01

Classe e Objeto

Transcrição

Vimos que o paradigma da Orientação a Objetos está relacionado com a organização do código, um conceito que começamos a aplicar no código. Juntamos as características — como `numero`, `titular`, `saldo` e `limite` — com as funcionalidades de uma conta.

Organizamos o código, porém, será que devemos continuar com essa abordagem? No mundo procedural, não somos obrigados a adotar OO, não há algo que reforça a necessidade de utilizarmos essa maneira mais organizada de escrever o código. Ou seja, cabe ao desenvolvedor decidir se quer adotar o paradigma OO.

Em um sistema maior, é grande a chance de que as funções fiquem separadas em arquivos e módulos diferentes do projeto. No entanto, pode ser trabalhoso encontrar onde está cada trecho do código e isso, pode resultar em retrabalho e escrever funções já existentes. O paradigma Orientado a Objetos nos incentiva a agrupar funcionalidades relacionadas em um mesmo lugar. Este é um dos principais problemas.

Mas existe ainda outro problema, nós temos a opção de acessar o saldo diretamente no console, sem chamar uma funcionalidade e definir um novo valor como veremos abaixo:

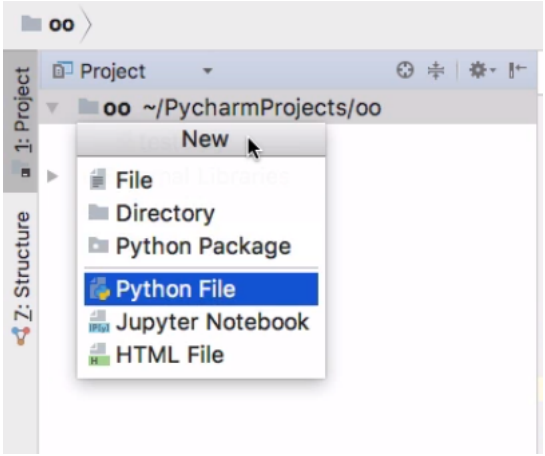
```
>>> conta["saldo"] = -300.0
```

Porém, ninguém deveria ter acesso ao saldo diretamente, sendo primeiramente necessário depositar ou sacar o dinheiro. Deveríamos manipular os dados somente por meio das funções.

Nós queremos resolver essas duas questões utilizando OO. Para isto, criaremos um objeto, no caso, falamos de uma conta. Trabalharemos com algo do mundo real, uma conta é algo concreto e inclui diversos dados. Porém, antes de termos um objeto, devemos definir as suas características.

A seguir faremos uma analogia: quando preparamos um bolo, geralmente, seguimos uma receita que define os ingredientes e o modo de preparação. A nossa conta é um objeto concreto assim como o bolo, e também precisamos definir antes uma receita. Porém, a "receita" no mundo OO recebe o nome de classe. Ou seja, antes de criarmos um objeto definiremos uma classe.

O próximo passo será gerar um novo arquivo Python, que receberá o nome de `conta.py`.



Dentro do arquivo gerado, definiremos a classe, que será antecedida pela palavra reservada `class`, utilizada na linguagem **Python** para definir a "receita". A classe vai receber o nome `Conta`, que por não ser uma palavra reservada, poderia ser outro qualquer.

Poderíamos ter adotado `ContaCorrente` como nome da classe, por exemplo. Desta forma, seguiríamos o padrão **CamelCase**, no qual cada palavra é iniciada com a letra maiúscula. Seguiremos esta boa prática e escreveremos `Conta` com a primeira letra em caixa alta, seguida pela pontuação `:`, com isto indicaremos que se trata do início de um bloco de código.

Todo o conteúdo adicionado após `:` fará parte desta classe. Para fazer o código funcionar, incluiremos a palavra-chave `pass` no arquivo `conta.py`.

```
class Conta:

    pass
```

Em seguida, reiniciaremos o console e importaremos do módulo conta.

```
>>> from conta import Conta
```

Para criarmos um primeiro objeto, no console, digitaremos o nome da classe `Conta` utilizando os parênteses (`()`) para que o código seja executado.

```
>>> Conta()
<conta.Conta object at 0x10715f518>
```

Ele imprimiu a informação de que temos um objeto baseado na classe `Conta`, localizado dentro do módulo `conta`. Observe que a letra maiúscula foi utilizada para diferenciar as duas nomenclaturas. O objeto foi criado em memória e imprime o endereço `0x10715f518`.

Seguiremos com a analogia do bolo... É como se tivéssemos passado a receita para uma fábrica, determinando a tarefa de fabricação do objeto para outro. No entanto, onde essa fábrica vai criar o bolo? Temos um endereço, mas não precisaremos manuseá-lo. A linguagem Python irá abstrair esse dado para nós.

Se quisermos trabalhar com o objeto, teremos que fazer algo a mais. Chamaremos a classe `Conta` novamente, guardando o retorno dentro da referência `conta`.

```
>>> conta = Conta()
```

Esta referência guardará o endereço em memória para localizar o objeto posteriormente. É como se a fábrica nos avisasse em qual armário o bolo foi guardado, porém, o endereço não é o objeto em si. Trata-se apenas de uma referência.

Se executarmos a linha `conta = Conta()` não teremos nenhum retorno, porque o resultado da execução da `Conta` foi atribuída a `conta`. Mas, se executarmos `conta` no console, teremos o seguinte retorno:

```
>>> conta
<conta.Conta object at  0x10715fe10>
```

Observe que o endereço retornado é diferente do que foi impresso na primeira execução. Isto ocorreu porque chamamos a classe e foi criado um segundo objeto.

Geramos dois objetos do tipo `conta` , baseada na mesma classe. Vale lembrar que uma classe pode gerar vários objetos, porém, queremos que ela tenha várias informações. É necessário que `Conta` trabalhe com vários dados para depositar e sacar. Faremos isto adiante.