TESTES

Explicação de cada linha do código, separado por tela de cada código

1. Arquivo Setup:
2. import tela1\_login: Esta linha importa um módulo ou arquivo Python chamado tela1\_login. Ao fazer isso, o código neste arquivo (setup) obtém acesso a todas as funções, classes e variáveis que foram definidas dentro de tela1\_login.py. É como trazer um conjunto de ferramentas específicas para usar aqui.
3. import tela3\_tela\_de\_trabalho: Similar à linha anterior, esta linha importa outro módulo Python chamado tela3\_tela\_de\_trabalho. Isso permite que o código neste arquivo use os componentes definidos em tela3\_tela\_de\_trabalho.py.
4. (linha em branco): Esta linha é apenas uma linha em branco. Em Python, linhas em branco são ignoradas pelo interpretador e são usadas apenas para melhorar a legibilidade do código.
5. def main():: Esta linha define uma função chamada main. Em muitos programas Python, a função main é onde a execução principal do programa começa. O código indentado abaixo desta linha faz parte desta função.
6. usuario\_logado = tela1\_login.executar\_fluxo\_login(): Dentro da função main, esta linha chama a função executar\_fluxo\_login() que está dentro do módulo tela1\_login. O resultado retornado por essa função (que, pelo nome, provavelmente é algo indicando se o login foi bem-sucedido e talvez os dados do usuário logado) é então armazenado na variável chamada usuario\_logado.
7. (linha em branco): Outra linha em branco para legibilidade.
8. if usuario\_logado:: Esta linha verifica uma condição. Ela checa se o valor armazenado na variável usuario\_logado é "verdadeiro". Em Python, a maioria dos valores é considerada "verdadeira", exceto coisas como None, False, o número 0, strings vazias (""), listas vazias ([]), dicionários vazios ({}), etc. Neste contexto, a condição if usuario\_logado: é verdadeira se o login na linha anterior foi bem-sucedido e a variável usuario\_logado contém alguma informação sobre o usuário (ou seja, não é None ou similar indicando falha).
9. tela3\_tela\_de\_trabalho.main(usuario\_logado): Se a condição na linha 7 for verdadeira (ou seja, se o usuário\_logado tiver um valor "verdadeiro"), esta linha será executada. Ela chama a função main() que está dentro do módulo tela3\_tela\_de\_trabalho. Além disso, ela passa o valor da variável usuario\_logado como um argumento para essa função. Isso significa que a tela de trabalho (tela3\_tela\_de\_trabalho.py) receberá a informação sobre o usuário que acabou de logar.
10. (linha em branco): Mais uma linha em branco.
11. if \_\_name\_\_ == "\_\_main\_\_":: Esta é uma construção padrão em Python que verifica se o script atual está sendo executado diretamente pelo interpretador Python. A variável especial \_\_name\_\_ é definida como "\_\_main\_\_" quando o script é o ponto de entrada principal, mas se o script for importado por outro script, \_\_name\_\_ será definido com o nome do módulo (neste caso, "setup(arquivo final) - não mexer").
12. main(): Se a condição na linha 10 for verdadeira (ou seja, se este script estiver sendo executado diretamente), esta linha chama a função main() que foi definida a partir da linha 4. Isso inicia o fluxo principal do programa, que é tentar fazer o login e, se bem-sucedido, ir para a tela de trabalho.
13. Arquivo Tela1\_login:

Com certeza! Vamos analisar o arquivo tela1\_login.py linha por linha, explicando literalmente o que cada uma faz e enumerando-as:

1. # tela1\_login.py: Esta linha é um comentário. Comentários em Python começam com # e são ignorados pelo interpretador. Eles servem apenas para adicionar notas ou explicações para quem lê o código. Neste caso, indica o nome do arquivo.
2. from db\_manager import conectar\_db: Esta linha importa especificamente a função conectar\_db do módulo (arquivo Python) chamado db\_manager. Isso significa que o código neste arquivo pode usar a função conectar\_db diretamente, sem precisar referenciá-la como db\_manager.conectar\_db.
3. import tela2\_registro: Esta linha importa o módulo completo chamado tela2\_registro. Agora, qualquer coisa definida em tela2\_registro.py pode ser acessada neste arquivo prefixando com tela2\_registro..
4. import os: Esta linha importa o módulo padrão os, que fornece uma maneira de interagir com o sistema operacional. É comumente usado para tarefas como limpar a tela do console, manipular caminhos de arquivo, etc.
5. # --- Importar a lógica da Cifra de Hill e Base64 ---: Esta linha é um comentário que serve como um cabeçalho, indicando que as próximas linhas importarão funcionalidades relacionadas à criptografia (Cifra de Hill) e codificação (Base64).
6. from hill\_cipher\_logic import decrypt as hill\_decrypt: Esta linha importa a função decrypt do módulo hill\_cipher\_logic e a renomeia para hill\_decrypt. Isso torna o nome mais descritivo neste contexto (descriptografia com Hill) e evita possíveis conflitos de nomes se houvesse outra função decrypt importada de outro lugar.
7. import base64: Esta linha importa o módulo padrão base64, que contém funções para trabalhar com a codificação Base64.
8. # --- Fim da importação ---: Este é outro comentário, marcando o fim da seção de importação de módulos relacionados à criptografia/codificação.
9. (linha em branco): Linha em branco para melhorar a legibilidade.
10. def limpar\_tela():: Esta linha define uma função chamada limpar\_tela. O código indentado abaixo dela pertence a esta função.
11. os.system('cls' if os.name == 'nt' else 'clear'): Dentro da função limpar\_tela, esta linha executa um comando no sistema operacional. Ela usa uma expressão condicional ('cls' if os.name == 'nt' else 'clear') para escolher o comando correto: se o sistema operacional for Windows (os.name == 'nt'), o comando é cls; caso contrário (para sistemas como Linux ou macOS), o comando é clear. O resultado é limpar o conteúdo visível do console.
12. (linha em branco): Linha em branco para legibilidade.
13. def buscar\_usuario\_por\_username(username):: Esta linha define uma função chamada buscar\_usuario\_por\_username que aceita um argumento chamado username. Esta função é projetada para encontrar um usuário no banco de dados usando o nome de usuário fornecido.
14. """Busca um usuário no banco de dados pelo username.""": Esta é uma docstring (string de documentação). Ela explica brevemente o que a função buscar\_usuario\_por\_username faz. Docstrings são usadas para documentar o código.
15. conexao = conectar\_db(): Dentro da função buscar\_usuario\_por\_username, esta linha chama a função conectar\_db() (que foi importada na linha 2) e armazena o objeto de conexão retornado na variável conexao.
16. if conexao is None:: Esta linha verifica uma condição: se o valor da variável conexao é None. A função conectar\_db() provavelmente retorna None se não conseguir estabelecer uma conexão com o banco de dados.
17. return None: Se a condição na linha 16 for verdadeira (a conexão falhou), esta linha é executada. Ela interrompe a execução da função buscar\_usuario\_por\_username e retorna o valor None para quem chamou a função.
18. (linha em branco): Linha em branco.
19. try:: Esta linha inicia um bloco try. O código dentro deste bloco será executado, mas se ocorrer um erro (uma exceção), a execução saltará para o bloco except correspondente. Isso é usado para lidar com erros de forma controlada, especialmente ao interagir com recursos externos como bancos de dados.
20. cursor = conexao.cursor(dictionary=True) # Retorna como dicionário: Dentro do bloco try, esta linha cria um objeto cursor a partir da conexão do banco de dados (conexao). O cursor é usado para executar comandos SQL. O argumento dictionary=True instrui o cursor a retornar as linhas do resultado como dicionários, onde as chaves são os nomes das colunas da tabela. O comentário # Retorna como dicionário reafirma isso.
21. query = "SELECT FROM usuarios WHERE username = %s": Esta linha define uma string de texto que contém uma consulta SQL. A consulta SELECT FROM usuarios seleciona todas as colunas () da tabela chamada usuarios. A cláusula WHERE username = %s filtra os resultados para incluir apenas as linhas onde a coluna username corresponde a um valor que será fornecido posteriormente (%s é um placeholder).
