

Programação Orientada a Objetos - Python

Emerson J. da Silva

PPGMC - Engenharia de Software

11 de Maio, 2018

Programação Orientada a Objetos - Python

Sumário:

- ❑ CRIAÇÃO DE CLASSES
- ❑ ENCAPSULAMENTO
- ❑ INSTÂNCIAS E ACESSO A OBJETOS
- ❑ CLASSE ABSTRATA
- ❑ HERANÇA E POLIMORFISMO
- ❑ SOBRECARGA DE OPERADORES

Programação Orientada a Objetos - Python

Antes de começar...


- ❑ Python não possui tipos primitivos, tudo é objeto;
- ❑ Em Python não manuseamos ponteiros, somente referências para objetos;
- ❑ Python é interpretado e dinâmico;
- ❑ Grande variedade de bibliotecas com fácil utilização.

Criação de Classes


Classe Old-Style

x

Classe New-Style (Python 2.2+)



```
1 class Veiculo:
2     'Classe base para todos os veiculos.'
3
4     def __init__(self, nome, marca):
5         self.nome = nome
6         self.marca = marca
7
8     def exibirVeiculo(self):
9         print "Nome: ", self._nome
10        print "Marca: ", self._marca
11
12    def __del__(self):
13        class_name = self.__class__.__name__
14        print class_name, "destruido!"
15
```



```
1 class Veiculo(object):
2     'Classe base para todos os veiculos.'
3
4     def __init__(self, nome, marca):
5         self.nome = nome
6         self.marca = marca
7
8     def exibirVeiculo(self):
9         print "Nome: ", self.nome
10        print "Marca: ", self.marca
11
12    def __del__(self):
13        class_name = self.__class__.__name__
14        print class_name, "destruido!"
15
```

Criação de Classes

Classe New-Style



```
1 class Veiculo(object):
2     'Classe base para todos os veiculos.'
3
4     def __init__(self, nome, marca):
5         self.nome = nome
6         self.marca = marca
7
8     def exibirVeiculo(self):
9         print "Nome: ", self.nome
10        print "Marca: ", self.marca
11
12    def __del__(self):
13        class_name = self.__class__.__name__
14        print class_name, "destruido!"
15
```

- ❑ Parâmetro `self`: Obrigatório em todos os métodos da classe;
- ❑ Funções ou atributos com 2 underlines antes e após o nome são nativos da classe;
 - ❑ Exemplos: `__init__`, `__del__`, `__doc__`, `__name__`, `__dict__`, etc.
- ❑ `__init__(self)`: Chama o construtor e inicializa os atributos da classe;
- ❑ `__del__(self)`: Destrutor.

Obs.: Objetos são destruídos automaticamente, processo denominado Garbage Collection.

Criação de Classes

Classe New-Style



```
1 class Veiculo(object):
2     'Classe base para todos os veiculos.'
3
4     def __init__(self, nome, marca):
5         self.nome = nome
6         self.marca = marca
7
8     def exibirVeiculo(self):
9         print "Nome: ", self.nome
10        print "Marca: ", self.marca
11
12    def __del__(self):
13        class_name = self.__class__.__name__
14        print class_name, "destruido!"
15
```

❑ `__init__` vs `__new__`:

- ❑ `__init__`: Recebe uma instância já construída. Sua função é inicializar os atributos da instância.
- ❑ `__new__`: Constrói o objeto em si, raramente precisamos implementar esse método.

Criação de Classes

Classe New-Style



```
1 class Veiculo(object):
2     'Classe base para todos os veiculos.'
3
4     def __init__(self, nome, marca):
5         self.nome = nome
6         self.marca = marca
7
8     def exibirVeiculo(self):
9         print "Nome: ", self.nome
10        print "Marca: ", self.marca
11
12    def __del__(self):
13        class_name = self.__class__.__name__
14        print class_name, "destruido!"
15
```

- ❑ Em Python é possível criar atributos dinamicamente. Assim, pode ser útil criar classes "vazias" (não muito comum).

Encapsulamento

```
1 class Veiculo(object):
2     'Classe base para todos os veiculos.'
3
4     def __init__(self, nome, marca):
5         self._nome = nome        #Protected
6         self.__marca = marca    #Private
7
8     def exibirVeiculo(self):
9         print "Nome: ", self._nome
10        print "Marca: ", self.__marca
11
12    def __del__(self):
13        class_name = self.__class__.__name__
14        print class_name, "destruido!"
15
```

❑ Visibilidade de atributos e métodos:

(CONVENÇÃO E NAME MANGLING)

- ❑ **Public:** sem underline no início;
- ❑ **Protected:** 1 underline no início;
- ❑ **Private:** 2 underlines no início;

Instâncias e Acesso a Objetos

```
1 class Veiculo(object):
2     'Classe base para todos os veiculos.'
3
4     def __init__(self, nome, marca):
5         self._nome = nome        #Protected
6         self.__marca = marca     #Private
7
8     def exibirVeiculo(self):
9         print "Nome: ", self._nome
10        print "Marca: ", self.__marca
11
12    def __del__(self):
13        class_name = self.__class__.__name__
14        print class_name, "destruido!"
15
```

❑ Instanciando e Acessando Objetos:

```
#Cria instâncias para Veiculo
v1 = Veiculo('Fusion', 'Ford')
v2 = Veiculo('Fusca', 'Volkswagen')

#Acessa atributos e métodos
v1.exibirVeiculo()
print v2._nome    #Acessa normalmente
print v2.__marca  #Erro
```

Mas `_nome` não deveria ser protegido?

