Mas não é comum acessá-lo dessa maneira. Estes métodos iniciados e terminados com dois *underscores* são chamados pelo interpretador e são conhecidos como métodos mágicos. Existe outra função embutida do Python, a função `vars()`, que chama exatamente o `__dict__` de uma classe. Obtemos o mesmo resultado usando `vars(conta)`:

```
>>> vars(conta)
{'saldo': 1000.0, 'numero': '123-4', 'titular': <__main__.Cliente object at 0x7f0b6d028f28>, 'limite'
: 1000.0}
```

Repare que o `__dict__` e o `vars()` retornam exatamente um dicionário de atributos de uma conta como tínhamos modelado no início deste capítulo. Portanto, nossas classes utilizam dicionários para armazenar informações da própria classe.

Os demais métodos mágicos estão disponíveis para uso e não utilizaremos por enquanto. Voltaremos a falar deles em um outro momento.

7.13 EXERCÍCIO: PRIMEIRA CLASSE PYTHON

1. Crie um arquivo chamado `conta.py` na pasta `oo` criada no exercício anterior.
2. Crie a classe Conta sem nenhum atributo e salve o arquivo.

```
class Conta:
    pass
```

3. Abra o terminal e vá até a pasta onde se encontra o arquivo `conta.py`. Abra o console do Python3 no terminal e importe a classe Conta do módulo conta.

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

4. Crie uma instância (objeto) da classe `Conta` e utilize a função `type()` para verificar o tipo do objeto:

```
>>> conta = Conta()
>>> type(conta)
<class 'conta.Conta'>
```

Além disso, crie alguns atributos e tente acessá-los.

1. Abra novamente o arquivo conta.py e escreva o método `__init__()` recebendo os atributos anteriormente definidos por nós que toda conta deve ter (numero titular, saldo e limite):

```
class Conta:

    def __init__(self, numero, titular, saldo, limite):
        self.numero = numero
        self.titular = titular
        self.saldo = saldo
        self.limite = limite
```

2. Reinicie o Python3 no terminal e importe novamente a classe `Conta` do módulo conta para

testarmos nosso código:

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

3. Tente criar uma conta sem passar qualquer argumento no construtor:

```
>>> conta = Conta()
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
TypeError: __init__() missing 4 required positional arguments: 'numero', 'titular', 'saldo', and
'limite
```

   Note que o interpretador acusou um erro. O método `__init__()` exige 4 argumentos 'numero', 'titular', 'saldo' e 'limite'.

4. Agora vamos seguir o exigido pela classe, pela receita de uma conta:

```
>>> conta = Conta('123-4', 'João', 120.0, 1000.0)
```

5. O interpretador não acusou nenhum erro. Vamos imprimir o `numero` e `titular` da conta:

```
>>> conta.numero
'123-4'
>>> conta.titular
'João'
```

6. Crie o método `deposita()` dentro da classe `Conta`. Esse método deve receber uma referência do próprio objeto e o valor a ser adicionado ao saldo da conta.

```
def deposita(self, valor):
    self.saldo += valor
```

7. Crie o método `saca()` que recebe como argumento uma referência do próprio objeto e o valor a ser sacado. Esse método subtrairá o valor do saldo da conta.

```
def saca(self, valor):
    self.saldo -= valor
```

8. Crie o método `extrato()`, que recebe como argumento uma referência do próprio objeto. Esse método imprimirá o saldo da conta:

```
def extrato(self):
    print("numero: {} \nsaldo: {}".format(self.numero, self.saldo))
```

9. Modifique o método `saca() fazendo retornar um valor que representa se a operação foi ou não bem sucedida. Lembre que não é permitido sacar um valor menor do que o saldo.

```
def saca(self, valor):
    if (self.saldo < valor):
        return False
    else:
        self.saldo -= valor
        return True
```

10. Crie o método `transfere_para()` que recebe como argumento uma referência do próprio objeto, uma `Conta` destino e o valor a ser transferido. Esse método deve sacar o valor do próprio objeto e

depositar na conta destino:

```
def transfere_para(self, destino, valor):
    retirou = self.saca(valor)
    if (retirou == false):
        return False
    else:
        destino.deposita(valor)
        return True
```

11. Abra o Python no terminal, importe o módulo conta, crie duas contas e teste os métodos criados.

12. (Opcional) Crie uma classe para representar um cliente do nosso banco que deve ter `nome`, `sobrenome` e `cpf`. Instancie uma `Conta` e passe um cliente como `titular` da conta. Modifique o método `extrato()` da classe `Conta` para imprimir, além do número e o saldo, os dados do cliente. Podemos criar uma `Conta` sem um `Cliente`? E um `Cliente` sem uma `Conta`?

13. (Opcional) Crie uma classe que represente uma data, com dia, mês e ano. Crie um atributo `data_abertura` na classe `Conta`. Crie uma nova conta e faça testes no console do Python.

14. (Desafio) Crie uma classe `Historico` que represente o histórico de uma `Conta` seguindo o exemplo da apostila. Faça testes no console do Python criando algumas contas, fazendo operações e por último mostrando o histórico de transações de uma `Conta`. Faz sentido criar um objeto do tipo `Historico` sem uma `Conta`?

Agora, além de funcionar como esperado, nosso código não permite criar uma conta sem os atributos que definimos anteriormente. Discuta com seus colegas e instrutor as vantagens da orientação a objetos até aqui.