22. cursor.execute(query, (username,)): Esta linha executa a consulta SQL definida na linha anterior usando o objeto cursor. A variável query contém o comando SQL, e a tupla (username,) fornece o valor para substituir o placeholder %s. A vírgula após username é crucial para que Python a trate como uma tupla contendo um único elemento.
23. return cursor.fetchone() # Retorna o dicionário do usuário ou None: Esta linha recupera o próximo resultado da execução da consulta. cursor.fetchone() retorna uma única linha de resultado. Como a consulta busca por um nome de usuário específico (que deve ser único), espera-se que retorne no máximo uma linha. Se uma linha for encontrada, ela será retornada como um dicionário (devido a dictionary=True na linha 20). Se nenhuma linha for encontrada, ela retornará None. O comentário # Retorna o dicionário do usuário ou None explica o comportamento.
24. except Exception as err:: Esta linha inicia um bloco except. Ele será executado se *qualquer tipo de erro* (Exception) ocorrer no bloco try acima. O erro capturado é armazenado na variável err.
25. print("Erro ao buscar usuário:", err): Se ocorrer um erro no bloco try, esta linha imprime uma mensagem de erro no console, incluindo a descrição do erro armazenada na variável err.
26. return None: Após imprimir a mensagem de erro, esta linha interrompe a execução da função buscar\_usuario\_por\_username e retorna o valor None, indicando que a busca pelo usuário falhou.
27. finally:: Esta linha inicia um bloco finally. O código dentro deste bloco sempre será executado, independentemente de ter ocorrido um erro no bloco try ou não. É frequentemente usado para garantir que recursos (como conexões de banco de dados) sejam fechados corretamente.
28. if conexao and conexao.is\_connected(): # Verifica se a conexão existe e está aberta: Dentro do bloco finally, esta linha verifica duas condições: primeiro, se a variável conexao existe e não é None (significa que a tentativa de conexão ocorreu), e segundo, se o método is\_connected() do objeto conexao retorna True (significa que a conexão com o banco de dados ainda está aberta). O comentário # Verifica se a conexão existe e está aberta explica a verificação.
29. if 'cursor' in locals() and cursor:: Se a conexão estiver aberta (condição da linha 28 verdadeira), esta linha verifica duas condições adicionais: primeiro, se a string 'cursor' está presente no dicionário de variáveis locais (locals()), o que indica se a variável cursor foi criada no bloco try; segundo, se a variável cursor existe e não é None. Isso evita tentar fechar um cursor que nunca foi criado ou que é None. O comentário # Verifica se o cursor foi criado explica.
30. cursor.close(): Se o cursor existir e não for None (condição da linha 29 verdadeira), esta linha chama o método close() no objeto cursor para fechar o cursor do banco de dados e liberar seus recursos associados.
31. conexao.close(): Se a conexão com o banco de dados estiver aberta (condição da linha 28 verdadeira), esta linha chama o método close() no objeto conexao para fechar a conexão com o banco de dados e liberar seus recursos.
32. (linha em branco): Linha em branco.
33. def executar\_fluxo\_login():: Esta linha define a função principal deste arquivo, chamada executar\_fluxo\_login. Esta função gerencia todo o processo de interação do usuário para login ou registro.
34. # Exibe a tela inicial com as opções de Login ou Novo Cadastro: Um comentário indicando o propósito da próxima seção de código.
35. limpar\_tela(): Dentro de executar\_fluxo\_login, esta linha chama a função limpar\_tela() (definida anteriormente) para limpar o console antes de exibir o menu de login/registro.
36. print("\n" + "=" 70): Esta linha imprime uma string que consiste em uma quebra de linha (\n) seguida de 70 caracteres de igual (=). Isso cria uma linha horizontal de separação visual.
37. print("\t🌿 Bem Vindo ao Sistema de Sustentabilidade Individual 🌿"): Esta linha imprime uma string contendo um título de boas-vindas, com tabulações (\t) no início para tentar centralizar e ícones de folha (🌿).
38. print("=" 70): Esta linha imprime outra linha horizontal com 70 caracteres de igual para formatação.
39. print("\t\t\t[1] Fazer Login "): Esta linha imprime uma string que representa a opção 1 do menu (Fazer Login), com tabulações no início para formatação.
40. print("\t\t\t[2] Novo Cadastro "): Esta linha imprime uma string que representa a opção 2 do menu (Novo Cadastro), com tabulações para formatação.
41. print("=" 70): Mais uma linha horizontal de formatação com 70 caracteres de igual.
42. (linha em branco): Linha em branco.
43. # Seleção da opção: Um comentário indicando o início da seção que lida com a escolha do usuário.
44. while True:: Esta linha inicia um loop infinito. O código dentro deste loop continuará a ser executado repetidamente até que uma instrução break seja encontrada (para sair do loop) ou uma instrução return seja encontrada (para sair da função). É usado aqui para garantir que o usuário escolha uma opção válida (1 ou 2).
45. (linha em branco): Linha em branco.
46. opcao = input("Escolha uma opção (1/2): ").strip(): Dentro do loop, esta linha solicita entrada do usuário. Exibe a mensagem "Escolha uma opção (1/2): ", espera que o usuário digite algo e pressione Enter, lê o que foi digitado, remove quaisquer espaços em branco no início e no final da string usando o método .strip(), e armazena o resultado na variável opcao.
47. (linha em branco): Linha em branco.
48. if opcao == '1':: Esta linha verifica uma condição: se o valor da variável opcao é exatamente a string '1'.
49. limpar\_tela(): Se a condição na linha 48 for verdadeira (o usuário escolheu 1), esta linha chama a função limpar\_tela() para limpar o console.
50. break: Se a condição na linha 48 for verdadeira, esta linha interrompe a execução do loop while True (iniciado na linha 44). Como o usuário escolheu fazer login, não precisa mais exibir o menu de opções.
51. elif opcao == '2':: Se a condição na linha 48 for falsa, esta linha verifica outra condição: se o valor da variável opcao é exatamente a string '2'.
52. limpar\_tela(): Se a condição na linha 51 for verdadeira (o usuário escolheu 2), esta linha chama a função limpar\_tela() para limpar o console antes de ir para a tela de registro.
53. # Chama a função main de registro e verifica se foi bem-sucedido: Um comentário explicando a próxima linha.
54. registro\_ok = tela2\_registro.main(): Se a condição na linha 51 for verdadeira, esta linha chama a função main() que está dentro do módulo tela2\_registro (importado na linha 3). Esta chamada inicia o processo de registro de novo usuário. O valor retornado pela função tela2\_registro.main() (que provavelmente indica se o registro foi bem-sucedido ou cancelado) é armazenado na variável registro\_ok.
55. limpar\_tela(): Após a função tela2\_registro.main() retornar, esta linha chama limpar\_tela() novamente para limpar o console.
56. # Se o registro foi cancelado ou falhou, volta ao menu inicial: Um comentário explicando a próxima condição.
57. if not registro\_ok:: Se a condição na linha 51 foi verdadeira (o usuário escolheu 2) e a função tela2\_registro.main() retornou um valor "falso" (como False), esta linha verifica essa condição. not registro\_ok é verdadeiro se registro\_ok for False ou qualquer outro valor "falso".
58. print("\nVoltando ao menu principal..."): Se a condição na linha 57 for verdadeira (o registro foi cancelado ou falhou), esta linha imprime uma mensagem indicando que o sistema está retornando ao menu principal.
59. return executar\_fluxo\_login(): Se a condição na linha 57 for verdadeira, esta linha interrompe a execução da função executar\_fluxo\_login atual e chama a si mesma novamente (executar\_fluxo\_login()). Isso cria uma chamada recursiva que efetivamente reinicia o fluxo de login/registro desde o início. O valor retornado por essa nova chamada será o valor final retornado por esta execução da função. O comentário # Reinicia o fluxo de login/registro explica o propósito.
