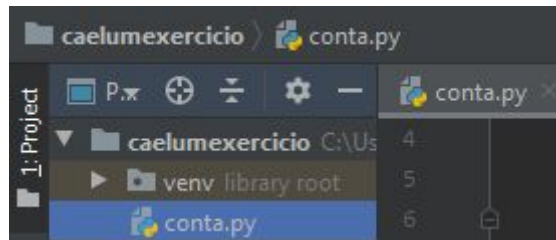


**Atividade Avaliativa 1**  
**Programação Orientada a Objetos**  
**Aluno: Josemar Rocha da Silva**

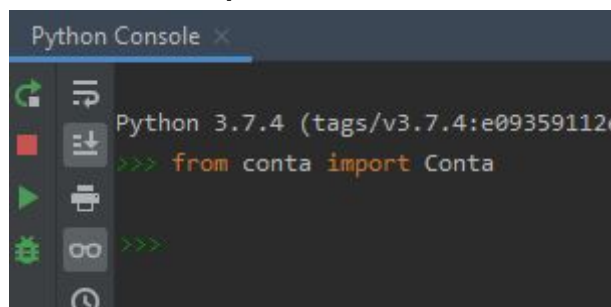
**1 - Criar um arquivo chamado conta.py.**



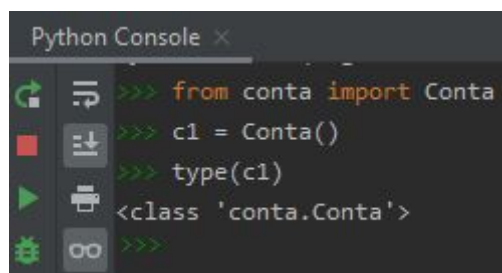
**2 - Cria uma classe sem atributo e salve o arquivo.**

```
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.**



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



**4.1, 4.2, 4.3, 4.4 e 4.5 - Além disso, crie alguns atributos e tente acessá-los.**

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

```
Python Console x
>>> from conta import Conta
>>> c1 = Conta('923-4', 'Josemar', 200.0, 500.0)
>>> print(c1.numero)
923-4
>>> print(c1.titular)
Josemar
>>> print(c1.saldo)
200.0
>>> print(c1.limite)
500.0
>>> |
```

4.6 - 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
```

4.7 - 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
```

4.8 - 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))
```

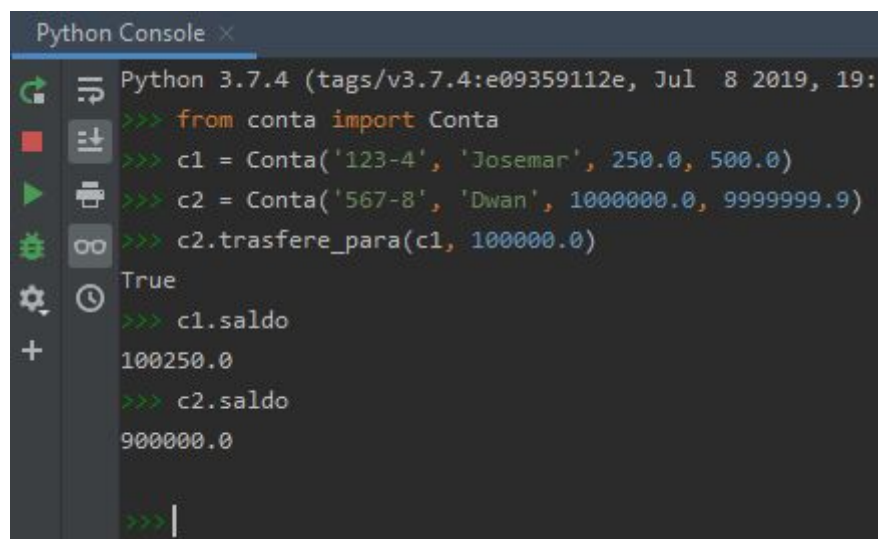
4.9 - Modifique o método `saca()` fazendo retornar um valor que representa se a operação foi ou não bem sucedida. Lembre que não é permitido sacar um valor menor do que o saldo.

