**Инициализатор \_\_init\_\_ и финализатор \_\_del\_\_**

[Смотреть материал на видео](https://www.youtube.com/watch?v=-J3Ou8-8vVk&list=PLA0M1Bcd0w8zPwP7t-FgwONhZOHt9rz9E" \t "_blank)

На этом занятии мы поговорим о начальной инициализации объектов в момент их создания и финализации при их удалении.

В каждом классе языка Python есть набор предопределенных «магических» методов. Да, это такое общепринятое название. Магические методы начинаются и заканчиваются двумя подчеркиваниями:

\_\_ имя метода\_\_

В частности существуют два таких метода:

* \_\_init\_\_(self) –  инициализатор объекта класса
* \_\_del\_\_(self) – финализатор класса

Первый вызывается сразу после создания экземпляра класса, а второй – перед непосредственным его удалением. Давайте посмотрим, как они работают и зачем нужны.

Я вернусь к примеру из прошлого занятия, где мы с вами определили класс с двумя свойствами color и circle и двумя методами set\_coords и get\_coords:

**class** Point:

    color = 'red'

    circle = 2

**def** set\_coords(self, x, y):

        self.x = x

        self.y = y

**def** get\_coords(self):

**return** (self.x, self.y)

Использовать на практике такой класс не очень удобно, так как после создания объекта:

pt = Point()

координат x, y в нем никаких не будет. Дополнительно для этого нужно еще вызывать метод:

pt.set\_coords(1, 2)

Было бы хорошо сделать эти действия сразу в момент создания экземпляра класса. И поможет нам в этом магический метод \_\_init\_\_. Давайте сначала мы его пропишем в самом простом виде:

**class** Point:

    color = 'red'

    circle = 2

**def** \_\_init\_\_(self):

**print**("вызов \_\_init\_\_")

        self.x = 0

        self.y = 0

    ...

Здесь первый параметр self является ссылкой на созданный экземпляр класса, через который мы создаем в этом новом объекте локальные свойства x, y. В результате, создавая экземпляр класса:

pt = Point()

мы увидим в консоли, что, во-первых, произошел вызов этого метода и, во-вторых, в объекте pt были созданы два локальных свойства x, y с нулевыми значениями:

**print**(pt.\_\_dict\_\_)

Детально это работает, следующим образом. Вначале происходит создание объекта в памяти устройства. Непосредственно перед его созданием вызывается магический метод \_\_new\_\_ (О нем мы еще будем говорить). Затем, после успешного создания объекта, вызывается магический метод \_\_init\_\_ для начальной инициализации созданного объекта. В результате, у нас появляются два локальных атрибута x и y.

Давайте теперь сделаем так, чтобы в момент создания экземпляра класса мы могли сразу указывать значения координат точки. Для этого в инициализаторе можно прописывать дополнительные параметры:

**def** \_\_init\_\_(self, a, b):

        self.x = a

        self.y = b

И при создании объекта с таким инициализатором мы уже должны в круглых скобках передавать эти два аргумента:

pt = Point(1, 2)

Если этого не сделать, то возникнет ошибка и объект создан не будет.

Обратите внимание, я специально в качестве дополнительных параметров указал имена a и b. При этом, в самом объекте создаются локальные свойства с именами x и y. То есть, то, что мы прописываем после self, то и является именем нового атрибута. Конечно, на практике, обычно, имена параметров в инициализаторе совпадают с именами создаваемых свойств и лучше записать инициализатор в таком виде:

**def** \_\_init\_\_(self, x, y):

        self.x = x

        self.y = y

Также всегда следует помнить, что метод \_\_init\_\_ - это обычная функция, поэтому мы в качестве дополнительных параметров можем указывать и фактические и формальные параметры. Например, так:

**def** \_\_init\_\_(self, x=0, y=0):

        self.x = x

        self.y = y

Тогда, при создании объектов мы можем вообще не указывать аргументы:

pt = Point()

Либо указать один или два:

pt = Point(10)

pt = Point(10, 20)

То есть, здесь все ровно так, как и с обычными функциями.

**Финализатор объекта класса**

Я, надеюсь, вы теперь хорошо себе представляете, как работает инициализатор объектов класса и зачем он нужен. Второй аналогичный магический метод \_\_del\_\_ автоматически вызывается непосредственно перед уничтожением экземпляра класса. Он называется **финализатор**.

Давайте пропишем такой магический метод в классе Point, следующим образом:

**def** \_\_del\_\_(self):

**print**("Удаление экземпляра: "+ str(self))

Здесь по-прежнему self – это ссылка на экземпляр класса, который будет удален.

После запуска программы видим, что объект в конце программы действительно был удален и вызван метод \_\_del\_\_. Но, здесь у вас может возникнуть вопрос: а когда и в какой момент вообще происходит удаление объектов. Мы же в программе это явно нигде не указываем? Работает все достаточно просто. Интерпретатор языка Python имеет, так называемый, **сборщик мусора**. Это алгоритм, который отслеживает объекты и как только они становятся ненужными, удаляет их. Но как он определяет нужный объект или ненужный? Все очень просто. Пока на какой-либо объект ведет хотя бы одна внешняя ссылка, то он считается используемым и сохраняется в памяти. Как только все внешние ссылки пропадают, то сборщик мусора его уничтожает (освобождает память, которую он занимал).

Итак, из этого занятия вы должны знать, что такое магический метод и как работают инициализатор объекта и его финализатор. А также в какой момент и когда они вызываются. Если все это понятно, то жду всех вас на следующем занятии.