60. # Se o registro foi OK, continua para a tela de login: Um comentário explicando o que acontece se o registro foi bem-sucedido.
61. break: Se a condição na linha 51 for verdadeira (o usuário escolheu 2) e a condição na linha 57 for falsa (o registro foi bem-sucedido), esta linha é executada. Ela interrompe a execução do loop while True (iniciado na linha 44). Como o usuário se registrou com sucesso, ele prosseguirá para a parte de autenticação de login. O comentário # Sai do loop de opção e vai para a autenticação explica o que acontece.
62. else:: Se nenhuma das condições anteriores (opcao == '1' ou opcao == '2') for verdadeira, este bloco else é executado. Isso acontece se o usuário digitou algo diferente de '1' ou '2'.
63. print("\n⚠ Opção inválida!"): Dentro do bloco else, esta linha imprime uma mensagem indicando que a opção digitada é inválida, precedida por uma quebra de linha e um ícone de aviso.
64. # Não limpa a tela aqui para o usuário ver a mensagem: Um comentário explicando por que a função limpar\_tela() não é chamada aqui. É para que o usuário veja a mensagem de erro.
65. # limpar\_tela() # Removido: Um comentário indicando que uma linha que chamaria limpar\_tela() foi removida intencionalmente neste ponto.
66. (linha em branco): Linha em branco.
67. # Processo de autenticação do usuário: Um comentário indicando o início da seção do código que lida com a autenticação (verificação de nome de usuário e senha).
68. UserTentativas = 3: Esta linha inicializa uma variável chamada UserTentativas com o valor 3. Isso define o número máximo de vezes que o usuário poderá tentar digitar um nome de usuário válido.
69. usuario\_correto = None # Armazenará o dicionário do usuário encontrado: Esta linha inicializa uma variável chamada usuario\_correto com o valor None. Esta variável será usada posteriormente para armazenar os dados do usuário (como um dicionário) se ele for encontrado no banco de dados. O comentário # Armazenará o dicionário do usuário encontrado explica seu propósito.
70. (linha em branco): Linha em branco.
71. while UserTentativas > 0:: Esta linha inicia um loop while. O código dentro deste loop será executado enquanto o valor da variável UserTentativas for maior que 0. Este loop gerencia as tentativas de entrada do nome de usuário.
72. print("=" 70): Dentro do loop, esta linha imprime uma linha horizontal com 70 sinais de igual para formatação.
73. print("🔒 Autenticação de Usuário 🔒".center(70)): Esta linha imprime uma string "🔒 Autenticação de Usuário 🔒" centralizada em um espaço de 70 caracteres.
74. print("=" 70): Mais uma linha horizontal de formatação.
75. username = input("Digite o nome de usuário: ").strip(): Esta linha solicita a entrada do usuário para o nome de usuário, lê, remove espaços e armazena em username.
76. (linha em branco): Linha em branco.
77. usuario = buscar\_usuario\_por\_username(username) # Busca o usuário no DB: Esta linha chama a função buscar\_usuario\_por\_username() (definida anteriormente) passando o username digitado pelo usuário como argumento. O resultado da busca (o dicionário do usuário ou None) é armazenado na variável usuario. O comentário # Busca o usuário no DB explica a ação.
78. if usuario:: Esta linha verifica uma condição: se o valor da variável usuario é "verdadeiro" (ou seja, se a função buscar\_usuario\_por\_username encontrou um usuário e retornou seu dicionário, não None).
79. usuario\_correto = usuario # Guarda os dados do usuário encontrado: Se a condição na linha 78 for verdadeira (usuário encontrado), esta linha copia o dicionário do usuário da variável usuario para a variável usuario\_correto. O comentário # Guarda os dados do usuário encontrado explica.
80. break # Usuário encontrado, sai do loop de username: Se a condição na linha 78 for verdadeira, esta linha interrompe a execução do loop while UserTentativas > 0 (iniciado na linha 71). Como o nome de usuário foi encontrado, não há necessidade de mais tentativas de username. O comentário explica a ação.
81. else:: Se a condição na linha 78 for falsa (o usuário não foi encontrado no banco de dados com o nome de usuário digitado).
82. UserTentativas -= 1: Dentro do bloco else, esta linha decrementa o valor da variável UserTentativas em 1.
83. limpar\_tela(): Esta linha chama limpar\_tela() para limpar o console antes de exibir a mensagem de erro.
84. print("\n" + "=" 70): Imprime uma linha horizontal de separação.
85. print("❌ Usuário não cadastrado!".center(70)): Se o usuário não foi encontrado, esta linha imprime uma mensagem indicando que o usuário não está cadastrado, centralizada e com um ícone de erro.
86. print(f"Tentativas restantes: {UserTentativas}".center(70)): Esta linha imprime uma string que mostra o número de tentativas restantes para digitar o nome de usuário, utilizando uma f-string para incluir o valor da variável UserTentativas, e centraliza a mensagem.
87. if UserTentativas == 0: # Verifica aqui se as tentativas acabaram: Dentro do bloco else (usuário não encontrado), esta linha verifica uma condição: se o valor de UserTentativas se tornou 0 após o decremento. O comentário explica o propósito da verificação neste ponto.
88. limpar\_tela(): Se a condição na linha 87 for verdadeira (acabaram as tentativas de nome de usuário), esta linha chama limpar\_tela().
89. print("\n" + "=" 70): Imprime uma linha horizontal de separação.
90. print("🚫 Acesso bloqueado!".center(70)): Se as tentativas de nome de usuário acabaram, esta linha imprime uma mensagem indicando que o acesso foi bloqueado, centralizada e com um ícone de bloqueio.
91. print("Usuário não encontrado após múltiplas tentativas.".center(70)): Imprime uma explicação para o bloqueio, centralizada.
92. print("=" 70 + "\n"): Imprime outra linha horizontal de separação seguida de uma quebra de linha.
93. return None # Termina a função se esgotou tentativas do username: Se as tentativas de nome de usuário acabaram, esta linha interrompe a execução da função executar\_fluxo\_login e retorna o valor None. Isso indica que o processo de login falhou porque o usuário não foi encontrado após as tentativas permitidas. O comentário explica o que acontece.
94. (linha em branco): Linha em branco.
95. # Se chegou aqui, o usuário foi encontrado (usuario\_correto não é None): Um comentário explicando que o código só chega a este ponto se o loop de autenticação de usuário (linhas 71-93) foi quebrado porque um usuário foi encontrado (a condição da linha 78 se tornou verdadeira), o que significa que usuario\_correto contém o dicionário do usuário.
96. # Autenticação de senha: Um comentário indicando o início da seção do código que lida com a autenticação da senha.
97. tentativas\_senha = 3: Esta linha inicializa uma variável chamada tentativas\_senha com o valor 3. Isso define o número máximo de vezes que o usuário poderá tentar digitar a senha correta.
98. # Pega a senha cifrada (em Base64) do banco de dados a partir do dicionário 'usuario\_correto': Um comentário explicando a próxima linha.
99. senha\_cifrada\_b64\_do\_db = usuario\_correto["senha"] # Certifique-se que a chave do dicionário é "senha": Esta linha acessa o valor associado à chave "senha" dentro do dicionário armazenado na variável usuario\_correto (que são os dados do usuário buscado no banco de dados). O valor obtido (que se espera que seja a senha do usuário armazenada no banco de dados em formato Base64) é armazenado na variável senha\_cifrada\_b64\_do\_db. O comentário # Certifique-se que a chave do dicionário é "senha" é um lembrete sobre a estrutura dos dados do usuário.
100. (linha em branco): Linha em branco.
101. while tentativas\_senha > 0:: Esta linha inicia um loop while. O código dentro deste loop será executado enquanto o valor da variável tentativas\_senha for maior que 0. Este loop gerencia as tentativas de entrada da senha.
102. print("=" 70): Dentro do loop de senha, esta linha imprime uma linha horizontal para formatação.
103. print("🔑 Autenticação de Senha 🔑".center(70)): Esta linha imprime uma string "🔑 Autenticação de Senha 🔑" centralizada em 70 caracteres.
