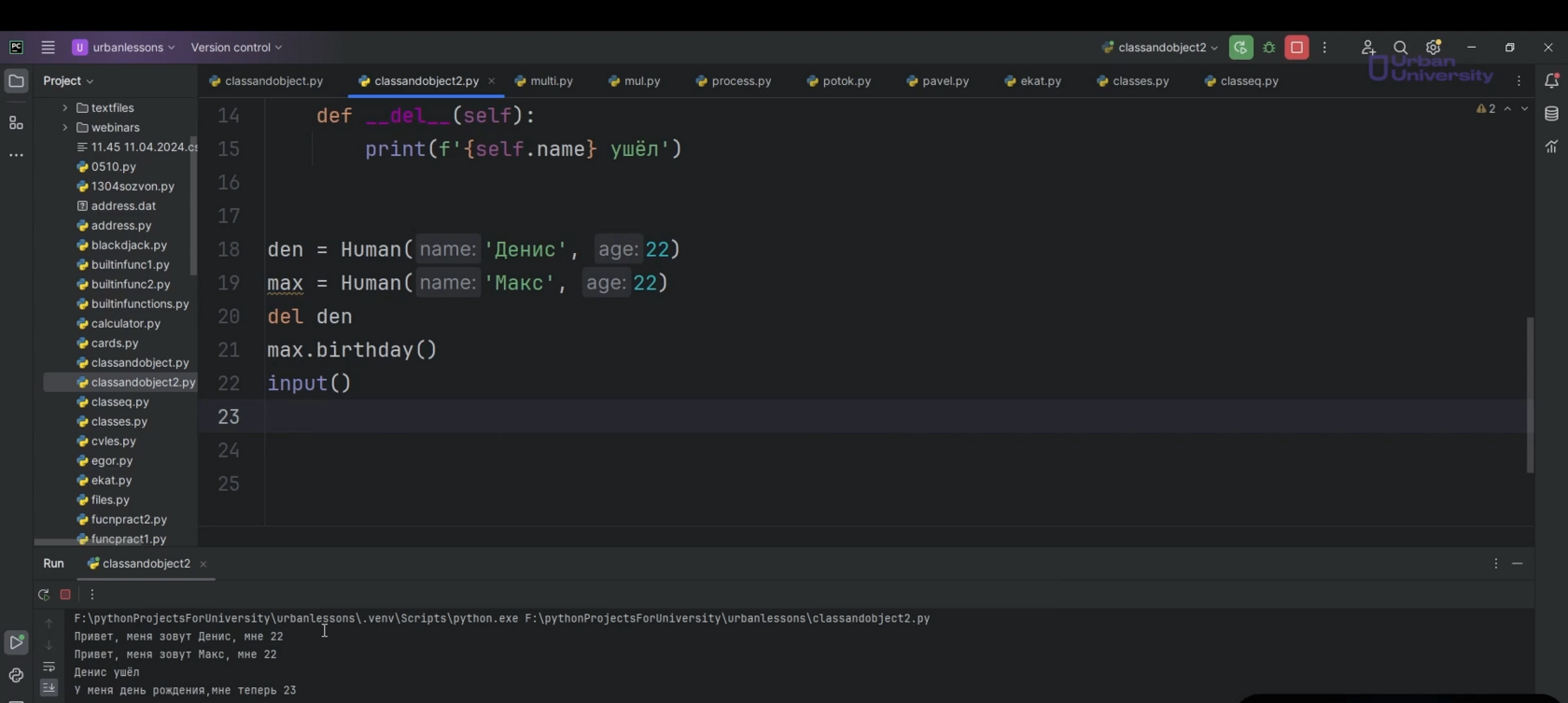


Рис.6

Однако видим, что программа еще не завершила свою работу, мы не видим exit кодов, потому что запрашивается ввод с клавиатуры. Он без пометок без комментариев. Это просто для того, чтобы мы могли наглядно посмотреть, как у нас это работает. Получается, после того как мы нажмем на enter, срабатывают уже деструктор для max(рис.7), несмотря на то, что вручную мы ничего не удаляем.

