02

Métodos estáticos

DOWNLOAD

Segue o link (https://s3.amazonaws.com/caelum-online-public/python/12-python.zip) com os arquivos desta aula.

Introdução

Aprendemos a ler e a escrever em arquivos, mas agora utilizaremos esta funcionalidade dentro do nosso modelo `Perfil`. A ideia é encapsular a leitura do arquivo CSV e criar um novo perfil para cada linha do arquivo. Lembrando da estrutura do nosso arquivo CSV:

```
Ana Gonçalves, 21-34345432, Amigas Ltda
Camila Almeida, 21-21215643, Auron Ltda
...
```

Ou seja, cada linha desse arquivo devemos criar um novo perfil. Por exemplo, para a primeira linha criaremos o perfil baseado nos dados da Ana:

```
>>> Perfil(nome='Ana Paula Gonçalves', telefone='21-34345432', empresa='Amigas Ltda')
```

O perfil criado guardaremos dentro de uma lista. Vamos criar uma lista nova chamada de `perfis`, usando o método `append` para adicionar o perfil na lista:

```
>>> perfis = []
>>> perfil = Perfil(nome='Ana Paula Gonçalves', telefone='21-34345432', empresa='Amigas Ltda')
>>> perfis.append(perfil)
```

Leitura de perfis e o método split

Também já sabemos ler o arquivo linha a linha. Basta abrirmos o arquivo no modo de leitura para depois iterar em cada linha usando um laço `for`:

```
>>> arquivo = open(nome_arquivo,'r')
>>> for linha in arquivo:
...   print(linha)
```

O problema ainda é que temos uma linha com os 3 valores: `nome`, `telefone` e `empresa`. É preciso separar essas valores pela vírgula, para instanciar um perfil. Felizmente a classe `string` possui um método `split` que sabe separar a `string` por um caracter. O caracter de separação é a vírgula, então o método `split` recebe este caracter:

```
>>> arquivo = open(nome_arquivo,'r')
>>> for linha in arquivo:
...    valores = linha.split(',')
```

A variável `valores` é um array com 3 elementos, na primeira posição tem o nome, na segunda o telefone e na última a empresa, exatamente o que precisamos para o construtor da classe `Perfil` :

```
>>> for linha in arquivo:
...     valores = linha.split(',')
...     Perfil(valores[0], valores[1], valores[2])
```

Repare que estamos passando os valores para o perfil na ordem ordinal (0, 1, 2). Porém, o Python tem uma forma mais enxuta de passar os valores no construtor. Podemos usar o símbolo * (asterisco) como atalho. Veja como fica elegante:

```
>>> for linha in arquivo:
...     valores = linha.split(',')
...     Perfil(*valores)
```

Acredite, funciona!

O nosso código está quase completo só falta fechar o arquivo:

```
>>> arquivo = open('perfis.csv','r')
>>> perfis = []
>>> for linha in arquivo:
...     valores = linha.split(',')
...        perfis.append(Perfil(*valores))
...
>>> arquivo.close()
```

Encapsulando e método estáticos

Perfeito! E para facilitar o reuso desse código vamos colocá-lo dentro de um método na classe Perfil. No final estamos criando perfis, é um comportamento relacionado a perfil. Vamos criar um novo método chamado `gerar_perfis` dentro da classe `Perfil` , que recebe o nome do arquivo como parâmetro e executa todo o código escrito anteriormente:

```
    def gerar_perfis(self, nome_arquivo):
         arquivo = open(nome_arquivo,'r')
         perfis = []
         for linha in arquivo:
             valores = linha.split(',')
             perfis.append(Perfil(*valores))
         arquivo.close()
         return perfis
```

Para usar o método devemos criar um novo perfil (!) para chamar o método. Mas faz sentido criarmos um novo perfil para chamarmos o método `gerar_perfis` ? Repare que o método `gerar_perfis` funciona como uma fábrica, é responsável pela criação então não faz sentido criar um novo perfil antes, certo?

