Python
Paradigma Procedural x Paradigma Orientado a Objetos

Vamos fazer um exemplo de conta no banco. Teremos a princípio 4 variáveis para a conta:

```
numero = 123
titular = "Fulano"
saldo = 12321.0
limite = 20000.0
```

E se fôssemos lidar com diversas contas???

Listas? Dicionários?

```
conta = {"numero": 123}
conta = {"numero": 123, "titular": "Fulano", "saldo": 12567.0, "limite": 20000.0}
conta2 = {"numero": 321, "titular": "Beltrano", "saldo": 100000.0, "limite": 200000.0}
```

Python
Paradigma Procedural x Paradigma Orientado a Objetos

Criação de uma função para nos ajudar!!!

```
def cria_conta():
    conta2 = {"numero": 321, "titular": "Fulano", "saldo": 12567.0, "limite": 20000.0}
```

Ainda assim o código parece meio chumbado, podemos modifica-lo para torna-lo mais dinâmico, retornando qualquer conta que queiramos criar:

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

Python
Paradigma Procedural x Paradigma Orientado a Objetos

Problemas do Muno Procedural:

- Lembrar nomes das chaves... (conta, numero, titular, limite e saldo)

- Adicionar funções relacionadas a uma conta: depositar, sacar, transferir, tirar extrato, etc.

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

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

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

Python
Paradigma Procedural x Paradigma Orientado a Objetos

A grande vantagem do paradigma OO é a junção das dados/características (número, titular, limite, saldo) e dos procedimentos/funcionalidades (sacar, depositar, tirar extrato) apresentados anteriormente!

E se eu quiser criar uma conta fora da função de criação de conta???

```
conta2 = {"numero": 321, "saldo": 1100.0}
```

Essa conta está certa?

```
conta3 = {"numero": 321, "limite": 1200.0}
```

E se tentarmos fazer um depósito na conta3?

As ligações no mundo procedural são muito frágeis, pensamos pensar BEM antes de fazê-las!

Python
Classes e Objetos

Um exemplo de ligação frágil do mundo procedural:

```
conta["saldo"] = conta["saldo"] + 100.0
extrato(conta)
```

Acabamos de realizar um depósito sem ao menos chamar a função depósito...

Nada te obriga a utilizar OO em seus projetos...

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.

Python
Classes e Objetos

Você ainda pode exibir o valor do saldo no console SEM CHAMAR a função de extrato...

Pode alterar o saldo de forma bem simples como mostrado anteriormente

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

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

Python
Classes e Objetos

Para fazer um bolo você segue uma receita. Para a conta o procedimento é análogo - precisamos ter uma receita para fazer nossa conta. A essa receita damos o nome de CLASSE!

arquivo conta.py:

```
class Conta:
    pass
```

No console, após a execução do código:

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

Foi passado para a fábrica que o bolo vai ser criado, mas onde ele vai ser armazenado?

```
conta = Conta()
```

Python
Classes o Objetos

Vimos que uma vez instanciado o python retorna para nós o endereço de memória onde o objeto está alocado na memória - chama-se referência!

Trabalhando com a função __init__:

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

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

O Python constrói o objeto, cria um lugar na memória e depois chama a função __init__

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

```
Construindo objeto ... <conta.Conta object at 0x1020d7f28>
```

Python
Classes e Objetos

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

```
class Conta:
    def __init__(self):
        print("CONSTRUÇÃO DE UM OBJETO: {}".format(self))
        self.numero = 123
        self.titular = "Fulano"
        self.saldo = 12567.0
        self.limite = 20000.0
```

Para deixar os atributos de construção dinâmicos tal qual fizemos com a função no modelo procedural:

```
class Conta:
    def __init__(self, numero, titular, saldo, limite):
        print("CONSTRUÇÃO DE UM OBJETO: {}".format(self))
        self.numero = numero
        self.titular = titular
        self.saldo = saldo
        self.limite = limite
```

Python
Construtores com Valores Padrão

Imagine que preciso criar 3 contas:

```
conta1 = Conta(1, "Fulano", 0.0, 1000.0)
conta2 = Conta(2, "Beltrano", 0.0, 1000.0)
conta3 = Conta(3, "Sicrano", 0.0, 2000.0)
```

O que ambas têm em comum? Como podemos configurar um limite padrão na conta?

