MC102 - Projeto 03: Gerenciador de Tarefas

Integrantes:

Leonardo Franco Silva (205007) - Turma W

Murilo Tsuda (239797) - Turma W

Como Utilizar o Programa

Para rodar o programa você deve ter o Python instalado. Navegue até a pasta do projeto e execute o seguinte comando no seu terminal:

```
python lista_de_tarefas.py
```

Funcionalidades Principais

Gestão de Tarefas

- **Adicionar Tarefas**: Permite a criação de novas tarefas com título, associação a uma lista, data de término, prioridade, tags, notas e frequência de repetição. O ID da tarefa é gerado automaticamente pelo sistema.
- **Editar Tarefas**: O usuário pode alterar qualquer informação de uma tarefa existente (exceto o ID), como título, notas, data, prioridade e a lista à qual pertence.
- **Concluir Tarefas**: É possível marcar tarefas como concluídas e, depois de marcadas como concluídas, é possível torná-las pendentes novamente
- **Remover Tarefas**: O usuário pode remover tarefas de forma individual ou em massa (por exemplo, remover todas as concluídas).

Gestão de Listas de Tarefas

- **Adicionar Listas**: Crie novas listas para organizar suas tarefas (ex: "Trabalho", "Estudos",

"Pessoal").
- **Editar Listas**: Altere o nome de listas já existentes.
- **Remover Listas**: É possível remover uma lista, o que também apaga todas as tarefas contidas nela. Por segurança, o sistema não permite a exclusão da última lista restante.

Visualização e Organização

- **Filtros**: Visualize tarefas com base em múltiplos critérios:
  - Por lista de tarefas específica.
  - Por uma `tag` específica.
  - Por status (concluídas, pendentes ou todas).
  - Por data (atrasadas, para hoje, para os próximos 7 dias).
- **Ordenação**: As tarefas podem ser ordenadas por data de término (padrão) ou por nível de prioridade.

Busca

- **Busca Rápida**: Encontre tarefas buscando por um termo que pode estar presente no título, nas notas ou nas tags da tarefa.

Persistência de Dados

- **Salvamento Automático**: Todas as alterações, como a criação de uma nova tarefa ou a edição de uma lista, são salvas automaticamente em um arquivo `dados_tarefas.json`. Isso garante que os dados não sejam perdidos ao fechar ou sair do programa.

Estrutura dos Arquivos

Este projeto é dividido em módulos para separar as responsabilidades:

1. `lista_de_tarefas.py`

Este é o **ponto de entrada principal** da aplicação. É o arquivo que você executa para iniciar o Gerenciador de Tarefas.

- **Responsabilidade**: Orquestrar o fluxo do programa. Ele contém o loop principal que exibe o menu inicial e direciona o usuário para as diferentes funcionalidades (visualizar, adicionar, buscar, etc.) com base na sua escolha.
- **Como funciona**: Ele cria uma instância do `TaskManager` e entra em um loop `while`,

chamando funções do módulo `ui` para interagir com o usuário e, em seguida, acionando os métodos apropriados no `gerenciador` para executar as ações.

Bibliotecas e Importações Utilizadas

- `from datetime import date, timedelta`: Utilizado para manipular datas. `date` é usado para obter a data atual (`today`) e `timedelta` para calcular períodos de tempo, como os "próximos 7 dias".
- `from typing import List`: Usado para "Type Hinting", que, no Python, é um meio de mostrar o tipo que é esperado do retorno de algo, ajudando a tornar o código mais legível e a evitar erros, especificando que uma variável deve ser uma lista de um determinado tipo (ex: `List[Tarefa]`).
- `from manager import TaskManager`: Importa a classe principal `TaskManager`, que contém toda a lógica de negócios, do arquivo `manager.py`.
- `from models import Tarefa`: Importa a classe `Tarefa` do arquivo `models.py` para que o código saiba como lidar com objetos de tarefa.
- `import ui`: Importa todo o módulo `ui.py`, que contém as funções responsáveis por exibir menus e interagir com o usuário.

2. `manager.py`

Este arquivo pode ser considerado o **cérebro da aplicação**. Ele contém a classe `TaskManager`, que centraliza toda a lógica de negócios do sistema.

- **Responsabilidade**: Gerenciar as operações de CRUD (Criar, Ler, Atualizar, Deletar) para tarefas e listas. Ele lida com a lógica de:
  - Adicionar e remover listas e tarefas.
  - Editar os dados de tarefas e listas.
  - Marcar tarefas como concluídas e lidar com a recorrência.
  - Buscar tarefas por termos.
- **Como funciona**: A classe `TaskManager` mantém o estado atual das listas e tarefas em memória (`self._listas`, `self._tarefas`). Sempre que uma alteração é feita, ela chama as funções do módulo `persistence` para salvar os dados no arquivo JSON.

