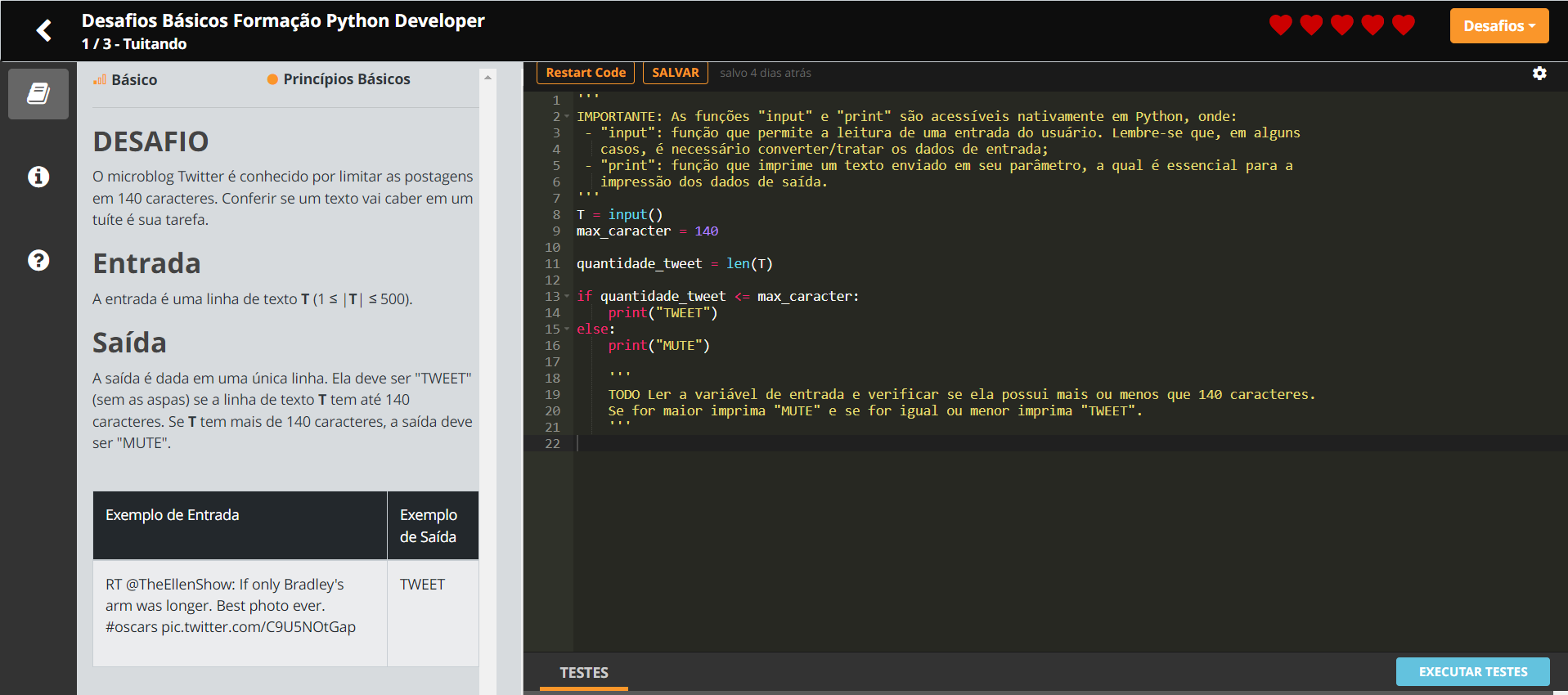
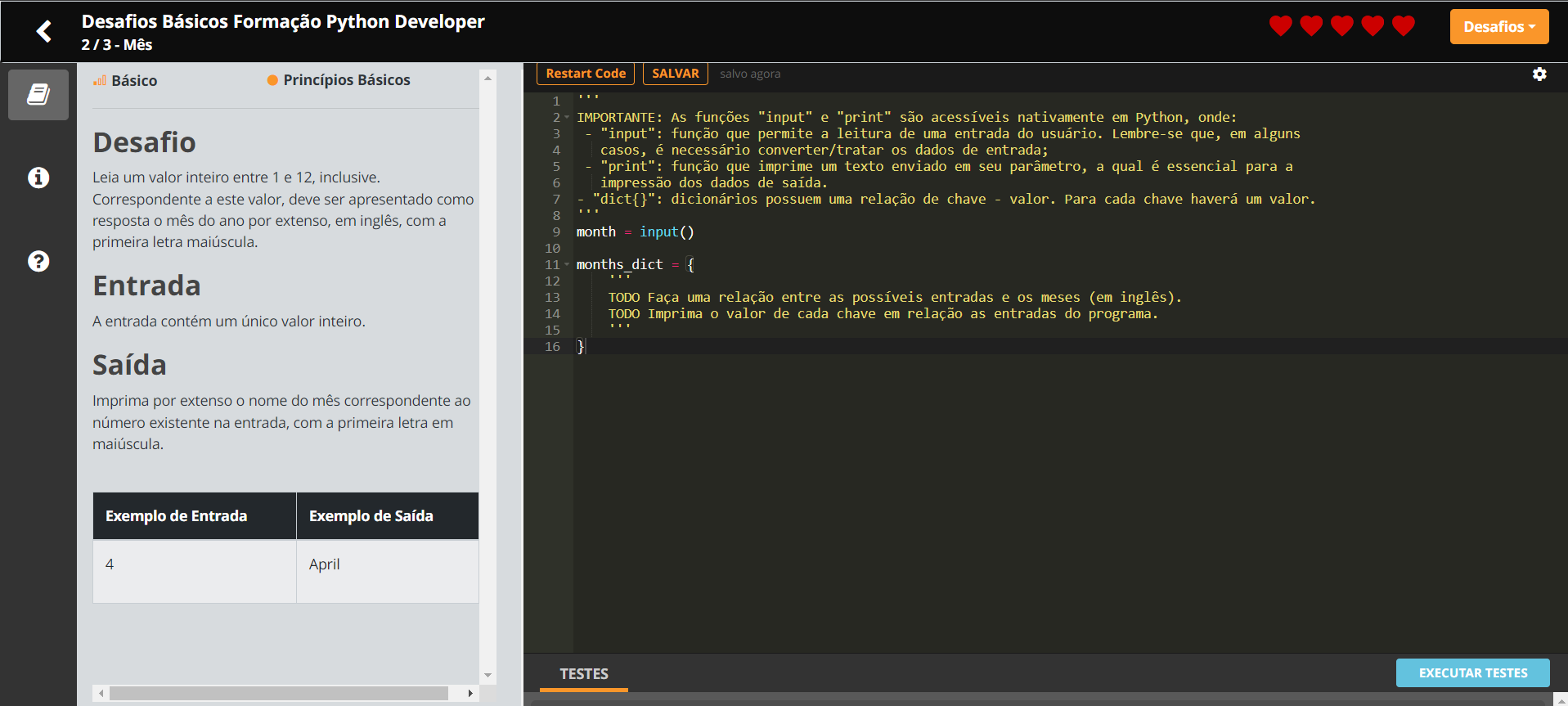
