Projeto “Econo-me Bank”

Mateus Marana Assuena, RA 22.123.026.1

Gustavo Oliveira Rodrigues, RA 22.222.007-1

Projeto feito nas aulas práticas de Fundamentos de Algoritmos com a professora Gabriela.

Foi nos pedidos a interface de um banco e as seguintes funções:

- Cadastro de novos clientes
- Apagar clientes
- Listar clientes
- Depositar valores
- Debitar valores
- Extrato de valores
- Transferir valores entre as contas
- Investimentos

O projeto teve início no dia 20/04 e foi feito até o dia 26/05

- ## 20/04 – Primeiro dia

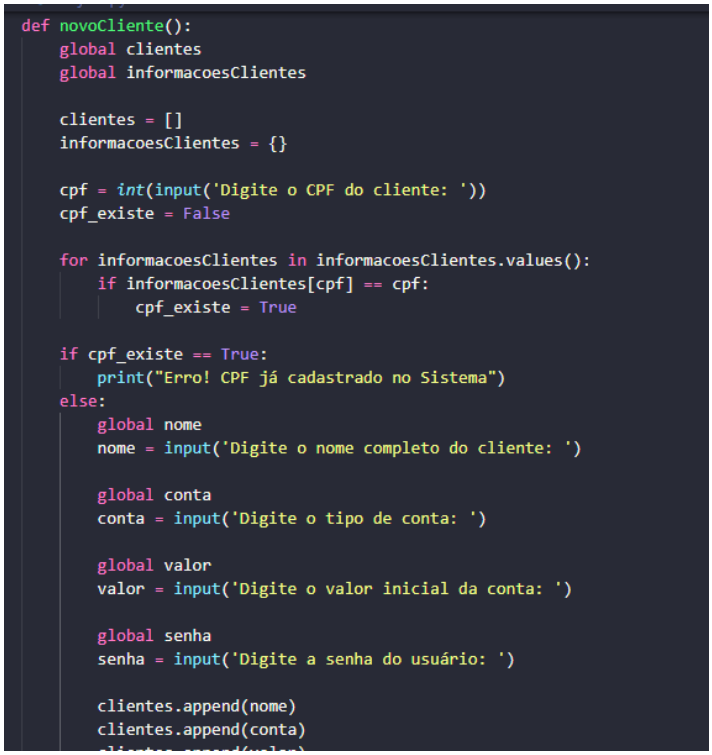
No dia 20/04 nós fizemos a interface e as funções de cadastrar e apagar os clientes.

Interface

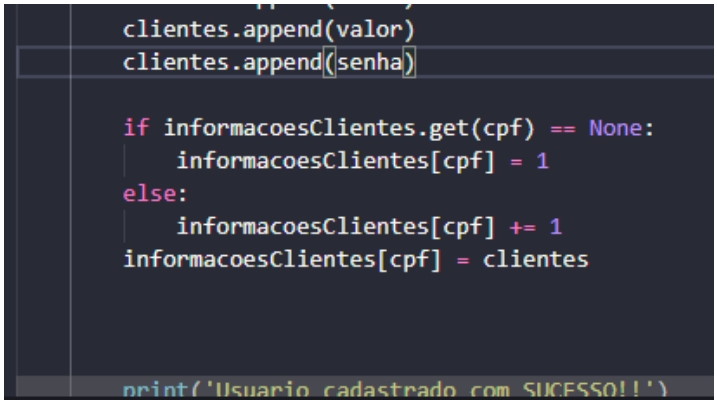
```
eto > Projeto.py
    #Painel
    while True:
        print('--------------------')
        print('$$ ECONO-ME BANK $$')
        print('--------------------')
        print()
        print('Para Cadastrar Novo Cliente - Digite 1')
        print('Para Apagar Cliente pelo Cpf - Digite 2')
        print('Para Listar os Clientes Existentes - Digite 3')
        print('Para Debitar um Valor na Conta - Digite 4')
        print('Para Depositar um Valor na Conta - Digite 5')
        print('Para Tirar Extrato - Digite 6')
        print('Para Transferir um Valor -Digite 7')
        print('Para Investir um Valor - Digite 8')
        print('Para Fechar o programa - Digite 9')
        print()
        decisao = input('O que Deseja?: ')
        if decisao == '1': novoCliente()
        if decisao == '2': apagarCliente()
        if decisao == '3': listarClientes()
        if decisao == '4': debitarValor()
        if decisao == '5': depositaValor()
        if decisao == '6': extratoDaConta()
        if decisao == '7': transferirValor()
        if decisao == '8': investimentos()
        if decisao == '9': break
```

Interface produzida dentro do laço WHILE, a partir da decisão escolhida pelo usuário chama a função correspondente. Se a decisão escolhida for igual a 9, o código se encerra.

