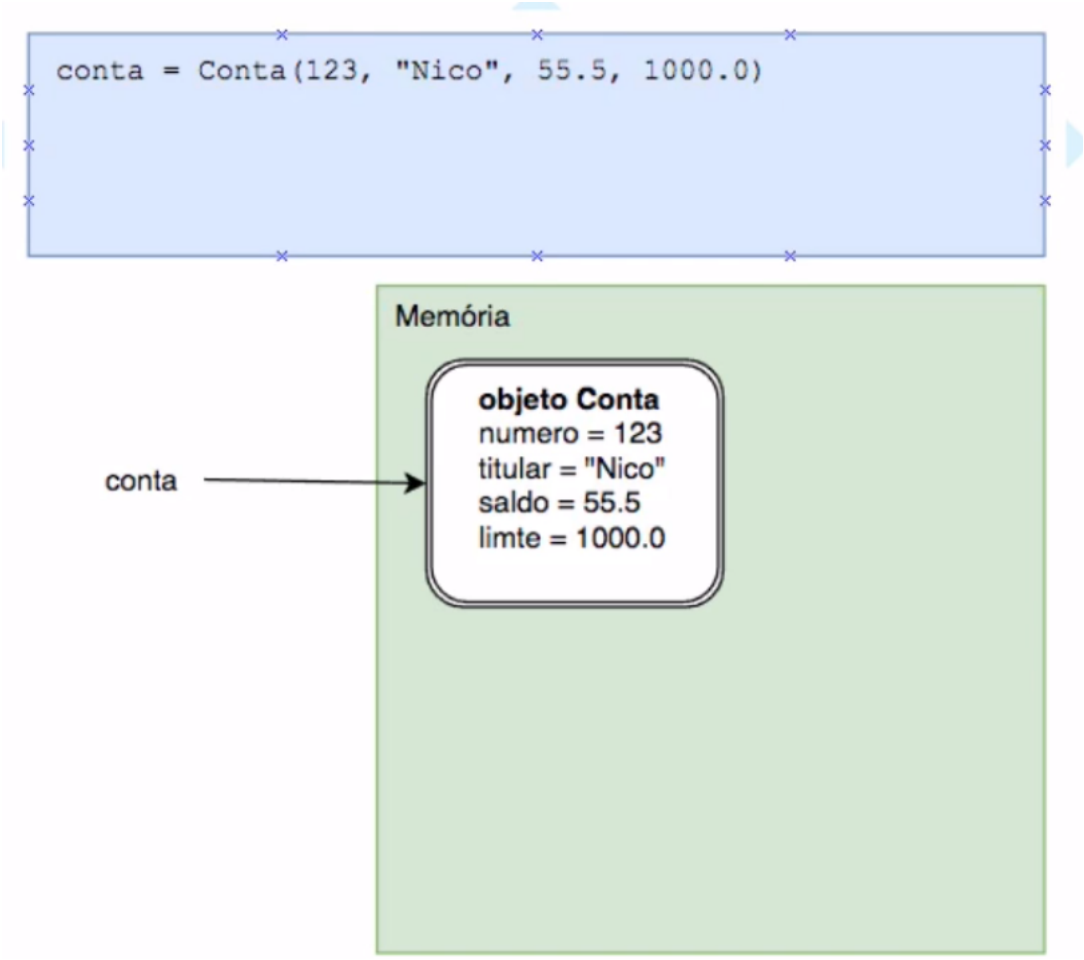
05

None e Coletor de lixo

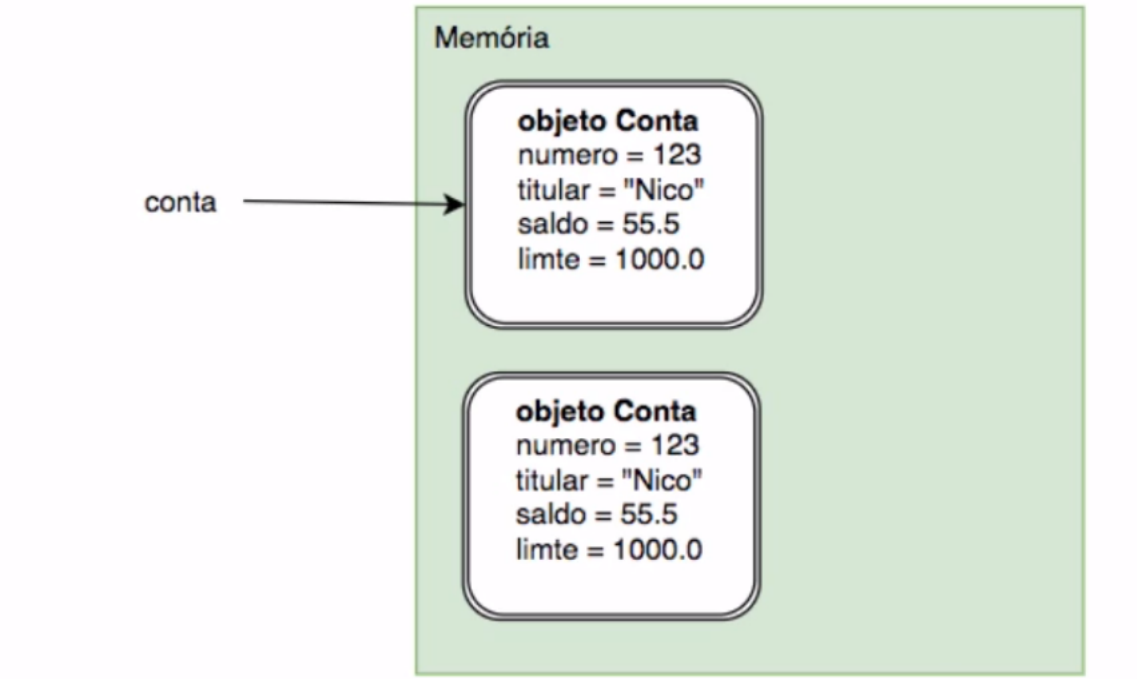
Transcrição

Vamos falar mais sobre os conceitos fundamentais como "referência" e "objeto". Criamos um novo diagrama:



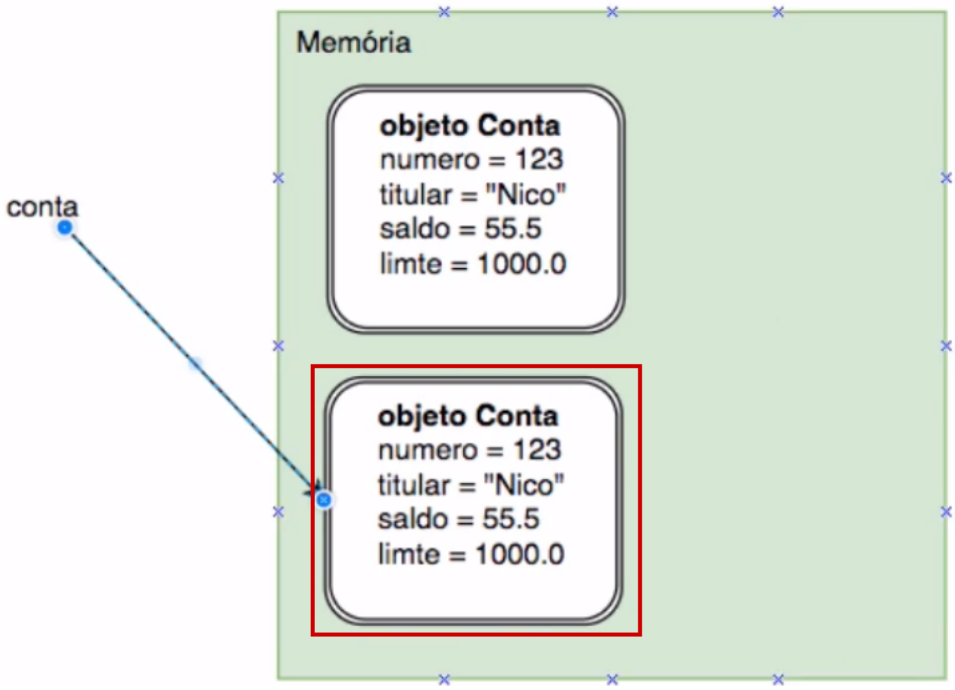
No desenho, mostramos a criação da conta e o objeto em memória. Ao chamarmos a função construtora `__init__` , por baixo dos panos, será gerado o objeto. Em seguida, duplicaremos as linhas inclusas nas duas partes do objeto, desta forma representaremos que temos duas contas e dois objetos, no qual estarão presentes os mesmos dados.

```
conta = Conta(123, "Nico", 55.5, 1000.0)
conta = Conta(123, "Nico", 55.5, 1000.0)
```