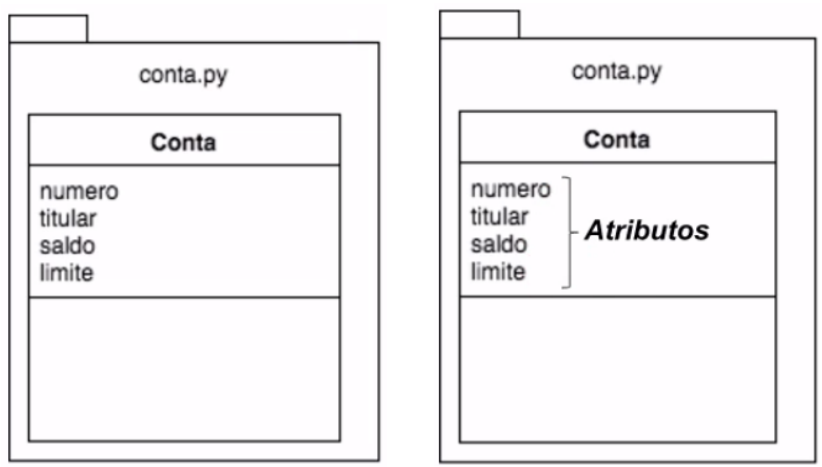
```
class Conta:
    def __init__(self, numero, titular, saldo, limite = 1000.0):
        self.numero = numero
        self.titular = titular
        self.saldo = saldo
        self.limite = limite
conta1 = Conta(1, "Fulano", 0.0)
conta2 = Conta(2, "Beltrano", 0.0)
conta3 = Conta(3, "Sicrano", 0.0, 2000.0)
```

Python
Acessando Atributos

Conta() se encontra dentro de um arquivo que, em python, chamamos de módulo - em outras linguagens pode ser chamado de pakcage e namespace também.

Nós chegaremos ao objeto por meio da referência conta, responsável por indicar onde se encontra o objeto. Precisamos dizer usando a linguagem Python "vai para esse objeto e acessa aquele atributo". O pedido de "vai" nas linguagens Orientadas a Objeto é indicado com **.**

```
>>> conta.saldo
55.5
>>> conta2.saldo
100.0
```



Python
Implementando Métodos

```
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
    def extrato(self):
        print("Saldo {} do titular {}".format(self.saldo, self.titular))
```

```
>>> conta.extrato()
Saldo 12567.5 do titular Fulano
```

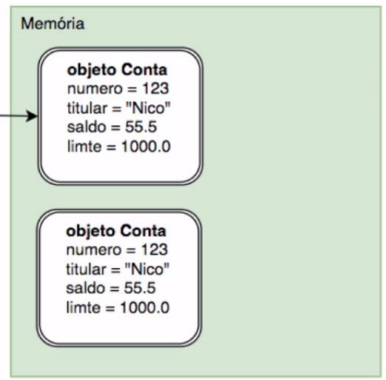
Python
Implementando Métodos

```
def extrato(self):
    print("Saldo de {} do titular {}".format(self.saldo, self.titular))
def deposita(self, valor):
    self.saldo += valor
def saca(self, valor):
    self.saldo -= valor
```

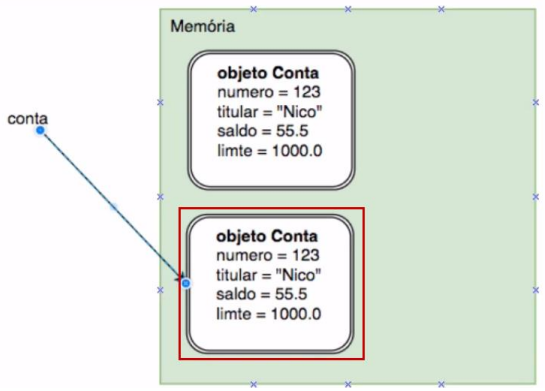
Encapsulamento!