Função De Cadastro Para Novos Clientes



```
def novoCliente():
    global clientes
    global informacoesClientes

    clientes = []
    informacoesClientes = {}

    cpf = int(input('Digite o CPF do cliente: '))
    cpf_existe = False

    for informacoesClientes in informacoesClientes.values():
        if informacoesClientes[cpf] == cpf:
            cpf_existe = True

    if cpf_existe == True:
        print("Erro! CPF já cadastrado no Sistema")
    else:
        global nome
        nome = input('Digite o nome completo do cliente: ')

        global conta
        conta = input('Digite o tipo de conta: ')

        global valor
        valor = input('Digite o valor inicial da conta: ')

        global senha
        senha = input('Digite a senha do usuário: ')

        clientes.append(nome)
        clientes.append(conta)
```



```
        clientes.append(valor)
        clientes.append(senha)

        if informacoesClientes.get(cpf) == None:
            informacoesClientes[cpf] = 1
        else:
            informacoesClientes[cpf] += 1
        informacoesClientes[cpf] = clientes



        print('Usuario cadastrado com SUCESSO!!')
```

Função de cadastrar novos clientes, ela cria um dicionário e uma lista dentro do dicionário. Há inputs que armazenam em variáveis os dados dos usuários e depois as variáveis criadas são armazenadas na lista, exceto o CPF que é a chave do dicionário.

Ou seja, a partir do CPF a lista, contendo as outras informações, é acessada.

Última parte da função é para conferir se existe o CPF cadastrado e caso existente não pode ser cadastrado de novo.

Função Para Apagar Clientes Cadastrados

```python
# Função apagar cliente
def apagarCliente():
    print()
    cpf = input('Digite o CPF a ser deletado: ')
    for x in clientes:
        if x == cpf:
            remover = input("Deseja realmente apagar o ccleliente pertencente ao CPF %s \n \nDigite 1 para confirmar \n \n[
            if(remover == "1"):
                #Apagar
                print("Tudo bem, cliente com CPF %s removido" % (cpf))
                del clientes[cpf]
                break
            else:
                #Nao Apagar
                print("Tudo bem, cliente não removido")
                break
        else:
            #cpf nao existe
            print("CPF nao condiz a algum cliente cadastrado")
            break
```

Função que apaga clientes ao digitar o CPF do mesmo, após a inserção a varável X dentro do laço FOR irá percorrer o dicionário e se for encontrado um CPF igual ao digitado, todas as informações serão deletadas.

- **27/04 –Segundo dia (Aniversário do Mateus)**

Função Para Listar Os Clientes Existentes

```python

    # Função Exibir os Clientes Cadastrados
    def listarClientes():
        if clientes == {}:
            print("Não há clientes cadastrados")
        else:
            print("Listagem de Clientes Cadastrados")
            print('--------------------')
            for x,y in informacoesClientes.items():
                    print('CPF = %s : %s' % (x,y))
                    break


```

Caso o dicionário for vazio vamos informar que não existe clientes cadastrados. Caso contrário (ELSE) irá listar os clientes através das variáveis X e Y que iteram nos itens presentes no dicionário.

Última função feita no nosso código antigo, pois estava com problemas para salvar novos clientes devido ao erro na lógica.

Estávamos utilizando a lógica de Lista dentro de Dicionário.

No novo código usamos a lógica de Dicionários dentro de Lista.

Toda documentação feita a partir deste ponto será usado o código novo.

