import datetime

import json

import os

from colorama import init, Fore, Style

init(autoreset=True)

TASK\_FILE = "tasks.json"

class Task:

def \_init\_(self, description, due\_date=None, priority="medium", completed=False):

self.description = description

self.due\_date = due\_date

self.priority = priority

self.completed = completed

def to\_dict(self):

return {

"description": self.description,

"due\_date": self.due\_date.strftime('%Y-%m-%d') if self.due\_date else None,

"priority": self.priority,

"completed": self.completed

}

@classmethod

def from\_dict(cls, data):

due\_date = datetime.datetime.strptime(data['due\_date'], '%Y-%m-%d').date() if data['due\_date'] else None

return cls(data['description'], due\_date, data['priority'], data['completed'])

def \_str\_(self):

status = Fore.GREEN + "Completed" if self.completed else Fore.YELLOW + "Pending"

due\_info = f" (Due: {self.due\_date.strftime('%Y-%m-%d')})" if self.due\_date else ""

priority\_color = {"high": Fore.RED, "medium": Fore.BLUE, "low": Fore.CYAN}

return f"- {self.description} [{priority\_color.get(self.priority, Fore.WHITE)}Priority: {self.priority}]{due\_info} [{status}]"

class TaskManager:

def \_init\_(self):

self.tasks = []

self.load\_tasks()

def save\_tasks(self):

with open(TASK\_FILE, 'w') as f:

json.dump([task.to\_dict() for task in self.tasks], f)

def load\_tasks(self):

if os.path.exists(TASK\_FILE):

with open(TASK\_FILE, 'r') as f:

self.tasks = [Task.from\_dict(t) for t in json.load(f)]

def add\_task(self, description, due\_date\_str=None, priority="medium"):

if not description.strip():

print("Description cannot be empty.")

return

due\_date = None

if due\_date\_str:

try:

due\_date = datetime.datetime.strptime(due\_date\_str, '%Y-%m-%d').date()

except ValueError:

print("Invalid date format. Please use YYYY-MM-DD.")

return

if priority not in ["high", "medium", "low"]:

priority = "medium"

task = Task(description, due\_date, priority)

self.tasks.append(task)

self.save\_tasks()

print(f"Task '{description}' added.")

def view\_tasks(self):

if not self.tasks:

print("No tasks to display.")

return

print("\n--- Your Tasks ---")

for i, task in enumerate(self.tasks):

print(f"{i+1}. {task}")

print("------------------")

def mark\_completed(self, task\_index):

if 0 <= task\_index < len(self.tasks):

self.tasks[task\_index].completed = True

self.save\_tasks()

print(f"Task '{self.tasks[task\_index].description}' marked as completed.")

else:

print("Invalid task number.")

def delete\_task(self, task\_index):

if 0 <= task\_index < len(self.tasks):

removed = self.tasks.pop(task\_index)

self.save\_tasks()

print(f"Task '{removed.description}' deleted.")

else:

print("Invalid task number.")

def edit\_task(self, task\_index, new\_desc=None, new\_due\_date=None, new\_priority=None):

if 0 <= task\_index < len(self.tasks):

task = self.tasks[task\_index]

if new\_desc:

task.description = new\_desc

if new\_due\_date:

try:

task.due\_date = datetime.datetime.strptime(new\_due\_date, '%Y-%m-%d').date()

except ValueError:

print("Invalid date format. Skipping date change.")

if new\_priority in ["high", "medium", "low"]:

task.priority = new\_priority

self.save\_tasks()

print(f"Task '{task.description}' updated.")

else:

print("Invalid task number.")

def prioritize\_tasks\_ai(self):

print("\n--- Prioritizing tasks (AI-driven simulation) ---")

self.tasks.sort(key=lambda t: (

{"high": 0, "medium": 1, "low": 2}.get(t.priority, 1),

t.due\_date if t.due\_date else datetime.date.max

))

self.view\_tasks()

print("------------------------------------------")

def main():

manager = TaskManager()

while True:

print("\n--- Task Manager Menu ---")

print("1. Add Task")

print("2. View Tasks")

print("3. Mark Task as Completed")

print("4. Prioritize Tasks (AI-driven)")

print("5. Edit Task")

print("6. Delete Task")

print("7. Exit")

choice = input("Enter your choice: ").strip()

if choice == '1':

description = input("Enter task description: ")

due\_date\_str = input("Enter due date (YYYY-MM-DD, optional): ")

priority = input("Enter priority (high, medium, low): ").lower()

manager.add\_task(description, due\_date\_str, priority)

elif choice == '2':

manager.view\_tasks()

elif choice == '3':

manager.view\_tasks()

try:

task\_num = int(input("Enter task number to mark as completed: "))

manager.mark\_completed(task\_num - 1)

except ValueError:

print("Please enter a valid number.")

elif choice == '4':

manager.prioritize\_tasks\_ai()

elif choice == '5':

manager.view\_tasks()

try:

index = int(input("Enter task number to edit: ")) - 1

new\_desc = input("New description (leave blank to skip): ")

new\_date = input("New due date (YYYY-MM-DD, leave blank to skip): ")

new\_priority = input("New priority (high/medium/low, leave blank to skip): ").lower()

manager.edit\_task(index, new\_desc or None, new\_date or None, new\_priority or None)

except ValueError:

print("Invalid input.")

elif choice == '6':

manager.view\_tasks()

try:

task\_num = int(input("Enter task number to delete: "))

manager.delete\_task(task\_num - 1)

except ValueError:

print("Invalid input.")

elif choice == '7':

print("Goodbye!")

break

else:

print("Invalid choice. Please try again.")

if \_name\_ == "\_main\_":

main()