Python
Coletor de Lixo / None

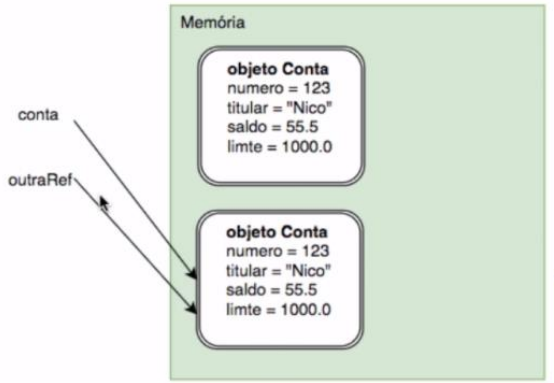
```
conta = Conta(123, "Nico", 55.5, 1000.0)
conta = Conta(123, "Nico", 55.5, 1000.0)
```



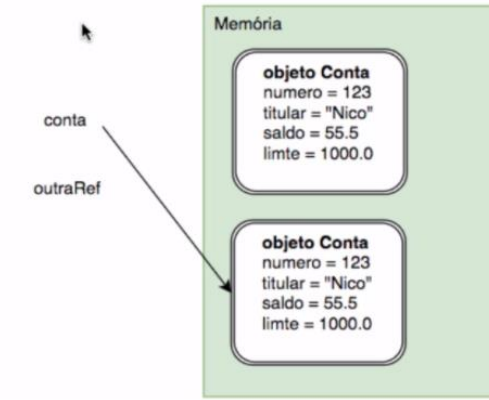

```
conta = Conta(123, "Nico", 55.5, 1000.0)
conta = Conta(123, "Nico", 55.5, 1000.0)
```




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




```
conta = Conta(123, "Nico", 55.5, 1000.0)
conta = Conta(123, "Nico", 55.5, 1000.0)
outraRef = conta
outraRef = None
```



Python
Encapsulamento – Atributos Privados

Acessando o atributo saldo:

```
>>> conta.saldo
12567.0
```

Alterando o atributo saldo:

```
>>> conta.saldo = 13000.0
>>> conta.extrato()
Saldo de 13000.0 do titular Fulano
```

Eu poderia fazer isso???

Se quiséssemos saber o nome de alguém, seria uma falta de educação pegar diretamente o documento de identificação da pessoa, sem pedir autorização. Da mesma forma, seria mais apropriado usarmos um método para identificar o saldo, em vez de acessá-lo diretamente.

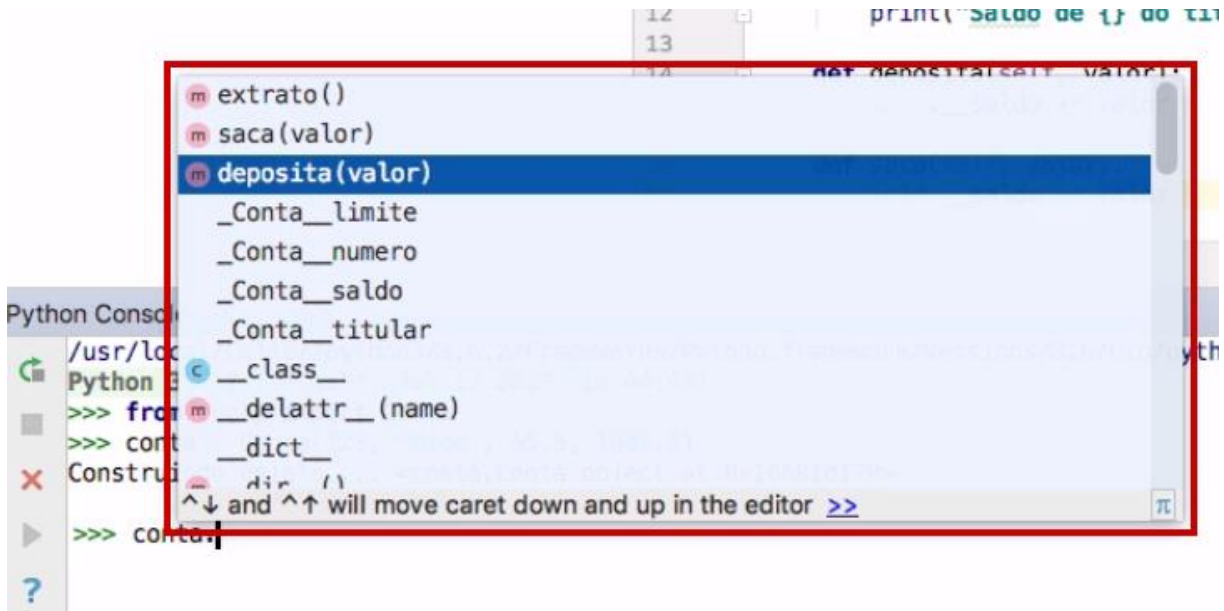
Não podemos acessar o atributo saldo do objeto diretamente. Teremos que usar os métodos responsáveis por encapsular o acesso ao objeto.

