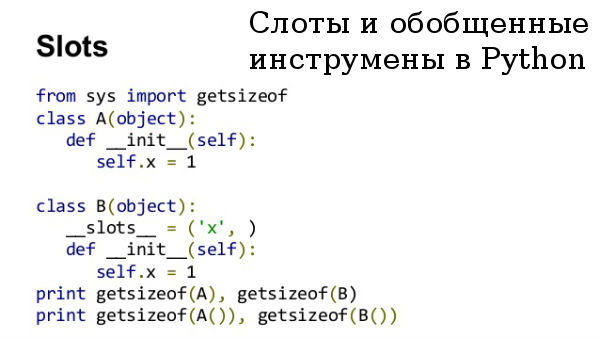
[](http://python-3.ru/uploads/sloty-i-obobshennye-instrumenty-v-python.jpg)

Фактически **некоторые экземпляры со слотами** вообще могут не иметь атрибут словаря \_\_dict\_\_, что может сделеать некоторые метапрограммы намного более сложными.

Обобщенные инструменты, которые получают списки атрибутов или обращаются к атрибутам, используя имена в виде строк, например, должны использовать более универсальные механизмы, чем атрибут \_\_dict\_\_. К таким механизмам можно отнести встроенные функции getattr, setattr и dir, способные отыскивать атрибуты в обоих хранилищах, \_\_dict\_\_ и \_\_slots\_\_. В некоторых случаях для полноты картины может потребоваться проверить оба источника атрибутов.

Например, экземпляры классов, где используются слоты, обычно не имеют атрибут словаря \_\_dict\_\_ - вместо него пространство для атрибутов в экземпляре выделяется с применением дескрипторов класса.

Только имена, перечисленные в списке \_\_slots\_\_, смогут использоваться как атрибуты экземпляра, однако значения этих атрибутов могут извлекаться и изменяться обычными способами.

class MyClass:

... \_\_slots\_\_ = ['a', 'b'] # По умолчанию наличие \_\_slots\_\_ означает отсутствие \_\_dict\_\_

>>> ZZZ = MyClass()

>>> ZZZ.a = 1

>>> ZZZ.a

1

>>> ZZZ.\_\_dict\_\_

AttributeError: 'ZZZ' object has no attribute '\_\_dict\_\_'

>>> getattr(ZZZ, 'a')

1

>>> setattr(ZZZ, 'b', 10)

>>> ZZZ.b

10

>>> 'a' in dir(ZZZ)

True

>>> 'b' in dir(ZZZ)

True

В отсутствие словаря с пространством имен невозможно присвоить значения атрибутам экземпляра, имена которых отсутствуют в списке слотов:

class MySecondClass:

... \_\_slots\_\_ = ['a', 'b']

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

... self.d = 4 # Невозможно добавить новый атрибут, когда отсутствует атрибут \_\_dict\_\_

>>> QQQ = MySecondClass()

AttributeError: 'QQQ' object has no attribute 'd'

Однако возможность добавлять новые атрибуты все-таки существует - для этого необходимо включить имя \_\_dict\_\_ в список \_\_slots\_\_, разрешить тем самым создать словарь с пространством имен. В этом случае действовать будут оба механизма хранения имен, однако обобщенные инструменты, такие как getattr, будут воспринимать их, как единое множество атрибутов:

>>> class MyThirdClass:

... \_\_slots\_\_ = ['a', 'b', '\_\_dict\_\_'] # Добавить \_\_dict\_\_ в слоты

... c = 250 # Атрибуты класса действуют как обычно

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

... self.d = 55 # Имя d будет добавлено в \_\_dict\_\_, а не в \_\_slots\_\_

...

>>> KKK = MyThirdClass()

>>> KKK.d

55

>>> KKK.\_\_dict\_\_ # Некоторые объекты имеют оба атрибута, \_\_dict\_\_ и \_\_slots\_\_

{'d':55} # getattr() может извлекать атрибуты любого типа

>>> KKK.\_\_slots\_\_

['a', 'b', '\_\_dict\_\_']

>>> KKK.c

250

>>> KKK.a # Все атрибуты экземпляра не определены

AttributeError: a # Пока им не будет присвоено значение

>>> KKK.a = 123

>>> KKK.b = 100

>>> getattr(KKK, 'a'), getattr(KKK, 'c'), getattr(KKK, 'd')

(123, 250, 55)

Если потребуется реализовать универсальный способ получения значений всех атрибутов экземпляров, необходимо учесть наличие двух форм хранения атрибутов или использовать функцию dir, которая дополнительно возвращает все унаследованные атрибуты(для получения ключей в следующем примере используется итератор словаря):

>>> for attr in list(KKK.\_\_dict\_\_) + KKK.\_\_slots\_\_:

... print(attr, ' => ', getattr(KKK, attr))

d => 55

a => 123

b => 100

\_\_dict\_\_ => {'d' : 55}

Поскольку любой из этих атрибутов может отсутствовать, более правильный способ выглядит, как показано ниже(функция getattr позволяет определять возвращаемое значение по умолчанию):

>>> for attr in list(getattr(KKK, '\_\_dict\_\_', [])) + getattr(KKK, '\_\_slots\_\_', []):

... print(attr, ' => ', getattr(KKK, attr))

d => 55

a => 123

b => 100

\_\_dict\_\_ => {'d' : 55}