104. print("=" 70): Mais uma linha horizontal de formatação.
105. senha\_digitada = input("Digite sua Senha: ").strip() # Senha que o usuário digita agora: Esta linha solicita a entrada do usuário para a senha, lê, remove espaços e armazena em senha\_digitada. O comentário # Senha que o usuário digita agora esclarece o conteúdo da variável.
106. (linha em branco): Linha em branco.
107. # --- Descriptografar a senha do DB e Comparar ---: Um comentário indicando o início da seção do código que lida com a descriptografia da senha armazenada e a comparação com a senha digitada.
108. try:: Esta linha inicia um bloco try para capturar possíveis erros durante o processo de decodificação Base64 ou descriptografia Hill Cipher.
109. print("\n🔄 Verificando credenciais...") # Mensagem opcional: Dentro do bloco try, esta linha imprime uma mensagem opcional indicando que as credenciais (usuário e senha) estão sendo verificadas, precedida por uma quebra de linha e um ícone de atualização/verificação. O comentário indica que a mensagem é opcional (para fins de depuração ou feedback ao usuário).
110. # 1. Decodificar Base64 (obtendo os bytes da senha cifrada original): Um comentário explicando o primeiro passo do processo de verificação da senha.
111. # Usa 'latin-1' para reverter o processo do registro: Um comentário explicando a codificação ('latin-1') que se espera que tenha sido usada durante o processo de registro antes de codificar em Base64, pois essa codificação precisa ser usada na ordem inversa para decodificar corretamente.
112. ciphertext\_bytes = base64.b64decode(senha\_cifrada\_b64\_do\_db): Esta linha chama a função base64.b64decode() para decodificar a string da senha armazenada no banco de dados (senha\_cifrada\_b64\_do\_db, que está em formato Base64) de volta para sua representação original em bytes. Os bytes decodificados são armazenados na variável ciphertext\_bytes.
113. ciphertext\_recovered = ciphertext\_bytes.decode('latin-1'): Esta linha chama o método .decode('latin-1') nos bytes decodificados (ciphertext\_bytes) para converter esses bytes de volta para uma string, usando a codificação 'latin-1'. A string resultante (que deve ser a senha cifrada original antes da codificação Base64) é armazenada na variável ciphertext\_recovered.
114. (linha em branco): Linha em branco.
115. # 2. Descriptografar com Hill Cipher: Um comentário explicando o segundo passo: usar a Cifra de Hill para descriptografar a senha cifrada.
116. senha\_decifrada\_do\_db = hill\_decrypt(ciphertext\_recovered): Esta linha chama a função hill\_decrypt() (que foi importada e renomeada na linha 6) passando a string da senha cifrada recuperada (ciphertext\_recovered) como argumento. A função hill\_decrypt realiza a descriptografia usando a lógica da Cifra de Hill. O resultado (a senha original em texto plano) é armazenado na variável senha\_decifrada\_do\_db.
117. print("✅ Credenciais processadas.") # Mensagem opcional: Esta linha imprime uma mensagem opcional indicando que as credenciais foram processadas (descodificadas/descriptografadas), com um ícone de checkmark e centralização.
118. (linha em branco): Linha em branco.
119. # 3. Comparar a senha decifrada do DB com a senha digitada AGORA: Um comentário explicando o terceiro e último passo da verificação: comparar a senha em texto plano obtida do banco de dados com a senha que o usuário acabou de digitar.
120. if senha\_digitada == senha\_decifrada\_do\_db:: Esta linha verifica uma condição: se a string armazenada na variável senha\_digitada (a senha que o usuário digitou) é exatamente igual à string armazenada na variável senha\_decifrada\_do\_db (a senha em texto plano obtida do banco de dados após decodificação e descriptografia).
121. limpar\_tela(): Se a condição na linha 120 for verdadeira (as senhas coincidem, login bem-sucedido), esta linha chama limpar\_tela().
122. print("\n" + "=" 70): Imprime uma linha horizontal de separação.
123. print("✅ Login bem-sucedido!"): Se o login for bem-sucedido, esta linha imprime uma mensagem de sucesso.
124. print(f"Bem-vindo, {usuario\_correto['username']}!".center(70)) # Exibe o nome de usuário: Esta linha imprime uma mensagem de boas-vindas personalizada que inclui o nome de usuário (acessado do dicionário usuario\_correto). Usa uma f-string para formatar a mensagem e centraliza o resultado. O comentário # Exibe o nome de usuário explica.
125. print("=" 70 + "\n"): Imprime outra linha horizontal de separação seguida de uma quebra de linha.
126. return usuario\_correto["username"] # Login OK, retorna o username: Se o login for bem-sucedido, esta linha interrompe a execução da função executar\_fluxo\_login e retorna o nome de usuário (acessado do dicionário usuario\_correto). Retornar o nome de usuário indica que o login foi bem-sucedido e fornece uma informação sobre o usuário logado para o código que chamou esta função. O comentário # Login OK, retorna o username explica.
127. else:: Se a condição na linha 120 for falsa (as senhas não coincidem, senha incorreta).
128. # Senha incorreta: Um comentário indicando que a senha digitada está incorreta.
129. tentativas\_senha -= 1: Dentro do bloco else (senha incorreta), esta linha decrementa o valor da variável tentativas\_senha em 1.
130. limpar\_tela(): Esta linha chama limpar\_tela() para limpar o console antes de exibir a mensagem de erro da senha.
131. print("\n" + "=" 70): Imprime uma linha horizontal de separação.
132. print("❌ Senha incorreta!".center(70)): Se a senha estiver incorreta, esta linha imprime uma mensagem indicando que a senha está incorreta, centralizada e com um ícone de erro.
133. print(f"Tentativas restantes: {tentativas\_senha}".center(70)): Esta linha imprime uma string que mostra o número de tentativas restantes para digitar a senha, utilizando uma f-string e centralizando a mensagem.
134. (linha em branco): Linha em branco.
135. except base64.binascii.Error:: Esta linha inicia um bloco except que será executado se ocorrer um erro específico (base64.binascii.Error) dentro do bloco try (iniciado na linha 108). Este tipo de erro geralmente indica um problema ao decodificar dados Base64, possivelmente porque a string não está no formato Base64 correto.
136. # Erro ao decodificar Base64 (senha no DB pode estar corrompida ou não ser Base64): Um comentário explicando a provável causa do erro capturado pelo bloco except anterior.
137. limpar\_tela(): Se ocorrer o erro de Base64, esta linha chama limpar\_tela().
138. print("\n" + "=" 70): Imprime uma linha horizontal de separação.
139. print("⚠️ Erro interno ao verificar senha (formato inválido no DB).".center(70)): Se ocorrer o erro de Base64, esta linha imprime uma mensagem de aviso indicando um erro interno relacionado ao formato da senha no banco de dados, centralizada e com um ícone de aviso.
140. print("Contacte o administrador.".center(70)): Imprime uma sugestão para o usuário contatar o administrador, centralizada.
141. tentativas\_senha -= 1 # Penaliza a tentativa: Esta linha decrementa o contador de tentativas de senha, mesmo em caso de erro interno na decodificação. O comentário # Penaliza a tentativa explica o motivo.
142. print(f"Tentativas restantes: {tentativas\_senha}".center(70)): Imprime as tentativas restantes após o erro.
143. except Exception as e:: Esta linha inicia outro bloco except que será executado se *qualquer outro tipo de erro* (Exception) ocorrer no bloco try (iniciado na linha 108) que não tenha sido capturado pelo except base64.binascii.Error anterior. O erro capturado é armazenado na variável e.
144. # Outro erro (ex: descriptografia falhou, talvez caractere inválido ou erro na lógica Hill): Um comentário explicando possíveis causas para este erro mais genérico (problemas na descriptografia Hill, caracteres inválidos, etc.).
145. limpar\_tela(): Se ocorrer este erro genérico, esta linha chama limpar\_tela().
146. print("\n" + "=" 70): Imprime uma linha horizontal de separação.