Faria mais sentido usar o método sem ter uma instância, algo assim:

```
>>> perfis = Perfil.gerar_perfis('perfis.csv')
```

Repare que só usamos o nome da classe. O método `gerar_perfis` é da classe e não da instância!

Para isso realmente funcionar, devemos declarar o método como **estático**. O Python oferece para tal configuração um **decorador**. O decorador **@staticmethod** torna o método, um método da classe:

```
@staticmethod
def gerar_perfis(nome_arquivo):
     arquivo = open(nome_arquivo,'r')
     perfis = []
     for linha in arquivo:
         valores = linha.split(',')
         perfis.append(Perfil(*valores))
     arquivo.close()
     return perfis
```

E agora sim podemos gerar perfis sem criar uma instância antes:

```
>>> perfis = Perfil.gerar_perfis('perfis.csv')
```

Decoradores @staticmethod e @classmethod

Repare que no método usamos a classe `Perfil` para chamarmos o construtor, já que isso foi o nosso objetivo. Imagine agora que queremos usar o mesmo método para criar perfis VIP, por exemplo:

```
>>> perfis_vip = Perfil_Vip.gerar_perfis("perfis.csv")
```

O nosso método vai criar apenas perfis padrão, pois já usamos o construtor da classe `Perfil` . Podemos criar um outro método na classe `Perfil_Vip` só que não queremos repetir código. Para resolver isso foi criado um outro decorador `@classmethod` que recebe sempre a classe que está sendo utilizada como primeiro parâmetro:

```
@classmethod
def gerar_perfis(classe, nome_arquivo):
    arquivo = open(nome_arquivo,'r')
    perfis = []
    for linha in arquivo:
      valores = linha.split(',')
      perfis.append(classe(*valores))
    arquivo.close()
    return perfis
```

Assim, sempre será criado o perfil correto pelo método `gerar_perfis` :

```
>>> perfis_vip = Perfil_Vip.gerar_perfis("perfis.csv")
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
  File "models.py", line 27, in gerar_perfis
```

```
    perfis.append(classe(*valores))
TypeError: __init__() takes exactly 5 arguments (4 given)
```

Opa! Recebemos uma mensagem de erro porque o construtor da classe `Perfil_Vip` recebe um apelido, que não foi definido no arquivo CSV. Não podemos simplesmente adicionar qualquer informação no arquivo, pois o apelido só existe em nossa rede social. Será que há alguma maneira de adotarmos um valor padrão para novos perfis VIP? Sim, há. O Python permite definir um valor padrão para os parâmetros de métodos, inclusive da nossa função `__init__`, logo vamos editar o parâmetro `apelido` do construtor da classe `Perfil_Vip`:

```
class Perfil_Vip(Perfil):
    'Classe padrão para perfis de usuários VIPs'

    def __init__(self, nome, telefone, empresa, apelido=''):
        super(Perfil_Vip, self).__init__(nome, telefone, empresa)
        self.apelido = apelido

    def obter_creditos(self):
        return super(Perfil_Vip, self).obter_curtidas() * 10.0
```

Repare que o parâmetro `apelido` de nosso construtor recebe uma atribuição, no caso **vazio**. Sendo assim, todos os perfis VIPs criados terão como padrão o valor vazio como apelido. Agora podemos testar novamente:

```
#parâmetro classe será Perfil_Vip, gera Perfis_Vip
>>> perfis_vip = Perfil_Vip.gerar_perfis("perfis.csv")
>>> type(perfis_vip[0])
<class 'models.Perfil_Vip'>
```

E se usamos a classe `Perfil`, será criado uma lista de perfis comuns:

```
#parâmetro classe será Perfil
>>> perfis = Perfil.gerar_perfis("perfis.csv")
>>> type(perfis[0])
<class 'models.Perfil'>
```