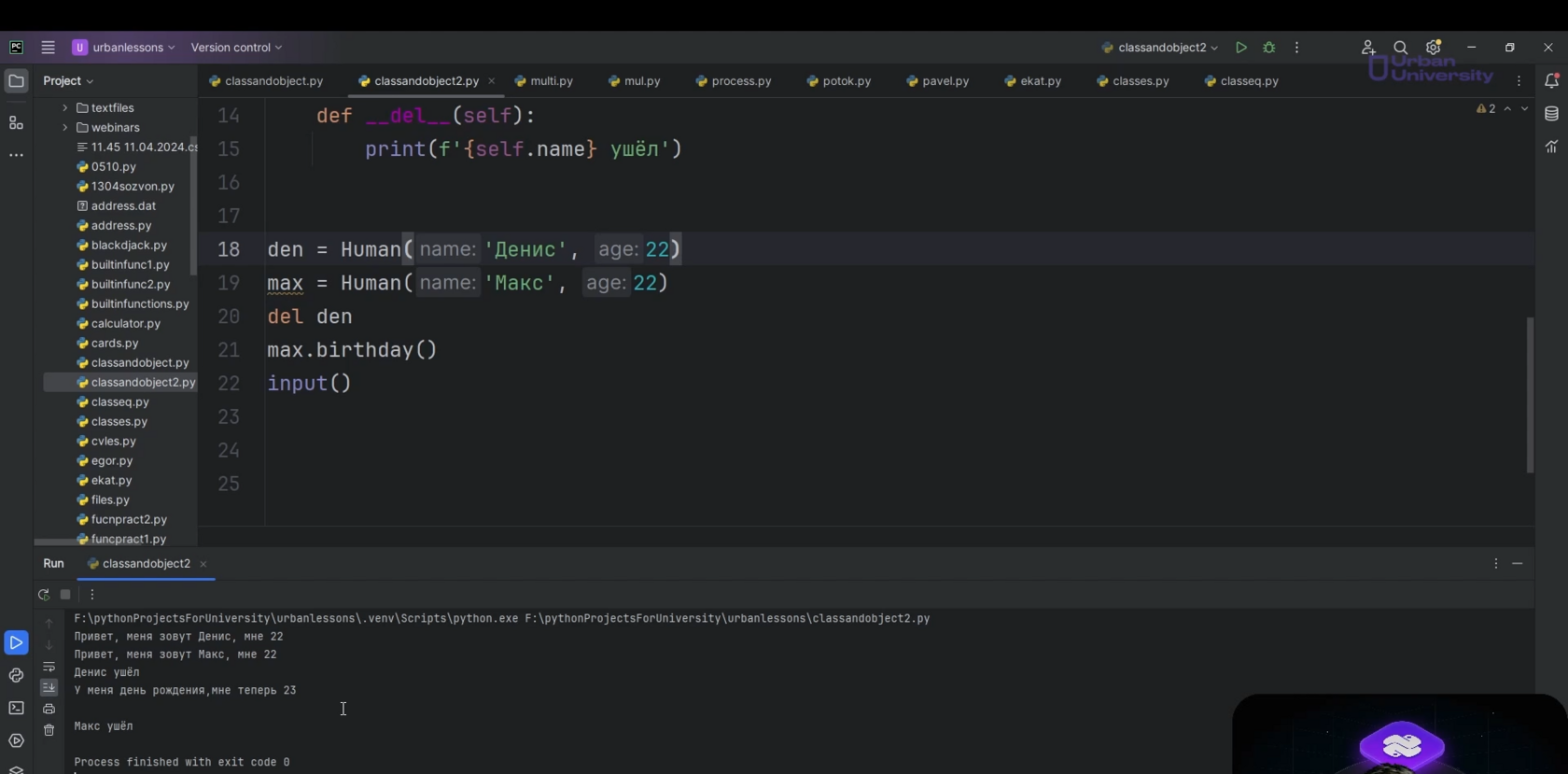


Рис.7

Объекты существуют до тех пор, пока есть хотя бы одна ссылка на этот объект. Если представить нашу память и реализацию того, как это выглядит в Python, мы с вами работали со словарями, у которых есть ключи и значения, и по ключу мы могли достать, получить определенное значение, так и здесь мы создаём какое-то имя, и этому имени присваиваем определенное значение, то есть в Python это выглядит примерно как словарь. То есть имя den вело на объект классах Human, который содержал атрибуты name и age. Имя max привело к объекту класса Human, который содержал атрибуты name и age, но когда эти ссылки у нас заканчиваются, у нас просто объекты исчезают.

**Остальных методов на самом деле очень большое количество.**

Давайте рассмотрим то, как будет работать логика количественной меры измерения количества наших людей. Например, если брать строки или списки, мы можем выяснить размер объекта, то есть количество, допустим, символов в нем, используя функцию**len()**. Что если мы хотим по отношению к нашим людям, тоже применять эту же самую функцию, как нам поступить?