147. print(f"⚠️ Erro CRÍTICO ao verificar senha: {e}".center(70)): Se ocorrer este erro genérico, esta linha imprime uma mensagem de erro CRÍTICO em vermelho, incluindo a descrição do erro (e), centralizada.
148. print("Contacte o administrador.".center(70)): Imprime uma sugestão para o usuário contatar o administrador, centralizada.
149. tentativas\_senha -= 1 # Penaliza a tentativa: Esta linha decrementa o contador de tentativas de senha, mesmo em caso deste erro genérico. O comentário # Penaliza a tentativa explica o motivo.
150. print(f"Tentativas restantes: {tentativas\_senha}".center(70)): Imprime as tentativas restantes após o erro.
151. (linha em branco): Linha em branco.
152. # --- Fim da Verificação ---: Um comentário marcando o fim da seção que verifica a senha.
153. (linha em branco): Linha em branco.
154. # Se esgotou as tentativas de senha: Um comentário indicando a próxima condição.
155. if tentativas\_senha == 0:: Após o loop while tentativas\_senha > 0 terminar (o que só acontece se o break for executado por senha correta, ou se tentativas\_senha chegar a 0), esta linha verifica uma condição: se o valor da variável tentativas\_senha é exatamente 0. Isso só será verdadeiro se o loop de senha terminou porque as tentativas acabaram.
156. limpar\_tela(): Se a condição na linha 155 for verdadeira (acabaram as tentativas de senha), esta linha chama limpar\_tela().
157. print("\n" + "=" 70): Imprime uma linha horizontal de separação.
158. print("🚫 Acesso bloqueado!".center(70)): Se as tentativas de senha acabaram, esta linha imprime uma mensagem indicando que o acesso foi bloqueado, centralizada e com um ícone de bloqueio.
159. print("Senha incorreta múltiplas vezes".center(70)): Imprime uma explicação para o bloqueio, centralizada.
160. print("=" 70 + "\n"): Imprime outra linha horizontal de separação seguida de uma quebra de linha.
161. return None # Termina a função: Se as tentativas de senha acabaram, esta linha interrompe a execução da função executar\_fluxo\_login e retorna o valor None. Isso indica que o processo de login falhou devido a múltiplas tentativas de senha incorretas. O comentário explica o que acontece.
162. (linha em branco): Linha em branco.
163. if \_\_name\_\_ == "\_\_main\_\_":: Esta é a construção padrão em Python que verifica se este script está sendo executado diretamente. Se for o caso, o código dentro deste bloco será executado.
164. resultado\_login = executar\_fluxo\_login(): Se o script for executado diretamente (condição da linha 163 verdadeira), esta linha chama a função executar\_fluxo\_login() (definida a partir da linha 33) para iniciar todo o processo de login/registro. O valor retornado por essa função (o nome de usuário em caso de login bem-sucedido, ou None em caso de falha ou cancelamento) é armazenado na variável resultado\_login.
165. if resultado\_login:: Se o script for executado diretamente, esta linha verifica uma condição: se o valor da variável resultado\_login é "verdadeiro". Isso será verdadeiro se a função executar\_fluxo\_login() retornou um nome de usuário (indicando login bem-sucedido).
166. print(f"Usuário '{resultado\_login}' logado com sucesso."): Se a condição na linha 165 for verdadeira (login bem-sucedido), esta linha imprime uma mensagem confirmando que o usuário (cujo nome está em resultado\_login) foi logado com sucesso, usando uma f-string.
167. # Aqui você chamaria a próxima parte do seu sistema: Um comentário indicando que, em um aplicativo completo, seria neste ponto que o código chamaria a próxima parte do sistema (como a tela principal ou de trabalho) após um login bem-sucedido.
168. else:: Se a condição na linha 165 for falsa (login falhou ou foi cancelado).
169. print("Falha no login ou usuário cancelou."): Se o login falhou ou foi cancelado, esta linha imprime uma mensagem indicando isso.
170. Arquivo Tela2\_registro:
171. # tela2\_registro.py: Esta linha é um comentário. É um texto explicativo que o interpretador Python ignora. Ele serve para identificar o nome do arquivo.
172. from db\_manager import criar\_usuario # Importa a função correta: Esta linha importa uma função específica chamada criar\_usuario de um módulo (arquivo Python) chamado db\_manager. O comentário # Importa a função correta sugere que talvez em algum momento a função errada tenha sido importada e isso foi corrigido.
173. import re: Esta linha importa o módulo padrão re, que fornece suporte para operações com expressões regulares. Expressões regulares são padrões usados para encontrar combinações de caracteres em strings, úteis aqui para validar o formato de dados como e-mail e CPF.
174. import os: Esta linha importa o módulo padrão os, que permite interagir com o sistema operacional. É usado para executar comandos específicos do sistema, como limpar a tela do console.
175. # --- Importar a lógica da Cifra de Hill e Base64 ---: Esta linha é um comentário usado como um cabeçalho para a próxima seção de importações, indicando que se referem a funcionalidades de criptografia e codificação.
176. from hill\_cipher\_logic import encrypt as hill\_encrypt: Esta linha importa a função encrypt do módulo hill\_cipher\_logic e a renomeia para hill\_encrypt. Isso facilita a leitura do código, indicando explicitamente que esta é a função de criptografia da Cifra de Hill, e evita conflitos de nomes se houver outra função encrypt em outro lugar.
177. import base64: Esta linha importa o módulo padrão base64, que fornece funções para codificar e decodificar dados usando o formato Base64. Isso é útil para converter dados binários (como uma senha criptografada que pode conter caracteres não imprimíveis) em um formato de texto seguro para armazenamento em bancos de dados ou transmissão.
178. # --- Fim da importação ---: Este é outro comentário que marca o fim da seção de importação de módulos relacionados à criptografia/codificação.
179. (linha em branco): Uma linha em branco para melhorar a legibilidade do código.
180. def limpar\_tela():: Esta linha define uma função chamada limpar\_tela. O código indentado abaixo dela constitui o corpo dessa função.
181. os.system('cls' if os.name == 'nt' else 'clear'): Dentro da função limpar\_tela, esta linha executa um comando no sistema operacional. A expressão 'cls' if os.name == 'nt' else 'clear' é uma forma concisa de escolher entre o comando cls (usado no Windows, onde os.name é 'nt') e clear (usado em outros sistemas operacionais como Linux ou macOS). O efeito é limpar o conteúdo exibido no terminal ou console.
182. (linha em branco): Outra linha em branco para legibilidade.
183. def validar\_cpf(cpf):: Esta linha define uma função chamada validar\_cpf que aceita um argumento chamado cpf. Esta função tem como objetivo verificar se uma string de CPF fornecida é válida de acordo com as regras matemáticas dos dígitos verificadores.
184. cpf = re.sub(r'[^0-9]', '', cpf): Dentro da função validar\_cpf, esta linha usa uma expressão regular (re.sub) para remover todos os caracteres que NÃO SÃO dígitos ([^0-9]) da string cpf e substitui esses caracteres por uma string vazia (''). O resultado, contendo apenas os números do CPF, é armazenado de volta na variável cpf. Isso garante que a validação seja feita apenas nos dígitos.
185. if len(cpf) != 11 or cpf == cpf[0] 11:: Esta linha verifica duas condições: primeiro, se o comprimento da string cpf (após remover caracteres não numéricos) não é igual a 11. Segundo, se a string cpf consiste em 11 dígitos iguais (por exemplo, "00000000000" ou "11111111111"). CPFs com dígitos repetidos não são considerados válidos pelas regras de validação.
186. return False: Se qualquer uma das condições na linha 15 for verdadeira (o CPF não tem 11 dígitos numéricos ou todos os dígitos são iguais), esta linha interrompe a execução da função validar\_cpf e retorna o valor False, indicando que o CPF é inválido.
187. (linha em branco): Linha em branco para legibilidade.
188. # Cálculo do primeiro dígito: Este é um comentário indicando o início do cálculo do primeiro dígito verificador do CPF.