Classe Abstrata

```
1 import abc
2
3 class Veiculo(object):
4     'Veiculo agora e uma classe abstrata.'
5
6     __metaclass__ = abc.ABCMeta #Utiliza modulo abc
7
8     def __init__(self, nome, marca):
9         self._nome = nome      #Protected
10        self.__marca = marca    #Private
11
12    #Método abstrato
13    @abc.abstractmethod
14    def exibirVeiculo(self):
15        return
16
17    def __del__(self):
18        class_name = self.__class__.__name__
19        print class_name, "destruído!"
```

❑ Python não tem interfaces.

- ❑ Mas possibilita herança múltipla e classes abstratas.

❑ Criação de classes abstratas:

- ❑ Utiliza-se o módulo abc (abstract base classes).
- ❑ O método `exibirVeiculo` agora é abstrato e assim, todas as classes derivadas de `Veiculo` devem implementar essa função.

Herança

Classes derivadas herdam os métodos da classe base!

```
1 class Veiculo(object):
2     'Veiculo: Classe base.'
3
4     def __init__(self, nome, marca):
5         self._nome = nome      #Protected
6         self._marca = marca    #Protected
7
8     def exibirVeiculo(self):
9         print "Nome: ", self._nome
10        print "Marca: ", self._marca
11
12    def alterarNome(self, novoNome):
13        self._nome = novoNome
14
15    def __del__(self):
16        class_name = self.__class__.__name__
17        print class_name, "destruido!"
18
```

```
20 class Caminhao(Veiculo):
21     'Caminhao: Classe derivada de veiculo'
22
23     def __init__(self, nome, marca, capacidadeBau):
24         #Veiculo.__init__(self, nome, marca)      #Alternativa para old-style
25         #super().__init__(nome, marca)            #Python 3
26         super(Caminhao, self).__init__(nome, marca) #Python 2
27         self.__capacidadeBau = capacidadeBau      #Private
28
29     def exibirVeiculo(self):
30         print "Nome: ", self._nome
31         print "Marca: ", self._marca
32         print "Capacidade do Bau: ", self.__capacidadeBau
33
34     def __del__(self):
35         class_name = self.__class__.__name__
36         print class_name, "destruido!"
37
```

Polimorfismo

```
1 class Veiculo(object):
2     'Veiculo: Classe base.'
3
4     def __init__(self, nome, marca):
5         self._nome = nome      #Protected
6         self._marca = marca    #Protected
7
8     def exibirVeiculo(self):
9         print "Nome: ", self._nome
10        print "Marca: ", self._marca
11
12    def alterarNome(self, novoNome):
13        self._nome = novoNome
14
15    def __del__(self):
16        class_name = self.__class__.__name__
17        print class_name, "destruido!"
18
```

```
20 class Caminhao(Veiculo):
21     'Caminhao: Classe derivada de veiculo'
22
23     def __init__(self, nome, marca, capacidadeBau):
24         #Veiculo.__init__(self, nome, marca)      #Alternativa para old-style
25         #super().__init__(nome, marca)            #Python 3
26         super(Caminhao, self).__init__(nome, marca) #Python 2
27         self.__capacidadeBau = capacidadeBau      #Private
28
29     def exibirVeiculo(self):
30         print "Nome: ", self._nome
31         print "Marca: ", self._marca
32         print "Capacidade do Bau: ", self.__capacidadeBau
33
34     def __del__(self):
35         class_name = self.__class__.__name__
36         print class_name, "destruido!"
37
```

```
38 #Cria instâncias
39 v = Veiculo('nomeVeiculo', 'nomeMarca')
40 c = Caminhao('nomeCaminhao', 'marcaCaminhao', '5 toneladas')
41 #Acessa métodos
42 v.exibirVeiculo()
43 c.exibirVeiculo()
44 c.alterarNome('novoNome')
```

Sobrecarga de Operadores

```
1 class Vector:
2     def __init__(self, a, b):
3         self.a = a
4         self.b = b
5
6     def __str__(self):
7         return 'Vector (%d, %d)' % (self.a, self.b)
8
9     def __add__(self, other):
10        return Vector(self.a + other.a, self.b + other.b)
11
12 v1 = Vector(2,10)
13 v2 = Vector(5,-2)
14 print v1 + v2
```

Saída: Vector(7, 8)

- ❑ O que acontece quando tentamos imprimir `v1 + v2` sem sobrecarregar o operador `__add__`?
- ❑ O que acontece quando tentamos imprimir `v1 + v2` sem sobrecarregar o operador `__str__`?