Bibliotecas e Importações Utilizadas

- `import copy`: Utilizado especificamente para a função `copy.deepcopy()`. Isso é crucial para criar uma cópia totalmente independente de uma tarefa ao lidar com tarefas recorrentes, evitando que a nova tarefa e a antiga compartilhem referências.
- `from datetime import timedelta`: Usado para calcular a data da próxima ocorrência de tarefas repetitivas (diária, semanal, etc.).
- `from typing import List, Optional, Dict, Any`: Importações para fazer o Type Hinting do

Python. `Optional` indica que um valor pode ser `None`, `Dict` para dicionários e `Any` para qualquer tipo.
- `from models import Tarefa, ListaDeTarefas`: Importa as classes de modelo de dados para criar, ler e manipular tarefas e listas.
- `import persistence`: Importa o módulo responsável por salvar e carregar os dados no arquivo JSON.

3. `models.py`

Este arquivo define as **estruturas de dados** do projeto. Ele contém as classes que representam os objetos principais do sistema: `Tarefa` e `ListaDeTarefas`.

- `Tarefa`: Representa uma tarefa individual com todos os seus atributos, como `id`, `titulo`, `data_termino`, `prioridade`, `tags`, etc.
- `ListaDeTarefas`: Representa uma lista que agrupa tarefas. Contém atributos como `id` e `nome`.
- **Funcionalidades Chave**: Ambas as classes possuem os métodos `to_dict()` e `from_dict()`, que convertem os objetos Python em um formato (dicionário) que pode ser facilmente salvo como JSON, e vice-versa.

Bibliotecas e Importações Utilizadas

- `from datetime import date`: Usado para tipar o atributo `data_termino` na classe `Tarefa` e para converter as datas entre o formato de string (para salvar em JSON) e objetos `date` do Python.
- `from typing import List, Optional, Dict, Any`: Usado para a tipagem dos atributos das classes, melhorando a clareza e a manutenibilidade do código.

4. `ui.py`

Este módulo é responsável por toda a **interação com o usuário**. Ele separa completamente a lógica de apresentação (o que o usuário vê) da lógica de negócios (o que o sistema faz).

- **Responsabilidade**:
  - **Exibir Menus**: Contém funções para mostrar todos os menus de navegação (principal, de ações, de filtros, etc.).
  - **Imprimir Dados**: Formata e imprime as listas de tarefas de maneira clara e legível no terminal.
  - **Capturar Entradas**: Contém funções para obter dados do usuário, como os detalhes de uma nova tarefa, o ID de uma tarefa a ser editada ou o termo para uma busca.
  - **Funções Auxiliares**: Inclui funções úteis como `clear_screen()` para limpar a tela do terminal e `pausar_e_limpar()` para melhorar a experiência do usuário.

Bibliotecas e Importações Utilizadas

- `import os`: Biblioteca padrão do Python para interagir com o sistema operacional. É usada na função `clear_screen()` para executar o comando `cls` (no Windows) ou `clear` (em Linux/macOS) e limpar a tela do terminal.
- `from datetime import date, datetime`: `date` é utilizado para verificar se uma tarefa está atrasada, comparando sua data de término com a data atual (`date.today()`). `datetime` é usado para permitir a conversão de strings de data em um formato personalizado. É usada com `datetime.strptime(data_str, '%d/%m/%Y')` para que o usuário possa digitar a data no formato `DD/MM/AAAA`.
- `from typing import List, Dict, Any, Optional`: Usado para tipar os parâmetros e os valores esperados de retorno das funções.
- `from manager import TaskManager`: Importado para fins de "Type Hinting", indicando que algumas funções recebem um objeto `TaskManager` como parâmetro.
- `from models import Tarefa`: Importado para que as funções que manipulam ou exibem tarefas (como `imprimir_tarefas`) saibam qual é a estrutura de um objeto `Tarefa`.

5. `persistence.py`

Este módulo lida com a **leitura e escrita de dados** no disco. Sua única responsabilidade é a persistência dos dados.

- **Responsabilidade**: Salvar o estado atual das tarefas e listas em um arquivo `dados_tarefas.json` e carregar esses dados quando o programa inicia.
- `salvar_dados()`: Recebe as listas de objetos `Tarefa` e `ListaDeTarefas`, converte-as em dicionários usando os métodos `to_dict()`, e as escreve no arquivo JSON.
- `carregar_dados()`: Lê o arquivo JSON, converte os dados de volta para objetos Python usando os métodos `from_dict()`, e os retorna para o `TaskManager`. Se o arquivo não existir, ele cria uma estrutura de dados padrão.