189. soma = sum(int(cpf[i]) (10 - i) for i in range(9)): Esta linha calcula a soma ponderada dos primeiros 9 dígitos do CPF. Ela itera pelos índices de 0 a 8 (range(9)). Para cada índice i, ela pega o dígito correspondente do CPF (cpf[i]), o converte para um inteiro (int()), e o multiplica por um peso (10 - i). A soma desses produtos é armazenada na variável soma.
190. digito1 = 11 - (soma % 11) if (soma % 11) > 1 else 0: Esta linha calcula o valor do primeiro dígito verificador. Ela pega o resto da divisão da soma por 11 (soma % 11). Se o resto for maior que 1, o digito1 é 11 menos o resto. Caso contrário, o digito1 é 0. O resultado é armazenado na variável digito1.
191. (linha em branco): Linha em branco para legibilidade.
192. # Cálculo do segundo dígito: Este é um comentário indicando o início do cálculo do segundo dígito verificador.
193. soma = sum(int(cpf[i]) (11 - i) for i in range(10)): Esta linha calcula a soma ponderada dos primeiros 10 dígitos do CPF (agora incluindo o primeiro dígito verificador que foi calculado anteriormente). Ela itera pelos índices de 0 a 9 (range(10)). Para cada índice i, ela pega o dígito correspondente (int(cpf[i])) e o multiplica por um peso (11 - i). A soma desses produtos é armazenada de volta na variável soma (sobrescrevendo o valor anterior).
194. digito2 = 11 - (soma % 11) if (soma % 11) > 1 else 0: Esta linha calcula o valor do segundo dígito verificador. Ela pega o resto da divisão da nova soma (dos 10 dígitos) por 11 (soma % 11). Se o resto for maior que 1, o digito2 é 11 menos o resto. Caso contrário, o digito2 é 0. O resultado é armazenado na variável digito2.
195. (linha em branco): Linha em branco para legibilidade.
196. return cpf[-2:] == f"{digito1}{digito2}": Esta linha compara os dois últimos caracteres da string cpf (após a limpeza, contendo apenas os 11 dígitos numéricos originais) usando fatiamento (cpf[-2:]) com uma string criada combinando os valores inteiros de digito1 e digito2 convertidos para string (f"{digito1}{digito2}"). A linha retorna True se os dois últimos dígitos do CPF original corresponderem aos dígitos verificadores calculados, indicando um CPF válido, e retorna False caso contrário.
197. (linha em branco): Linha em branco.
198. def main():: Esta linha define a função principal deste arquivo, chamada main. Esta função contém a lógica para o fluxo de registro de usuário.
199. limpar\_tela(): Dentro da função main, esta linha chama a função limpar\_tela() (definida na linha 10) para limpar o console antes de exibir a tela de registro.
200. print("╔" + "═" 40 + "╗"): Esta linha imprime uma string que forma a parte superior de uma "caixa" ou moldura usando caracteres especiais de linha dupla (╔, ═, ╗) e o operador de repetição ().
201. print("║" + "🌟 REGISTRO DE USUÁRIO 🌟".center(38) + "║"): Esta linha imprime uma string que forma o título " REGISTRO DE USUÁRIO " centralizado em um espaço de 38 caracteres (.center(38)) e envolto em caracteres de linha vertical (║). Ícones de estrela (🌟) são adicionados antes e depois do texto centralizado.
202. print("╚" + "═" 40 + "╝"): Esta linha imprime uma string que forma a parte inferior da caixa ou moldura usando caracteres especiais (╚, ═, ╝).
203. (linha em branco): Linha em branco.
204. try:: Esta linha inicia um bloco try. O código dentro deste bloco será monitorado para a ocorrência de certas exceções (erros). Se a exceção KeyboardInterrupt ocorrer, a execução saltará para o bloco except KeyboardInterrupt no final da função.
205. # Seção de cadastro: Um comentário indicando o início da seção onde os dados do usuário serão solicitados.
206. print("\033[1mDados Pessoais\033[0m".center(50)): Esta linha imprime uma string "Dados Pessoais". Os caracteres \033[1m ativam a formatação em negrito no terminal, e \033[0m a desativam. A string formatada é então centralizada em um espaço de 50 caracteres.
207. print("─" 42): Esta linha imprime uma linha horizontal com 42 caracteres de traço (─) para separação visual.
208. (linha em branco): Linha em branco.
209. # Validação do username: Um comentário indicando o início da seção de validação do nome de usuário.
210. while True:: Esta linha inicia um loop infinito. O código dentro deste loop continuará a ser executado até que uma instrução break seja encontrada (para sair do loop) ou uma instrução return (para sair da função). É usado aqui para forçar o usuário a digitar um nome de usuário que atenda aos critérios de validação.
211. username = input("│ ► Nome de Usuário: ").strip(): Dentro do loop, esta linha solicita a entrada do usuário para o nome de usuário. Exibe a mensagem "│ ► Nome de Usuário: ", lê a entrada, remove espaços no início e fim (.strip()), e armazena na variável username.
212. if len(username) < 3:: Esta linha verifica uma condição: se o comprimento (len()) da string username é menor que 3 caracteres.
213. print("│ ⚠️ \033[31mNome deve ter pelo menos 3 caracteres!\033[0m"): Se a condição na linha 42 for verdadeira (nome de usuário muito curto), esta linha imprime uma mensagem de erro em vermelho (\033[31m ... \033[0m) com um ícone de aviso (⚠️) e o prefixo de formatação (│).
214. print("├" + "─" 38): Imprime uma linha horizontal de formatação (├ seguido de 38 traços) dentro da estrutura de caixa.
215. continue: Se a condição na linha 42 for verdadeira, esta linha pula o resto do código dentro do loop while True e volta para o início do loop (linha 40), solicitando o nome de usuário novamente.
216. break: Se a condição na linha 42 for falsa (o nome de usuário tem 3 ou mais caracteres), esta linha interrompe a execução do loop while True (iniciado na linha 40), pois a validação do nome de usuário foi bem-sucedida.
217. (linha em branco): Linha em branco.
218. # Validação do email: Um comentário indicando o início da seção de validação do e-mail.
219. while True:: Esta linha inicia outro loop infinito para a validação do e-mail, com a mesma lógica do loop anterior (repetir até que um break seja encontrado).
220. email = input("\n│ ► E-mail: ").strip(): Dentro do loop, esta linha solicita a entrada do usuário para o e-mail, lê, remove espaços e armazena em email. Uma quebra de linha (\n) é adicionada antes do prefixo para separação visual.
221. if not re.match(r'^[\w\.-]+@[\w\.-]+\.[a-zA-Z]{2,}$', email):: Esta linha verifica uma condição: se a string email não corresponde (not) ao padrão de expressão regular fornecido (re.match). A expressão regular r'^[\w\.-]+@[\w\.-]+\.[a-zA-Z]{2,}$' é um padrão comum para verificar se uma string tem um formato básico de e-mail (algo@algo.dominio).
222. print("│ ⚠️ \033[31mFormato de e-mail inválido! Ex: user@exemplo.com\033[0m"): Se a condição na linha 51 for verdadeira (formato de e-mail inválido), esta linha imprime uma mensagem de erro formatada em vermelho, com ícone de aviso e um exemplo de formato correto.
223. print("├" + "─" 38): Imprime uma linha horizontal de formatação dentro da caixa.
224. continue: Se a condição na linha 51 for verdadeira, esta linha volta para o início do loop while True (linha 49), solicitando o e-mail novamente.
225. break: Se a condição na linha 51 for falsa (o formato do e-mail é válido), esta linha interrompe a execução do loop while True (linha 49), pois a validação do e-mail foi bem-sucedida.
226. (linha em branco): Linha em branco.
227. # Validação do CPF: Um comentário indicando o início da seção de validação do CPF.
228. while True:: Esta linha inicia outro loop infinito para a validação do CPF.
229. cpf1 = input("\n│ ► CPF (apenas números): ").strip(): Dentro do loop, esta linha solicita a entrada do usuário para o CPF (sugerindo apenas números), lê, remove espaços e armazena em cpf1.