Python
Encapsulamento – Atributos Privados

```
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
```

Eu ainda consigo acessar esses itens?

```
>>> conta._Conta__limite
1000.0
>>> conta._Conta__saldo
55.5
```



Python
Encapsulamento – Métodos

Como eu retiro dinheiro de uma conta e envio dinheiro para outra conta?

```
>>> valor = 10.00
>>> conta2.saca(valor)
>>> conta.deposita(valor)
```

Nosso encapsulamento foi quebrado com a ação de transferência de dinheiro entre contas...

```
def transfere(self, valor):
    conta2.saca(valor)
    conta.deposita(valor)
```

Isso está certo???

```
def transfere(self, valor, origem, destino):
    origem.saca(valor)
    destino.deposita(valor)
```

Python
Encapsulamento – Métodos

```
def transfere(self, valor, destino):
    self.saca(valor)
    destino.deposita(valor)
```

Python
Coesão – Classes e Responsabilidades (Únicas)

É possível criar um método que verifique que o usuário é inadimplente?

```
def eh_inadimplente(self, cliente):
```

Quais as vantagens e desvantagens de utilizar esse método?

Python
Getters e Setters

Vocês acham que a função extrato estava devidamente correta?

```
def extrato(self):
    print("Saldo de {} do titular {}".format(self.saldo, self.titular))
```

O que poderia ser melhorado???

```
def pega_saldo(self):
    return self.__saldo


def devolve_titular(self):
    return self.__titular
def retorna_limite(self):
    return self.__limite
```

A pergunta que não quer calar: Anda está certo??? 🤔

Python
Getters e Setters

Podemos colocar o nome "get" atrelado ao nome do método!

```
def transfere(self, valor, destino):
    self.saca(valor)
    destino.deposita(valor)

def get_saldo(self):
    return self.__saldo

def get_titular(self):
    return self.__titular
```

Python
Getters e Setters

Da mesma forma que fizemos com os getters também faremos o mesmo com a propriedade "set":

```
def set_limite(self, limite):
    self.__limite = limite
```

Utilizar getters e setters com bom senso -> somente quando necessários!

Python
Properties

Criar um arquivo "cliente.py":

```
class Cliente:
    def __init__(self, nome):
        self.nome = nome
```

Podemos alterar a propriedade “nome” de fora da classe???

```
>>> cliente.nome = "Beltrano"
>>> cliente.nome
'Beltrano
```

Python
Properties

Vamos criar um método onde o acesso ao parâmetro será dado por uma função que se chamará nome() e que retornará o nome do titular da conta com a primeira letra maiúscula, no entanto sem utilizar os parênteses!

```
class Cliente:
    def __init__(self, nome):
        self.nome = nome
    @property
    def nome(self):
        return self.nome.title()
```

Python
Properties

Podemos fazer de forma parecida para um setter:

```
@nome.setter
def nome(self, nome):
    print("CHAMANDO O SETTER nome()")
    self.__nome = nome
```

```
>>> from cliente import Cliente
>>> cliente = Cliente("Fulano")
>>> cliente.nome = "Beltrano"
chamando setter nome()
>>> cliente.nome
chamando @property nome()
'Beltrano'
```

