Python para Modelagem Baseada em Agentes `aula 9` Classes

Furtado, Bernardo Alves

February 4, 2019

ipea Institute for Applied Economic Research

Menu do dia

- Classes
  - + exemplos
  - Debugging in PyCharm
  - Class exercise Transactions: walkthrough

Quase lá: o que não vimos

- Pandas, numpy
- Matplotlib
- Web scraping
- Networkx, graph
- Database, SQL
- SciPy
- ...

`class`: conceitos

- É que como se você criasse seu próprio Python object
- Por exemplo: listas e dicionários?
- São python classes, com métodos especificos. Teste.
- Para seu caso especial, um objeto, com atributos específicos, pode ser necessário
- Encaixa como uma luva no conceito de agentes

`class`: exemplo `my_first_class.py`

- Classes contém MÉTODOS (funções, alterações nos atributos)
- E dados. Informação
- Animal: nome, idade, ato de envelhecer, o que come
- Conceito de `self`
- refere-se a UMA versão do tipo classe tal.
- classe (cerveja): self: (uma instância) = um objeto cerveja. Que tem marca, gradação alcoólica, ...

`class`: exemplo `my_second_class.py`

- Métodos internos __add__
- __repr__
- __eq__
- `my_first_real_class.py`

Template

- `class_template.py`

Show how to stop programme and see what happens

Demonstrate debug within PyCharm!

Accounts I

1. Objetivo: criar agentes e interações. Agentes vão às compras e ganham satisfação. Lojas, com capacidade limitada recebem os clientes. Vendem satisfação. Análise da interação (simples) na medida em que se aumentam clientes.

Accounts II

2. Três classes distintas: `Account, Shop, Agent`
3. Account is simple. Inicia-se com `self.balance = 0` e um número de registro `self.registration = i`
4. Account também deverá ter dois métodos:
   - 4.1 Um que recebe recursos `amount` e acrescenta ao balanço: `self.balance += amount`
   - 4.2 E outro que retira recursos (amount) do balanço (e retorna a quantia retirada) `self.balance -= amount`
5. Tanto as classes `Agent`, quanto `Shop` terão uma `account`
6. E precisarão de um número de registro `i`

Accounts III

7. Portanto, o primeiro método `__init__` de `Agent` e `Shop` deverá constar como uma instância da class `Account`
   - Como precisa de um parâmetro, o registro, fica `self.account = Account(i)`
   - Notem que, para acessar o método depósito da `account`, será necessário algo como:
   - `bob.account.deposit(quantia)`
   - Do mesmo modo, na `Shop`
   - `loja.account.deposit(quantia)`

Accounts IV

8. A class `Shop` deverá ter ainda no seu `__init__`, três variáveis: `capacity, fun, cost`.
9. Todas três podem ser geradas, no momento de criação da instância, por meio do `random.randrange(início, fim)`

Accounts V

10. Crie métodos, de forma que:
    - 10.1 Sempre que ocorrer uma visita de um cliente, diminua a capacidade por 1: `self.capacity -= 1`
    - 10.2 Se a capacidade chegar a zero, retorne `False`. Senão, retorne `True`
    - 10.3 Antes de permitir a visita e a compra, check se a capacidade permite
    - 10.4 Por fim, note que, dado o acesso às `accounts` do cliente e da loja, a transação não ocorre na classe, mas fora dela.

Accounts VI

11. O `Agent` tem pelo menos um método para acumular a `fun`, satisfação que recebe ao visitar cada `Shop`
12. Adicionalmente, pode-se ter um método que verifica se os recursos do `Agent` são suficientes para bancar o `Shop.cost`

Interactions I

- Outro módulo, `import` as classes e contém algumas funções:
- Uma para criar os agentes e as lojas, a partir de um número `n` e controla o id específico `i`
- Criam-se listas
- Faz-se um `loop` utilizando `n, i`, `append` às listas, retorna
- Outra função, acessa as contas dos agentes e deposita recursos, por exemplo:
- `a[i].account.deposit(random.randrange(10, 50))`
- Por fim, talvez a função mais difícil, a interação entre agentes e lojas.

Interactions II

- Por exemplo, para todos os agentes, escolhe-se uma loja aleatória: `random.choice(listalojas)`
- Então, verifica-se a capacidade da loja, os recursos do agente, faz-se a visita, computa-se o gasto e adquire-se satisfação!
- Por exemplo:
- `s1.account.deposit(a[i].account.pay(s1.cost))`
  `a[i].get_fun(s1.fun)`
- Por fim, escreva uma função que tire a média da satisfação do agente, do custo das lojas e das contas.

Generalization I

- Simplesmente, crie uma rotina que testa números de agentes e de lojas, roda a interação e verificam-se os resultados:
- `Average fun, average balance, average cost`
- Salve em um arquivo.
- Possivelmente, depois, será possível realizar plots(gráficos)
- Um pouco mais detalhado:
  - Crie uma lista com vários números de `Agents`
  - Claro, com alguma relação óbvia. Por exemplo, aumentando linearmente de tamanho. Ou exponencialmente

Generalization II

- Crie uma lista com vários números de `Shops`
- Em um `for loop`, chame a função principal do módulo interactions (que recebe com parâmetros número de agentes e lojas).