В принципе очень просто, мы сами можем воспользоваться тоже специальным методом len(), который будет возвращать возраст(рис.8). Получается теперь при попытке измерить размер объекта, мы будем возвращать его возраст. Размер человека определяется его возрастом, мы решили это определить таким образом.

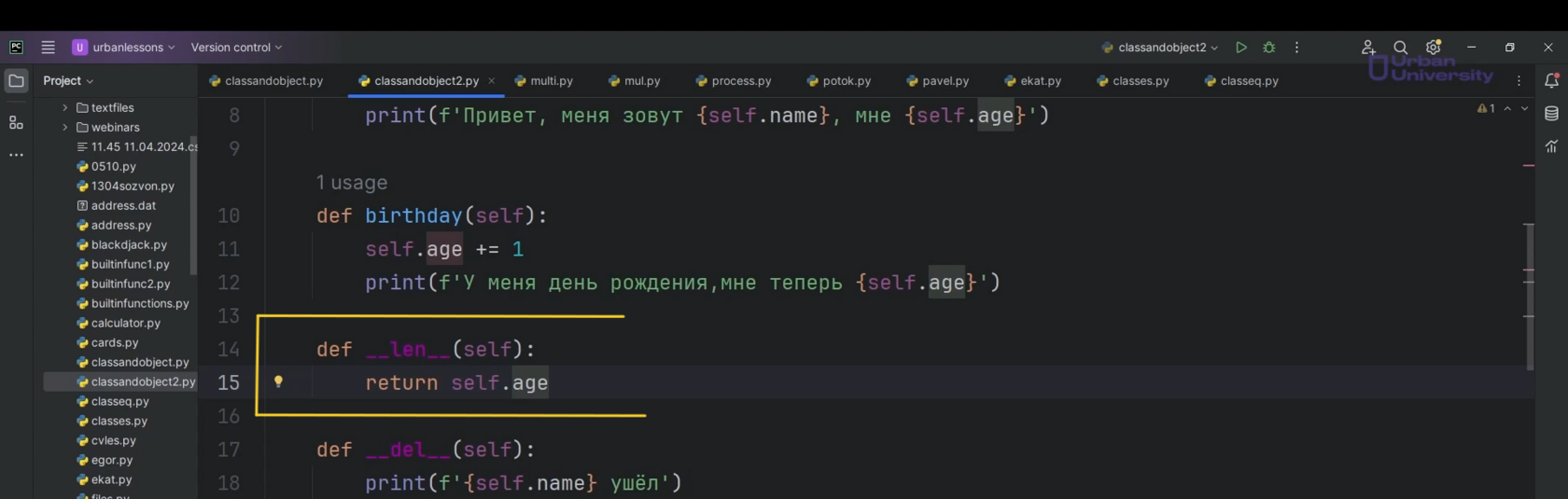


Рис.8

При вызове функции мы будем возвращать возраст человека. Если мы в программе добавим строчку, например print(len(den)), то увидим 22(рис.9).

