02-POO-encapsulamento

March 3, 2020

1 Programação Orientada aos Objetos (POO) - parte II

Pedro Cardoso
(ISE/UAlg - pcardoso@ualg.pt)

1.1 Encapsulamento

- Em Python NÃO existem variáveis e métodos protegidos ou privados
- Para os atributos/métodos "protegidos" é usada uma convenção: nomes que comecem com *underscore* ("_")
- Para os atributos/métodos "privados" faz-se o *Name Mangling* que aos nomes que iniciam com dois *underscore* acrescenta no início um *underscore* e o nome da classe.

```
In [1]: class Carro:
            def __init__(self, cor, marca, modelo, dono, consumo, kms):
                self._cor = cor           # atributo protegido... nao devemos aceder diretamen
                self._marca = marca       # idem ...
                self._modelo = modelo
                self._dono = dono
                self._consumo = consumo
                self._kms = kms

            def __repr__(self):
                return 'A {} tem um {} {} de cor {} que gasta {}l/100Km e tem {}kms. Logo gast
                    self._dono, self._marca, self._modelo, self._cor, self._consumo, self._kms

            def __metodo_quase_privado(self):
                return 'Nao e facil chegar aqui!'

            def print_info(self):
                print('A {} tem um {} {} que gasta {}l/100Km e tem {}kms. Logo gastou {}l desd
                    self._dono, self._marca, self._modelo, self._cor, self._consumo, self._kms
```

Vejamos quais são os métodos que `Carro` tem

```
In [2]: dir(Carro)  # repare on _Carro__metodo_quase_privado'
```

```
Out[2]: ['_Carro__metodo_quase_privado',
         '__class__',
         '__delattr__',
         '__dict__',
         '__dir__',
         '__doc__',
         '__eq__',
         '__format__',
         '__ge__',
         '__getattribute__',
         '__gt__',
         '__hash__',
         '__init__',
         '__init_subclass__',
         '__le__',
         '__lt__',
         '__module__',
         '__ne__',
         '__new__',
         '__reduce__',
         '__reduce_ex__',
         '__repr__',
         '__setattr__',
         '__sizeof__',
         '__str__',
         '__subclasshook__',
         '__weakref__',
         'print_info']
```

Podemos aceder a todos os atributos e métodos ("privados" ou "protegidos") mas devemos ter cuidado. A ideia é que "somos todos adultos"

```
In [3]: carro_a = Carro('vermelha', 'Fiat', '500', 'Claudia', 6, 20000)

        print(carro_a) # o "mesmo que" print(carro_a.__repr__())
```

```
A Claudia tem um Fiat 500 de cor vermelha que gasta 6l/100Km e tem 20000kms. Logo gastou 1200.0
```

```
In [4]: carro_a.print_info()
```

```
A Claudia tem um Fiat 500 que gasta vermelhal/100Km e tem 6kms. Logo gastou 20000l desde que o
```

Não deviamos aceder a um atributo "protegido"

```
In [5]: carro_a._cor

Out[5]: 'vermelha'
```

E muito menos aceder a algo "privado"

```
In [6]: # carro_a.__metodo_quase_privado() não existe
        carro_a._Carro__metodo_quase_privado()
```

```
Out[6]: 'Nao e facil chegar aqui!'
```

1.1.1 getter e setters

Para aceder às variáveis "protegidas"/"privadas" usam-se pois, por norma, getters e setters. E em python temos ainda *properties*

```
In [7]: class Carro:

            def __init__(self, cor, marca):
                self.cor = cor  # chama a propriedade (valida dados). E guarda o valor em self
                self.marca = marca # chama a propriedade (valida dados).E guarda o valor em se

            def get_cor(self):
                """devolve o valor de __cor"""
                return self.__cor

            def set_cor(self, cor):
                """define o valor de __cor.

                 Parameters
                ----------
                param1 : valores admissiveis ['vermelha', 'branca', 'amarela']
                    valor da cor
                """
                if cor.lower() in ['vermelha', 'branca', 'amarela']:
                    print('Cor válida')
                    self.__cor = cor
                else:
                    print('Cor inválida')
                    raise

            def get_marca(self):
                """devolve o valor de __marca"""
                return self.__marca

            def set_marca(self, marca):
                """define o valor de __marca

                 Parameters
                ----------
                param1 : valores admissiveis ['audi', 'fiat', 'seat', 'ferrari']
                    valor da marca
                """
```

```
            if marca.lower() in ['audi', 'fiat', 'seat', 'ferrari']:
                self.__marca = marca
            else:
                raise

        cor = property(get_cor, set_cor)
        marca = property(get_marca, set_marca)
```

"Dentro" carro temos métodos e ...

```
In [8]: print(dir(Carro))
```

```
['__class__', '__delattr__', '__dict__', '__dir__', '__doc__', '__eq__', '__format__', '__ge__
```