Novo código

- **Feito do dia 03/05 (Fora de Aula) até o dia 26/05**

Imports

econo-me.py > carregar_dados_arquivo

```python
# Gustavo Rodrigues e Mateus Marana

import random
from datetime import datetime

```

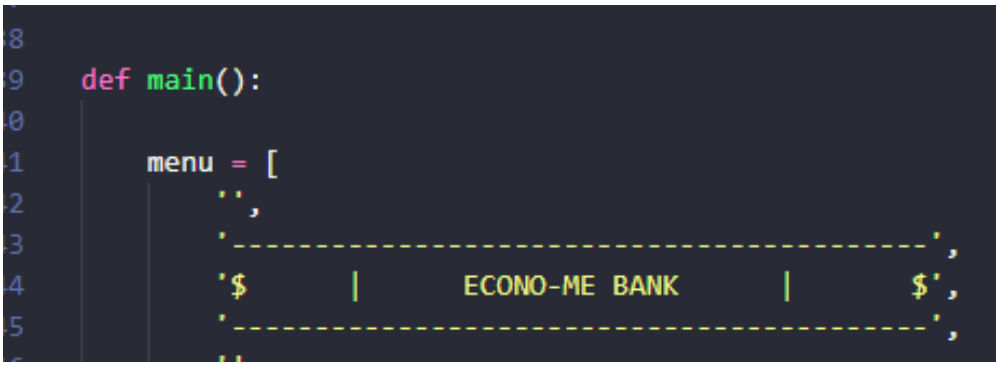
Função IMPORT RANDOM e IMPORT DATETIME, usadas para a função de investimentos e para salvar a hora e a data na função de extrato, respectivamente.

Salvar Dados Em Arquivos

```python

def salvar_dados_arquivo(dados, nome_arquivo):
    with open(nome_arquivo, 'w') as arquivo:
        for usuario, conta in zip(dados['usuarios'], dados['contas']):
            nome = usuario['nome']
            cpf = usuario['cpf']
            tipo_conta = conta['tipoConta']
            senha = usuario['senha']
            saldo = conta['saldo']
            transacoes = ','.join(str(t) for t in conta['transacoes'])
            linha = f"{nome},{cpf},{tipo_conta},{senha},{saldo},{transacoes}\n"
            arquivo.write(linha)

```

Ela recebe dois parâmetros: dados, que contém a estrutura de dados com as informações dos usuários e contas do banco, e NOME_ARQUIVO, que é o nome do arquivo onde os dados serão salvos. A função abre o arquivo especificado em modo de escrita **('w')** usando o OPEN(). Em seguida, percorre os elementos de DADOS[‘USUARIOS’] e DADOS[‘CONTAS’] simultaneamente usando a função ZIP(). Essa função permite percorrer duas listas em paralelo. Para cada par de elementos (usuário e conta), são extraídas as informações relevantes para serem salvas no arquivo. As informações incluem o nome do usuário, CPF, tipo da conta, senha, saldo e transações. O saldo é convertido em uma string usando STR(saldo). Nesse caso, cada transação é convertida em uma string usando STR(t) e separadas por vírgula. Com todas as informações necessárias, é criada uma linha no formato especificado para cada usuário e conta. A linha é criada usando f-string, onde as informações são inseridas nos devidos lugares. Cada linha é escrita no arquivo usando o método WRITE(). Após percorrer todos os elementos, a função fecha o arquivo.

Main



Função que contém todo o código que será executado.

Lista Menu

```
menu = [
    '',
    '----------------------------------------',
    '$      |      ECONO-ME BANK      |      $',
    '----------------------------------------',
    '',
    'Usuarios',
    ' Digite 1 - Cadastrar Novo Cliente',
    ' Digite 2 - Apagar Cliente pelo CPF',
    ' Digite 3 - Listar os Clientes Existentes',
    '',
    'Conta',
    ' Digite 4 - Debitar um Valor na Conta',
    ' Digite 5 - Depositar um Valor na Conta',
    ' Digite 6 - Tirar Extrato',
    ' Digite 7 - Transferir um Valor',
    ' Digite 8 - Investir um Valor',
    '',
    'Sair',
    ' Digite 9 - Fechar o programa',
    ''
]
```

Lista criada para armazenar a interface e será utilizada mais para frente no código.

AppBanco E Investimentos Menu

```
appBanco = {
    'usuarios': [],
    'contas': []
}

investimentos_menu = [
    '',
    ' Digite 1 - Criptmoedas',
    ' Digite 2 - Fundo Conservador',
    ' Digite 3 - Fundo Imobiliario',
    ' Digite 4 - Fundo de Ações',
    ' Digite 5 - Poupança',
    ' Digite 6 - Cancelar',
    ''
]
```

AppBanco é um dicionário que armazena os dados de cada usuário, a chave USUARIOS é destinada ao nome, CPF e senha. Já a chave CONTAS é destinada ao dados da conta, como valor, tipo e as transações feitas.

