UNIVERSIDADE DE FORTALEZA

CURSO DE ANÁLISE E DESENVOLVIMENTO DE SISTEMAS DISCIPLINA:

PROGAMAÇÃO FUNCIONAL

RELATÓRIO FINAL

EQUIPE:

DÉBORA AYRA DA COSTA OLIVEIRA (2315922)

JARDEL PONCIANO MENDES (2318767)

JOSIANE RODRIGUES DA SILVA (2317660)

JÚLIO LINCOLN CRUZ DE LIMA (2315237)

MARIA ROBERTA CARVALHO DE ALMEIDA (2319535)

NICOLAS ANDRADE BARBOZA (2314020)

FORTALEZA-CE MARÇO/2025

Relatório do Projeto: Gerenciador de Tarefas com Tkinter

1. Introdução:

Este relatório descreve o desenvolvimento de um gerenciador de tarefas com interface gráfica Tkinter, implementado em Python. O projeto foi desenvolvido seguindo os requisitos especificados na proposta do trabalho, com foco em programação funcional e interface amigável.

2. Definição de Papéis:

- Implementação do código: JÚLIO LINCOLN e JOSIANE RODRIGUES
- Criação do documento de requisitos: MARIA ROBERTA e DÉBORA AYRA
- Realização dos testes: JARDEL PONCIANO e NICOLAS ANDRADE

3. Documento de Requisitos:

- **Requisitos Funcionais:**
    - O sistema deve permitir a criação de novas tarefas com uma descrição e status inicial "pendente".
    - O sistema deve permitir a visualização de todas as tarefas, exibindo a descrição e o status (concluída ou pendente).
    - O sistema deve permitir marcar uma tarefa como concluída.
    - O sistema deve permitir a exclusão de tarefas da lista.
    - O sistema deve permitir a filtragem de tarefas por status (concluída ou pendente).
    - O sistema deve fornecer uma interface gráfica intuitiva para interagir com as tarefas.
- **Requisitos Não Funcionais:**
    - O sistema deve ser fácil de usar e ter uma interface gráfica intuitiva.
    - O sistema deve lidar com índices inválidos ao atualizar ou excluir tarefas, exibindo mensagens de erro informativas.
    - O sistema deve utilizar os conceitos de programação funcional solicitados (funções lambda, list comprehension, closures e funções de alta ordem).

- **Mapeamento de Requisitos:**
    - Criação de tarefas: função *criar_tarefa*
    - Visualização de tarefas: função *visualizar_tarefas*
    - Atualização de tarefas: função *atualizar_tarefa (dentro da closure criar_atualizador_tarefa)*
    - Exclusão de tarefas: função *excluir_tarefa*
    - Filtragem de tarefas: função *filtrar_tarefas*
    - Interface gráfica: funções *criar_tarefa_ui, atualizar_lista_tarefas, concluir_tarefa_ui e excluir_tarefa_ui*

- **Construções de Programação Funcional:**
    - Função Lambda: usada na função *filtrar_tarefas* para filtrar tarefas por status.
    - List Comprehension: usada na função *visualizar_tarefas* para formatar a saída.
    - Closure: usada na função *criar_atualizador_tarefa* para criar funções de atualização personalizadas.
    - Função de Alta Ordem: usada na função *aplicar_funcao_tarefas* para aplicar uma função a todas as tarefas.

