



Classes e Objetos

fmasanori@gmail.com

Baixar os códigos abaixo

- `tv*.py`
- `tatu*.py`
- `teste*.py`

Classes e objetos

- Classes associam dados (atributos) e operações (métodos) em uma só estrutura
- Um objeto é uma variável cujo tipo é uma classe, ou seja, um objeto é uma instância de uma classe
- Veremos apenas o básico da orientação à objetos

Classes e objetos

```
class Televisão:  
    def __init__(self):  
        self.ligada = False  
        self.canal = 2
```

```
>>> tv_quarto = Televisão()
```

```
>>> tv_sala = Televisão()
```

```
>>> tv_quarto.ligada
```

```
False
```

```
>>> tv_quarto.canal
```

```
2
```

```
>>> tv_sala.ligada = True
```

```
>>> tv_sala.canal = 5
```

Classes e objetos

- Quando declaramos uma classe, estamos criando um novo tipo de dados
- Da mesma forma que quando criamos uma lista ou uma string, estamos instanciando ou criando uma instância dessas classes
- É a mesma coisa fazer `lista = []` ou `lista = list()`
- O método `__init__` é chamado construtor e é chamado na criação do objeto

Classes e objetos

- O parâmetro `self` significa o objeto televisão em si
- `self.ligada` é um valor de `self`, ou seja, do objeto televisão
- Sempre que criamos atributos do objeto, devemos associá-los a `self`.
- Caso contrário, se escrevêssemos apenas `ligada = False`, `ligada` seria apenas uma variável local do método e não um atributo

Classes e objetos

```
class Televisão:
    def __init__(self):
        self.ligada = False
        self.canal = 2
    def muda_canal_para_baixo(self):
        self.canal -= 1
    def muda_canal_para_cima(self):
        self.canal += 1
```

```
>>> tv = Televisão()
>>> tv.muda_canal_para_cima()
>>> tv.muda_canal_para_cima()
>>> tv.canal
4
>>> tv.muda_canal_para_baixo()
>>> tv.canal
3
```

Classes e objetos

- Você irá informatizar o banco Tatú, controlando o saldo das contas correntes
- Cada conta corrente pode ter um ou mais clientes como titular
- O banco controla apenas o nome e telefone
- A conta corrente apresenta um saldo e um extrato de operações de saques e depósitos
- Não há contas especiais, logo o cliente não pode sacar mais do que têm no saldo

Arquivo tatu.py

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

class Conta:
    def __init__(self, clientes, número, saldo = 0):
        self.saldo = saldo
        self.clientes = clientes
        self.número = número
    def resumo(self):
        print('CC Número: %s Saldo: %10.2f' %
              (self.número, self.saldo))
    def saque(self, valor):
        if self.saldo >= valor:
            self.saldo -= valor
    def deposito(self, valor):
        self.saldo += valor
```

Arquivo teste.py

```
from tatu import Cliente
from tatu import Conta
joão = Cliente('João da Silva', '777-1234')
maria = Cliente('Maria da Silva', '555-4321')
print ('Nome: %s. Telefone: %s.'
        %(joão.nome, joão.telefone))
print ('Nome: %s. Telefone: %s.'
        %(maria.nome, maria.telefone))
conta1 = Conta([joão], 1, 1000)
conta2 = Conta([maria, joão], 2, 500)
conta1.resumo()
conta2.resumo()
```

Saída de teste.py

>>>

Nome: João da Silva. Telefone: 777-1234.

Nome: Maria da Silva. Telefone: 555-4321.

CC Número: 1 Saldo: 1000.00

CC Número: 2 Saldo: 500.00

>>>

Extrato de operações

- Altere o método `resumo` da classe `Conta` para extrato, imprimindo agora uma lista de operações de saques e depósitos feitas
- Altere o método `__init__` para que utilize o método `depósito` para inicializar o saldo

Arquivo tatu2.py (apenas Conta)