A lista INVESTIMENTOS_MENU é usada como interface para escolher o investimento que será realizado.

Função De Gerar Valor E Opções Investimentos

```
def gerar_valor_aleatorio():
    return random.random()

opcoes_investimentos = {
    '1': gerar_valor_aleatorio,
    '2': gerar_valor_aleatorio,
    '3': gerar_valor_aleatorio,
    '4': gerar_valor_aleatorio,
    '5': gerar_valor_aleatorio,
}
```

Função Gerar Valor gera um valor aleatório que é usado no dicionário Opções Investimentos. Será usado na função de investimentos.

Função Calcular Retorno Investimento

```
cono-me.py > ...

    def calcular_retorno_investimento(valor_investido, taxa_retorno):
        retorno = random.randint(-1, 2)
        return valor_investido * (retorno * taxa_retorno)
```

Função utilizada para calcular o retorno dos investimentos, podendo ser positivo ou negativo. Também usada na função de investimentos.

Funções Limpar Tela e Exibir Menu

```
def limpar_tela():
    linhas_terminal = 40  # número de linhas do terminal
    for _ in range(linhas_terminal):
        print()

def exibir_menu():
    for linha in menu:
        print(linha)
```

A função Limpar Tela limpa o terminal após a utilização, sempre que entramos em umas das funções disponíveis no menu.

A função Exibir Menu irá percorrer a lista MENU e irá mostrar na tela o seu conteúdo, nesse caso:

$ | ECONO-ME BANK | $

Usuários

Digite 1 - Cadastrar Novo Cliente

Digite 2 - Apagar Cliente pelo CPF

Digite 3 - Listar os Clientes Existentes

Conta

Digite 4 - Debitar um Valor na Conta

Digite 5 - Depositar um Valor na Conta

Digite 6 - Tirar Extrato

Digite 7 - Transferir um Valor

Digite 8 - Investir um Valor

Sair

Digite 9 - Fechar o programa

Escolha uma opção:

Novo Cliente

```python
def novo_cliente():
    nome = input("Digite o nome do usuário: ")
    cpf = input("Digite o CPF do usuário: ")

    tipo_conta = input("\nTipo da Conta\n\nDigite 1 - Conta Premium\nDigite 2 - Conta Comum\nDigite o tipo da conta: ")
    if tipo_conta == '1':
        print('Cliente Premium!')
        tipo_conta = 'Premium'

    elif tipo_conta == '2':
        print('Cliente Comum!')
        tipo_conta = 'Comum'
    else:
        print("Opção inválida. Digite um número válido.")
        return
    senha = input("Digite a senha do usuário: ")
    valor = float(input("Digite o valor inicial da conta: "))

    for usuario in appBanco['usuarios']:
        if usuario['cpf'] == cpf:
            print("\nCPF já cadastrado. O usuário não será cadastrado novamente.")
            return

    usuario = {
        'nome': nome,
        'senha': senha,
        'cpf': cpf
    }
    conta = {
        'usuario': usuario,
        'saldo': valor,
        'tipoConta': tipo_conta,
        'transacoes': []
    }
    appBanco['usuarios'].append(usuario)
    appBanco['contas'].append(conta)
    print("\nUsuário cadastrado com sucesso!\n")
```

A função inicia solicitando ao usuário as informações necessárias para o cadastro: nome, CPF, tipo de conta, senha e valor inicial da conta. O tipo de conta é verificado para determinar se é uma "Conta Premium" ou "Conta Comum". O usuário é informado sobre o tipo de conta selecionado. A senha e o CPF são verificados para garantir que o CPF não esteja duplicado no sistema. Se um CPF já estiver cadastrado, uma mensagem de aviso é exibida e o usuário não será cadastrado novamente. Se o CPF for único, um novo dicionário de usuário (usuário) e conta (conta) é criado com base nas informações fornecidas. O dicionário do usuário é adicionado à lista de usuários (APPBANCO[‘USUARIOS’]), e o dicionário da conta é adicionado à lista de contas (APPBANCO['CONTAS']). Uma mensagem de confirmação é exibida, informando que o usuário foi cadastrado com sucesso. Essa função permite o cadastro de novos clientes no sistema bancário, armazenando suas informações na estrutura de dados AppBanco. Essas informações poderão ser utilizadas posteriormente em outras operações do sistema, como depósitos, saques e consulta de saldo.

