# **Как работает \_\_slots\_\_ с property и при наследовании**

Курс по Python ООП: <https://stepik.org/a/116336>

[Смотреть материал на видео](https://www.youtube.com/watch?v=7xXPuIdfoUU&list=PLA0M1Bcd0w8zPwP7t-FgwONhZOHt9rz9E)

Мы продолжаем изучение коллекции \_\_slots\_\_. Как мы говорили на предыдущем занятии, эта коллекция позволяет ограничивать список локальных свойств в экземплярах класса, в котором прописана. Например, для класса точки на плоскости мы разрешали только свойства с именами x и y:

**class** Point2D:     \_\_slots\_\_ = ('x', 'y')       **def** \_\_init\_\_(self, x, y):         self.x = x         self.y = y

А теперь, сделаем следующее. Добавим в эту коллекцию еще одно имя:

\_\_slots\_\_ = ('x', 'y', 'length')

Тогда, очевидно, мы его тоже сможем использовать в нашем классе, например, в конструкторе:

**class** Point2D:     \_\_slots\_\_ = ('x', 'y', 'length')       **def** \_\_init\_\_(self, x, y):         self.x = x         self.y = y         self.length = (x\*x + y\*y) \*\* 0.5

И, далее, создавая экземпляр этого класса:

pt = Point2D(1, 2)

автоматически получаем свойство length с расстоянием до точки (относительно нуля):

pt.length

Но что если в этом классе прописать свойство (property) с тем же самым именем length? А в коллекции \_\_slots\_\_ задать приватное свойство:

\_\_slots\_\_ = ('x', 'y', '\_\_length')

Давайте добавим это свойство и посмотрим, что получится:

    @property     **def** length(self):         **return** self.\_\_length       @length.setter     **def** length(self, value):         self.\_\_length = value

И, затем, после создания экземпляра класса, мы можем использовать такой синтаксис:

pt = Point2D(1, 2)**print**(pt.length)

Смотрите, мы здесь обращаемся к свойству length, хотя в \_\_slots\_\_ нет такого разрешенного имени. И при этом нет никаких ошибок выполнения. В принципе, так и должно быть, потому что length – это не локальная переменная экземпляра класса, а атрибут класса Point2D. Коллекция \_\_slots\_\_ не накладывает ограничения на атрибуты класса, только на локальные атрибуты его экземпляров. Благодаря этому и появляется свойство length, которое работает как геттер и сеттер класса Point2D.

## **Поведение \_\_slots\_\_ при наследовании классов**

Следующий важный момент – это особенности работы коллекции \_\_slots\_\_ при наследовании классов. Я сразу приведу пример, чтобы показать первую особенность этой коллекции. Допустим, мы создаем еще один класс Point3D, который наследует от класса Point2D и он пока будет пустым:

**class** Point3D(Point2D):     **pass**

Тогда, для экземпляров этого класса будет разрешено создание локальных свойств, несмотря на то, что в базовом классе присутствует коллекция \_\_slots\_\_:

pt3 = Point3D(10, 20) pt3.z = 30

То есть, по умолчанию эта коллекция не наследуется дочерними классами и они ведут себя как обычные, без каких-либо ограничений, сохраняя все локальные свойства, используя коллекцию:

pt3.\_\_dict\_\_

Однако, если мы ее пропишем даже без указания каких-либо элементов:

**class** Point3D(Point2D):     \_\_slots\_\_ = ()

то ограничения вступят в свои права и она будет как бы унаследована от базового класса Point2D. То есть, в классе Point3D сейчас доступны только два локальных свойства x и y.

Но нам нужно разрешить еще одно имя z. Для этого достаточно одно его и прописать в этой коллекции:

**class** Point3D(Point2D):     \_\_slots\_\_ = 'z',

Тогда имена x, y будут наследоваться от класса Point2D, а z – добавляться классом Point3D. Также обратите внимание на висячую запятую. Она означает, что z – это элемент кортежа, а не просто строка. Предпочтительно записывать в таком виде. Хотя, и без запятой тоже все будет работать.

Очевидно, следующим шагом надо добавить конструктор в дочерний класс. Запишем его в следующем виде:

**class** Point3D(Point2D):     \_\_slots\_\_ = 'z',       **def** \_\_init\_\_(self, x, y, z):         super().\_\_init\_\_(x, y)         self.z = z

И экземпляр будем создавать уже с тремя параметрами:

pt3 = Point3D(10, 20, 30)

Теперь, наш дочерний класс содержит уникальный код, расширяющий функционал базового класса без каких-либо повторений. При этом, в обоих классах используется ограничение, но с разным набором локальных свойств.

Надеюсь, из этих занятий вы стали лучше понимать работу этой коллекции и при необходимости применять ее в своих проектах.

Курс по Python ООП: <https://stepik.org/a/116336>