

# Banco Python

Nome: Tális Breda (22102202)

## 1. Execução da aplicação

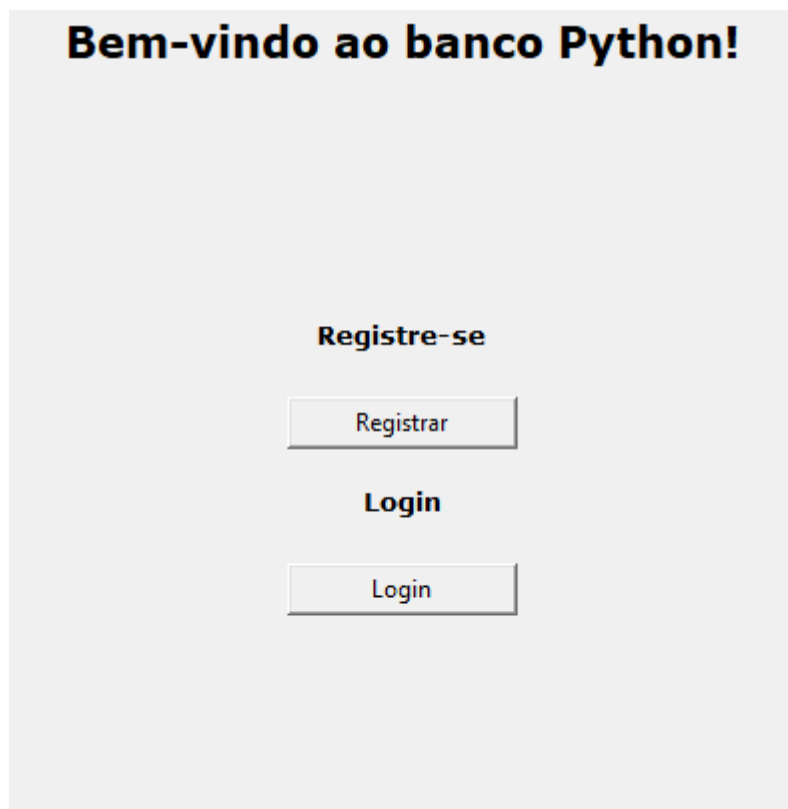
A aplicação contém quatro arquivos .py, sendo dois relacionado às classes (account e client), um relacionado às operações back-end/API (main) e outro responsável pela geração e gerenciamento das interfaces.

Para executar corretamente o programa, deve-se executar o arquivo **interfaceModules.py**.

## 2. Interface

Toda a interface da aplicação foi construída utilizando a biblioteca Tkinter do Python, e cada uma das telas/módulos é uma classe própria, apesar de todas estarem no mesmo arquivo.

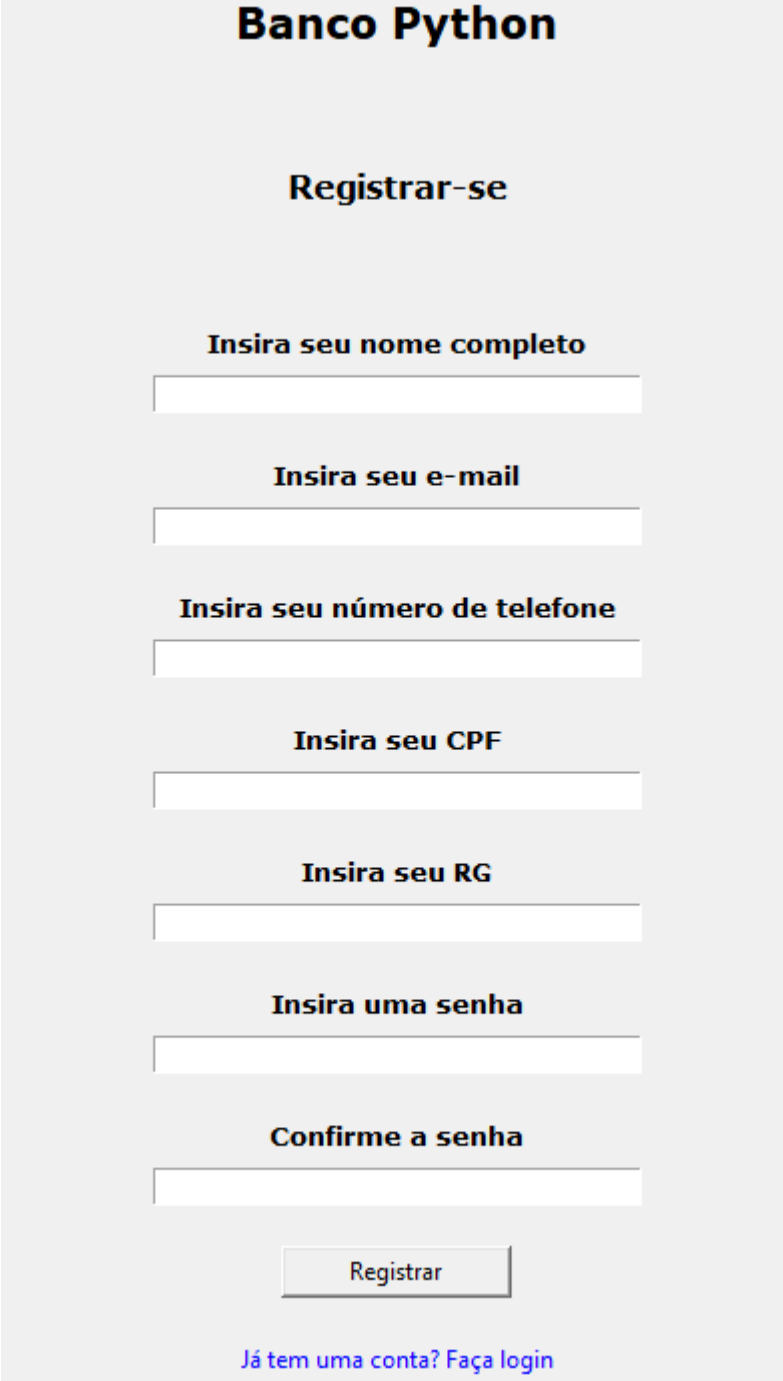
### 2.1. Tela inicial



A primeira tela a ser mostrada ao executar a aplicação. Dá ao usuário opções de registro e de login. Foi construída utilizando dois Frames, três Labels e dois Buttons.

Ao clicar no botão "Registrar", o usuário será direcionado para a tela de registro/criação de conta, e ao clicar no botão "Login", será direcionado para a tela de login.

## 2.2. Tela de registro



**Banco Python**

**Registrar-se**

**Insira seu nome completo**

**Insira seu e-mail**

**Insira seu número de telefone**

**Insira seu CPF**

**Insira seu RG**

**Insira uma senha**

**Confirme a senha**


[Já tem uma conta? Faça login](#)

Nessa tela, o usuário insere todos os dados necessários para criar uma conta, sendo estes nome, e-mail, telefone, RG, CPF e senha, além de uma confirmação da senha. Esse módulo foi construído utilizando dez Labels, sete Entries e um Button.

Ao finalizar a inserção de dados nos inputs, o usuário pode clicar no botão de registro, e o programa fará uma tentativa de inserir os dados no banco de dados.

Caso algum input esteja vazio, será mostrada uma mensagem pedindo ao usuário que preencha todas as entradas:


 Erro ×

 Por favor preencha todos os campos

OK

Caso o e-mail já exista, será mostrado um aviso com a mensagem “E-mail já está em uso”:


 Erro ×

 E-mail já está em uso

OK

Caso o formato do e-mail esteja fora do padrão, será exibido outro aviso com a mensagem “E-mail inválido”:


 Erro ×

 E-mail inválido

OK

Caso os campos de telefone, CPF ou RG estejam vazios ou contenham caracteres que não sejam números, será mostrada uma mensagem alertando o usuário que esses campos devem conter apenas números:

 Erro ×

 Os campos telefone, CPF e RG devem conter APENAS números

OK

Caso a senha inserida no primeiro campo não corresponda com a senha inserida no segundo, outra mensagem de aviso será mostrada, alertando que as senhas não correspondem.

Erro ✕



As senhas não correspondem

OK

Caso o CPF inserido seja inválido, um erro será mostrado com a mensagem “CPF inválido”:

Erro ✕



CPF é inválido

OK

Caso nenhum desses casos ocorra, os dados serão inseridos no banco de dados e uma nova conta será criada.

No rodapé da página, há um link “Já tem uma conta? Faça login”. Ao clicar nesse link, o usuário será direcionado à tela de login.

### 2.3. Tela de login



The image shows a login form for 'Banco Python'. At the top, the title 'Banco Python' is displayed in a bold, black font. Below it, the word 'Login' is centered. The form consists of two input fields: the first is labeled 'E-mail' and the second is labeled 'Senha' (Password). Both labels are centered above their respective input boxes. Below the password field is a 'Login' button. At the bottom of the form, there is a link that says 'Não tem conta? Registre-se' (Don't have an account? Register).

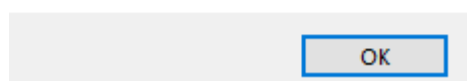
Na tela de login, há dois campos de input, correspondentes ao e-mail e senha, que deverão ser preenchidos pelo usuário. Esse módulo contém cinco Labels, dois Entries e um Button. Ao finalizar o preenchimento dos inputs, o usuário aperta no botão “Login” e o programa tenta realizar a autenticação.

Caso as credenciais não correspondam a nenhuma conta no banco de dados, será mostrada um erro com a mensagem “Credenciais estão incorretas”

 Erro ×



Credenciais estão incorretas

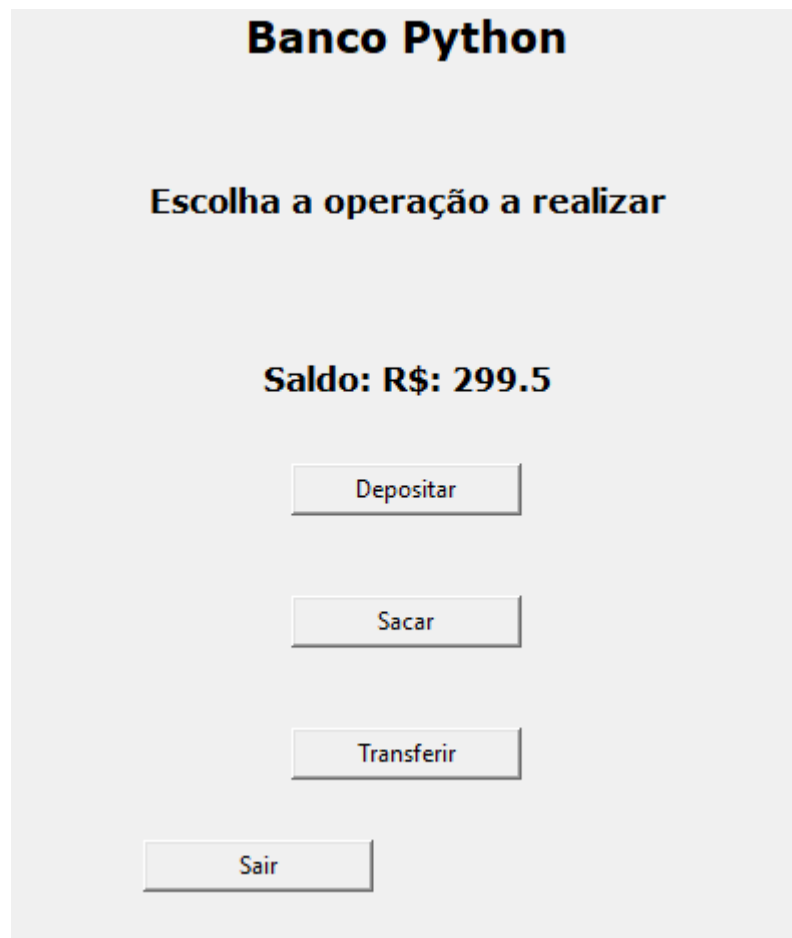


A rectangular button with a blue border and the text 'OK' in the center.

Caso ambas credenciais correspondam a uma conta do banco de dados, a autenticação será efetuada.

No rodapé da página, há um link escrito “Não tem uma conta? Registre-se”. Ao clicar, o usuário será redirecionado à tela de registro/criação de conta.

## 2.4. Tela principal/operações



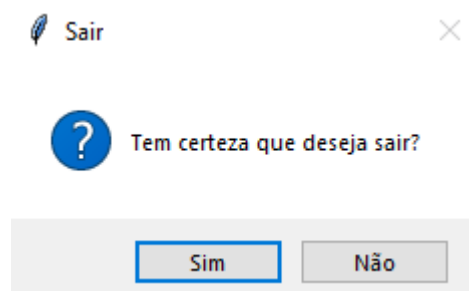
Essa tela contém quatro botões que podem ser clicados, além de mostrar o saldo da conta acessada. Foi construída utilizando três Labels e quatro Buttons. O usuário pode escolher a operação a ser realizada, ou sair/fazer logout.

Ao clicar em “Depositar”, o usuário será direcionado a uma tela onde pode inserir um valor a ser depositado.

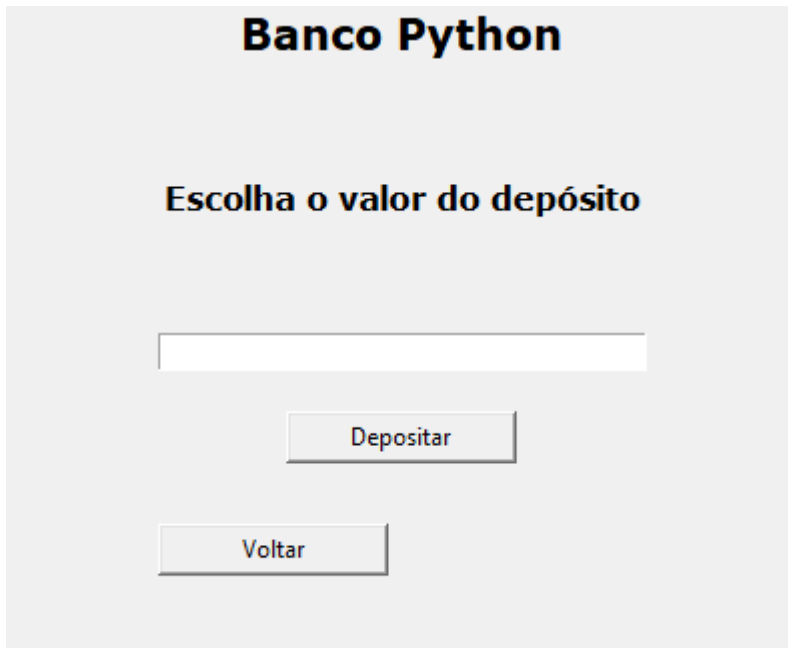
Ao clicar em “Sacar”, o usuário será direcionado a uma tela onde pode inserir um valor a ser retirado.

Ao clicar em “Transferir”, o usuário será direcionado a uma tela onde pode escolher, além de um valor para transferir, uma conta para onde o valor será transferido.

Ao clicar em “Sair”, será mostrada uma caixa de confirmação, perguntando se o usuário realmente deseja sair. Ao clicar em “Sim”, ele será redirecionado à tela inicial. Ao clicar em “Não”, permanecerá na mesma tela.



## 2.5. Tela de depósito



**Banco Python**

**Escolha o valor do depósito**

Depositar

Voltar

Esse módulo contém dois Labels, um Entry e dois Buttons. O usuário pode digitar um valor a ser depositado e clicar em “Depositar”, ou clicar em “Voltar” e voltar para a tela principal.

Ao clicar em “Depositar”, o programa tentará depositar o valor inserido no input na conta do usuário. Caso o valor do input seja nulo ou não-numérico, aparecerá um erro na tela com a mensagem “Favor inserir um número válido”:

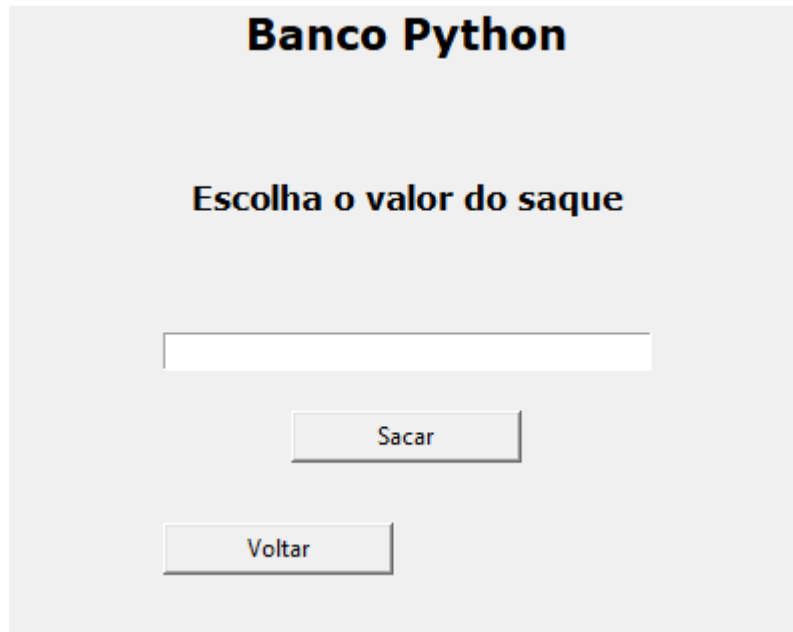
✎ Erro ✕



Favor inserir um número válido

OK

## 2.6. Tela de saque



**Banco Python**

**Escolha o valor do saque**

Sacar

Voltar

Esse módulo contém dois Labels, um Entry e dois Buttons. O usuário pode digitar um valor a ser retirado e clicar em “Sacar”, ou clicar em “Voltar” e voltar para a tela principal.

Ao clicar em “Sacar”, o programa tentará retirar o valor inserido do saldo da conta do usuário.

Caso o input seja nulo ou não-numérico, será mostrado um aviso com a mensagem “Favor inserir um número válido”

✎ Erro ✕



Favor inserir um número válido

OK

Caso o input contenha um valor numérico maior que o saldo atual da conta, será mostrado um aviso com a mensagem “Saldo insuficiente”

✎ Erro ✕

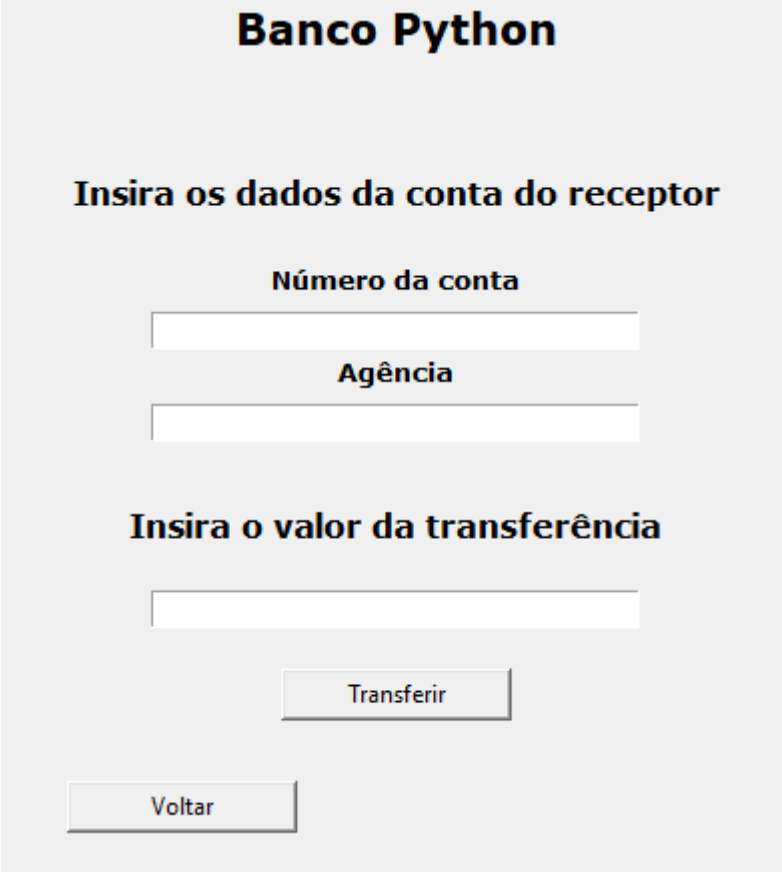


Saldo insuficiente

OK



## 2.7. Tela de transferência



**Banco Python**

**Insira os dados da conta do receptor**

Número da conta

Agência

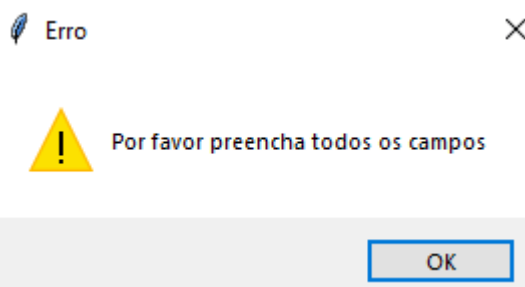
**Insira o valor da transferência**

Transferir

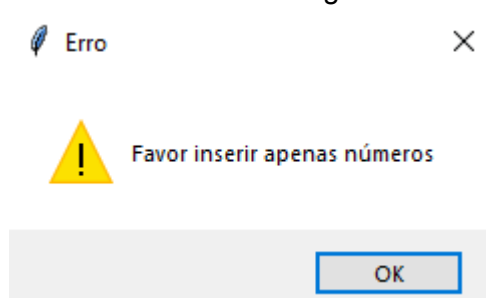
Voltar

Esse módulo contém cinco Labels, três Entries e dois Buttons. Nessa tela, o usuário pode escolher uma conta e um valor para transferir. Ao preencher os três inputs e clicar em “Transferir”, o programa tentará realizar a transferência do valor para a conta indicada.

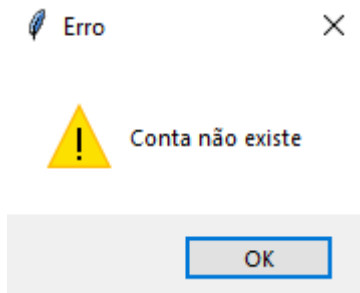
Caso pelo menos um dos campos esteja vazio, será mostrado um aviso contendo a mensagem “Por favor preencha todos os campos”



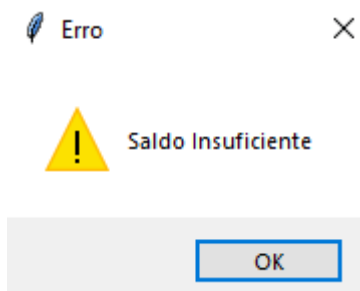
Caso pelo menos um dos inputs seja não-numérico, será exibido um aviso contendo a mensagem:



Caso os dados da conta inseridos não correspondam a nenhuma conta existente no banco de dados, será exibido outro erro com a mensagem “Conta não existe”:

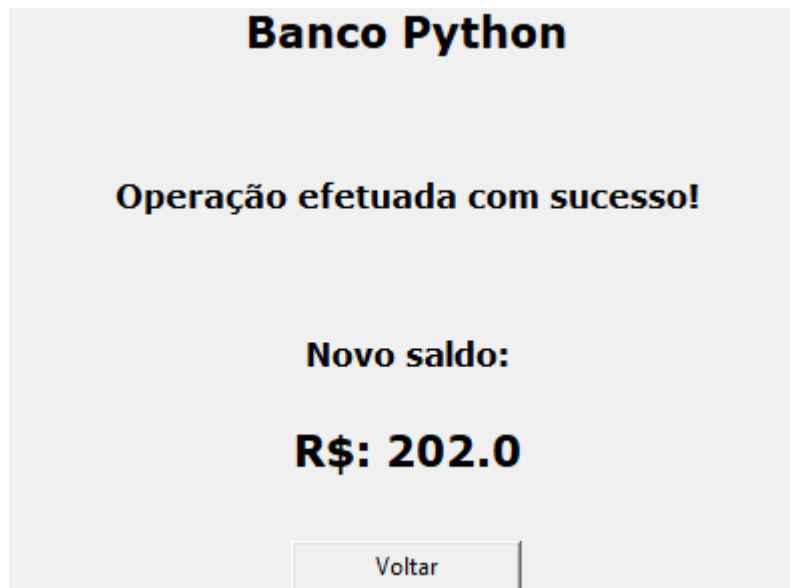


Caso o valor inserido seja maior que o saldo atual da conta, será exibido outro erro com a mensagem “Saldo insuficiente”:



Caso nenhum dos casos anteriores aconteça, a transação será efetivada.

## 2.8. Novo saldo



Essa tela contém quatro Labels e um botão, e é apenas mostrada quando uma operação é realizada com sucesso, mostrando o novo saldo. Ao clicar no botão “Voltar”, o usuário será redirecionado à tela principal.

## 3. Back-end

O back-end da aplicação foi construído com base na biblioteca SQLAlchemy do Python, cuja funcionalidade é prover maneiras de conectar a aplicação em Python com

um banco de dados, com suporte para várias linguagens de gerenciamento de banco de dados. No caso desta aplicação, a biblioteca foi utilizada com MySQL.

Resumidamente, o funcionamento do SQLAlchemy se dá através da criação de uma engine, onde é inserida a URL do banco de dados. Uma engine é capaz de suportar várias conexões ao mesmo tempo. No caso desta aplicação, o banco de dados já existia previamente, então para a conexão e operações foi utilizada a propriedade `text()` da engine. Com essa propriedade pode-se realizar operações inserindo um texto em SQL.

Também é possível criar um banco de dados diretamente pelo Python utilizando essa linguagem, e nesse caso não é necessário utilizar o `text()`. A escolha de criar um banco de dados separadamente é por preferência e por afinidade.

### 3.1. Main

A função `main` contém diversas funções que são executadas fora do escopo das classes do programa. Por exemplo, as funções de autenticação, criação de usuário, criação de conta, busca de usuário e busca de conta estão contidas no `main.py`.

#### 3.1.1. `createNewClient()`

Na operação de registro de usuário, a função `createNewClient()` é chamada. Ela recebe todos os dados do usuário (nome, e-mail, telefone, cpf, rg e senha). Entre suas responsabilidades estão:

- Gerar um ID aleatório de 6 dígitos e conferir sua existência no banco de dados através da função `checkId()`;
- Criptografar a senha utilizando SHA256;
- Checar a validade e a existência do e-mail no banco de dados através da função `checkEmail()`;
- Inserir uma nova linha na tabela `client` no banco de dados;
- Instanciar uma classe `Client` com os mesmos dados;
- Retornar a função `bank()` que, nesse caso, cria uma nova linha na tabela `account` do banco de dados.

#### 3.1.2. `authenticate()`

Na operação de login, a função `authenticate()` é chamada. Essa função busca no banco de dados a senha relacionada com o e-mail inserido pelo usuário, e compara esta com a senha inserida. Caso a comparação retorne `True`, a função `accessClient()` é chamada. Caso a comparação retorne `False`, um erro é criado, que será tratado na interface, com a mensagem “Credenciais estão incorretas”.

#### 3.1.3. `accessClient()`

A função `accessClient()` é chamada dentro de `authenticate()`, e é responsável por buscar os dados do cliente no banco de dados e instanciar uma classe utilizando os dados obtidos. Após isso, chama a função `bank()`, que acessa a conta bancária relacionada a este cliente

#### 3.1.4. `bank()`

Essa função é chamada em **createNewClient()** e em **accessClient()**, e é responsável por criar e acessar contas bancárias dos clientes.

- Caso o cliente seja recém-criado, a função **createNewAccount()** é chamada para criação de uma nova conta bancária no banco de dados, e o status de cliente recém criado é desativado
- Caso o cliente já exista previamente, a função **accessAccount()** é chamada, buscando no banco de dados a conta relacionada ao cliente.

#### 3.1.5. **createNewAccount()**

Essa função é chamada dentro da função **bank()**, e tem algumas responsabilidades:

- Gerar um ID e um número de conta aleatórios, e checar a existência destes no banco de dados
- Inserir uma nova linha na tabela account do banco de dados, utilizando os dados que recebeu
- Instanciar uma classe Account utilizando os mesmos dados

#### 3.1.6. **accessAccount()**

Essa função é chamada dentro da função **bank()**, e tem como responsabilidade buscar no banco de dados as informações da conta bancária, e instanciar uma classe Account utilizando os dados obtidos.

#### 3.1.7. **checkId()**

Essa função é chamada durante a inserção de uma linha em alguma tabela, e recebe um número como parâmetro. Sua função é buscar no banco de dados alguma linha cujo ID seja igual ao que recebeu. Caso essa linha exista, é gerado um novo ID aleatório, que será conferido novamente. O processo se repete até que não seja encontrada uma linha com ID igual, e neste caso a operação continuará.