Apagar Cliente

```python
def apagar_cliente():
    cpf = input("Digite o CPF do usuário a ser excluído: ")
    print()
    for i, usuario in enumerate(appBanco['usuarios']):
        if usuario['cpf'] == cpf:
            del appBanco['usuarios'][i]
            del appBanco['contas'][i]
            print("Usuário excluído com sucesso!")
            return
    print("\nUsuário não encontrado.")
```

A função solicita ao usuário o CPF do cliente a ser excluído. Em seguida, um loop for é utilizado juntamente com a função ENUMERATE() para percorrer a lista de usuários (APPBANCO [‘USUARIOS’]) juntamente com seus índices. Para cada usuário, é verificado se o CPF fornecido corresponde ao CPF do usuário atual. Se um usuário com o CPF correspondente for encontrado, os dados desse usuário são removidos das listas APPBANCO[‘USUARIOS’] e APPBACNO[‘CONTAS’] utilizando o comando del. Uma mensagem de confirmação é exibida informando que o usuário foi excluído com sucesso. Caso nenhum usuário com o CPF correspondente seja encontrado, uma mensagem de aviso é exibida informando que o usuário não foi encontrado. Essa função permite a exclusão de um cliente do sistema bancário, removendo suas informações das listas APPBANCO[‘USUARIOS’] e APPBANCO['CONTAS']. É importante ressaltar que a exclusão de um cliente também implica na exclusão da conta associada a ele.

Listar Clientes

```
def listar_clientes():
    if len(appBanco['usuarios']) == 0 or len(appBanco['contas']) == 0:
        print("Não há clientes cadastrados.")
        return

    print("-----------------------------------------")
    print("|   Nome   |      CPF       |     Valor  |")
    print("-----------------------------------------")
    for usuario, conta in zip(appBanco['usuarios'], appBanco['contas']):
        nome = usuario['nome']
        cpf = usuario['cpf']
        saldo = conta['saldo']
        print(f"| {nome:<8} | {cpf:<14} | R${saldo:<14.2f}")
    print("-----------------------------------------")
```

A função verifica se há clientes cadastrados verificando o tamanho das listas APPBANCO[‘USUARIOS’] e APPBANCO['CONTAS']. Se alguma das listas estiver vazia, uma mensagem é exibida informando que não há clientes cadastrados, e a função é encerrada. Caso existam clientes cadastrados, é exibido um cabeçalho de tabela com as colunas "Nome", "CPF" e "Valor”. Utilizando um loop for juntamente com a função ZIP(), percorremos simultaneamente as listas de usuários (APPBANCO['USUARIOS']) e contas (APPBANCO['CONTAS']). Para cada par de usuário e conta, são obtidos o nome do usuário, CPF e saldo da conta. Os dados são formatados e exibidos em linhas da tabela. Ao final, é exibida uma linha de separação da tabela. Essa função permite visualizar os clientes cadastrados no sistema bancário e suas informações básicas. É uma forma conveniente de listar os clientes e seus respectivos saldos de conta.

Debitar Valor

```python
def debitar_valor():
    print("Entrou na função Debitar Valor")
    cpf = input("Digite o CPF do usuário: ")
    senha = input("Digite a senha do usuário: ")
    valor = float(input("Digite o valor a ser debitado: "))
    data_hora = datetime.now().strftime("%d/%m/%Y %H:%M:%S")
    VALOR_TARIFA_PREMIUM = 0.03
    VALOR_TARIFA_COMUM = 0.05

    tarifa_comum = valor * VALOR_TARIFA_COMUM
    tarifa_premium = valor * VALOR_TARIFA_PREMIUM
    conta_comum = valor + tarifa_comum
    conta_premium = valor + tarifa_premium

    for usuario, conta in zip(appBanco['usuarios'], appBanco['contas']):
        if usuario['cpf'] == cpf and usuario['senha'] == senha:
            if conta['saldo'] >= valor:
                if conta['tipoConta'] == 'Premium':
                    conta['saldo'] -= conta_premium
                    conta['transacoes'].append({'tipo': 'debito', 'valor': conta_premium, 'tarifa': tarifa_premium, 'data_hora': data_hora})
                    print('Valor debitado com sucesso. Tarifa de R$%.2f cobrada' % tarifa_premium)
                else:
                    conta['saldo'] -= conta_comum
                    conta['transacoes'].append({'tipo': 'debito', 'valor': conta_comum, 'tarifa': tarifa_comum, 'data_hora': data_hora})
                    print('Valor debitado com sucesso. Tarifa de R$%.2f cobrada' % tarifa_comum)
                return
            else:
                print("Saldo insuficiente para debitar valor.")
                return

    print("\nCPF ou senha incorretos.")
```

