Ana Clara Morais de Carvalho

https://github.com/anaclaramcarvalho/LIPAI

AULA 01

```
arquivo =
open("C:/Users/anacl/OneDrive/Documents/UFU/LIPAI/Semana03/src/Depuração/Manipulação_arquivos/teste.txt", "r")

print(arquivo.readable())

print(arquivo.read())

print(arquivo.readline())

lista = arquivo.readlines()

print(lista)

print(lista[3])


 arquivo =
open("C:/Users/anacl/OneDrive/Documents/UFU/LIPAI/Semana03/src/Depuração/Manipulação_arquivos/teste.txt", "a")

 arquivo.write("C\n")

 arquivo.write("Terraform\n")


arquivo =
open(""C:/Users/anacl/OneDrive/Documents/UFU/LIPAI/Semana03/src/Depuração/Manipulação_arquivos/teste2.txt", "w")


 arquivo =
open("C:/Users/anacl/OneDrive/Documents/UFU/LIPAI/Semana03/src/Depuração/Manipulação_arquivos/teste3.txt", "x")

 arquivo.write("python\n")

 arquivo.write("Terraform\n")


arquivo.close()
```

```
import os

 if
os.path.exists("C:/Users/anacl/OneDrive/Documents/UFU/LIPAI/Semana03/src/Depuração/Manipulação_arquivos/teste3.txt"):

    "se existir ele remove"

    os.remove("C:/Users/anacl/OneDrive/Documents/UFU/LIPAI/Semana03/src/Depuração/Manipulação_arquivos/teste3.txt")

else:

print("Arquivo não existe")


os.rmdir("C:/Users/anacl/OneDrive/Documents/UFU/LIPAI/Semana03/src/Depuração/Manipulação_arquivos/novo_arquivo")
```

AULA 02

```
arq = open("teste.txt","w")

print(arq)


string = "ola Ana!"

x = ["ola", "Ana","!"]

#arq.write(x)#uma string

arq.writelines(x)#interavel de string


for nome in x:

   arq.write(nome + '\n')


arq.close()


with open('arquivo.txt', 'w') as arq:

   arq.write('teste')
```

```
    print('fim')


with open('arquivo.txt', 'a') as arq:
    arq.write('\nCaio')


with open('arquivo.txt', 'r') as arq:
    x = arq.read()#let tudo e trazer em uma string
    arq.readline()


with open('arquivo.txt', 'rB') as arq:
    x = arq.read()
    print(x)
```

EXERCÍCIO 01

```
def carregar_dados_alunos(nome_arquivo):
    lista_alunos = []


    try:
        with open(nome_arquivo, 'r', encoding='utf-8') as arquivo:
            for linha in arquivo:
                linha_limpa = linha.strip()
                if not linha_limpa:
                    continue
                campos = linha_limpa.split(',')
                aluno = {
                    'prontuario': campos[0],
                    'nome': campos[1],
                    'email': campos[2]
```

```
            }
            lista_alunos.append(aluno)

    except FileNotFoundError:
        print(f"Erro: O arquivo '{nome_arquivo}' não foi encontrado.")
        return ()
    except IndexError:
        print("Erro: O arquivo parece estar com o formato incorreto (faltando vírgulas).")
        return ()
    return tuple(lista_alunos)

resultado = carregar_dados_alunos("C:/Users/anacl/OneDrive/Documents/UFU/LIPAI/Semana03/src/Depuração/Exercicios/alunos.txt")

for aluno in resultado:
    print(aluno)

"""with open(...): Esta é a forma de abrir arquivos em Python. Ela garante que o arquivo seja fechado automaticamente, mesmo se ocorrer um erro durante o processamento.

strip(): Essencial para remover o caractere invisível de "quebra de linha" (\n) que fica no final de cada frase dentro do arquivo de texto.

split(','): Como o formato do arquivo é CSV (valores separados por vírgula), usamos este método para transformar a string única da linha em uma lista de strings individuais.
```

tuple(lista_alunos):é mais eficiente adicionar itens a uma lista (que é mutável) e, ao final, convertê-la para tupla (que é imutável), atendendo ao requisito da função."""

EXERCÍCIO 02

```
def carregar_dados_projetos(nome_arquivo):
  lista_projetos = []


  try:
    with open(nome_arquivo, 'r', encoding='utf-8') as arquivo:
      for linha in arquivo:
        linha_limpa = linha.strip()
        if not linha_limpa:
          continue
        campos = linha_limpa.split(',')
        projeto = {
          'codigo': int(campos[0]),
          'titulo': campos[1],
          'responsavel': campos[2]
        }


        lista_projetos.append(projeto)


  except FileNotFoundError:
    print(f"Erro: O arquivo '{nome_arquivo}' não foi encontrado.")
    return ()
  except (ValueError, IndexError):
    print("Erro: Falha ao processar os dados. Verifique o formato do arquivo.")
    return ()
```

```
    return tuple(lista_projetos)


caminho = "C:/Users/anacl/OneDrive/Documents/UFU/LIPAI/Semana03/src/Depuração/Exercicios/projetos.txt"
resultado = carregar_dados_projetos(caminho)


for projeto in resultado:
    print(projeto)
```

EXERCÍCIO 03

```
def linha_para_dict(linha, chaves):
    valores = linha.strip().split(',')
    dicionario = {}
    for i in range(len(chaves)):
        chave = chaves[i]
        valor = valores[i]
        dicionario[chave] = valor


    return dicionario


"""EXEMPLOS"""
linha1 = "SP000001,Maria da Silva,maria@email.com"
chaves1 = ['prontuario', 'nome', 'email']
print("Saída Exemplo 1:", linha_para_dict(linha1, chaves1))


linha2 = "banana,3"
chaves2 = ['item', 'quantidade']
```

print("Saída Exemplo 2:", linha_para_dict(linha2, chaves2))

Reflexão Arquivo

01 - Qual a vantagem de transformar cada linha do arquivo em dicionários em vez de trabalhar apenas com strings?

Trabalhar com dicionários oferece três vantagens principais:

Semântica e Clareza:Em vez de acessar um dado pela posição (ex: 'aluno[1]'), acessamos por uma chave nomeada (ex: 'aluno['nome']'). Isso torna o código muito mais legível e fácil de manter.

Facilidade de Manipulação:Com dicionários, podemos facilmente passar os dados de um aluno para outras funções, converter para JSON ou salvar em bancos de dados.

Isolamento da Lógica de Parse:Uma vez que a string foi convertida em dicionário, o restante do programa não precisa saber se o separador original era uma vírgula, um ponto-e-vírgula ou um espaço.

02 - Em que situações pode ser útil retornar uma tupla de registros (como nos exercícios ex01 e ex02) em vez de apenas uma lista de linhas?

uso de tuplas e registros processados é ideal quando:

Imutabilidade e Segurança:Uma tupla é imutável. Ao carregar dados de um arquivo, geralmente queremos garantir que essa "foto" inicial dos dados não seja alterada acidentalmente por outras partes do código durante a execução.

Prontidão para Uso:Retornar uma lista de strings puras obrigaria qualquer outra parte do sistema a "quebrar" a string (usar '.split()') toda vez que precisasse de um dado. Ao retornar registros (dicionários) já processados, o dado está pronto para ser consumido (ex: exibir em uma interface ou filtrar por ID).

03 - O que você achou mais desafiador: ler o arquivo ou transformar as linhas em estruturas de dados (dicionários)?

Geralmente, o maior desafio é a transformação e validação das linhas. Enquanto a leitura do arquivo é um processo quase sempre padrão ('open', 'read', 'close'), a transformação exige:

Lidar com caracteres invisíveis (como o '\n' no final das linhas).

Garantir que cada coluna vá para a chave correta.

Tratar exceções, como linhas em branco ou campos faltando, que podem quebrar o programa se não forem previstos.

A função genérica 'linha_para_dict' foi um passo importante para simplificar esse desafio, centralizando a complexidade em um único lugar.