Python
Properties

Transformando os getters e setters de limite da classe Conta em properties:

```
def get_titular(self):
    return self._titular
@property
def limite(self):
    return self.__limite
@limite.setter
def limite(self, limite):
    self.__limite = limite
```

```
>>> from conta import Conta
>>> conta = Conta(123, "Fulano", 12567.5, 20000.0)
Construindo objeto ... <conta.Conta   o bject at
0x1019df3c8>
>>> conta.limite
20000.0

>>> conta.limite = 100000.0
>>> conta.limite
100000.0
```

Python
Métodos Privados

Analisando a Classe Conta;

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

def transfere(self, valor, destino):
    self.saca(valor)
    destino.deposita(valor)
```

Supondo que a conta do Fulano tenha R$ 1250,00 de saldo e um limite de R$ 10.000,00

Qual o valor máximo de saque que podemos fazer? Teoricamente, só poderíamos sacar R$ 11250.00. Mas é possível fazer uma malandragem e sacar mais:

```
>>> conta.saca(21250.0)
>>> conta.saldo
-10000.0
```

Python
Métodos Privados

Passo 1 - fazer uma validação dentro do próprio método saca():

```
def saca(self, valor):
    if(valor <= (self.__saldo + self.__limite)):
        self.__saldo -= valor
    else:
        print("O valor {} passou o limite".format(valor))
```

Tentar realizar a chamativa do método saca() no terminal com um valor maior do que o criado pela condicional:

```
O valor 40000 passou o limite
```

Python
Métodos Privados

Podemos deixar o código um pouco melhor??? Mais expressivo -> mais fácil de entender...

```
def pode_sacar(self):
    pass
def sacar(self, valor):
    if(pode_sacar(self)):
        self.__saldo -= valor
    else:
        print("O valor {} passou o limite".format(valor))
```

Faz sentido um método que é utilizado por uma função de dentro da classe ser acessado de fora dela???

```
def __pode_sacar(self, valor_a_sacar):
```

Python
Métodos Estáticos

Vamos criar uma nova variável que faz referência ao código do banco!

```
self.__codigo_banco = "001"

@property
def codigo_banco(self):
    return self.__codigo_banco
```

Já que estamos criando todas as contas como sendo de um banco, faz sentido eu precisar criar um objeto para saber o banco que esse objeto vai fazer parte?

basta remover o self???

Eu consigo continuar utilizando uma property sem fazer referência ao self???

```
@staticmethod
def codigo_banco():
    return "001"
```

Pode remover o atributo __codigo_banco da __init__()

Python
Métodos Estáticos

E se eu quisesse devolver o código e o nome de vários bancos?

```
@staticmethod
def codigos_bancos():
    return {'BB': '001', 'Caixa': '104', 'Bradesco':'237'}
```

Usar com parcimônia! Lembrar que estamos trabalhando com **Orientação a Objetos**!

Normalmente são utilizados quando todos os objetos têm alguma característica comum, como nesse caso.

Python
Python x Java

```
class Conta {
    //atributos
    private int numero;
    private String titular;
    private double saldo;
    private double limite;
    //construtor
    Conta(int numero, String titular, double saldo, double limite) {
        this.numero = numero;
        this.titular = titular;
        this.saldo = saldo;
        this.limite = limite;
    }
}

Conta contaDoFulano = new Conta(123, "Fulano", 12567.0, 20000.0);
contaDoFulano.deposita(1000.0);
```

Python
Python x Java