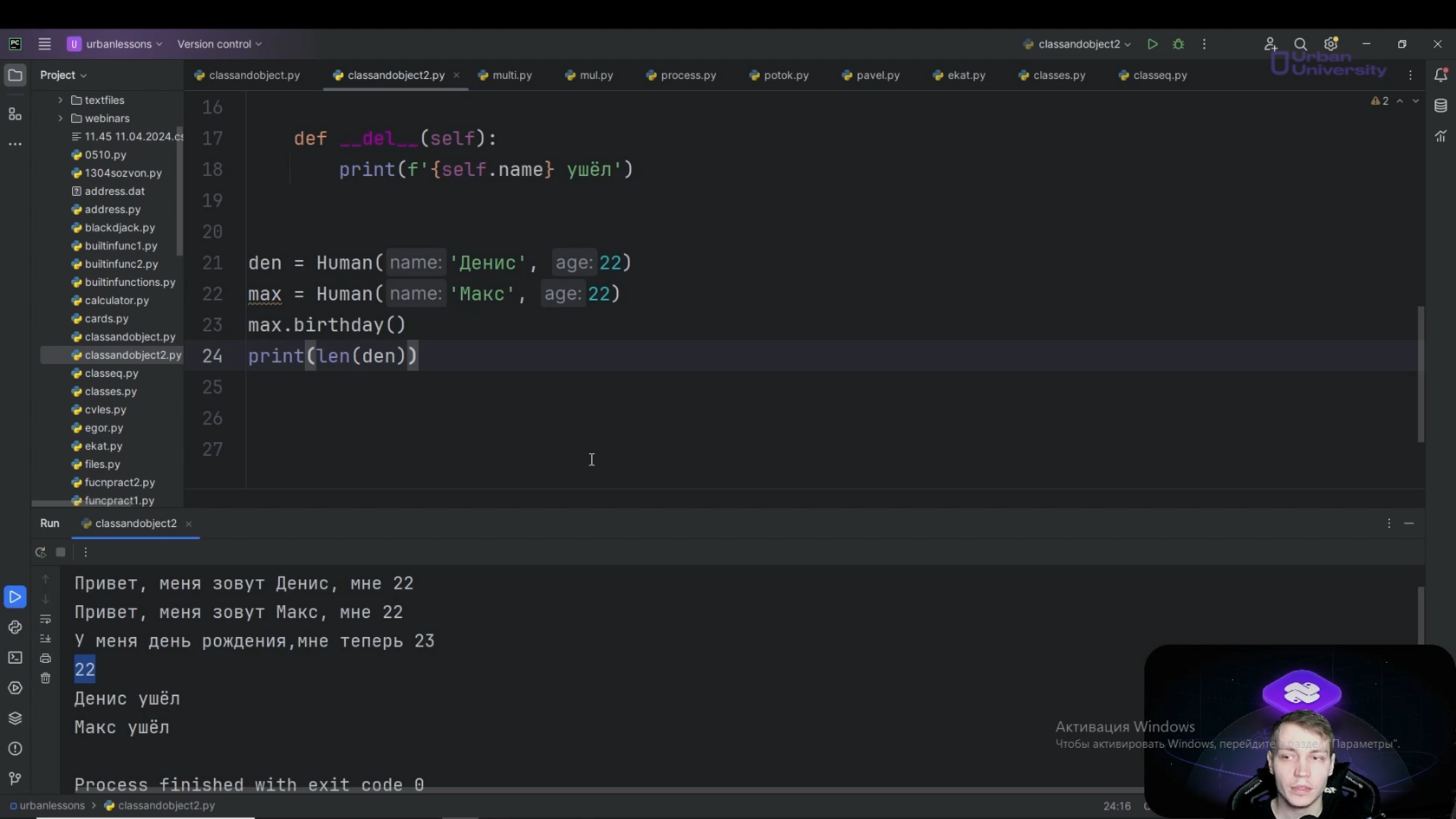


Рис.9

Это тоже один из большого множества этих магических методов. В этом вы можете убедиться сами, введя 2 нижних подчеркивания и увидев здесь огромное количество(рис.10).