230. cpf = cpf1.replace('.','').replace('-','').replace('/',''): Esta linha cria uma nova string chamada cpf removendo todos os pontos (.), hifens (-) e barras (/) da string cpf1 usando o método .replace(). Isso garante que a validação subsequente seja feita apenas nos dígitos numéricos.
231. if not validar\_cpf(cpf):: Esta linha chama a função validar\_cpf() (definida na linha 13) passando a string cpf (apenas números) como argumento. Ela verifica uma condição: se o valor retornado pela função validar\_cpf() não é True (ou seja, se a função retornou False porque o CPF é inválido).
232. print("│ ⚠️ \033[31mCPF inválido! Digite 11 números válidos.\033[0m"): Se a condição na linha 61 for verdadeira (o CPF é inválido), esta linha imprime uma mensagem de erro formatada em vermelho, com ícone de aviso e uma instrução.
233. print("├" + "─" 38): Imprime uma linha horizontal de formatação dentro da caixa.
234. continue: Se a condição na linha 61 for verdadeira, esta linha volta para o início do loop while True (linha 58), solicitando o CPF novamente.
235. # Formata o CPF para o padrão 000.000.000-00: Um comentário explicando a próxima linha.
236. cpf\_formatado = f"{cpf[:3]}.{cpf[3:6]}.{cpf[6:9]}-{cpf[9:]}": Se a condição na linha 61 for falsa (o CPF é válido), esta linha cria uma nova string chamada cpf\_formatado. Ela usa uma f-string para formatar a string cpf (que contém os 11 dígitos numéricos) no padrão comum de CPF com pontos e hífen, usando fatiamento de strings ([:3], [3:6], etc.).
237. break: Se a condição na linha 61 for falsa (o CPF é válido), esta linha interrompe a execução do loop while True (linha 58), pois a validação e formatação do CPF foram bem-sucedidas.
238. (linha em branco): Linha em branco.
239. # Validação da senha (obter a senha em texto plano): Um comentário indicando o início da seção de validação da senha.
240. while True:: Esta linha inicia outro loop infinito para a validação da senha.
241. # Renomeado para 'senha\_plana' para clareza interna, mas pode manter 'senha': Um comentário explicando que a variável que armazenará a senha digitada será chamada senha\_plana para maior clareza interna, distinguindo-a de uma senha criptografada ou hash.
242. senha\_plana = input("\n│ ► Senha (mínimo 6 caracteres): ").strip(): Dentro do loop, esta linha solicita a entrada do usuário para a senha (sugerindo um mínimo de 6 caracteres), lê, remove espaços e armazena em senha\_plana. Uma quebra de linha (\n) é adicionada antes do prefixo.
243. if len(senha\_plana) < 6:: Esta linha verifica uma condição: se o comprimento da string senha\_plana é menor que 6 caracteres.
244. print("│ ⚠️ \033[31mSenha muito curta! Use mais caracteres.\033[0m"): Se a condição na linha 73 for verdadeira (senha muito curta), esta linha imprime uma mensagem de erro formatada em vermelho com ícone de aviso.
245. print("├" + "─" 38): Imprime uma linha horizontal de formatação dentro da caixa.
246. continue: Se a condição na linha 73 for verdadeira, esta linha volta para o início do loop while True (linha 70), solicitando a senha novamente.
247. break: Se a condição na linha 73 for falsa (a senha tem 6 ou mais caracteres), esta linha interrompe a execução do loop while True (linha 70), pois a validação da senha foi bem-sucedida.
248. (linha em branco): Linha em branco.
249. # Confirmação final: Um comentário indicando o início da seção de confirmação dos dados digitados pelo usuário.
250. limpar\_tela(): Esta linha chama a função limpar\_tela() para limpar o console antes de exibir a tela de confirmação.
251. print("╔" + "═" 38 + "╗"): Imprime a parte superior de uma caixa de confirmação.
252. print("║" + "🔍 CONFIRA SEUS DADOS 🔍".center(36) + "║"): Imprime o título " CONFIRA SEUS DADOS " centralizado em 36 caracteres e envolto em linhas verticais, com ícones de lupa (🔍).
253. print("╟" + "─" 38 + "╢"): Imprime uma linha horizontal de separação dentro da caixa usando caracteres especiais (╟, ─, ╢).
254. print("║" + f" Usuário: \033[34m{username}\033[0m".center(47) + "║"): Esta linha imprime uma string que exibe o rótulo " Usuário: " seguido do nome de usuário digitado (username). O nome de usuário é formatado em azul (\033[34m ... \033[0m). Toda a string resultante (incluindo a formatação de cor) é centralizada em um espaço de 47 caracteres e envolta em linhas verticais.
255. print("║" + f" E-mail: \033[34m{email}\033[0m".center(47) + "║"): Similar à linha anterior, esta linha imprime o rótulo " E-mail: " seguido do e-mail digitado (email), formatado em azul, centralizado e dentro da caixa.
256. print("║" + f" CPF: \033[34m{cpf\_formatado}\033[0m".center(47) + "║"): Similar às linhas anteriores, esta linha imprime o rótulo " CPF: " seguido do CPF formatado (cpf\_formatado), formatado em azul, centralizado e dentro da caixa.
257. # Não mostre a senha na confirmação!: Um comentário IMPORTANTE alertando para não exibir a senha em texto plano na tela de confirmação por motivos de segurança.
258. print("╚" + "═" 38 + "╝"): Imprime a parte inferior da caixa de confirmação.
259. (linha em branco): Linha em branco.
260. confirmacao = input("\n│ ❓ Confirmar cadastro? (S/N): ").upper(): Esta linha solicita a entrada do usuário para confirmar o cadastro. Exibe a mensagem "│ ❓ Confirmar cadastro? (S/N): ", lê a entrada, converte toda a string para maiúsculas usando .upper(), e armazena o resultado na variável confirmacao. Uma quebra de linha (\n) é adicionada antes do prefixo.
261. if confirmacao != 'S':: Esta linha verifica uma condição: se o valor da variável confirmacao não é igual à string 'S'. Isso é verdadeiro se o usuário digitou qualquer coisa diferente de 'S' (indicando que não quer confirmar).
262. print("\n\033[31m✖ Cadastro cancelado!\033[0m"): Se a condição na linha 91 for verdadeira (o usuário não confirmou), esta linha imprime uma mensagem " Cadastro cancelado!" formatada em vermelho (\033[31m ... \033[0m) com um ícone de cruz (✖), precedida por uma quebra de linha.
263. return False: Se a condição na linha 91 for verdadeira, esta linha interrompe a execução da função main e retorna o valor False. Isso indica que o processo de registro foi cancelado pelo usuário.
264. (linha em branco): Linha em branco.
265. # --- Criptografar a senha com Hill Cipher antes de salvar ---: Um comentário indicando o início da seção de criptografia da senha antes de salvá-la no banco de dados.
266. try:: Esta linha inicia um bloco try para monitorar por possíveis erros durante o processo de criptografia.
267. print("\n🔒 Criptografando senha...") # Mensagem opcional: Dentro do bloco try, esta linha imprime uma mensagem opcional indicando que a senha está sendo criptografada, precedida por uma quebra de linha e um ícone de cadeado (🔒).
268. senha\_cifrada\_raw = hill\_encrypt(senha\_plana): Esta linha chama a função hill\_encrypt() (importada e renomeada na linha 6) passando a senha em texto plano digitada pelo usuário (senha\_plana) como argumento. O resultado da criptografia (a senha cifrada bruta) é armazenado na variável senha\_cifrada\_raw.
269. # Codificar em Base64 para armazenamento seguro no DB: Um comentário explicando o próximo passo: codificar a senha cifrada em Base64 para que possa ser armazenada de forma segura no banco de dados como texto.
270. # Usa 'latin-1' para garantir que todos os bytes 0-255 sejam preservados: Um comentário explicando a codificação ('latin-1') usada antes de codificar em Base64. 'latin-1' (ISO-8859-1) mapeia todos os 256 valores possíveis de um byte para caracteres, garantindo que quaisquer bytes resultantes da criptografia Hill Cipher (que podem não ser caracteres ASCII imprimíveis) sejam representados como caracteres antes de serem codificados em Base64.