Neste caso, teremos dois objetos que representam a conta `123`. Na realidade, só podemos ter uma conta com esse número, mas em um sistema, se chamamos duas vezes o mesmo construtor, teremos dois objetos. Além disso, observem que, com o objeto criado, atribuímos o endereço à mesma referência. Desta forma, a referência `conta` relacionada com um objeto apontará especificamente para o segundo.

```
conta = Conta(123, "Nico", 55.5, 1000.0)
conta = Conta(123, "Nico", 55.5, 1000.0)
```



A referência consegue encontrar o objeto criado mais recentemente. Porém, como faremos para alcançar o primeiro objeto? Ficamos sem referência para ele e, de fato, não temos como alcançá-lo. O primeiro objeto permanecerá ocupando espaço, mas sem ser acessado.

Fazendo uma analogia, seria como se anotássemos um endereço específico em um papel e ao jogarmos essa anotação fora, o local será esquecido e nunca mais localizado.

Quando criamos um programa, são gerados diversos objetos que em algum momento serão abandonados. Dentro da máquina virtual, na execução do Python, existe um processo que procura esses objetos esquecidos. Os itens inutilizados serão apagados e o espaço livre em memória será reutilizado. No caso, o responsável por jogar fora esses objetos em desuso é o **coletor de lixo** (*garbage collector*, em inglês) do Python.

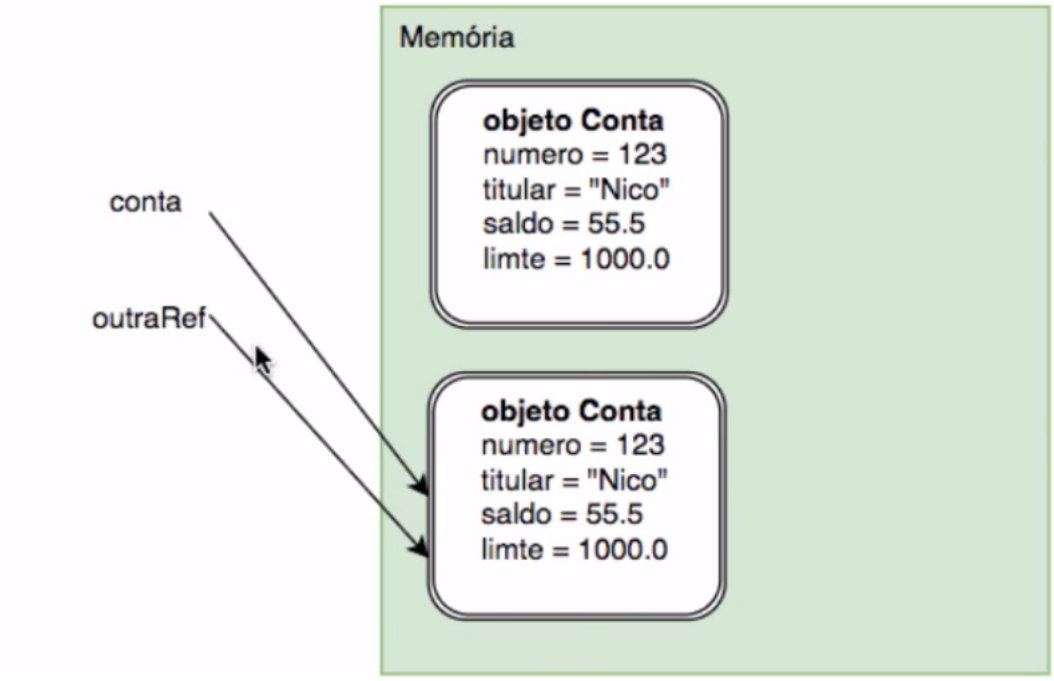
Em seguida, criaremos uma terceira variável, que receberá o nome de `outraRef`. Será para ela que atribuiremos o valor da referência `conta`.

```
>>> conta = Conta(123, "Nico", 55.5, 1000.0)
>>> conta = Conta(123, "Nico", 55.5, 1000.0)

>>> outraRef = conta
```

O valor da referência `conta` fica com a referência `outraRef`. Usando novamente a analogia do endereço anotado em um papel, é como se tivéssemos feito uma fotocópia do papel. Em linguagem UML, o diagrama ficaria assim:

```
conta = Conta(123, "Nico", 55.5, 1000.0)
conta = Conta(123, "Nico", 55.5, 1000.0)
outraRef = conta
```