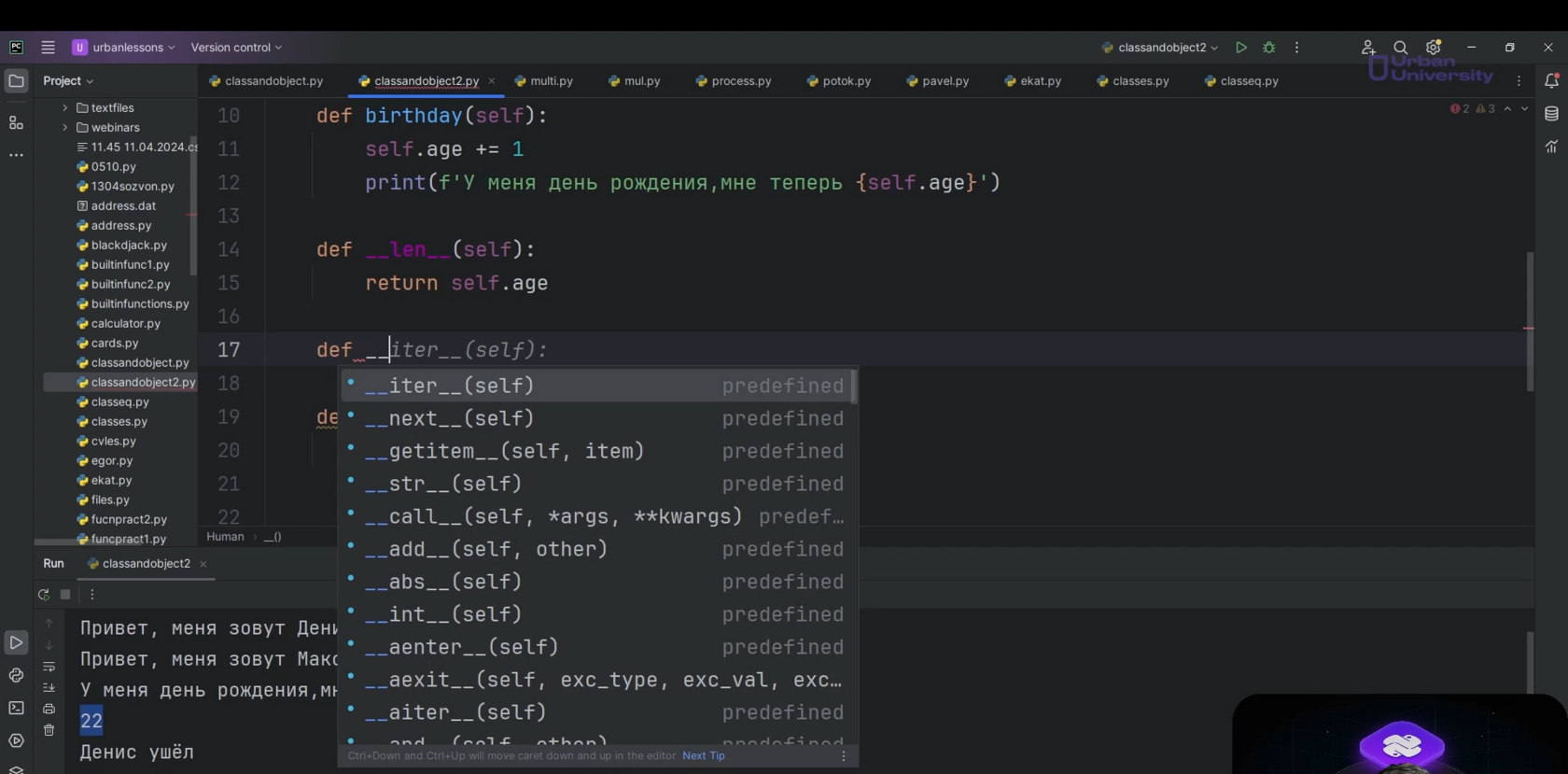


Рис.10

Кроме методов, которые принимают не только self, но и other, мы поговорим с вами в будущем. Но вы должны понимать, что эти специальные методы с двойным подчеркиванием, позволяют нам творить некую магию. Они не просто так называются магическими, потому что, когда мы создаём свою структуру данных, работаем с собственным типом, у нас иногда возникает необходимость переопределять поведение этих объектов тех или иных ситуациях. Например, в случае вычисления размера наших объектов, в случае сравнения двух объектов, в случае сложения, вычитания или применения каких-то других функций, которые мы с вами привыкли применять к уже существующим типам, но только в случае с нашим собственным типом. Именно эти методы dunder с двойным подчёркиванием помогают нам реализовывать вот эту логику.

По сути классов не стоит бояться, если не пытаться в это дело очень сильно углубляться, чтобы себе усложнять жизнь. Наша задача, по сути, научиться с ними работать, уметь создавать собственные классы и постепенно вырабатывать именно объектно-ориентированное мышление, которое позволит нам реализовывать большие программы, с помощью как раз объектов и связей между ними.

Если вы обратите внимание, то большая часть библиотек и их использование тоже завязано на классах. То есть мы будем создавать какие-то объекты. Если взять урок по функциям, где мы с вами создавали графический интерфейс, там реализация точно такая же. Мы создавали главное окошко с помощью **Tk()**, мы создавали объект класса Tk(). В библиотеке Tkinter прописаны классы, которые представляют собой виджеты для взаимодействия с нашим главным окном, например, кнопки, текстовые поля и другие. Также мы используем объекты этих классов и методы, которые не заложены, например **place()** - для размещения элементов на экране, **geometry()** - для изменения размера нашего окошка. Это вот такая реализация в виде классов, и это далеко не последняя библиотека, в которой применяются классы.

На самом деле это очень частое явление, потому что это эффективно, здесь можно придерживаться объектно-ориентированного программирования. **Оно содержит в себе или состоит из трёх принципов:**

* Наследование
* Инкапсуляция
* Полиморфизм

Все эти принципы позволяют нам реализовывать нашу программу эффективно:

* какие-то позволяет нам создавать очень много классов описаний, но которые будут либо дополнять базовый класс, либо же как-то изменять его поведение, то есть делать наши классы более уникальными;
* какие-то будут позволять пользователям взаимодействовать только с тем, что мы хотим, скрыв важные внутренние особенности с целью обезопасить работу.

На этом знакомство с магическими методами заканчивается. Также есть [ссылка](https://docs.python.org/3/reference/datamodel.html#basic-customization) на документацию, в которой можно будет прочитать некоторые особенности по отношению этих методов.