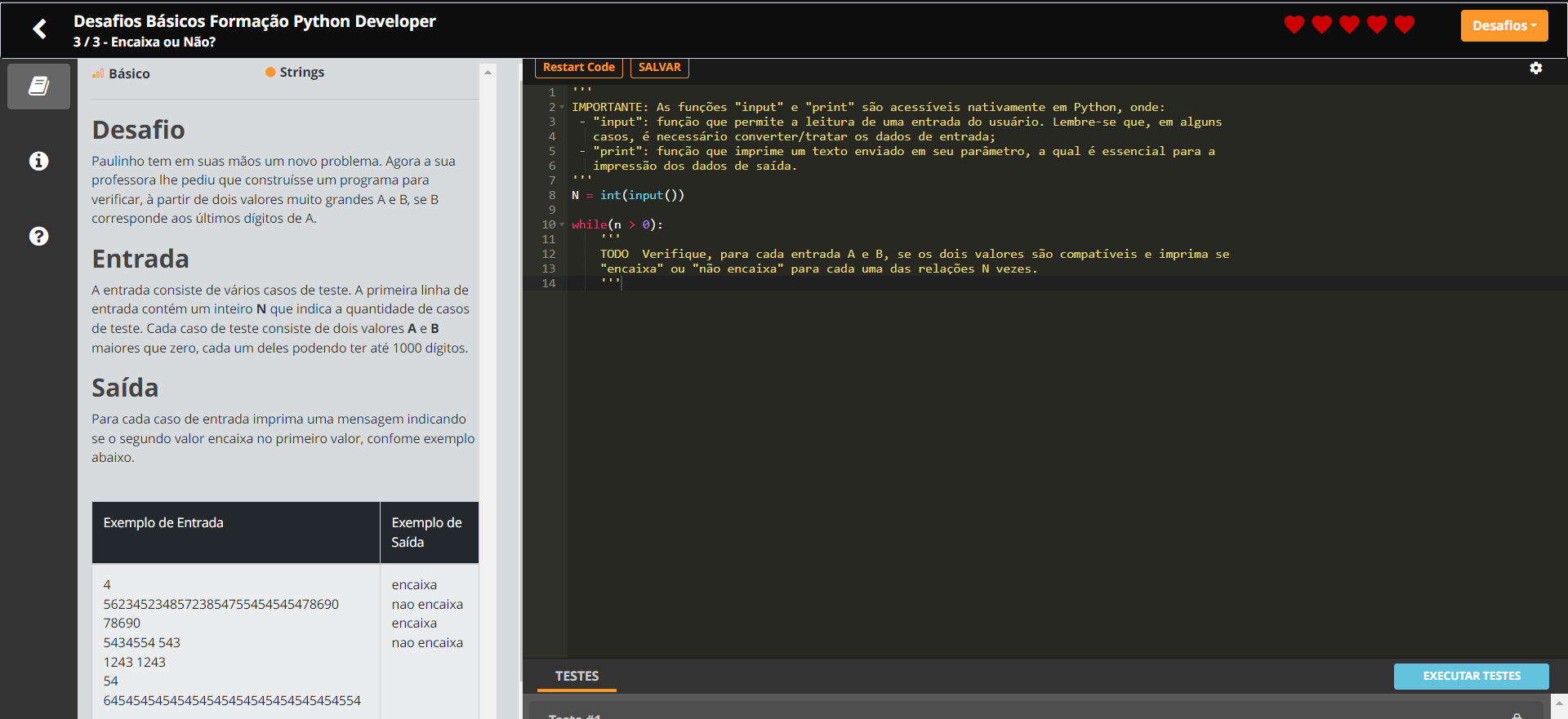
**DESAFIOS BÁSICOS FORMAÇÃO PYTHON DEVELOPER**