271. senha\_cifrada\_b64 = base64.b64encode(senha\_cifrada\_raw.encode('latin-1')).decode('utf-8'): Esta linha realiza o processo de codificação em Base64. Primeiro, senha\_cifrada\_raw.encode('latin-1') converte a string senha\_cifrada\_raw (a senha cifrada) em bytes usando a codificação 'latin-1'. Em seguida, base64.b64encode(...) codifica esses bytes em Base64. O resultado é novamente um objeto de bytes. Finalmente, .decode('utf-8') decodifica esses bytes resultantes do Base64 para uma string usando a codificação 'utf-8'. O resultado final (a string da senha criptografada e codificada em Base64) é armazenado na variável senha\_cifrada\_b64.
272. print("🔑 Senha processada.") # Mensagem opcional: Esta linha imprime uma mensagem opcional indicando que a senha foi processada (criptografada e codificada), com um ícone de chave (🔑).
273. except Exception as e:: Esta linha inicia um bloco except que será executado se qualquer tipo de erro (Exception) ocorrer no bloco try (iniciado na linha 96) durante o processo de criptografia. O erro capturado é armazenado na variável e.
274. print(f"\n\033[31m✖ Erro CRÍTICO ao criptografar a senha: {e}\033[0m"): Se ocorrer um erro na criptografia, esta linha imprime uma mensagem de erro CRÍTICO formatada em vermelho, incluindo a descrição do erro armazenada em e, precedida por uma quebra de linha e um ícone de cruz.
275. print("\033[31m✖ Cadastro não pode ser concluído.\033[0m"): Se ocorrer um erro na criptografia, esta linha imprime uma mensagem adicional em vermelho indicando que o cadastro não pôde ser concluído.
276. # Poderia ser um caractere inválido ou erro na lógica Hill: Um comentário sugerindo possíveis causas para o erro de criptografia.
277. return False: Se ocorrer um erro CRÍTICO durante a criptografia, esta linha interrompe a execução da função main e retorna o valor False, indicando que o registro falhou neste ponto.
278. # --- Fim da Criptografia ---: Um comentário marcando o fim da seção de criptografia.
279. (linha em branco): Linha em branco.
280. # Insere os dados do usuário no banco de dados.: Um comentário indicando que a próxima ação é inserir os dados no banco de dados.
281. # AGORA envia a senha criptografada em Base64: Um comentário enfatizando que a senha que será enviada para o banco de dados é a versão criptografada e codificada em Base64.
282. # IMPORTANTE: Certifique-se que o campo 'senha' no seu DB (tabela 'usuarios'): Um comentário IMPORTANTE fornecendo uma instrução crucial sobre a configuração do banco de dados: o campo onde a senha será armazenada ('senha' na tabela 'usuarios').
283. # pode armazenar uma string longa (VARCHAR(100+), TEXT, etc).: Continuação do comentário anterior, especificando que o tipo de dado para o campo de senha no banco de dados deve ser capaz de armazenar uma string longa, como VARCHAR com um tamanho adequado (100 ou mais) ou um tipo como TEXT. Isso é necessário porque a senha criptografada e codificada em Base64 pode ser significativamente mais longa que a senha original em texto plano.
284. sucesso = criar\_usuario(username, username, cpf\_formatado, email, senha\_cifrada\_b64): Esta linha chama a função criar\_usuario() (importada na linha 2). Ela passa como argumentos o nome de usuário (duas vezes, possivelmente para diferentes campos no banco de dados como login e nome de exibição), o CPF formatado, o e-mail e a senha criptografada e codificada em Base64. O valor retornado por criar\_usuario() (que deve ser True se a inserção no banco de dados foi bem-sucedida e False caso contrário) é armazenado na variável sucesso.
285. (linha em branco): Linha em branco.
286. if sucesso:: Esta linha verifica uma condição: se o valor da variável sucesso é "verdadeiro" (ou seja, se a função criar\_usuario retornou True, indicando que o usuário foi criado no banco de dados).
287. limpar\_tela(): Se a condição na linha 116 for verdadeira (cadastro bem-sucedido), esta linha chama limpar\_tela().
288. print("\n\033[32m╔══════════════════════════════════════╗"): Se o cadastro for bem-sucedido, esta linha imprime a parte superior de uma caixa de sucesso, formatada em verde (\033[32m ... \033[0m) e usando caracteres especiais.
289. print("║ ✅ CADASTRO REALIZADO COM SUCESSO! ║"): Imprime a mensagem " CADASTRO REALIZADO COM SUCESSO! " centralizada, em verde e com um ícone de checkmark (✅), dentro da caixa.
290. print("╚══════════════════════════════════════╝\033[0m"): Imprime a parte inferior da caixa de sucesso e redefine a cor do terminal para o padrão.
291. input(print("Pressione ENTER para seguir...")): Esta linha imprime uma mensagem "Pressione ENTER para seguir..." e espera que o usuário pressione a tecla Enter. O uso de input(print(...)) é um pouco redundante; input("Pressione ENTER para seguir...") sozinho já funcionaria para exibir a mensagem e esperar.
292. return True: Se a condição na linha 116 for verdadeira (cadastro bem-sucedido), esta linha interrompe a execução da função main e retorna o valor True, indicando que o processo de registro foi concluído com sucesso.
293. else:: Se a condição na linha 116 for falsa (a função criar\_usuario retornou False, indicando que ocorreu um erro ao inserir no banco de dados).
294. limpar\_tela(): Dentro do bloco else, esta linha chama limpar\_tela().
295. print("\n\033[31m✖ Ocorreu um erro ao cadastrar o usuário!\033[0m"): Se o cadastro falhou, esta linha imprime uma mensagem de erro em vermelho (\033[31m ... \033[0m) com um ícone de cruz, precedida por uma quebra de linha.
296. print("\033[31m✖ Seu Email, CPF ou Username já podem estar cadastrados! \033[0m"): Imprime uma mensagem adicional em vermelho, sugerindo possíveis causas para a falha (como duplicidade de e-mail, CPF ou nome de usuário no banco de dados).
297. print("\033[31m✖ Tente novamente com outros dados.\033[0m"): Sugere que o usuário tente o registro novamente, possivelmente com dados diferentes.
298. input(print("Pressione ENTER para seguir...")): Imprime a mensagem "Pressione ENTER para seguir..." e espera por Enter.
299. return False: Se o cadastro falhou, esta linha interrompe a execução da função main e retorna o valor False, indicando que o processo de registro não foi bem-sucedido.
300. (linha em branco): Linha em branco.
301. except KeyboardInterrupt:: Esta linha inicia um bloco except que será executado se a exceção KeyboardInterrupt ocorrer dentro do bloco try (iniciado na linha 34). KeyboardInterrupt é geralmente levantada quando o usuário pressiona Ctrl+C no terminal, indicando uma tentativa de interromper a execução do programa.
302. print("\n\033[33m⚠️ Operação interrompida pelo usuário!\033[0m"): Se a exceção KeyboardInterrupt ocorrer, esta linha imprime uma mensagem de aviso em amarelo (\033[33m ... \033[0m) com um ícone de aviso, indicando que a operação foi interrompida pelo usuário.
303. return False: Se a exceção KeyboardInterrupt ocorrer, esta linha interrompe a execução da função main e retorna o valor False, indicando que o processo de registro foi cancelado/interrompido pelo usuário.
304. (linha em branco): Linha em branco.
305. if \_\_name\_\_ == "\_\_main\_\_":: Esta é a construção padrão em Python que verifica se este script está sendo executado diretamente. Se for o caso, o código dentro deste bloco será executado.
306. main(): Se o script for executado diretamente (condição da linha 135 verdadeira), esta linha chama a função main() (definida a partir da linha 28) para iniciar o fluxo de registro de usuário.
307. Arquivo Tela3\_tela\_de\_trabalho: