Aula 15: Arquivos, parâmetros do programa

Luís Felipe

UFF

24 de Novembro de 2025

Luís Felipe

24/11/25

Programas interativos e arquivos

- Programas interativos: leem dados do teclado e exibem resultados na tela.
- Úteis para poucos dados ou quando há interação humana.
- Para grandes quantidades de dados, usam-se arquivos para armazenar entrada e saída.
- Arquivos de entrada e saída permitem processar dados sem interação contínua do usuário.

Luís Felipe

24/11/25

Arquivos texto

- Dois tipos básicos: texto e binário.
- Arquivo texto: sequência de caracteres organizada em linhas, com um nome.
- Pode ser criado e editado por editores de texto.
- Armazenado como sequência de caracteres, incluindo:
  - `\n` $\rightarrow$ fim de linha
  - `\0` $\rightarrow$ fim do arquivo

Luís Felipe

24/11/25

Abrindo arquivos em Python

```
# Abrindo arquivo
dados = open("teste.txt", "r")      # leitura
dados = open("teste.txt", "w")      # escrita (sobrescreve)
dados = open("teste.txt", "a")      # escrita no final
```

- "r": leitura (padrão, se omitido = "r").
- "w": escrita, apaga conteúdo anterior.
- "a": escrita no final.
- "r+": leitura e escrita (não visto aqui).

Luís Felipe

24/11/25

Modos de abertura de arquivos em Python

```
1 dados = open("teste.txt", "r")
```

- "r": leitura
  - Se existir: abre o arquivo e posiciona a cabeça de leitura no início.
  - Se não existir: gera `FileNotFoundError`.

```
1 dados = open("teste.txt", "w")
```

- "w": escrita (sobrescreve)
  - Se existir: apaga o conteúdo anterior e posiciona a cabeça de escrita no início.
  - Se não existir: cria o arquivo e posiciona a cabeça de escrita no início.

```
1 dados = open("teste.txt", "a")
```

- "a": escrita no final (append)
  - Se existir: abre para escrita e posiciona a cabeça no final do arquivo.
  - Se não existir: cria o arquivo e posiciona a cabeça no final (início do arquivo).

Luís Felipe

24/11/25

Fechando arquivos

Obs.: Sempre fechar um arquivo para garantir que os dados sejam gravados:

```
1 dados = open("exemplo.txt", "w")
2 # ... operações de escrita ...
3 dados.close()
```

Após `close()`, o arquivo não pode mais ser usado sem reabertura.

Luís Felipe

24/11/25

Leitura com `readline()`

A operação readline(), aplicada sobre um arquivo texto aberto, retorna uma linha completa do arquivo, incluindo o fim de linha: \n.

A cabeça de leitura avança para a próxima linha. Uma string vazia é retornada quando o fim de arquivo é encontrado.

`readline()` lê uma linha completa (incluindo \n).

```
1 dados = open("exemplo.txt", "r")
2 linha = dados.readline()
3 print(linha, end="")
4 dados.close()
```

Retorna "" (string vazia) ao chegar no fim do arquivo.

Luís Felipe

24/11/25

Lendo arquivo linha a linha

O programa abaixo pede ao usuário que escolha um nome de arquivo, existente em seu diretório, e exibe seu conteúdo na tela.

```
1  nomeArquivo = input("Digite o nome do arquivo: ")
2  dados = open(nomeArquivo, "r")
3  linha = dados.readline()
4  while linha != "":
5      print(linha, end="")
6      linha = dados.readline()
7  dados.close()
```

Luís Felipe

24/11/25

Lendo todas as linhas de uma vez

`readlines()` carrega todas as linhas para uma lista (funciona apenas para arquivos pequenos, até algumas dezenas de megabytes).

```
1  nomeArquivo = input("Digite o nome do arquivo: ")
2  dados = open(nomeArquivo, "r")
3  linhas = dados.readlines()
4  for linha in linhas:
5      print(linha, end="")
6  dados.close()
```

Luís Felipe

24/11/25

Escrevendo com `write()`

Para escrever uma sequência de caracteres em um arquivo texto, no modo “w” ou “a”, podemos utilizar o método `write()`.

`write("texto")` escreverá a string desejada (texto) a partir do ponto em que a cabeça de escrita do arquivo estiver posicionada.

Ao final, a cabeça de escrita ficará posicionada após o último caractere da String desejada.

```
1 dados = open("teste.txt", "w")
2 dados.write("qualquer dado pode ser escrito.")
3 dados.close()
```

Para escrever uma linha, incluir \n:

```
1 dados = open("teste.txt", "w")
2 dados.write("qualquer dado\n")
3 dados.close()
```

Luís Felipe

24/11/25

Criando arquivo com múltiplas linhas

O programa abaixo pede ao usuário que escolha um nome de arquivo e quantidade de linhas que deseja escrever, em seguida os seus conteúdos são lidos do teclado e escritos no arquivo.

```
1  nomeArquivo = input("Digite o nome do arquivo: ")
2  quantasLinhas = int(input("Quantas linhas: "))
3  dados = open(nomeArquivo, "w")
4
5  for i in range(quantasLinhas):
6      nova = input("Linha " + str(i+1) + ": ")
7      dados.write(nova + "\n")
8
9  dados.close()
```

Escreve o conteúdo linha por linha, adicionando \n ao final.