... proriedades

```
In [9]: type(Carro.cor)
```

```
Out[9]: property
```

E se instanciamos a classe

```
In [10]: c1 = Carro('vermelha', 'Fiat')
```

```
Cor válida
```

passamos a ter também atributos *mangled*

```
In [11]: print(dir(c1))
```

```
['_Carro__cor', '_Carro__marca', '__class__', '__delattr__', '__dict__', '__dir__', '__doc__',
```

para redefenir a cor podemos usar a *property* ou aceder diretamente ao setter

```
In [12]: c1.cor = 'branca' # ok!
         c1.set_cor('branca') # ok!
```

```
Cor válida
Cor válida
```

do mesmo modo podemos aceder ao valor da cor

```
In [13]: c1.cor
```

```
Out[13]: 'branca'
```

ao usar o setter (a *property*) podemos validar as entradas

```
In [14]: try:
             c1.cor = 'verde'
         except:
             print('Erro: essa cor nao existe')
```

```
Cor inválida
Erro: essa cor nao existe
```

e obviamente podemos fazer igual raciocinio para a marca

```
In [15]: c1.marca = 'Seat'  # ok!
```

```
In [16]: try:
             c1.marca = 'Ferrary'
         except:
             print('Erro: essa marca nao existe')
```

```
Erro: essa marca nao existe
```

Exercícios
Reimplemente a classe Carro encampsulando as variáveis da mesma

```
class Carro:
    def __init__(self, cor, marca, modelo, dono, consumo, kms):
        self.cor = cor
        self.marca = marca
        self.modelo = modelo
        self.dono = dono
        self.consumo = consumo
        self.kms = kms

    def print_info(self):
        print('A {} tem um {} {} que gasta {}l/100Km e tem {}kms.'.format(
            self.dono, self.marca, self.modelo, self.consumo, self.kms))
```

1.2 Solução mais *Pythoniana*

Uma solução mais Pythoniana usa decoradores

```
In [17]: class Carro:

             def __init__(self, cor, marca):
                 self.cor = cor  # chama a propriedade (valida dados). E guarda o valor em sel
                 self.marca = marca # chama a propriedade (valida dados).E guarda o valor em s

             @property
             def cor(self):
```

```
            return self.__cor

        @cor.setter
        def cor(self, cor):
            print('debug: setting a cor')
            if cor.lower() in ['vermelha', 'branca', 'amarela']:
                self.__cor = cor
            else:
                raise

        @cor.deleter
        def cor(self):
            print('debug: a colocar a cor a None')
            self.__cor = None

        @property
        def marca(self):
            return self.__marca

        @marca.setter
        def marca(self, marca):
            print('debug: setting a marca')
            if marca.lower() in ['audi', 'fiat', 'seat', 'ferrari']:
                self.__marca = marca
            else:
                raise
```

```
In [18]: c = Carro('vermelha', 'fiat')
```

```
debug: setting a cor
debug: setting a marca
```

```
In [19]: c.cor='branca'
```

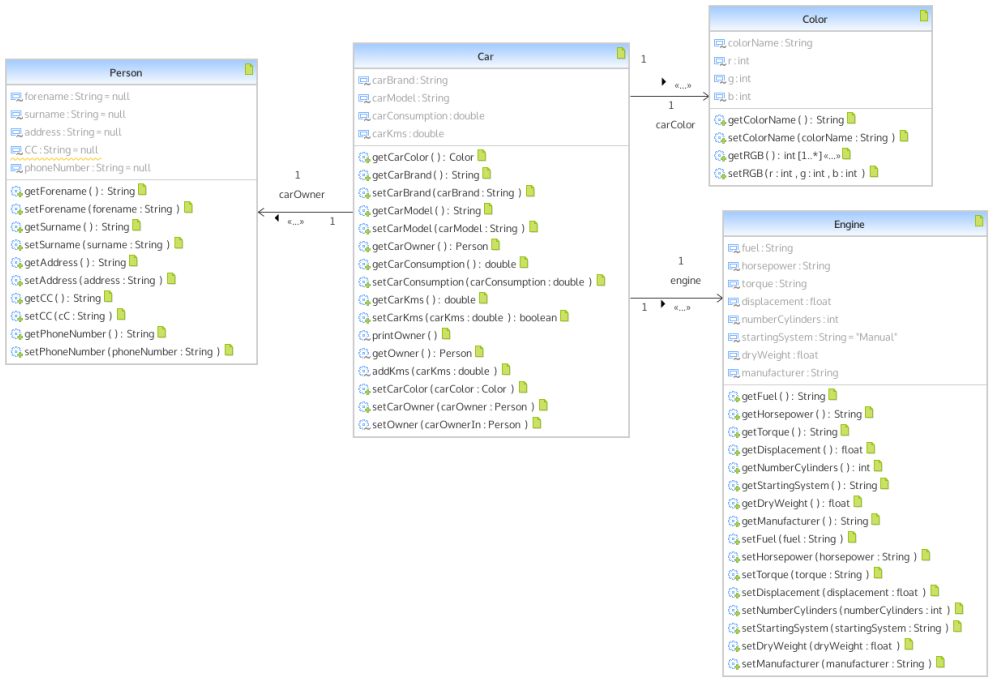
```
debug: setting a cor
```

c.cor

```
In [20]: print(c.cor)
```

```
branca
```

```
In [21]: try:
             c.cor = 'azul'
         except:
             print("Essa cor é inválida!")
```



title

```
debug: setting a cor
Essa cor é inválida!
```

```
In [22]: del(c.cor)
```

```
debug: a colocar a cor a None
```

```
In [23]: print(c.cor)
```

```
None
```

2 Exercício

Implemente as classes apresentadas no diagrama da Figura (use o pycharm ou outro IDE avançado). Crie um programa que permita de modo interativo listar, inserir, remover, e editar carros de uma lista. Crie ainda uma opção para gravar essa lista num ficheiro (veja o pacote pickle)

```
In [ ]:
```