

Teste de Software para Web

Revisão Python - Classes e Objetos

MSc. Jonathas Silva dos Santos



Definindo classes

- Objetos fazem parte do mundo real;
- Abstração é a palavra-chave;

E tudo são objetos

- Tudo em Python é no fim um objeto;
 - `"hello".upper()`
 - `list3.append('a')`
 - `dict2.keys()`
- Eles parecem como chamadas de método em Java ou C++ !
- Novos objetos (classes) podem ser facilmente adicionados aos tipos de dados já existentes em Python!
- De fato, programar em Python é normalmente feito de forma orientada objetos!



Definindo uma classe

- **Classe:** Carro
 - **Objeto:** Ford Ka
 - **Atributos:** Cor, Marca, Ano, Modelo
 - **Métodos:** Acelerar, Frear, Ligar, Desligar
- **Classe:** Celular
 - **Objeto:** Iphone 13
 - **Atributos:** SO, Cor
 - **Métodos:** Chamar, Mandar Mensagem



Métodos em classes

- Definir um método em uma classe , basta incluir a definição da função seguindo o escopo de bloco da classe.
- Em todos métodos associados à instância definido dentro de uma classe devem ter o argumento `self` definido como primeiro argumento.
- Há geralmente um método especial `__init__` definido na maioria das classes.



Definição de uma classe

Automovel
+ placa : str
__init__(str) : None
get_placa() : str
dirigir(int) : None

métodos

```
class Automovel:
```

construtor

```
    def __init__(self, placa='XX-123'):  
        self.placa = placa
```

```
    def get_placa(self):  
        return self.placa
```

self

```
    def dirigir(self, velocidade):  
        print 'Estou dirigindo a %d' \  
              ' km/h' % velocidade
```

Criando e Deletando instâncias

Instanciando Objetos

- Não há “**new**” como feito em Java!
- `a = student(“Sheldon”, 34)` (** sem o operador new!)
- “**__init__**” serve como construtor de uma classe. Geralmente faz o trabalho de inicialização.
- Não há limite para o número de argumentos passados para o método `__init__`. Como em qualquer outra função, os argumentos podem ser definidos com valores default, tornando-os assim opcionais ao chamador

Instanciando Objetos

- **self** : O primeiro argumento de qualquer método é a referência para a própria instância da classe
- Em “__init__” self referencia o objeto criado recentemente, e em outros métodos, referencia a instância de qual o método foi invocado.
- Similar ao `this` usado em Java ou C++
- Porém Python usa mais `self` do que Java com `this`

`__init__`

```
>>> class Carro:
...     def __init__(self):
...         self._nrodas = 4
...     def set_nrodas(self, n):
...         self._nrodas = n
>>> gol = Carro()
>>> gol._nrodas
4
>>> gol.set_nrodas(10)
>>> gol._nrodas
10
```

self

- Não é necessário incluí-lo no método que faz a chamada do mesmo, apenas na definição!
- Python passa ele automaticamente.
- `a = Automovel()`
- `print a.get_placa()`

Deletando instâncias

- Quando estiver finalizado com o objeto, você não precisa deletá-lo ou liberá-lo explicitamente.
- Python possui *garbage collection* de forma automática.
- Python irá automaticamente detectar quando todas as referências para um trecho de memória estiver não sendo mais referenciado. Automaticamente, a memória é liberada.
- Poucos *leaks* de memória, e não há métodos “destrutores” em Python!

Acesso de atributos e métodos

Acessibilidade

```
a = Automovel()
```

```
print a.n_rodas
```

Acesso de métodos e atributos


- Diretamente objeto.atributo ou por algum método objeto.getAtributo()

Acessibilidade

Atributos (class e ou instâncias)

- Privados
 - Atributos e métodos só podem ser acessados dentro da classe, usa-se “__” no início do nome.
- Protected
 - Apenas convenção e usa-se apenas um “_” no nome de métodos ou atributos

```
>>> class Carro: #Classe
...     _nrodas = 4
...     __nparafusos = 3000
...
>>> gol = Carro() #Instância
>>> gol.__nparafusos #Error
```



Como declarar os
membros de uma
classe ?!

Atributos

- Exceto métodos, todos os demais dados dentro de uma classe são armazenados como atributos.
- Atributos de instância
 - Variáveis que pertencem a uma instância particular da classe
 - Cada instância tem o seu próprio valor para o atributo
 - Os mais freqüentes em classes
- Atributos de classe
 - Variáveis que pertencem à classe como um todo.
 - Todas as instâncias da classe compartilham o mesmo atributo (valor).
 - Conhecidos como “estáticos” em outras linguagens




Atributos

- Atributos de instância são criados e inicializados pelo método `__init__()`
- Simplesmente atribuindo um valor a um rótulo
- Dentro da classe, referir-se ao atributo usando `self`
- Exemplo: `self.full_name`

Atributos

- Atributos de classe são compartilhados (apenas uma cópia) por todas as instâncias da classe.
- Qualquer instância alterá-lo, o valor é alterado para todas instâncias.
- Atributos de classe são definidas:
 - Dentro da definição de uma classe
 - Fora de quaisquer métodos da classe
- Já que estes atributos são compartilhados por todas instâncias de uma classe, eles são acessados através de uma notação diferente:
 - `self.__class__.name`



Python é uma
linguagem de
programação...



Herança

- Uma classe pode **herdar** a definição de outra classe
 - Permite o uso ou a extensão de métodos e atributos previamente definidos por outra classe.
 - Nova classe: subclasse. Original: classe pai, ancestral ou superclasse
 - Para definir uma subclasse, coloque o nome da superclasse entre parênteses depois do nome da subclasse na primeira linha da definição.
 - Python não tem a palavra 'extends' como em Java
 - Múltipla herança é suportada





Herança

<#>

```
>>> class Veiculo:
...     def andar(self): print "andei"
...
>>> class Carro(Veiculo):
...     _nrodas = 4
...
>>> gol = Carro()
>>> gol.andar()
andei
```