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

4.10 - 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, conta2, valor):
    retirou = self.saca(valor)
    if (retirou == False):
        return False
    else:
        conta2.deposita(valor)
        return True
```

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



```
Python Console x
Python 3.7.4 (tags/v3.7.4:e09359112e, Jul 8 2019, 19:
>>> from conta import Conta
>>> c1 = Conta('123-4', 'Josemar', 250.0, 500.0)
>>> c2 = Conta('567-8', 'Dwan', 1000000.0, 9999999.9)
>>> c2.transfere_para(c1, 100000.0)
True
>>> c1.saldo
100250.0
>>> c2.saldo
900000.0
>>> |
```

```
>>> c1.saca(250.0)
True
>>> c1.saldo
100000.0
```

```
>>> c2.extrato()
numero: 567-8
saldo: 900000.0
```

**4.12 - (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?**

Criando classe Cliente e passando como parâmetro na classe Conta:

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

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

Modificando o método extrato para imprimir informações do cliente:

```
def extrato(self):
    print("numero: {} \nsaldo: {} \nnome: {} \nsobrenome: {} \ncpf: {}".format(self.numero, self.saldo, self.titular.nome,
                                                                              self.titular.sobrenome, self.titular.cpf))
```

Teste das novas modificações:

```
>>> from conta import Conta, Cliente
>>> cliente1 = Cliente('Josemar', 'Rocha', '123456789-01')
>>> conta1 = Conta('123-4', cliente1, 250.0, 500.0)
>>> conta1.extrato()
numero: 123-4
saldo: 250.0
nome: Josemar
sobrenome: Rocha
cpf: 123456789-01
>>>
```

Não é possível criar uma conta sem um cliente, pois estamos passando o mesmo como parâmetro no *init* da própria classe Conta, mas é possível criar um Cliente sem uma Conta.

**4.13 - (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.

Criação da classe `Data`:

```
class Data:
    def __init__(self):
        self.data_abertura = datetime.datetime.today()

    def imprime_data(self):
        print("data abertura: {}".format(self.data_abertura))

class Conta:
    def __init__(self, numero, cliente, saldo, limite, data):
        self.numero = numero
        self.titular = cliente
        self.saldo = saldo
        self.limite = limite
        self.data = Data()
```

Teste para verificar se o atributo está funcionando corretamente:

```
>>> from conta import Conta, Cliente, Data
>>> cliente1 = Cliente('Josemar', 'Rocha', '123456789-01')
>>> data1 = Data()
>>> conta1 = Conta('123-4', cliente1, 250.0, 500.0, data1)
>>> conta1.data.imprime_data()
data abertura: 2019-09-13 16:35:50.217585

>>> |
```

**4.14 - (Desafio)** Crie uma classe `Histórico` 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 `Histórico` sem uma `Conta`?

Criação da classe `Histórico`:

```
class Historico:
    def __init__(self):
        self.transacoes = []

    def imprime(self):
        print("transacoes: ")
        for t in self.transacoes:
            print("-", t)
```

Exemplo de modificação nos métodos da classe Conta para armazenar dados no histórico:

```
def deposita(self, valor):  
    self.saldo += valor  
    self.historico.transacoes.append("depósito de {}".format(valor))
```

Testes:

```
>>> from conta import Conta, Cliente, Data, Historico  
>>> cliente1 = Cliente('Josemar', 'Rocha', '123456789-01')  
>>> cliente2 = Cliente('Filipe', 'Dwan', '234567890-12')  
>>> conta1 = Conta('123-4', cliente1, 250.0, 500.0)  
>>> conta2 = Conta('567-8', cliente2, 3000.0, 5000.0)  
>>> conta1.deposita(100.0)  
>>> conta1.saca(50.0)  
True  
>>> conta2.trasfere_para(conta1, 1000.0)  
True  
>>> conta1.trasfere_para(conta2, 500.0)  
True  
>>> conta1.historico.imprime()  
transacoes:  
- depósito de 100.0  
- saque de 50.0  
- depósito de 1000.0  
- saque de 500.0  
- transferencia de 500.0 para conta 567-8  
  
>>>
```

Não vejo sentido em criar histórico sem uma conta, pois não há o que armazenar no mesmo, tanto que passo histórico como atributo de uma conta, desta forma cada conta tem um histórico.