**CRIANDO UM SISTEMA BANCÁRIO EM PYTHON**

**Objetivo Geral:**

Criar um sistema bancário com as operações: sacar, depositar e visualizar extrato.

**DESAFIO:**

Fomos contratados por um grande banco para desenvolver o seu novo sistema. Esse banco deseja modernizar suas operações e para isso escolheu a linguagem Python. Para a primeira versão do sistema devemos implementar apenas 3 operações: depósito, saque e extrato.

**Operação de depósito:**

Deve ser possível depositar valores positivos para a minha conta bancária. A v1 do projeto trabalha apenas com 1 usuário, dessa forma não precisamos nos preocupar em identificar qual é o número da agência e conta bancária. Todos os depósitos devem ser armazenados em uma variável e exibidos na operação de extrato.

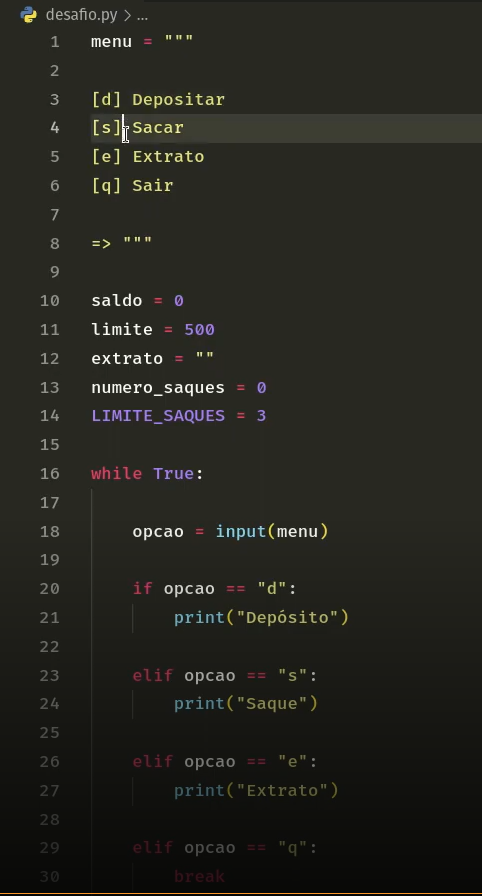
**Operação de saque:**

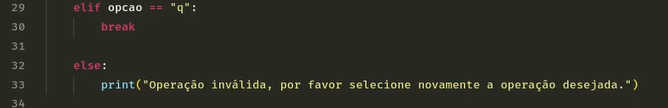
O sistema deve permitir realizar 3 saques diários com limite máximo de R$ 500,00 por saque. Caso o usuário não tenha saldo em conta, o sistema deve exibir uma mensagem informando que não será possível sacar o dinheiro por falta de saldo. Todos os saques devem ser armazenados em uma variável e exibidos na operação de extrato.

**Operação de extrato:**

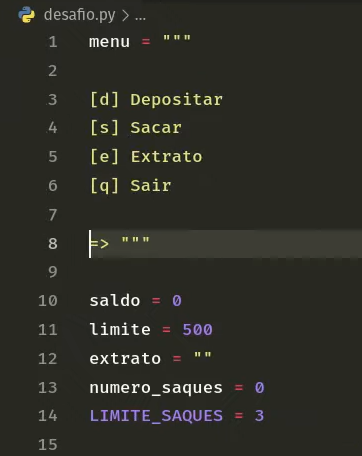
Essa operação deve listar todos os depósitos e saques realizados na conta. No fim da listagem deve ser exibido o saldo atual da conta. Se o extrato estiver em branco, exibir a mensagem: Não foram realizadas movimentações. Os valores devem ser exibidos utilizando o formato R$ xxx.xx

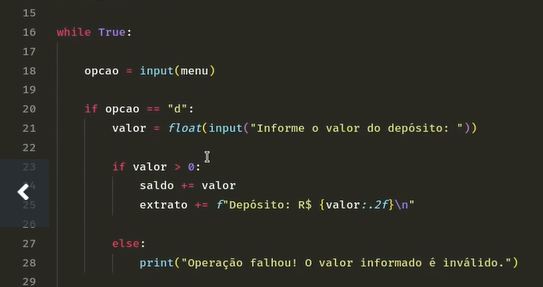
Exemplo: 1500.45 = R$ 1500.45

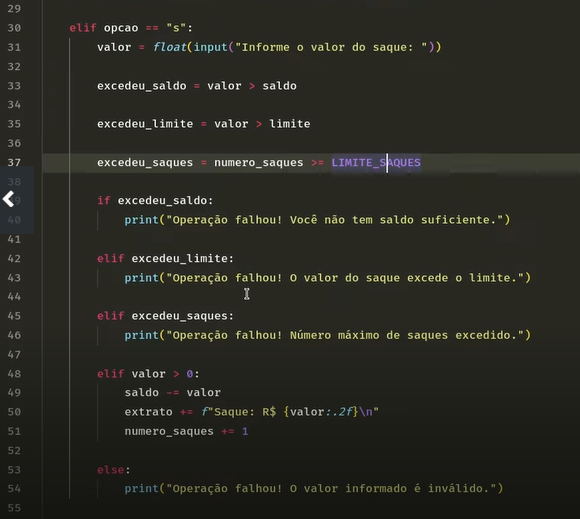


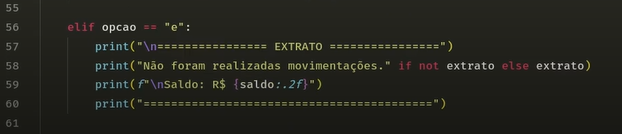


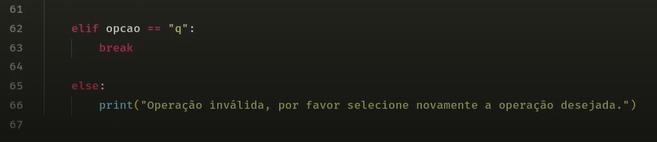
**RESOLUÇÃO**:



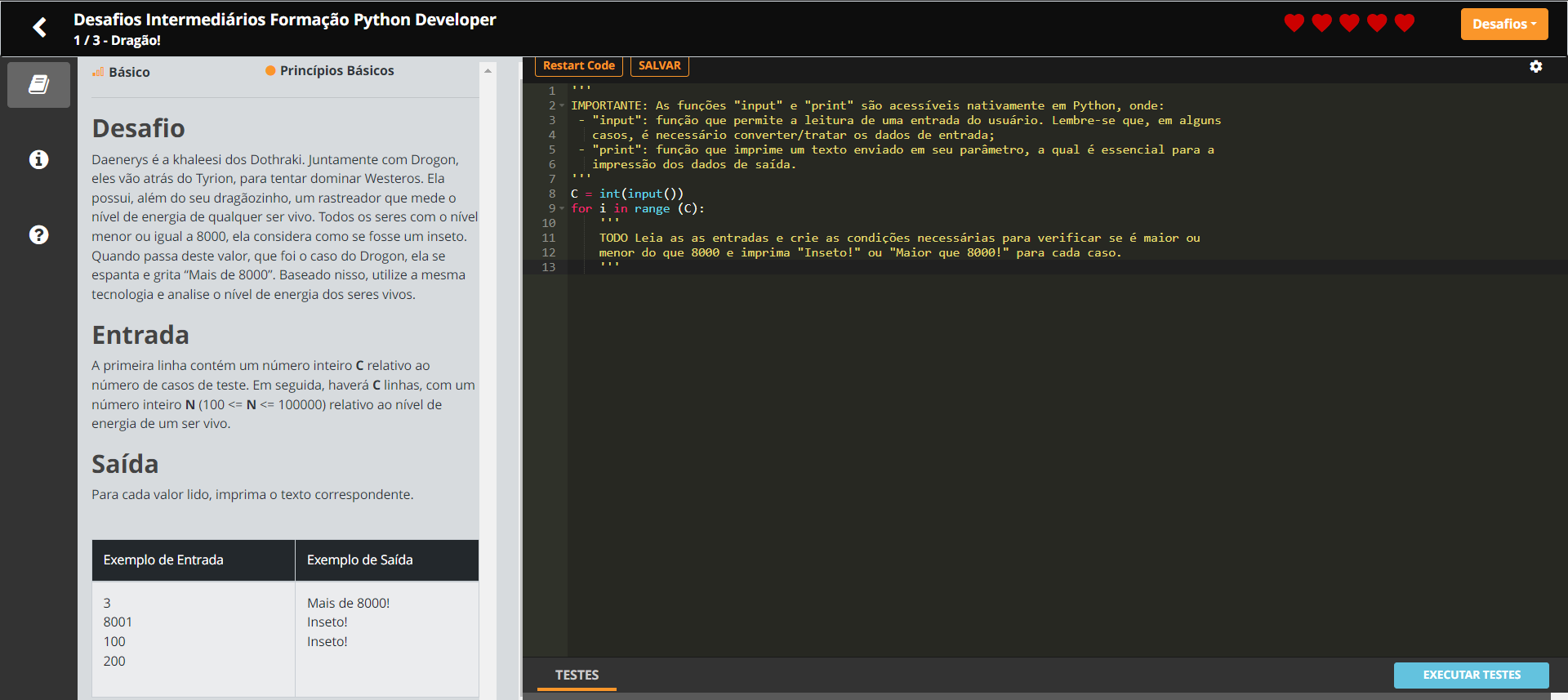


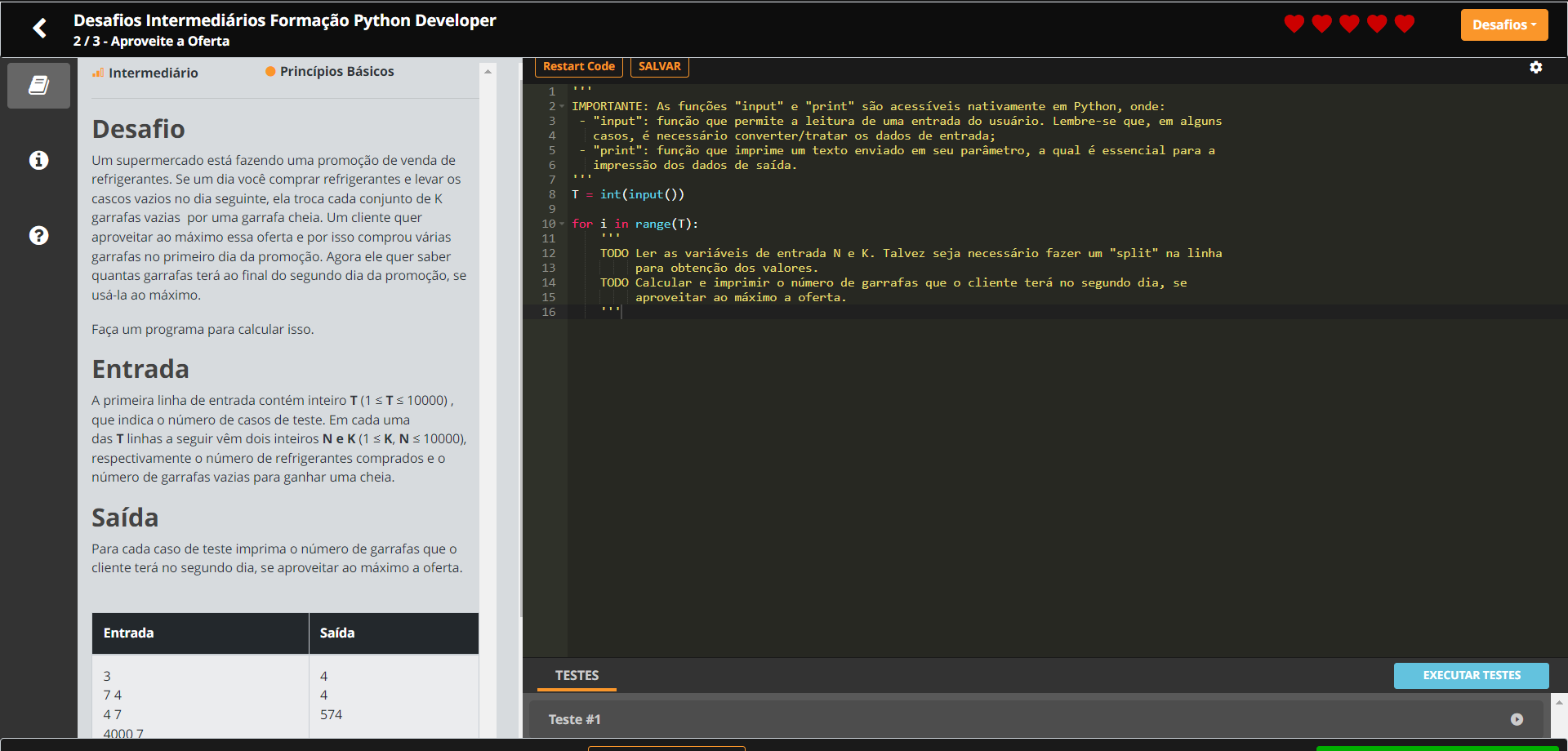


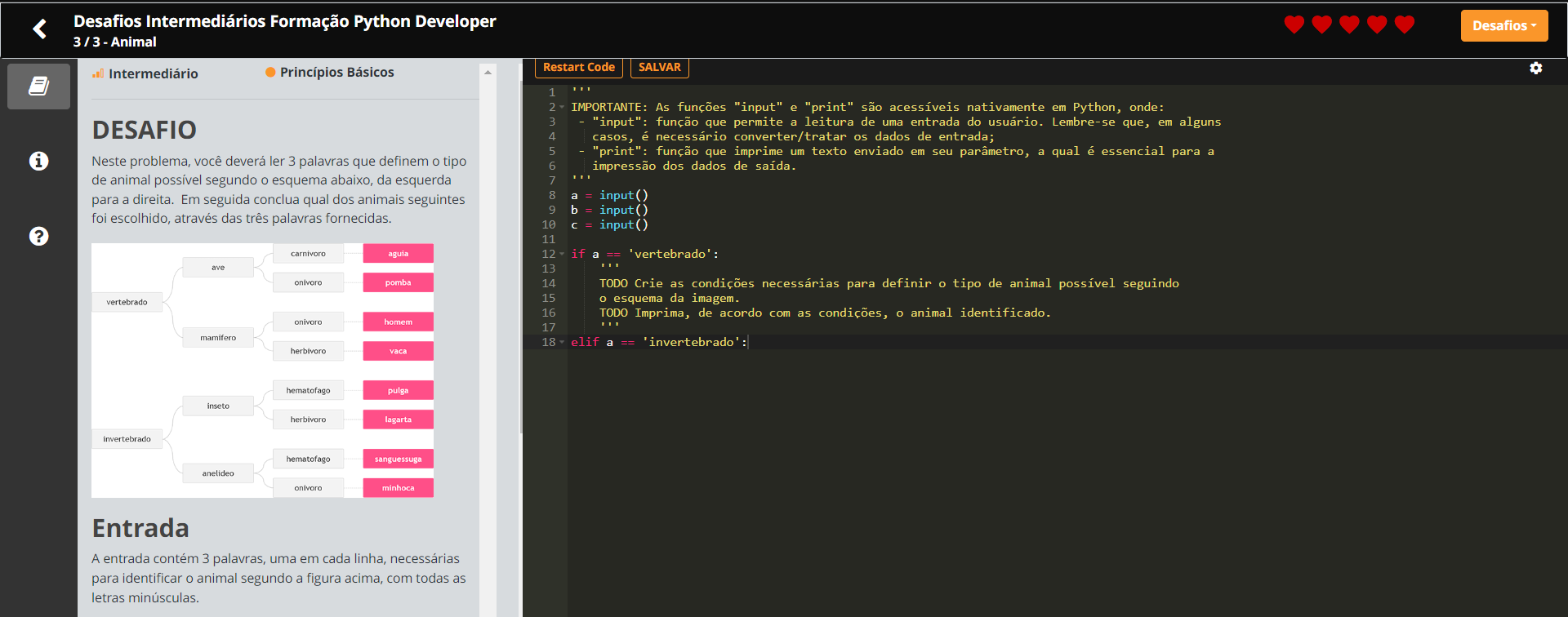


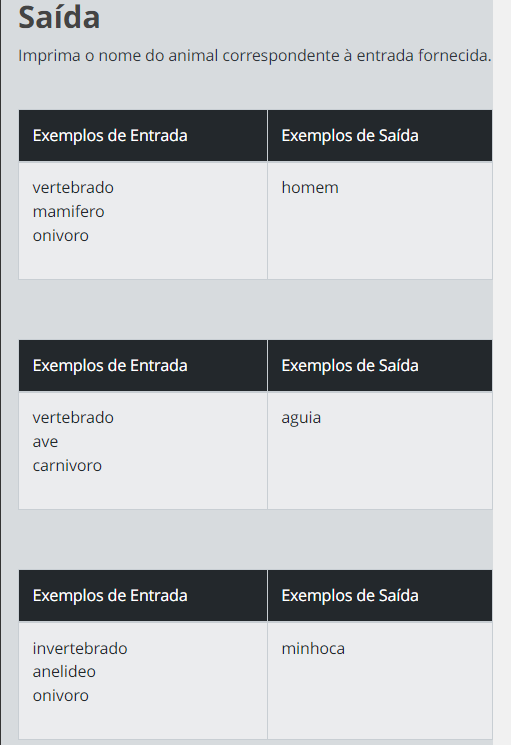


**DESAFIOS INTERMEDIÁRIOS FORMAÇÃO PYTHON DEVELOPER**









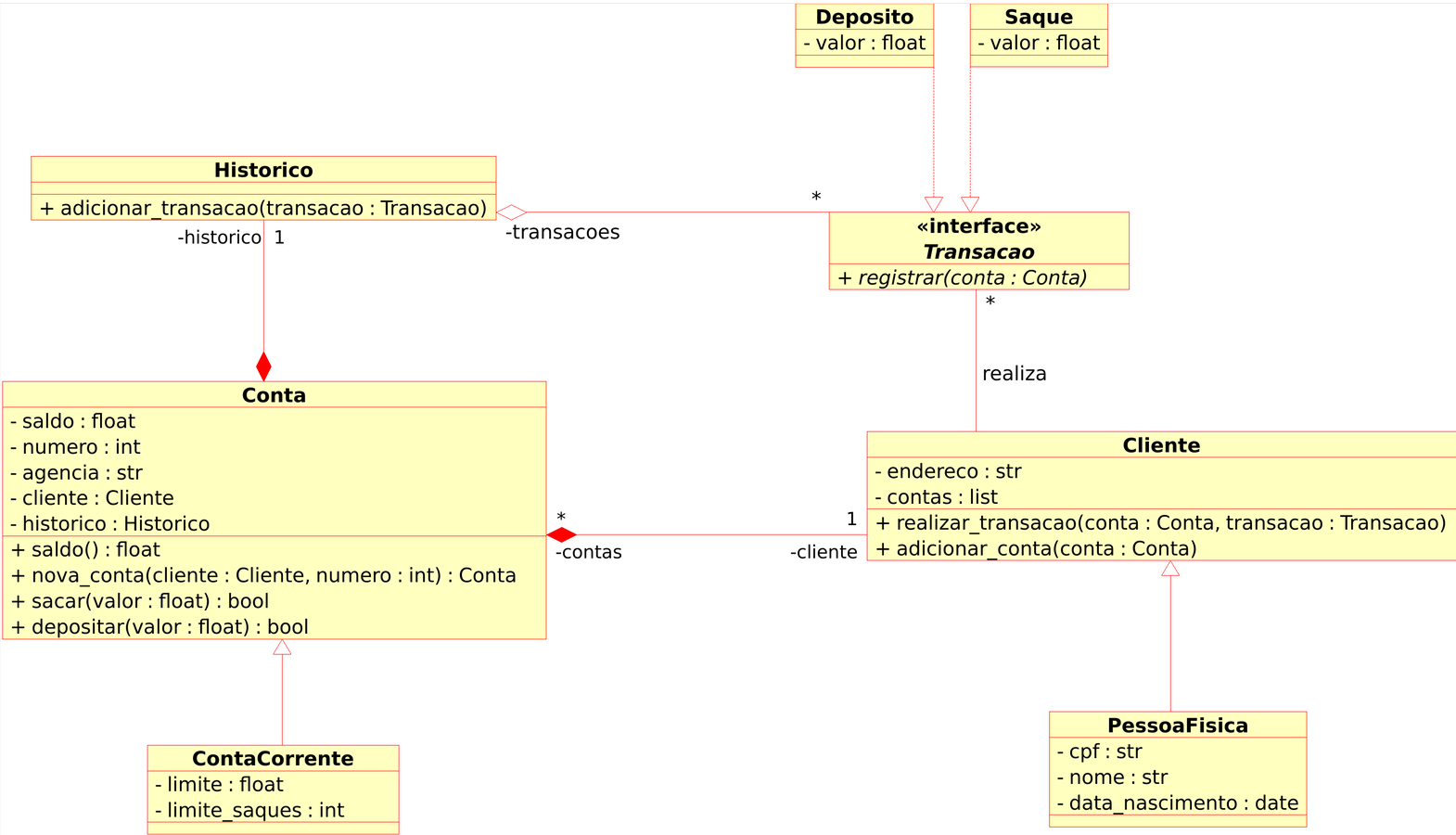
**OTIMIZANDO O SISTEMA BANCÁRIO COM FUNÇÕES PYTHON**

**Objetivo Geral:**

Separar as funções existentes de saque, depósito e extrato em funções. Criar duas novas funções: cadastro de usuários: (clientes) e cadastrar conta bancária.

**DESAFIO:**

Precisamos deixar nosso código mais modularizado, para isso vamos criar funções para as operações existentes: sacar, depositar e visualizar extrato. Além disso, para a versão 2 do nosso sistema precisamos criar duas novas funções: criar usuários (cliente do banco) e criar conta corrente (vincular com usuário).