```
class Conta:
    def __init__(self, clientes, número, saldo = 0):
        self.saldo = 0
        self.clientes = clientes
        self.número = número
        self.operacoes = []
        self.deposito(saldo)
    def resumo(self):
        print('CC N°%s Saldo: %10.2f' %
              (self.número, self.saldo))
    def saque(self, valor):
        if self.saldo >= valor:
            self.saldo -= valor
            self.operacoes.append(['Saque', valor])
    def deposito(self, valor):
        self.saldo += valor
        self.operacoes.append(['Depósito', valor])
    def extrato(self):
        print('Extrato CC N° %s' % self.número)
        for op in self.operacoes:
            print('%10s %10.2f' % (op[0], op[1]))
        print('%10s %10.2f\n' % ('Saldo=', self.saldo))
```

Arquivo teste2.py

```
from tatu2 import Cliente
from tatu2 import Conta
joão = Cliente('João da Silva', '777-1234')
maria = Cliente('Maria da Silva', '555-4321')
conta1 = Conta([joão], 1, 1000)
conta2 = Conta([maria, joão], 2, 500)
conta1.saque(50)
conta2.deposito(300)
conta1.saque(190)
conta2.deposito(95.15)
conta2.saque(250)
conta1.extrato()
conta2.extrato()
```

Saída de teste2.py

>>>

Extrato CC N° 1

Depósito	1000.00
Saque	50.00
Saque	190.00
Saldo=	760.00

Extrato CC N° 2

Depósito	500.00
Depósito	300.00
Depósito	95.15
Saque	250.00
Saldo=	645.15

Herança

- A orientação a objetos permite modificar nossas classes, adicionando ou modificando atributos e métodos, tendo como base a classe anterior
- Vamos criar contas especiais, onde podemos sacar mais dinheiro que o saldo, até um determinado limite
- As operações depósito, extrato e resumo continuam como uma conta normal

Adicionar ContaEspecial tatu3.py

```
class ContaEspecial(Conta):
    def __init__(self, clientes, número, saldo=0, limite=0):
        Conta.__init__(self, clientes, número, saldo)
        self.limite = limite
    def saque(self, valor):
        if self.saldo + self.limite >= valor:
            self.saldo -= valor
            self.operacoes.append(['Saque', valor])
```

ContaEspecial

- Observe que escrevemos Conta entre parênteses
- ContaEspecial herda os métodos e atributos de Conta
- `self.limite` será criado apenas para classes do tipo ContaEspecial
- Observe que estamos substituindo completamente o método saque em ContaEspecial

Arquivo teste3.py

```
from tatu3 import Cliente
from tatu3 import Conta, ContaEspecial
joão = Cliente('João da Silva', '777-1234')
maria = Cliente('Maria da Silva', '555-4321')
conta1 = Conta([joão], 1, 1000)
conta2 = ContaEspecial([maria, joão], 2, 500, 1000)
conta1.saque(50)
conta2.deposito(300)
conta1.saque(190)
conta2.deposito(95.15)
conta2.saque(1500)
conta1.extrato()
conta2.extrato()
```

Saída teste3.py

>>>

Extrato CC N° 1

Depósito	1000.00
Saque	50.00
Saque	190.00
Saldo=	760.00

Extrato CC N° 2

Depósito	500.00
Depósito	300.00
Depósito	95.15
Saque	1500.00
Saldo=	-604.85

Vantagens da herança

- Modificamos muito pouco o nosso programa, mantendo a funcionalidade anterior e adicionando novos recursos
- Foi possível fazer o reuso dos métodos de Conta
- Assim a definição da classe ContaEspecial foi bem menor, incluindo apenas o comportamento diferente

Outro exemplo OOP

```
import datetime
class Pessoa():
    def __init__(self, nome, nascimento):
        self.nome = nome
        self.nascimento = nascimento
    def idade(self):
        delta = datetime.date.today() - self.nascimento
        return int(delta.days/365)
    def __str__( self ):
        return '%s, %d anos' %(self.nome, self.idade())

masanori = Pessoa('Fernando Masanori', datetime.date(1980, 9, 1))
print (masanori.idade())
print (masanori)
```