4. Código Implementado e Compilável:

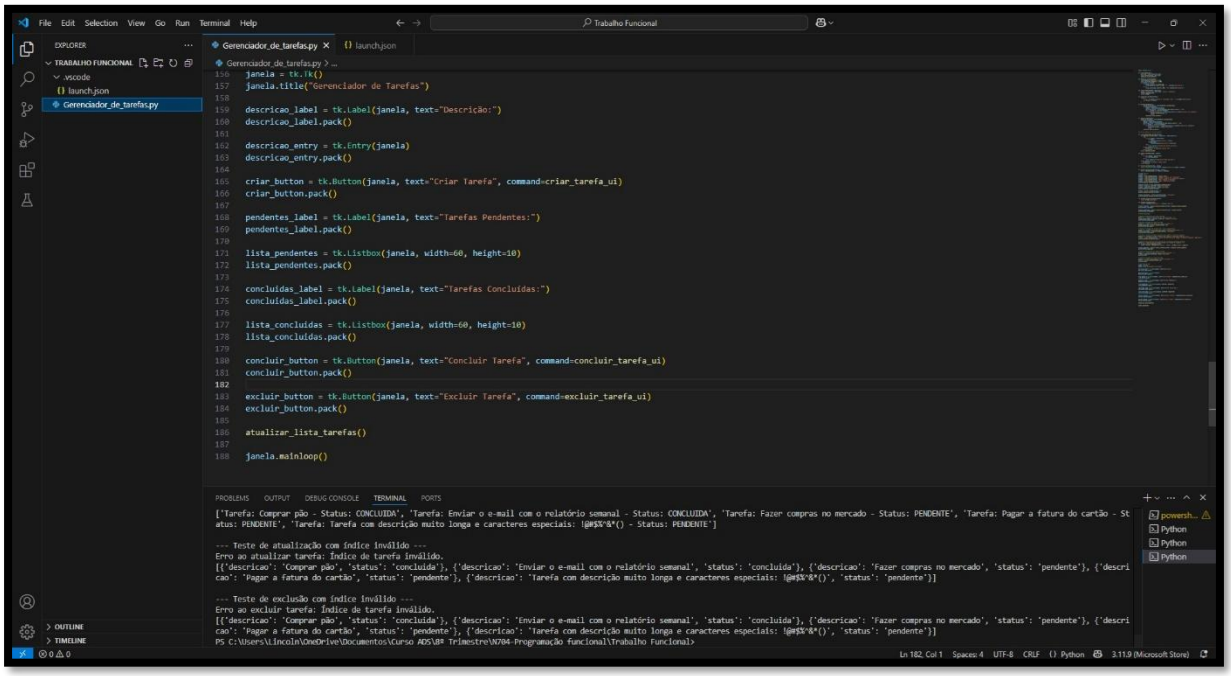
O código-fonte completo do projeto está disponível no GitHub. Ele foi implementado em Python e utiliza a biblioteca Tkinter para a interface gráfica. O código é compilável e executa todas as funcionalidades descritas nos requisitos.

5. Casos de Teste:

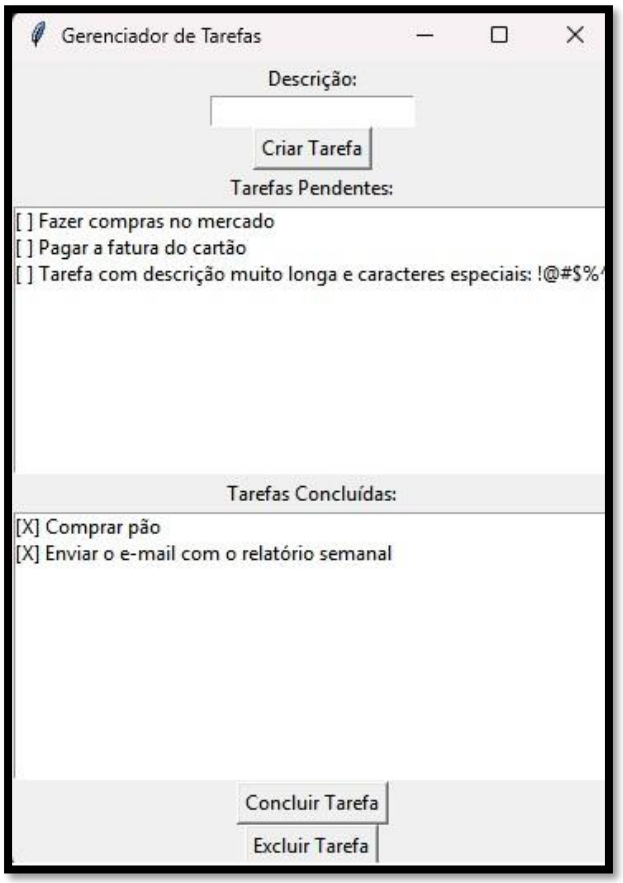
O código inclui testes para os seguintes cenários:

- Criação de tarefas com descrições longas e caracteres especiais.
- Atualização e exclusão de tarefas com índices válidos e inválidos.
- Filtragem de tarefas por status existente e inexistente.
- Aplicação de funções de formatação simples e complexas às tarefas.
- Testes da interface gráfica, verificando a criação, conclusão e exclusão de tarefas