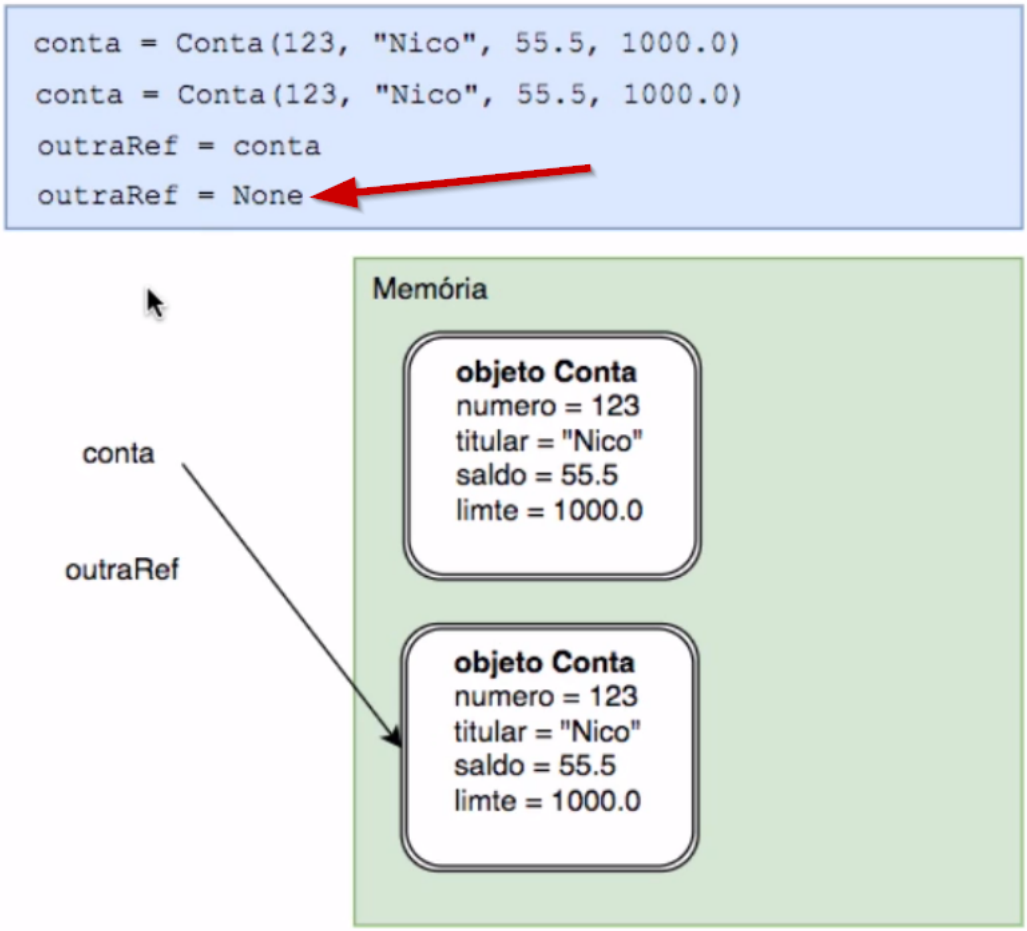


Observem que temos uma nova referência, no entanto, não criamos um novo objeto. Nós podemos ter diversas referências apontando para um mesmo objeto. Neste caso, podemos usar tanto `outroRef` ou `conta` para acessar um atributo.

O que acontece se quisermos desfazer uma referência, por exemplo, desreferenciar `outraRef` ? Para isto, podemos usar a palavra especial `None` :

```
>>> outraRef = None
```

Nosso diagrama ficará assim:



Com o uso do `None`, indicamos que a variável já não aponta para um objeto. A palavra `None` é equivalente a palavra-chave `null` nas linguagens C# ou Java. Nós também removemos a seta que apontava a referência `outraRef` para o objeto `Conta`, porque já não é possível acessá-lo usando a referência `outraRef`.

Revisando: Vimos que os objetos abandonados são removidos pelo coletor de lixo do Python e que podemos ter mais de uma referência apontando para o mesmo objeto. Inclusive, podemos desfazer a referência para um objeto, como fizemos com `outraRef`.

Estamos criando uma base sólida de conceitos, mas falta vermos muita coisa.