03

Dados e comportamento

Transcrição

Vamos apresentar alguns problemas sobre o mundo procedural. Quando o paradigma da Orientação a Objetos nasceu a programação procedural era predominante. A abordagem OO surgiu para resolver os problemas do mundo procedural. Quais são eles?

Vimos que precisamos lembrar de detalhes como nome das chaves: `conta`, `numero`, `titular`, `limite` e `saldo`.

```
>>> from teste import cria_conta
>>> conta = cria_conta(123, "Nico", 55.0, 1000.0)
```

O próximo passo é apresentarmos uma conta, descrevemos as suas características, mas ela tem funcionalidades associadas como depositar, sacar, transferir, tirar extrato, enfim, outras ações possíveis.

Vamos focar nas principais funcionalidades. A cada nova que adicionarmos, criaremos uma nova função. A seguir, definiremos a função `deposita()`:

```
def deposita(conta, valor):
    conta ["saldo"] += valor
```

Nós adicionamos o `valor` de uma forma simplificada no código, e em vez de repetir `conta["saldo"]`, nós utilizamos `+=`. A função recebeu `conta` e o `valor` como parâmetros. Nós precisamos acessar `conta` por meio da chave `saldo`, que foi definida mais acima.

Faremos algo semelhante com a função `saca()`, porém, desta vez iremos subtrair usando `-=`.

```
def saca(conta,valor):
    conta["saldo"] -= valor
```

Optamos por fazer uma implementação simples. Em seguida, adicionaremos `extrato()`, que será responsável por imprimir as informações:

```
def extrato(conta):
    print("Saldo é {}".format(conta["saldo"]))
```

O extrato imprime as informações da `conta`, e usando o `print` imprimiremos o saldo exibindo junto com a mensagem `Saldo é`, juntamente com o retorno da função `format()`, passando o `saldo` da conta.

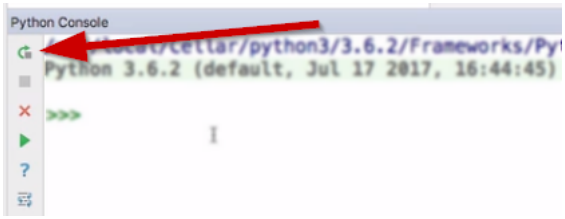
Vamos testar. Limparemos o console e importamos `cria_conta, deposita, saca, extrato`.

```
>>> from teste import cria_conta, deposita, saca, extrato
```

No entanto, teremos um retorno de mensagem de erro:

```
Traceback (most recent call last):
File "<input>", line 1, in <module>
ImportError: cannot import name 'deposita'
```

Precisamos reiniciar o console.



Pronto. Criamos novamente a conta, passando os valores.

```
>>> from teste import cria_conta, deposita, saca, extrato
>>> conta = cria_conta(123, "Nico", 55.0, 1000.0)
>>> deposita(conta, 15.0)
>>> extrato(conta)
Saldo é 70.0
>>> saca(conta, 20.0)
>>> extrato(conta)
Saldo é 50.0
```

Acrescentamos algumas funções como `deposita()`, `saca(conta, 20.0)`, `extrato(conta)`, obtendo o retorno `Saldo é 50.0`.

Nosso código funcionou conforme o esperado. Definimos as características e algumas funcionalidades de uma conta no mundo procedural.

Qual a relação com o mundo de Orientação a Objetos? A ciência deste paradigma é juntar dados e procedimentos, enfim, funcionalidades. O que fizemos agora foi baseado no conhecimento procedural que tínhamos no Python 3. No exemplo da conta, juntamos as características (número, titular, limite, saldo) e funcionalidades (sacar, depositar, tirar extrato).

Podemos programar Orientação a Objetos no mundo procedural? Não, por quê? Apesar de forçar esta ligação, ela é muito frágil na abordagem procedural. Não é obrigatório colocar as funcionalidades em um lugar só. Poderíamos colocar as funções `deposita()`, `saca()` e `extrato()` em outro arquivo. O importante é organizar o projeto, pois quando ele crescer (este código tem quatro funções, mas imagine um código com mil funções), ficará mais complexo.

Além disso, a ligação é frágil pois persiste o problema de alterar a conta, aumentando o `saldo` sem utilizar a função `deposita()`.

```
>>> conta["saldo"] = conta["saldo"] + 100.0
>>> extrato(conta)
Saldo é 150,0
```

Se você quer trabalhar com uma conta, o que pode ser feito? Depositar, sacar, tirar extrato. Temos funcionalidades planejadas, mas não precisamos criar uma conta com estas características. Podemos criar contas completamente diferentes.

Por isso, vamos criar uma segunda conta.

```
>>> conta2 = {"numero": 321, "saldo": 200.0}
```

Esta nova conta também terá `numero` e `saldo`, mas não terá `limite` e `titular`. Segundo a definição no nosso programa, esta nova conta não será considerada correta.

Se aplicarmos a função `deposita()`, passando como parâmetros `conta2, 200.0` dentro do parênteses, ela funciona. Mas se criarmos uma **conta que não possui** `saldo`, por exemplo, `conta3`:

```
>>> conta3 = {"numero": 321, "limite": 200.0}
```

Se tentarmos acessar `conta3`, receberemos uma mensagem de erro:

```
>>> conta3 = {"numero": 321, "limite": 200.0}
>>> deposita(conta3, 2000.0)

Traceback (most recent call last):
    File "<input>", line 1, in <module>
    File "/Users/caelum/PycharmProjects/oo/teste.py", line 8, in deposita
        conta["saldo"] += valor
KeyError: 'saldo'
```

Reforçamos o que uma conta tem e pode fazer, e que o mundo procedural não oferece essa ligação reforçada.

Precisamos pensar sobre o que escrevemos para não errar ligações frágeis entre funções. Faremos melhorias por meio do paradigma Orientado a Objetos.