Luís Felipe

24/11/25

Exemplo: leitura e escrita de arquivos

Objetivo: criar um arquivo com nomes e notas de alunos, depois ler e exibir seu conteúdo.

```
# Escrita no arquivo
dados = open("alunos.txt", "w")

dados.write("Ana, 9.5\n")
dados.write("Bruno, 7.8\n")
dados.write("Carla, 8.2\n")

dados.close()

# Leitura do arquivo
dados = open("alunos.txt", "r")
linhas = dados.readlines()

for linha in linhas:
    nome, nota = linha.strip().split(", ")
    print(f"Aluno: {nome} - Nota: {nota}")
dados.close()
```

Saída:

```
Aluno: Ana - Nota: 9.5
Aluno: Bruno - Nota: 7.8
Aluno: Carla - Nota: 8.2
```

Luís Felipe

24/11/25

Abrindo arquivos com `with`

A construção `with open(...) as f` é uma forma prática e confiável de trabalhar com arquivos em Python.

- Garante que o arquivo seja fechado automaticamente ao final do bloco.
- Ajuda a evitar erros comuns, como esquecer `close()`.
- Deixa o código mais organizado e fácil de ler.

```
1 with open("dados.txt", "r") as f:
2     for linha in f:
3         print(linha, end="")
```

Luís Felipe

24/11/25

Codificação de caracteres (encoding)

- Um arquivo texto é armazenado como bytes no disco.
- O encoding (codificação) define como esses bytes são convertidos em caracteres (e vice-versa).
- Exemplos comuns de encoding:
  - `UTF-8` (padrão em muitos sistemas modernos);
  - `cp1252`, `latin1` (muito usados em Windows antigos).
- Se o encoding usado para ler o arquivo não for o mesmo usado para gravar, podem ocorrer:
  - Erros de decodificação (`UnicodeDecodeError`);
  - Caracteres estranhos: `Ã`, `§`, etc.

Luís Felipe

24/11/25

Por que usar encoding="utf-8"?

- Se não indicarmos o encoding, o Python usa um padrão do sistema:
  - Em muitos Linux/macOS: geralmente utf-8;
  - Em Windows: pode ser cp1252 ou outro.
- Isso significa que o mesmo código pode:
  - Funcionar em um computador e falhar em outro;
  - Mostrar acentos corretamente em um sistema e corrompidos em outro.
- Ao escrever:

```
1 with open(nome_arquivo, "r", encoding="utf-8") as f:
2     ...
```

estamos:

- Padronizando a forma de ler o arquivo;
- Garantindo que acentos e caracteres especiais sejam tratados corretamente;
- Tornando o programa mais portável entre diferentes máquinas.

Luís Felipe

24/11/25

Exemplo com e sem encoding

Sem especificar encoding:

```
1  # Pode funcionar em um sistema e falhar em outro
2  with open("carta.txt", "r") as f:
3      for linha in f:
4          print(linha, end="")
```

- Depende do encoding padrão do sistema.
- Pode gerar `UnicodeDecodeError` ou mostrar acentos incorretos.

Com encoding explícito (UTF-8):

```
1  with open("carta.txt", "r", encoding="utf-8") as f:
2      for linha in f:
3          print(linha, end="")
```

- Assume que o arquivo foi gravado em UTF-8 (padrão comum em editores modernos).
- Torna o comportamento do programa mais previsível e consistente em diferentes sistemas.

Luís Felipe

24/11/25

Processamento de Arquivos Texto com Números Reais

Objetivo: Processar arquivos texto não vazios cujas linhas contêm um ou mais números de ponto flutuante. O programa deve calcular estatísticas sem carregar o arquivo inteiro na memória.

Entrada: Um nome de arquivo válido. Cada linha possui números reais separados por espaços. O arquivo não estará vazio.

Saída:

- Exibir o conteúdo completo do arquivo (linha por linha);
- Calcular a média de todos os números;
- Contar quantos números são estritamente maiores que essa média.

Restrição: O arquivo pode ser maior que a memória principal. O programa deve manter apenas uma linha em memória por vez. Exemplo:

```
Conteúdo em alfa:
10 20 33.33 22.1
-43.29 87.1111 13.05
8 -77.12

Média dos Números em alfa: 8.131233333333334
Quantidade Acima de 8.131233333333334 em alfa: 6
```

Luis Felipe
24/11/25

```python
def imprime_e_calcula_soma_contagem(nome_arquivo):
    soma = 0.0
    cont = 0
    with open(nome_arquivo, "r", encoding="utf-8") as f:
        for linha in f:
            # imprime o conteúdo exatamente como está no arquivo
            print(linha, end="")   # linha já tem '\n' no final

            # processa os números da linha
            partes = linha.split()
            for p in partes:
                # converte para float
                valor = float(p)
                soma += valor
                cont += 1
    return soma, cont

def conta_acima_da_media(nome_arquivo, media):
    qtd_acima = 0
    with open(nome_arquivo, "r", encoding="utf-8") as f:
        for linha in f:
            partes = linha.split()
            for p in partes:
                valor = float(p)
                if valor > media:
                    qtd_acima += 1
    return qtd_acima

def main():
    nome_arquivo = input("Digite o nome do arquivo: ")
    # 1a passagem: imprimir conteúdo, somar e contar números
    soma, cont = imprime_e_calcula_soma_contagem(nome_arquivo)
    if cont == 0:
        print("\nNenhum número encontrado no arquivo.")
        return
    # 2a passagem: calcular média e contar quantos estão acima
    media = soma / cont
    print("\n\nMédia dos números no arquivo:", media)
    qtd_acima = conta_acima_da_media(nome_arquivo, media)
    print("Quantidade de números acima da média:", qtd_acima)

if __name__ == "__main__":  # só executa este bloco quando o arquivo é rodado diretamente
    main()                  # não roda ao ser importado por outro código; chama a função principal
```