- Teste no terminal do VS Code:



- Teste na interface gráfica Tkinter:



6. Utilização de Conceitos de Programação Funcional:

O código-fonte utiliza os seguintes conceitos de programação funcional:

- **Funções Lambda:** A função filtrar_tarefas utiliza uma função lambda para filtrar as tarefas por status.
- **List Comprehension:** A função visualizar_tarefas utiliza list comprehension para formatar a saída das tarefas.
- **Closures:** A função criar_atualizador_tarefa utiliza closures para criar funções de atualização personalizadas.
- **Funções de Alta Ordem:** A função aplicar_funcao_tarefas utiliza funções de alta ordem para aplicar uma função a todas as tarefas.

7. Uso de Chatbot (Gemini 2.0 flash)

Durante o desenvolvimento do código e da documentação, o **Gemini 2.0 flash** foi utilizado para **auxiliar na verificação das funções**, ajudando a garantir que a implementação estivesse correta e conforme os requisitos da atividade. As sugestões feitas foram analisadas e ajustadas pela equipe, de acordo com as necessidades do projeto.

8. Conclusão:

O projeto foi desenvolvido com sucesso, atendendo a todos os requisitos especificados na proposta do trabalho. O código-fonte é compilável, legível e utiliza conceitos de programação funcional de forma eficiente. A interface gráfica Tkinter torna o sistema fácil de usar e intuitivo.

9. Instruções para Executar o Código:

1. Clone o repositório do GitHub.
2. Certifique-se de ter o Python instalado.
3. Execute o arquivo Gerenciador_de_Tarefas.py.