**Separando em Funções:**

Devemos criar funções para todas as operações do sistema. Para exercitar tudo o que aprendemos neste módulo, cada função vai ter uma regra na passagem de argumentos. O retorno e a forma como serão chamadas, pode ser definida por você da forma que achar melhor.

**Saque:**

A função saque deve receber os argumentos apenas por nome (keyword only). Sugestão de argumentos: saldo, valor, extrato, limite, numero\_saques, limite\_saques. Sugestão de retorno: saldo e extrato.

**Depósito:**

A função depósito deve receber os argumentos apenas por posição (positional only). Sugestão de argumentos: saldo, valor, extrato. Sugestão de retorno: saldo e extrato.

**Extrato:**

A função extrato deve receber os argumentos por posição e nome (positional only e keyword only). Argumentos posicionais: saldo; argumentos nomeados: extrato.

**Novas Funções:**

Precisamos criar duas novas funções: criar usuário e criar conta corrente. Fique a vontade para adicionar mais funções, exemplo: listar contas.

**Criar Usuário (Cliente):**

O programa deve armazenar os usuários em uma lista, um usuário é composto por: nome, data de nascimento, CPF e endereço. O endereço é uma String com o formato: logradouro, nro – bairro – cidade/sigla estado. Deve ser armazenado somente os números do CPF. Não podemos cadastrar 2 usuários com o mesmo CPF.

**Criar conta corrente:**

O programa deve armazenar contas em uma lista, uma conta é composta por: agência, número da conta e usuário. O número da conta é sequencial, iniciando em 1. O número da agência é fixo: “0001”. O usuário pode ter mais de uma conta, mas uma conta pertence a somente um usuário.

**Dica:**

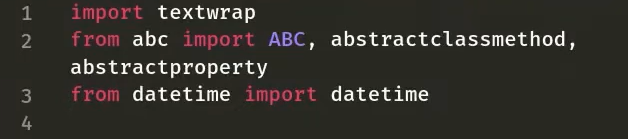
Para vincular um usuário a uma conta, filtre a lista de usuários buscando o número do CPF informado para cada usuário da lista.

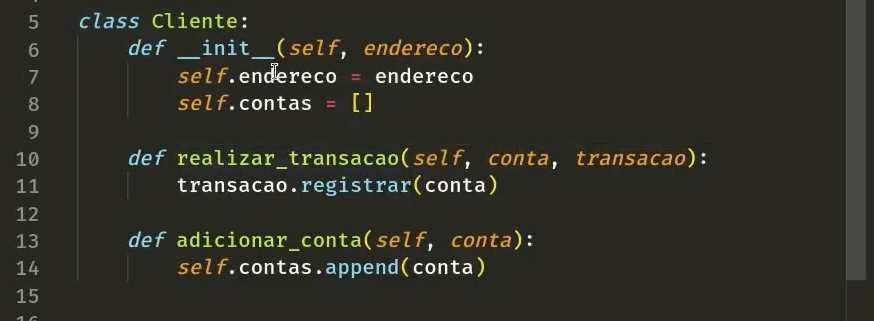
**Desafio extra:**

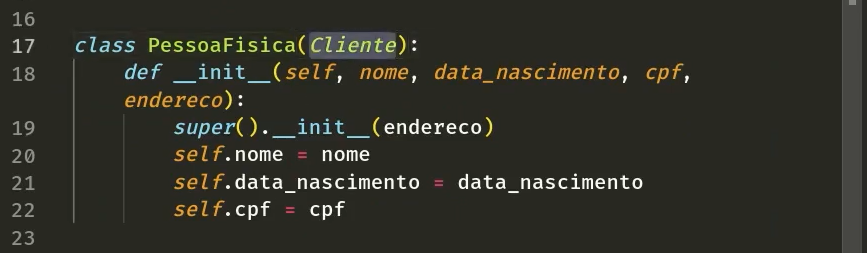
Após concluir a modelagem das classes e a criação dos métodos. Atualizar os métodos que tratam as opções do menu, para funcionarem com as classes modeladas

