# **Коллекция \_\_slots\_\_**

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

[Смотреть материал на видео](https://www.youtube.com/watch?v=6LiIT0Uz6Vw&list=PLA0M1Bcd0w8zPwP7t-FgwONhZOHt9rz9E)

На этом занятии мы с вами познакомимся со специальной коллекцией \_\_slots\_\_, которую можно прописывать в любом классе языка Python. Для простоты восприятия информации возьмем простой класс для представления точки на плоскости:

**class** Point:     **def** \_\_init\_\_(self, x, y):         self.x = x         self.y = y

Пока в этом классе нет коллекции \_\_slots\_\_, поэтому при создании его экземпляра можно свободно оперировать существующими свойствами:

pt = Point(1, 2) pt.x pt.y = 100

И создавать новые:

pt.z = 4

Как известно, список всех локальных свойств экземпляра класса хранится в его магическом списке \_\_dict\_\_:

pt.\_\_dict\_\_

Но что, если мы хотим объявить класс точки на плоскости, чтобы у его экземпляров были только два свойства x и y и никакие другие. Как это сделать? Для этого, как раз и применяется коллекция \_\_slots\_\_.

Давайте я создам для сравнения еще один класс, в котором укажу эту коллекцию:

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

Смотрите, мы в кортеже перечисляем атрибуты с именами x и y (указываются в виде строки) и только такие локальные свойства могут существовать в экземплярах этого класса. Проверим это. Создадим экземпляр:

pt2 = Point2D(10, 20)

Выведем свойства x и y, убедимся, что они присутствуют в экземпляре:

pt2.x pt2.y

Но вот добавить новое уже не получится:

pt2.z = 30

Это произошло как раз по тому, что в классе прописана коллекция \_\_slots\_\_ и мы ее можем даже вывести:

pt2.\_\_slots\_\_

А вот привычная нам коллекция

pt2.\_\_dict\_\_

будет отсутствовать. То есть, класс и его экземпляры ведут себя немного по-другому. Мы совершенно спокойно можем изменять, удалять и добавлять локальные свойства x, y:

pt2.x = 50**del** pt2.y pt2.y = 100

Но только их и никакие другие. Причем, обратите внимание, речь идет только о локальных свойствах экземпляров. В сам класс мы по-прежнему можем добавлять любые атрибуты, например, MAX\_COORD:

**class** Point2D:     \_\_slots\_\_ = ('x', 'y')     MAX\_COORD = 100       **def** \_\_init\_\_(self, x, y):         self.x = x         self.y = y

Тогда в экземплярах этого класса появится это дополнительное свойство со значением 100.

Коллекция \_\_slots\_\_ помимо ограничения создаваемых локальных свойств, еще уменьшает объем памяти, занимаемый экземпляром класса. Смотрите, если создать  два экземпляра:

pt = Point(1, 2) pt2 = Point2D(10, 20)

То первый будет содержать ссылку, как на объект класса, так и на коллекцию \_\_dict\_\_. А второй – только на пространство имен класса Point2D:

**print**(pt.\_\_sizeof\_\_(), pt.\_\_dict\_\_.\_\_sizeof\_\_())         **print**(pt2.\_\_sizeof\_\_())

И еще коллекция \_\_slots\_\_ ускоряет работу с локальными свойствами экземпляров класса. Например, если добавить в каждый класс метод:

**def** calc(self):         self.x += 1         **del** self.y         self.y = 0

И замерить скорость его работы:

t1 = timeit.timeit(pt.calc) t2 = timeit.timeit(pt2.calc)   **print**(t1, t2)

То мы увидим, что t1 больше, чем t2, то есть, класс Point2D работает быстрее с локальными свойствами, чем класс Point.

Вот такие три особенности дает коллекция \_\_slots\_\_ экземплярам класса:

* ограничение создаваемых локальных свойств;
* уменьшение занимаемой памяти;
* ускорение работы с локальными свойствами.

На следующем занятии мы продолжим эту тему и посмотрим, как эта коллекция ведет себя при наследовании классов.

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