- **Código Fonte Completo:**

```
import tkinter as tk

    def criar_tarefa_ui():
        descricao = descricao_entry.get()
        criar_tarefa(tarefas, descricao)
        atualizar_lista_tarefas()

    def atualizar_lista_tarefas():
        lista_pendentes.delete(0, tk.END)
        lista_concluidas.delete(0, tk.END)
        for tarefa in tarefas:
            if tarefa["status"] == "pendente":
                lista_pendentes.insert(tk.END, f"[ ] {tarefa['descricao']}")
            else:
                lista_concluidas.insert(tk.END, f"[X] {tarefa['descricao']}")

    def criar_tarefa(tarefas, descricao):
        tarefa = {"descricao": descricao, "status": "pendente"}
        tarefas.append(tarefa)
        return tarefas

    def visualizar_tarefas(tarefas):
        return [
            f"[{'X' if tarefa['status'] == 'concluida' else ' '}] {tarefa['descricao']}"
            for tarefa in tarefas
        ]

    def concluir_tarefa_ui():
        indice_selecionado = lista_pendentes.curselection()
        if indice_selecionado:
            indice = indice_selecionado[0]
            descricao_tarefa = lista_pendentes.get(indice).split("] ")[1]
            for i, tarefa in enumerate(tarefas):
                if tarefa["descricao"] == descricao_tarefa and tarefa["status"] == "pendente":
                    atualizar_tarefa(i, status="concluida")
                    break
            atualizar_lista_tarefas()

    # Início do código da lista de tarefas:

    def criar_atualizador_tarefa(tarefas):
        def atualizar_tarefa(indice, status=None, descricao=None):
            try:
                if 0 <= indice < len(tarefas):
                    if status:
                        tarefas[indice]["status"] = status
                    if descricao:
                        tarefas[indice]["descricao"] = descricao
                else:
                    raise IndexError("Índice de tarefa inválido.")
            except IndexError as e:
                print(f"Erro ao atualizar tarefa: {e}")
            return tarefas
        return atualizar_tarefa

    def excluir_tarefa(tarefas, indice):
        try:
            if 0 <= indice < len(tarefas):
```

```
            del tarefas[indice]
        else:
            raise IndexError("Índice de tarefa inválido.")
        except IndexError as e:
            print(f"Erro ao excluir tarefa: {e}")
        return tarefas

def filtrar_tarefas(tarefas, status):
    return list(filter(lambda tarefa: tarefa["status"] == status, tarefas))

def aplicar_funcao_tarefas(tarefas, funcao):
    return [funcao(tarefa) for tarefa in tarefas]

# Exemplo de uso
tarefas = []
tarefas = criar_tarefa(tarefas, "Comprar pão")
tarefas = criar_tarefa(tarefas, "Agendar consulta com o dentista")
tarefas = criar_tarefa(tarefas, "Enviar o e-mail com o relatório semanal")
tarefas = criar_tarefa(tarefas, "Fazer compras no mercado")
tarefas = criar_tarefa(tarefas, "Pagar a fatura do cartão")
print(visualizar_tarefas(tarefas))

atualizar_tarefa = criar_atualizador_tarefa(tarefas)
tarefas = atualizar_tarefa(0, status="concluida")
tarefas = atualizar_tarefa(2, status="concluida")
print(visualizar_tarefas(tarefas))

tarefas = excluir_tarefa(tarefas, 1)
print(visualizar_tarefas(tarefas))

tarefas_concluidas = filtrar_tarefas(tarefas, "concluida")
print(visualizar_tarefas(tarefas_concluidas))

def formatar_tarefa_resumida(tarefa):
    return tarefa['descricao']

def formatar_tarefa(tarefa):
    return f"{tarefa['descricao']} - {tarefa['status']}"

tarefas_resumidas = aplicar_funcao_tarefas(tarefas, formatar_tarefa_resumida)
print(tarefas_resumidas)

tarefas_formatadas = aplicar_funcao_tarefas(tarefas, formatar_tarefa)
print(tarefas_formatadas)

# Testes adicionais

# Teste de atualização com índice inválido
print("\n--- Teste de atualização com índice inválido ---")
tarefas_atualizadas = atualizar_tarefa(10, status="concluida")
print(tarefas_atualizadas)

# Teste de exclusão com índice inválido
print("\n--- Teste de exclusão com índice inválido ---")
tarefas_excluidas = excluir_tarefa(tarefas, 10)
print(tarefas_excluidas)

# Teste de filtragem sem tarefas com o status especificado
print("\n--- Teste de filtragem sem tarefas com status inexistente ---")
tarefas_canceladas = filtrar_tarefas(tarefas, "cancelada")
```

```
print(tarefas_canceladas)

# Teste de criação de tarefas com descrições longas e caracteres especiais
print("\n--- Teste de criação com descrições longas e caracteres especiais ---")
tarefas = criar_tarefa(tarefas, "Tarefa com descrição muito longa e caracteres especiais: !@#$%^&*()")
print(visualizar_tarefas(tarefas))

# Teste da função aplicar_funcao_tarefas com função de formatação complexa
print("\n--- Teste de aplicar_funcao_tarefas com formatação complexa ---")
def formatar_tarefa_complexa(tarefa):
    return f"Tarefa: {tarefa['descricao']} - Status: {tarefa['status'].upper()}"

tarefas_complexas = aplicar_funcao_tarefas(tarefas, formatar_tarefa_complexa)
print(tarefas_complexas)

# Teste de atualização com índice inválido
print("\n--- Teste de atualização com índice inválido ---")
tarefas = atualizar_tarefa(10, status="concluida")
print(tarefas)

# Teste de exclusão com índice inválido
print("\n--- Teste de exclusão com índice inválido ---")
tarefas = excluir_tarefa(tarefas, 10)
print(tarefas)

# Interface gráfica
janela = tk.Tk()
janela.title("Gerenciador de Tarefas")

descricao_label = tk.Label(janela, text="Descrição:")
descricao_label.pack()

descricao_entry = tk.Entry(janela)
descricao_entry.pack()

criar_button = tk.Button(janela, text="Criar Tarefa", command=criar_tarefa_ui)
criar_button.pack()

pendentes_label = tk.Label(janela, text="Tarefas Pendentes:")
pendentes_label.pack()

lista_pendentes = tk.Listbox(janela, width=60, height=10)
lista_pendentes.pack()

concluidas_label = tk.Label(janela, text="Tarefas Concluídas:")
concluidas_label.pack()

lista_concluidas = tk.Listbox(janela, width=60, height=10)
lista_concluidas.pack()

concluir_button = tk.Button(janela, text="Concluir Tarefa", command=concluir_tarefa_ui)
concluir_button.pack()

atualizar_lista_tarefas()

janela.mainloop()
```