```
void extrato() {
    System.out.println("Saldo de " + this.saldo)
}
public double getLimite() {
    return limite;
}
public void setLimite(double limite) {
    this.limite = limite;
}

public double getNumero() {
    return numero;
}
```

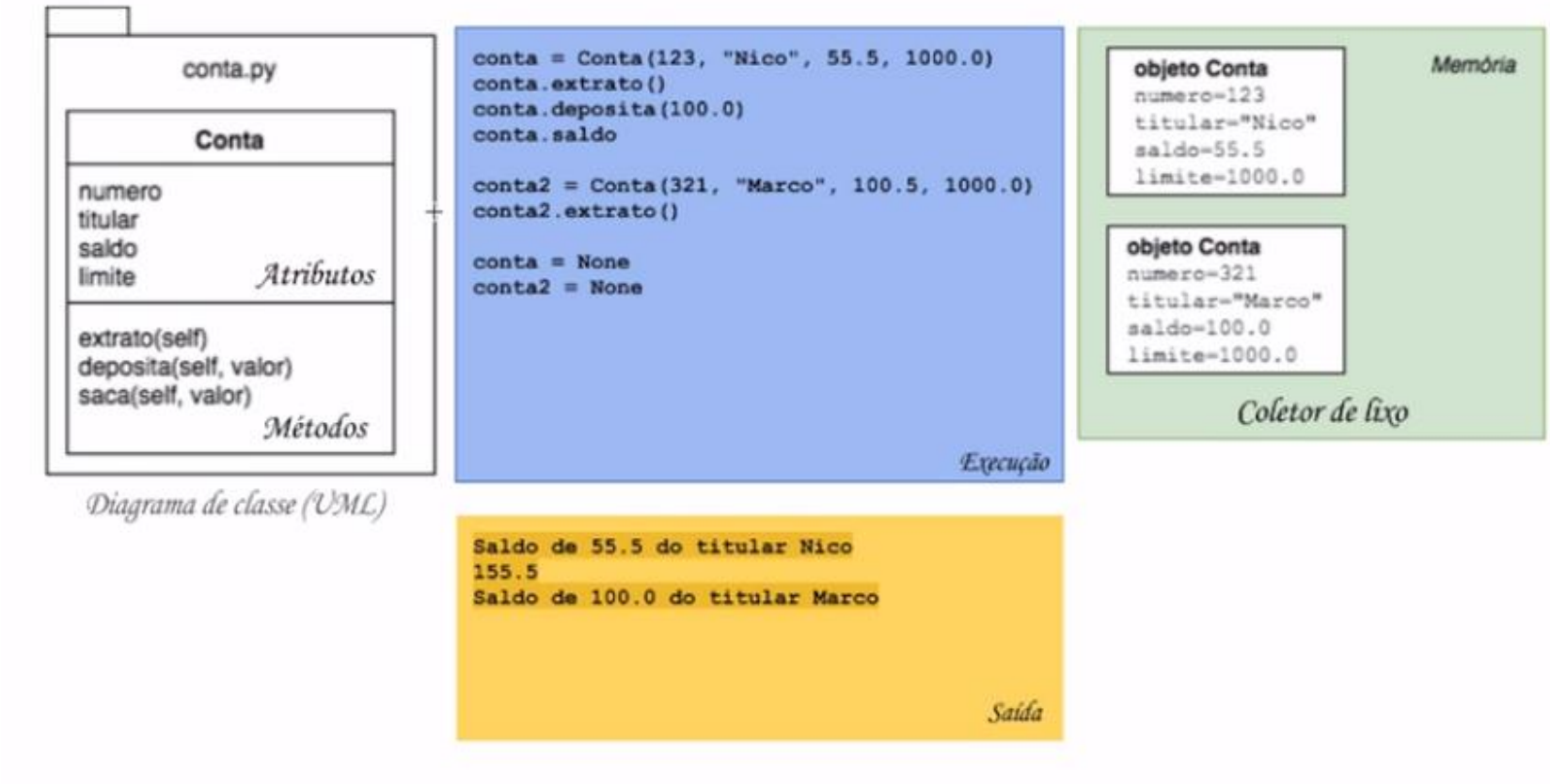
```
public void getTitular() {
    return titular;
}

public double getSaldo() {
    return saldo;
}

public static String codigo() {
    return "001"
}
```

Python

Python
Revisão



Python
Atributos Estáticos

Suponha uma classe Circulo:

```
class Circulo:
    @staticmethod
    def obter_pi():
        return 3.14
```

Podemos deixar isso mais simples?

```
class Circulo:
    PI = 3.14
    @staticmethod
    def obter_pi():
        return Circulo.PI
```