03

Construtor

Transcrição

Seguimos com nossa introdução ao mundo de Orientação a Objetos, vimos três conceitos fundamentais. A classe `Conta` está praticamente vazia, mas ela já existe e funciona como a receita do projeto, que é construído a partir da seguinte execução:

```
>>> conta = Conta()
>>> conta
<conta.Conta object at 0x10715fef0>
```

Chamaremos a classe como se fosse uma função. O objeto é criado em memória por baixo dos panos pelo Python. Ele abstrai este processo, portanto não devemos nos preocupar com isso.

O objeto será criado na hora de chamar a classe `Conta` . Em memória, o objeto é criado e o Python devolve o endereço que será guardado dentro da variável `conta` .

Em Orientação a Objetos, as variáveis são denominadas: **referências**.

Estes conceitos servem para diversas linguagens, além de Python. Se trabalharmos com PHP ou Java, faremos o mesmo. Teremos uma referência, uma classe, uma construção de objeto que poderemos aplicar em outra linguagem.

Nossa classe continua vazia, a seguir, definiremos o conteúdo dela, definindo quais são suas **características**. No mundo Orientação a Objetos, essas características são chamadas de **atributos**.

Os atributos da conta são: `numero` , `titular` , `saldo` e `limite` . Na hora de criar um objeto, o Python pode executar uma função, automaticamente, dentro da classe para definir os atributos.

Sendo uma função automática, receberá um nome especial. Adicionaremos dois caracteres `_` antes e depois do nome da função construtora, para criarmos `__init__` . O Python constrói o objeto, cria um lugar na memória e depois chama a função `__init__` . Como demonstração, segue o código:

```
class Conta:
    def __init__(self):
        print("Construindo objeto...")
```

Em seguida, reiniciaremos o console e importaremos novamente:

```
>>> from conta import Conta
>>> conta = Conta()
Construindo objeto ...
```

Automaticamente ele retornou `Construindo objeto...` , pois o Python cria o objeto em memória, encontra um espaço e, depois, a função construtora é chamada.

Criamos uma função, mas a ideia é definir os atributos e as características. Para isso, precisaremos da variável `self`, que está dentro da função `__init__()`.

Em seguida, no arquivo `conta.py`, usaremos interpolação e `format(self)` dentro de `print()`. O Python cria automaticamente `self`.

```
class Conta:
    def __init__(self):
        print("Construindo objeto...{}".format(self))
```

Reiniciaremos o console e testaremos o código.

```
>>> from conta import Conta
>>> conta = Conta()
Construindo objeto ... <conta.Conta object at 0x1020d7f28>
```

Repare que ele já conhece esta saída. Localizamos a conta, o objeto e o endereço. Considerando que é a mesma saída, ele imprimirá o valor da referência.

`self` é a referência que sabe encontrar o objeto construído em memória.

Agora que temos o endereço, utilizaremos `self` para acessar o objeto e definir seus atributos e características.

```
class Conta:

    def __init__(self):
        print("Construindo objeto...{}".format(self))
        self.numero = 123
        self.titular = "Nico"
        self.saldo = 55.0
        self.limite = 1000.0
```

Em `self.numero`, o caractere "ponto" (`.`) é um comando de ida ao objeto e `numero`, `titular`, `saldo` e `limite` são atributos.

Reiniciaremos o console e testaremos novamente, agora, com os atributos. Tudo continua funcionando, sem mudanças porque não utilizamos os atributos.

No entanto, não queremos deixar o valor dos atributos fixos, o ideal é que eles variem de acordo com a conta que está sendo criada.

Em `teste.py`, nós havíamos definido alguns parâmetros na função `cria_conta()`,:

```
def cria_conta(numero, titular, saldo, limite):
    conta = {"numero": numero, "titular": titular, "saldo": saldo, "limite": limite}
    return conta
```

De volta a `conta.py`, o próximo passo será também definir parâmetros para a função `__init__()`, aproveitando os mesmo dados da função `cria_conta()`:

```
class Conta:

    def __init__(self, numero, titular, saldo, limite):
        print("Construindo objeto...{}".format(self))
        self.numero = 123
        self.titular = "Nico"
        self.saldo = 55.0
        self.limite = 1000.0
```

Além do `self` que é passado automaticamente, queremos que os dados sejam alterados, por isso, adicionaremos os nomes dos parâmetros.

```
class Conta:

    def __init__(self, numero, titular, saldo, limite):
        print("Construindo objeto...{}".format(self))
        self.numero = numero
        self.titular = titular
        self.saldo = saldo
        self.limite = limite
```

Desta forma, o limite do objeto (`self.limite`) será o `limite` recebido do parâmetro da função `__init__()`. E qual é a origem dos dados? O valor do `self` é passado pelo Python, responsável pela criação final do objeto em memória. No entanto, os valores dos parâmetros como `numero` deverão ser definidos usando a aplicação.

Iremos importar `Conta` no console, criar a conta e especificar os valores:

```
>>> from conta import Conta

>>> conta = Conta(123, "Nico", 55.5, 1000.0)
Construindo objeto ... <conta.Conta object at 0x11077bcc0>
```

Observe que passamos os parâmetros da conta e nosso código foi executado. Isso é um bom sinal e, por isso, criaremos novos objetos.

```
>>> conta2 = Conta(321, "Marco", 100.0, 1000.0)
```

Será gerada a `conta2`, sendo possível acessar as duas contas criadas pelo console.

```
>>> conta
<conta.Conta object at 0x11077bcc0>

>>> conta2
<conta.Conta object at 0x1109e1da0>
```

Cada conta possui um número, titular, saldo e limite, agora, falta trabalharmos com esses atributos. Nós faremos isso a seguir.