Bibliotecas e Importações Utilizadas

- `import json`: Biblioteca essencial para a codificação e decodificação de dados no formato JSON. `json.dump()` é usado para escrever no arquivo, e `json.load()` para ler.
- `import os`: Usado para interagir com o sistema de arquivos. `os.path.exists()` verifica se o arquivo de dados já existe, e `os.path.getsize()` verifica se o arquivo não está vazio antes de tentar lê-lo.
- `from typing import List, Tuple`: Usado para tipar os valores de retorno das funções, indicando que `carregar_dados` retorna uma tupla contendo duas listas.
- `from models import Tarefa, ListaDeTarefas`: Importa as classes de modelo para poder recriar os objetos Python (`Tarefa` e `ListaDeTarefas`) a partir dos dados lidos do arquivo JSON.

6. `dados_tarefas.json`

Este arquivo funciona como o **banco de dados** da sua aplicação.

- **Responsabilidade**: Armazenar todas as listas e tarefas criadas pelo usuário em formato JSON.
- **Como funciona**: É um arquivo de texto simples que mantém os dados de forma estruturada, com uma chave para `listas` e outra para `tarefas`.

Exemplo de Uso: Adicionando e Visualizando uma Tarefa

Passo 1: Iniciar o programa

Para começar, execute o script principal no seu terminal.

```
python lista_de_tarefas.py
```

Você irá se deparar com o menu principal:

```
========================================
        Gerenciador de Tarefas
========================================

1. Visualizar Tarefas
2. Adicionar Tarefa
3. Buscar Tarefas
4. Gerenciar Listas
5. Sair

Escolha uma opção:
```

Passo 2: Adicionar uma nova tarefa

Vamos adicionar uma tarefa para "Terminar o projeto 3 de MC102".

1. Digite `2` e pressione Enter para escolher "Adicionar Tarefa".
2. O programa pedirá os detalhes da tarefa. Preencha-os como no exemplo abaixo.

```
======================================
         Adicionar Nova Tarefa
======================================

Listas disponíveis:
  ID: 1 - Geral
  ID: 2 - Teste

Digite o ID da lista para a nova tarefa: 1
Título da tarefa: Terminar o projeto 3 de MC102
Data de término (DD/MM/AAAA, opcional): 04/07/2025
Prioridade (alta, media, baixa, opcional): alta
Tags (separadas por vírgula, opcional): universidade, projeto
Notas (opcional): Focar na UI/UX
Repetição (diaria, semanal, mensal, anual, opcional):

Salvando dados...
Dados salvos com sucesso!

Tarefa adicionada com sucesso!

Pressione Enter para continuar...
```

Após pressionar Enter, a tela será limpa e você retornará ao menu principal.

Passo 3: Visualizar a tarefa criada

Agora, vamos visualizar a tarefa que acabamos de criar.

1. No menu principal, digite `1` e pressione Enter para escolher "Visualizar Tarefas".
2. O programa apresentará as opções de visualização. Vamos escolher ver todas as tarefas.

```
======================================
        Opções de Visualização
======================================

Como você deseja visualizar as tarefas?
1. Todas as Tarefas (geral)
2. Por Lista de Tarefas
3. Por Tag
4. Voltar

Escolha uma opção: 1
```

3. Em seguida, escolha o filtro. Para este exemplo, vamos ver todas, sem filtrar.

```
Aplicar qual filtro?
1. Ver todas as tarefas neste contexto
2. Apenas tarefas para hoje (e atrasadas)
3. Apenas tarefas para os próximos 7 dias (e atrasadas)
4. Apenas tarefas não concluídas
5. Apenas tarefas concluídas

Escolha uma opção de filtro: 1
```

4. Finalmente, escolha a ordenação (padrão por data).

```
Escolha a ordem de visualização:
1. Ordenar por Data (Padrão)
2. Ordenar por Prioridade
Escolha uma opção de ordenação (padrão é 1): 1
```

O sistema exibirá a lista de tarefas, incluindo a que acabamos de adicionar.

```
========================================
            Todas as Tarefas
========================================

[ ] ID: 3    | Terminar o back-end da tela de login do projeto | Data: 30/06/2025 (Atrasada!)
    Notas: Ainda falta implementar a lógica de verificar o domínio do email do usuário para v
[ ] ID: 4    | Terminar o projeto 3 de MC102  | Data: 04/07/2025     | Lista: Geral
    Notas: Focar na UI/UX
[ ] ID: 1    | Estudar para a prova           | Data: 07/07/2025     | Lista: Geral
[ ] ID: 2    | Levar cachorro para passear    | Data: Sem data       | Lista: Geral

Ações disponíveis:
1. Concluir uma tarefa
2. Desmarcar uma tarefa (tornar pendente)
3. Editar uma tarefa
4. Remover uma tarefa
5. Voltar

Escolha uma opção:
```

A partir daqui, você pode escolher uma ação, como concluir ou editar a tarefa, ou voltar ao menu principal.