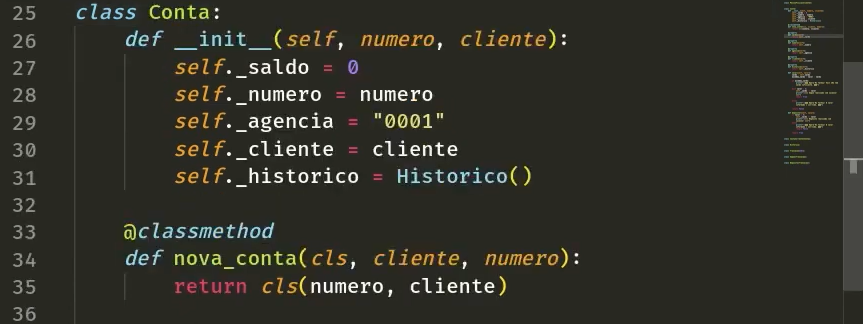
**RESOLUÇÃO: parte 1**

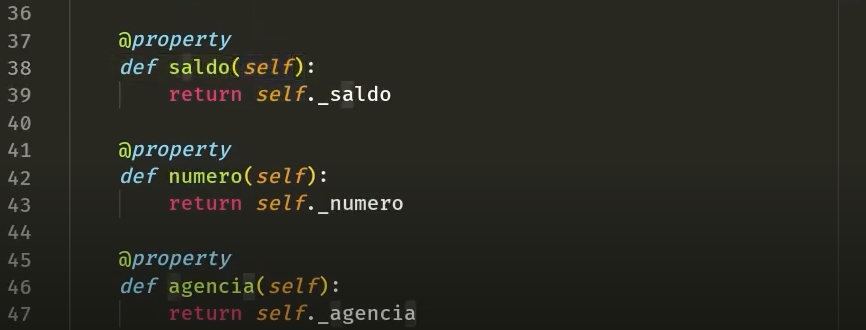


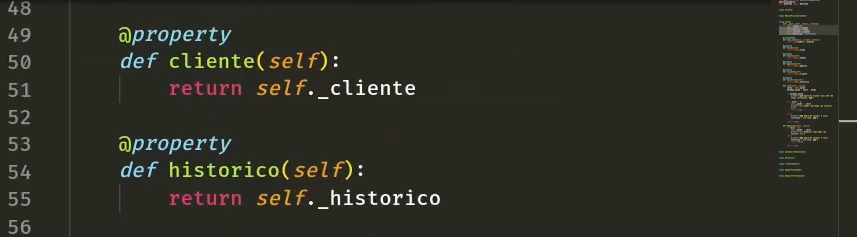
 ALTERAÇÃO

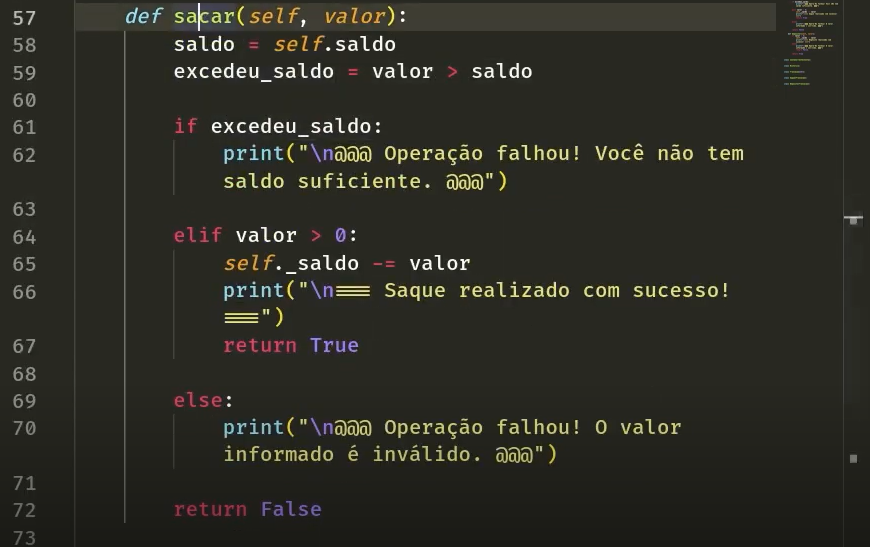


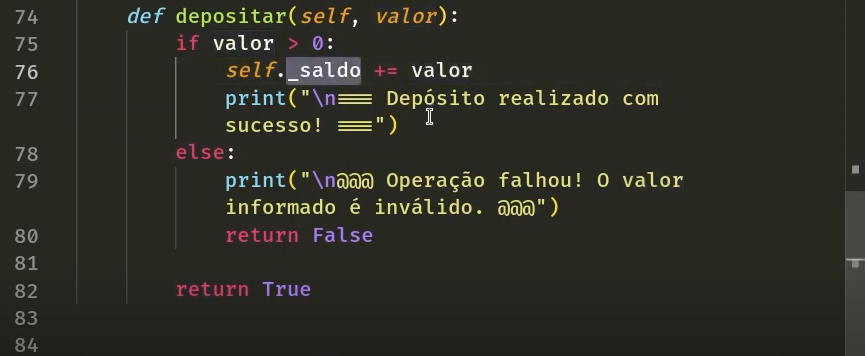


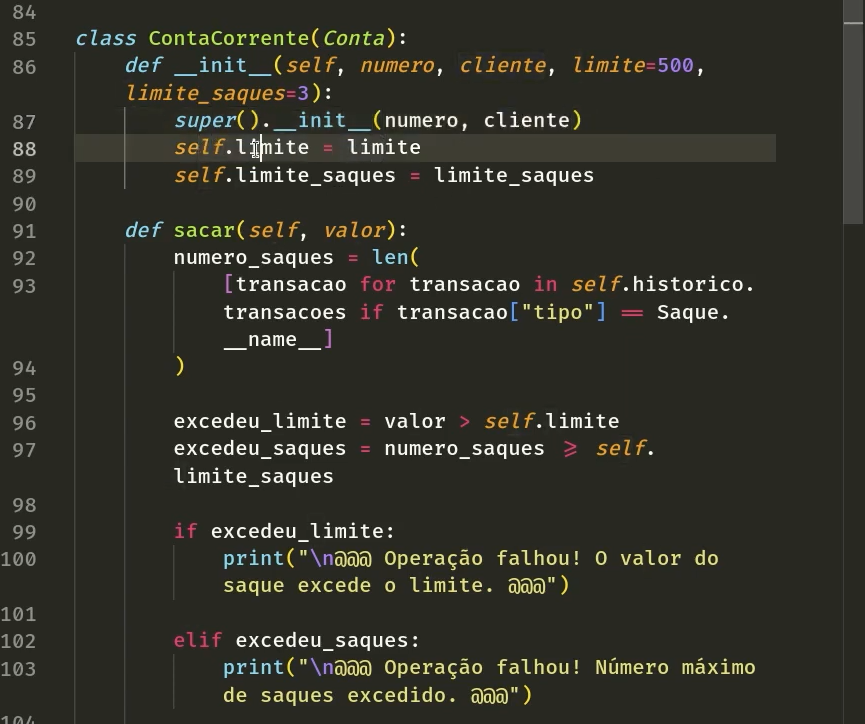


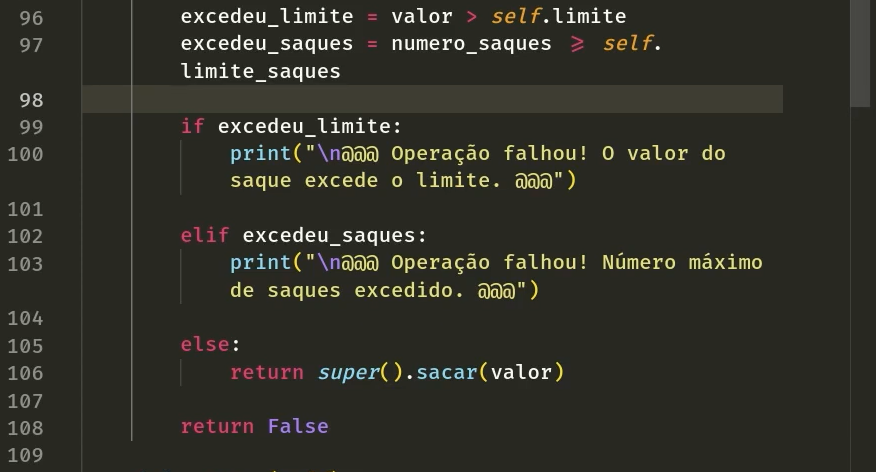


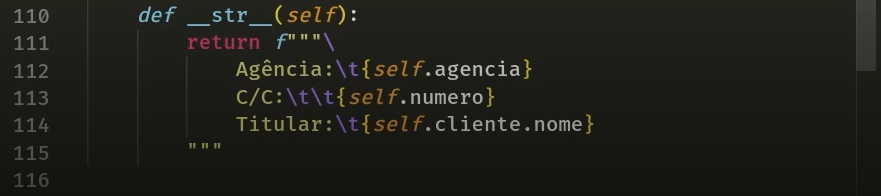






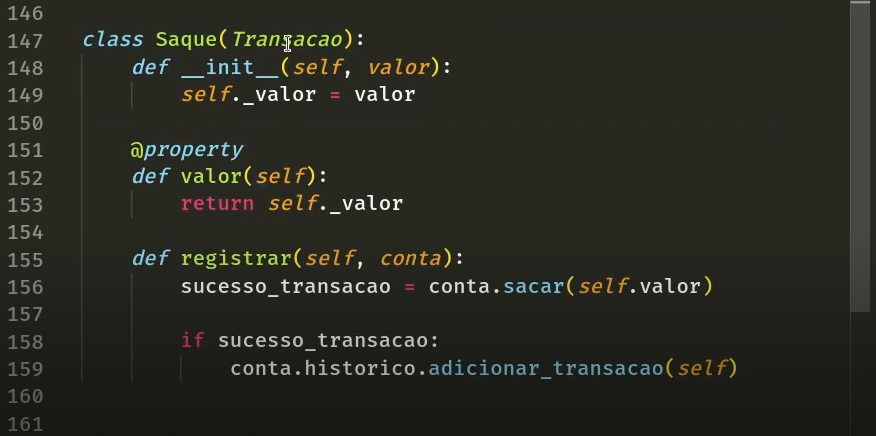


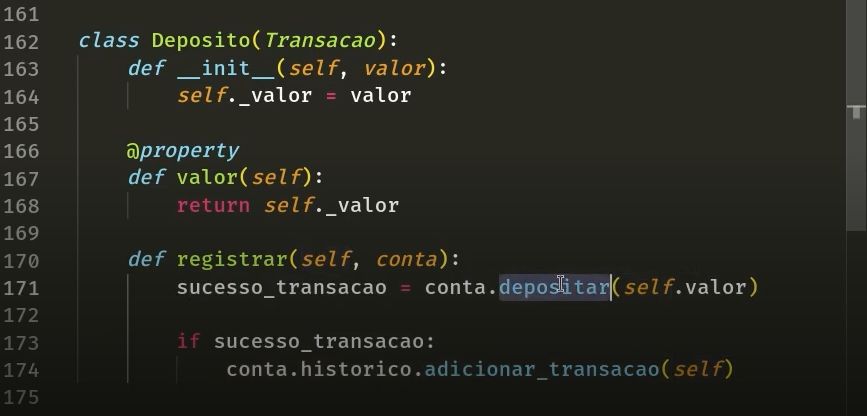












**RESOLUÇÃO: parte 2**