#### 3.1.8. **checkAcc()**

Tem a mesma função de **checkId()**, porém checa número da conta e não ID.

#### 3.1.9. **checkEmail()**

Essa função é chamada durante a criação de um novo cliente. Ela é responsável por buscar alguma linha cujo e-mail seja igual ao que está sendo testado. Além disso, ela checa a sintaxe e o formato do e-mail, e retorna um erro caso este seja inválido.

#### 3.1.10. **checkCPF()**

Essa função é chamada durante a criação de um novo cliente. Ela é responsável por fazer a validação do CPF inserido, utilizando a [fórmula](#) baseada nos dígitos registradores

### 3.2. **Account**

Account é uma classe criada para a realização das operações que envolvem alterações nos dados do banco, geralmente o saldo.

- Id: um número inteiro de 6 dígitos
- Client: recebe uma classe Client com todos os dados do cliente dono da conta
- Balance: saldo da conta

```
class Account():  
  
    def __init__(self, client, id) -> None:  
        self.id = id  
        self.client = client  
        self.balance = 0
```

#### 3.2.1. retrieveBalance()

Função cuja responsabilidade é buscar no banco de dados o saldo da conta com o ID informado

#### 3.2.2. updateBalance()

Função cuja responsabilidade é atualizar o saldo no banco de dados com o saldo informado

#### 3.2.3. deposit()

Essa função é responsável por chamar a função **retrieveBalance()**, inserir seu resultado na variável balance, somar essa variável com o valor informado, e chamar a função **updateBalance()** para atualizar o saldo no banco de dados.

#### 3.2.4. withdraw()

Essa função é responsável por chamar a função **retrieveBalance()** e inserir seu resultado na variável balance. Caso o valor informado seja maior que o saldo obtido, retorna um erro com a mensagem "Saldo insuficiente". Caso não, subtrai do saldo o valor informado e chama **updateBalance()** para atualizar o saldo no banco de dados.

#### 3.2.5. transfer()

Essa função é responsável por chamar a função **retrieveBalance()** e inserir seu resultado na variável balance. Caso o valor informado seja maior que o saldo obtido, retorna um erro com a mensagem "Saldo insuficiente". Caso não, subtrai do saldo o valor informado e chama a função **transferToReceiver()**, que adiciona o valor informado ao saldo da conta de destino. Por fim, chama **updateBalance()** para que o saldo desta seja atualizado.

#### 3.2.6. transferToReceiver()

Essa função é responsável por atualizar o saldo da conta que receberá a transferência. Suas ações são:

- Buscar no banco de dados o saldo da conta cujos dados são os informados
- Somar o valor informado com o saldo obtido na busca

- Atualizar o saldo da conta com o novo valor obtido da soma

Caso algum dos passos anteriores retorne um erro, significa que a conta informada pelo usuário não existe, e um erro informando isso será mostrado ao usuário.

### 3.3. Client

Função que existe para armazenar os dados do cliente durante a execução do programa. Seus atributos são id, name, email, phone, cpf, rg e new (conta recém criada ou não). Essa classe não tem funções além das padrões get e set da classe, pois existe para armazenar os dados do usuário na duração da execução.

```
class Client():
    def __init__(self, id, name, email, phone, cpf, rg, new) -> None:
        self.id = id
        self.name = name
        self.email = email
        self.phone = phone
        self.cpf = cpf
        self.rg = rg
        self.new = new
```

## 4. Banco de dados

O banco de dados da aplicação é bastante simples, contendo apenas duas tabelas com uma chave estrangeira em account. Foi feito utilizando MySQL, através da IDE MySQL Workbench.

A tabela client possui as colunas:

- idClient (chave primária, ID)
- Passcode (senha)
- Fullname (nome completo)
- Email
- Phone (telefone)
- CPF
- RG
- 

A tabela account possui as colunas:

- idAccount (chave primária, ID)
- Holder (ID do usuário dono da conta)
- AccNumber (número da conta)
- Agency (número da agência)
- accType (tipo de conta)
- Balance (saldo)