Primeiro, são solicitados o CPF, a senha e o valor a ser debitado ao usuário. Em seguida, a data e hora do momento são obtidas utilizando a função DATE.NOW().STRFTIME("%d/%m/%Y %H:%M:%S"). Essa informação será utilizada para registrar a transação no histórico. São definidos os valores das tarifas para contas Premium e Comum. Aqui, assumimos os valores de 0.03 (3%) para contas Premium e 0.05 (5%) para contas Comum.Com base no valor informado pelo usuário, são calculados os valores a serem debitados da conta, considerando as tarifas correspondentes a cada tipo de conta. Em seguida, é feita uma iteração sobre a lista de usuários e contas do banco para verificar se o CPF e a senha fornecidos correspondem a um usuário válido. Se o usuário for encontrado e a senha estiver correta, é verificado se o saldo da conta é suficiente para realizar o débito. Se o saldo for suficiente, o débito é realizado, deduzindo o valor correspondente da conta e adicionando a transação ao histórico de transações da conta. Uma mensagem é exibida informando o sucesso do débito e o valor da tarifa cobrada. Se o saldo não for suficiente, uma mensagem é exibida informando que o saldo é insuficiente para realizar o débito. Se o usuário não for encontrado ou a senha estiver incorreta, é exibida uma mensagem informando que o CPF ou a senha estão incorretos. Essa função permite debitar um valor da conta de um usuário, aplicando as tarifas correspondentes a cada tipo de conta. Além disso, registra a transação no histórico da conta, incluindo a data e hora do momento em que o débito foi realizado.

Depositar Valor

```python
def depositar_valor():
    cpf = input("Digite o CPF do usuário: ")
    senha = input("Digite a senha do usuário: ")
    valor = float(input("Digite o valor a ser depositado: "))
    data_hora = datetime.now().strftime("%d/%m/%Y %H:%M:%S")

    for usuario, conta in zip(appBanco['usuarios'], appBanco['contas']):
        if usuario['cpf'] == cpf and usuario['senha'] == senha:
            conta['saldo'] += valor
            conta['transacoes'].append({'tipo': 'deposito', 'valor': valor, 'tarifa': [], 'data_hora': data_hora})
            print("Valor depositado com sucesso!")
            return

    print("\nCPF ou senha incorretos.")
```

São solicitados o CPF, a senha e o valor a ser depositado ao usuário. A data e hora do momento são obtidas utilizando a função DATE.NOW().STRFTIME ("%d/%m/%Y %H:%M:%S"). Essa informação será utilizada para registrar a transação no histórico. É feita uma iteração sobre a lista de usuários e contas do banco para verificar se o CPF e a senha fornecidos correspondem a um usuário válido. Se o usuário for encontrado e a senha estiver correta, o valor é adicionado ao saldo da conta. Além disso, é adicionada uma nova transação ao histórico de transações da conta, com o tipo "deposito", o valor depositado, uma lista vazia para a tarifa (pois não há cobrança de tarifa no depósito) e a data e hora do depósito. Uma mensagem é exibida informando que o valor foi depositado com sucesso. Se o usuário não for encontrado ou a senha estiver incorreta, é exibida uma mensagem informando que o CPF ou a senha estão incorretos. Essa função permite realizar o depósito de um valor na conta de um usuário, atualizando o saldo da conta e registrando a transação no histórico, incluindo a data e hora do momento em que o depósito foi realizado.

