**Documentation**

**Medical Expert App**

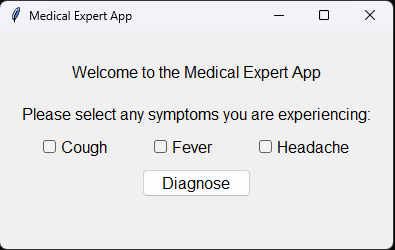


**Submitted by:**

Carl Michael C. De Taza

Sue C. Sinangote

David Frank Edison Regodos





**Prompt:** Introduction

Medical Expert App is an application crafted in Python utilizing the Tkinter GUI library. This innovative program serves as a valuable tool for users seeking to assess common illnesses based on presented symptoms. Through a user-friendly interface, individuals can input symptoms, receiving precise and informed analyses.



**Prompt:** Advantage and Disadvantage

**Advantages of the Medical Expert App:**

* **User-Friendly Interface:** The application promotes modularity, making the code easier to understand, maintain, and extend. Each method within the class handles a specific aspect of the application's functionality.

**Disadvantages of the Medical Expert App:**

* **Diagnostic Enhancement**: The provided code implements a basic rule-based diagnosis system suitable for simple scenarios. However, as the complexity of diagnostic rules and the number of symptoms increase, the code's scalability may become limited. Moreover, it lacks validation to ensure proper user interactions. For instance, it does not handle cases where users might select conflicting symptoms.



**Prompt:** Recommendations

**Improve Symptom List:** For a more reliable result, consider expanding the list of symptoms and incorporate a broader range of medical conditions. This can enhance the application's utility and provide users with a more comprehensive self-diagnosis tool.

**Integration of Health Information:** Explore the integration of health databases or APIs to enhance the accuracy of the diagnosis. Real-time access to medical information can contribute to more informed and up-to-date results.



**Prompt:** System Requirements

**Hardware Requirements:**

* No specific hardware requirements beyond a standard computer with a display.

**Software Requirements:**

|  |  |
| --- | --- |
| **Name** | **Descriptions** |
| Python 3.x | Python 3.x, the latest version, brings key improvements like Unicode support, enhanced syntax, and asyncio. It's actively maintained, secure, and backward incompatible with Python 2.x. Widely used in web development, AI, and more, Python 3.x is the recommended choice for new projects. |
| Tkinter | Tkinter is the standard GUI (Graphical User Interface) toolkit that comes with Python. It provides a set of tools and widgets for creating graphical user interfaces and is based on the Tk GUI toolkit. Tkinter allows developers to create windows, dialogs, buttons, textboxes, and other GUI elements for their Python applications. |



**Prompt:** Installation

**Python**

Ensure that Python 3.x is installed on your system. You can download it from the official Python website: [Python Downloads](<https://www.python.org/downloads/>)

**Create a virtual environment(optional)**

* **Open Command Prompt:**

Open the Command Prompt on your Windows desktop.

* **Navigate to the Desired Directory:**

Use the **cd** command to navigate to the directory where you want to create the virtual environment. For example:



* **Create Virtual Environment:**

Run the following command to create a virtual environment named **env**:

****

* **Activate Virtual Environment:**

To activate the virtual environment, run the appropriate activation script. In the Command Prompt:****

**Clone Repository from Github**

* **Get the Repository URL:**

On the GitHub repository page, click on the "Code" button. Make sure to select the "HTTPS" option. Copy the repository URL provided: <https://github.com/Chael07/drill4_Medical_Expert_System.git>

* **Open Terminal Command Prompt (Windows):**

Open a terminal or command prompt on your local machine.

* **Navigate to the Directory Where You Want to Clone the Repository:**

Use the cd command to navigate to the directory where you want to store the cloned repository. For example:****

* **Clone the Repository:**

Use the git clone command followed by the repository URL. Replace <https://github.com/Chael07/drill4_Medical_Expert_System.git> with the URL you copied in step 1.

**Required Python Packages**

Install the required Python packages using the following commands:





**Prompt:** Application Overview

**Code Structure**

The code is organized into a single Python script (`medical.py`) containing a class `MedicalExpertSystem`. Here's an overview of the code structure:

**Imported Libraries:**

* `tkinter`: Tkinter library for GUI.

