Private, protected and public in Python

(*Resource*: http://radek.io/2011/07/21/private-protected-and-public-in-python/)

In C++ and Java, things are pretty straight-forward. There are 3 magical and easy to remember **access modifiers**, that will do the job (**public, protected and private**). But there's no such a thing in Python. That might be a little confusing at first, but it's possible too. We'll have look at how to handle this in Python.

Python does not have any mechanisms that would effectively restrict you from accessing a variable or calling a method. All of this is a matter of culture and **convention**.

Public

All class fields (attributes) and methods are public by default in Python. So when you want to make your attribute or method public, you just do nothing. See the example below:

```
class Cup:
    def __init__(self):
        self.color = None
        self.content = None

    def fill(self, beverage):
        self.content = beverage

    def empty(self):
        self.content = None
```

We have a `class Cup`. And here's what we can do with it:

```
redCup = Cup()
redCup.color = "red"
redCup.content = "tea"
redCup.empty()
redCup.fill("coffee")
```

All of this is good and acceptable, because all the attributes and methods are **public**.

Protected

Protected fields and methods are (in C++ and Java) accessible **only** from within the class and its subclasses. How to accomplish this in Python? The answer is – **by convention**. By prefixing the name of your field or method with **a single underscore**, you're telling others "*don't touch this, unless you're a subclass*". See the example below:

```
class Cup:
    def __init__(self):
        self.color = None
        self._content = None # protected variable prefixed by a single underscore

    def fill(self, beverage):
        self._content = beverage

    def empty(self):
        self._content = None
```

Same example as before, but the content of the cup is now protected. This changes virtually nothing, you'll still be able to access the variable from outside the class, only if you see something like this:

```
cup = Cup()
cup._content = "tea"
```

you explain politely to the person responsible for this, that the variable is protected and he should not access it or even worse, change it from outside the class.

Private

By declaring your fields and methods private you mean, that nobody should be able to access it from outside the class, i.e. strong *you can't touch this* policy. Prefix your field or method with **two underscores.** Python supports a technique called **name mangling**. This feature turns every field or method name (member of class) **prefixed with at least two underscores and suffixed with at most one underscore** into `_<className><memberName>` . So how to make your field or method private? Let's have a look at the example below:

```
class Cup:
    def __init__(self, color):
        self._color = color    # protected variable (with a single underscore)
        self.__content = None  # private variable (with two underscores)

    def fill(self, beverage):
        self.__content = beverage

    def empty(self):
        self.__content = None
```

Our cup now can be only filled and poured out by using `fill()` and `empty()` methods. Note, that if you try accessing `__content` from outside, you'll get an error.