Extrato Da Conta

```python
def extrato_da_conta():
    cpf = input("Digite o CPF do usuário: ")
    senha = input("Digite a senha do usuário: ")
    for usuario, conta in zip(appBanco['usuarios'], appBanco['contas']):
        if usuario['cpf'] == cpf and usuario['senha'] == senha:
            print(f"Extrato da conta de {usuario['nome']}:")
            print(f"Portador do CPF {usuario['cpf']}")
            print(f"Com a conta {conta['tipoConta']}")
            print("Histórico de transações:")

            for transacao in conta['transacoes']:
                tipo = transacao['tipo']
                valor = transacao['valor']
                tarifa = transacao['tarifa']
                data_hora = transacao['data_hora']
                if transacao['tipo'] == 'deposito':
                    print(f"| {data_hora} | {tipo:<8} | {valor:<14.2f} |")
                else:
                    print(f"| {data_hora} | {tipo:<8} | {valor:<14.2f} | R${tarifa:<14.2f}")
            return
    print("\nCPF ou senha incorretos.")
```

São solicitados o CPF e a senha do usuário. É feita uma iteração sobre a lista de usuários e contas do banco para verificar se o CPF e a senha fornecidos correspondem a um usuário válido. Se o usuário for encontrado e a senha estiver correta, o extrato da conta é exibido. São exibidas informações básicas da conta, como o nome do titular, CPF e tipo de conta. Em seguida, é exibido o histórico de transações da conta. Para cada transação no histórico, são obtidos os detalhes, como tipo (depósito ou débito), valor, tarifa (se houver) e data e hora. As informações são formatadas e exibidas na tela, incluindo a data e hora da transação. Se o usuário não for encontrado ou a senha estiver incorreta, é exibida uma mensagem informando que o CPF ou a senha estão incorretos. Essa função permite visualizar o extrato da conta de um usuário, mostrando

todas as transações realizadas, incluindo a data e hora em que cada transação ocorreu.

Transferir Valor

```python
def transferir_valor():
    origem = input("Digite o CPF da conta de origem: ")
    destino = input("Digite o CPF da conta de destino: ")
    valor = float(input("Digite o valor a ser transferido: "))

    conta_origem = None
    conta_destino = None

    for conta in appBanco['contas']:
        if conta['usuario']['cpf'] == origem:
            conta_origem = conta
        if conta['usuario']['cpf'] == destino:
            conta_destino = conta

    if conta_origem is None:
        print("Conta de origem não encontrada.")
        return
    if conta_destino is None:
        print("Conta de destino não encontrada.")
        return
    if conta_origem['saldo'] >= valor:
        conta_origem['saldo'] -= valor
        conta_destino['saldo'] += valor
        print("Transferência realizada com sucesso!")
    else:
        print("Saldo insuficiente para realizar a transferência.")
```

São solicitados o CPF da conta de origem, o CPF da conta de destino e o valor a ser transferido. São inicializadas as variáveis CONTA_ORIGEM e CONTA_DESTINO como NONE. É feita uma iteração sobre a lista de contas do banco para encontrar as contas de origem e destino com base nos CPFs fornecidos. Se uma conta de origem correspondente for encontrada, a variável CONTA_ORIGEM é atualizada com a conta encontrada. Se uma conta de destino correspondente for encontrada, a variável CONTA_DESTINO é atualizada com a conta encontrada. Se a conta de origem não for encontrada, é exibida uma mensagem informando que a conta de origem não foi encontrada, e a função retorna. Se a conta de destino não for encontrada, é exibida uma mensagem informando que a conta de destino não foi encontrada, e a função retorna. Se a conta de origem tiver saldo suficiente para realizar a transferência, o valor é deduzido da conta de origem e adicionado à conta de destino. É exibida uma mensagem informando que a transferência foi realizada com sucesso. Se a conta de origem não tiver saldo suficiente para realizar a transferência, é exibida uma mensagem informando que o saldo é insuficiente, e a transferência não é realizada. Essa função permite transferir um valor entre duas contas, verificando se as contas existem, se a conta de origem tem saldo suficiente e, em caso afirmativo, realizando a transferência. Caso contrário, são exibidas mensagens apropriadas informando sobre os problemas encontrados.