**Class ‘MedicalExpertSystem’**

* Represents the blueprint for instances of the medical expert system application.
* Encapsulates the entire functionality, including GUI setup and symptom-based diagnosis.
* Facilitates a modular and organized structure for managing different aspects of the application.

**\_\_init\_\_(self, root)**

* Initializes an instance of the class.
* Sets up the graphical user interface (GUI) elements, such as labels, checkboxes, and buttons.
* Configures a custom style for a modern and visually appealing look.

**diagnose(self)**

* Performs the diagnosis based on selected symptoms.
* Retrieves the state of symptom variables (cough, fever, headache) using the get() method.
* Applies a simple rule-based logic to determine a potential diagnosis.
* Updates the result label in the GUI with the diagnosis.

**Main Execution:**

* The if \_\_name\_\_ == "\_\_main\_\_": block orchestrates the execution when the script is run directly.
* Creates an instance of the MedicalExpertSystem class.
* Initializes the Tkinter root window as the main window for the application.
* Initiates the Tkinter event loop with root.mainloop(), allowing the application to respond to user interactions.
* Facilitates the seamless integration of the class into the application's runtime environment, ensuring a user-friendly and interactive experience.



**Prompt:** Conclusion

The Medical Expert App offers a user-friendly interface for assessing current health conditions and diagnosing common illnesses based on presented symptoms. By expanding and refining its features, the application aims to provide an even more comprehensive and personalized user experience.



**Prompt:** Source Code Used in the Project

import tkinter as tk

from tkinter import ttk

class MedicalExpertSystem:

    def \_\_init\_\_(self, root):

        self.root = root

        self.root.title("Medical Expert App")

        # Create a style for a more modern look

        style = ttk.Style()

        style.configure('TLabel', font=('Arial', 12))

        style.configure('TCheckbutton', font=('Arial', 12))

        style.configure('TButton', font=('Arial', 12))

        # Create a frame for better organization

        frame = ttk.Frame(root, padding="20")

        frame.grid(row=0, column=0, sticky=(tk.W, tk.E, tk.N, tk.S))

        # Add a title label

        title\_label = ttk.Label(frame, text="Welcome to the Medical Expert App", style='TLabel')

        title\_label.grid(row=0, column=0, columnspan=3, pady=10)

        # Add instructions

        instruction\_label = ttk.Label(frame, text="Please select any symptoms you are experiencing:", style='TLabel')

        instruction\_label.grid(row=1, column=0, columnspan=3, pady=10)

        # Symptoms checkboxes

        self.cough\_var = tk.BooleanVar()

        self.cough\_checkbox = ttk.Checkbutton(frame, text="Cough", variable=self.cough\_var, style='TCheckbutton')

        self.cough\_checkbox.grid(row=2, column=0, padx=10)

        self.fever\_var = tk.BooleanVar()

        self.fever\_checkbox = ttk.Checkbutton(frame, text="Fever", variable=self.fever\_var, style='TCheckbutton')

        self.fever\_checkbox.grid(row=2, column=1, padx=10)

        self.headache\_var = tk.BooleanVar()

        self.headache\_checkbox = ttk.Checkbutton(frame, text="Headache", variable=self.headache\_var, style='TCheckbutton')

        self.headache\_checkbox.grid(row=2, column=2, padx=10)

        # Diagnose button

        self.diagnose\_button = ttk.Button(frame, text="Diagnose", command=self.diagnose, style='TButton')

        self.diagnose\_button.grid(row=3, column=0, columnspan=3, pady=10)

        # Result label

        self.result\_label = ttk.Label(frame, text="", style='TLabel')

        self.result\_label.grid(row=4, column=0, columnspan=3)

    def diagnose(self):

        cough = self.cough\_var.get()

        fever = self.fever\_var.get()

        headache = self.headache\_var.get()

        # Simple rule-based logic using if-else statements

        if cough and fever:

            result = "You may have the flu."

        elif cough and not fever and headache:

            result = "You may have a cold."

        elif not cough and fever and headache:

            result = "You may have a migraine."

        else:

            result = "You seem to be healthy."

        # Update the result label

        self.result\_label.config(text=result)

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

    root = tk.Tk()

    app = MedicalExpertSystem(root)

    root.mainloop()

**END**