Investimentos – Função Livre

```python
def investimentos():
    cpf = input("Digite o CPF do usuário: ")
    senha = input("Digite a senha do usuário: ")
    for usuario, conta in zip(appBanco['usuarios'], appBanco['contas']):
        if usuario['cpf'] == cpf and usuario['senha'] == senha:
            print(f"Saldo atual: R$ {conta['saldo']:.2f}")
            for investimento in investimentos_menu:
                print(investimento)
            decisao = input("Escolha uma opção de investimento: ")
            if decisao == '6':
                return
            elif decisao in opcoes_investimentos:
                valor_investido = float(input("Digite o valor a ser investido: "))
                conta['saldo'] -= valor_investido
                taxa_retorno = opcoes_investimentos[decisao]()
                valor_retorno = calcular_retorno_investimento(valor_investido, taxa_retorno)
                if(valor_retorno > 0):
                    print(f"Valor de retorno: R${valor_retorno:.2f}")
                    conta['saldo'] += valor_retorno
                else:
                    print(f"Valor de retorno: R${valor_retorno:.2f}")
                    print("Seu investimento não lucro, infelizmente você perdeu seu dinheiro")
                return
            else:
                print("Opção inválida. Digite um número válido.")
                return
    print("\nCPF ou senha incorretos.")
```

São solicitados o CPF do usuário e a senha. É feita uma iteração sobre a lista de usuários e contas para encontrar a correspondência do CPF e senha fornecidos. Se um usuário correspondente for encontrado, o saldo da conta associada é exibido. É exibido o menu de opções de investimento contido na lista INVESTIMENTOS_MENU. É solicitada uma decisão do usuário em relação à opção de investimento desejada. Se a decisão for igual a '6', a função retorna, encerrando a execução. Se a decisão estiver presente no dicionário OPCOES_INVESTIMENTOS, o usuário é solicitado a digitar o valor a ser investido. O valor investido é deduzido do saldo da conta. É obtida a taxa de retorno do investimento através da função correspondente à decisão selecionada. É calculado o valor de retorno do investimento chamando a função CALCULAR_RETORNO_INVESTIMENTOS() com o valor investido e a taxa de retorno. Se o valor de retorno for maior que zero, é exibido o valor de retorno e ele é adicionado ao saldo da conta. Se o valor de retorno for menor ou igual a zero, é exibido o valor de retorno e uma mensagem informando que o investimento não teve lucro. A função retorna. Se a conta de usuário não for encontrada, é exibida uma mensagem informando que o CPF ou senha estão incorretos. Essa função permite que um usuário faça investimentos a partir de sua conta bancária, deduzindo o valor investido do saldo e adicionando o valor de retorno (se houver) ao saldo. Ela também realiza verificações de CPF, senha e opções de investimento válidas, fornecendo feedback adequado ao usuário.

Interface

```python
opcoes_menu = {
    '1': novo_cliente,
    '2': apagar_cliente,
    '3': listar_clientes,
    '4': debitar_valor,
    '5': depositar_valor,
    '6': extrato_da_conta,
    '7': transferir_valor,
    '8': investimentos
}

appBanco = carregar_dados_arquivo('dados_banco.txt')
while True:
    exibir_menu()
    salvar_dados_arquivo(appBanco, 'dados_banco.txt')
    decisao = input("Escolha uma opção: ")

    if decisao == '9':

        break
    elif decisao in opcoes_menu:
        print()
        limpar_tela()
        opcoes_menu[decisao]()
    else:
        print("Opção inválida. Digite um número válido.")
```

O dicionário OPCOES_MENU mapeia os números de opção do menu às funções correspondentes. A função CARREGAR_DADOS_ARQUIVO() é chamada para carregar os dados do arquivo 'dados_banco.txt' e atribuí-los à variável APPBANCO.É iniciado um loop WHILE TRUE, que executa continuamente até que seja explicitamente interrompido. A função EXIBIR_MENU() é chamada para exibir as opções disponíveis ao usuário. A função SALVAR_DADOS_ARQUIVO() é chamada para salvar os dados atualizados na variável APPBANCO no arquivo 'dados_banco.txt.'.O usuário é solicitado a fazer uma escolha de opção digitando um número. Se a decisão for igual a '9', o loop é interrompido com a instrução break, encerrando o programa. Se a decisão estiver presente no dicionário OPCOES_MENU, a função correspondente é chamada. Caso contrário, é exibida uma mensagem informando que a opção é inválida. O fluxo continua novamente no início do loop WHILE TRUE, onde o menu é exibido novamente e o ciclo se repete. Esse trecho de código permite a interação contínua do usuário com o banco, realizando diferentes operações bancárias conforme as opções selecionadas. Os dados são carregados do arquivo no início e salvos novamente no arquivo sempre que houver alterações. O loop é interrompido quando o usuário seleciona a opção para sair do programa.

Fim Da Função Main

```python
if __name__ == "__main__":
    main()
```