FINAL PROJECT: AI ASSISTANT FOR CARDIAC REHABILITATION

Brandon Everett

JULY 6, 2025

CSC-510

Module 8

AI Use - Assistant for Cardiac Rehabilitation

For my final project, I built an artificial intelligence program to help people who are recovering from serious heart problems. The idea came from a conversation with my brother, who works in cardiac education. He was talking about how AI might change his job one day, and we thought, why not try to build something helpful ourselves. Patients leaving the hospital get a lot of information, and it can be really overwhelming. They might forget what their doctor said or feel unsure about their recovery at home. My program is designed to be a helpful assistant that can answer their questions using the information from their official cardiac rehabilitation guide (Krames, 2021). It's not meant to be a doctor, but rather a supportive tool that provides clear information and helps patients feel more confident in managing their health.

Getting the Knowledge into the Program

The first big challenge was getting the information from the physical booklet, "A Guide to Cardiac Rehabilitation," into a format the computer could understand. I ended up scanning every single page by hand, which took longer than I expected! This booklet is the main source of knowledge for my AI, since it is the booklet that cardiac patients are sent home after a heart procedure in the hospital. I started by taking the book apart and scanning each page one by one. After scanning, I had a PDF file. Converting this to a Microsoft Word document was a simple first step. The real challenge was getting the text from that Word document into a usable and organized data structure for my program.

To get the text out of the Word document, I used a Python library called python-docx (python-docx, n.d.). This made it easy for me to write a script called book_extract.py that automatically pulled all the text out. My script looked through the document and figured out what the section headings were. Each heading became a key, and the text under it became the value, which was saved in a JSON file. JSON is like a simple dictionary that my program can quickly look through to find information.

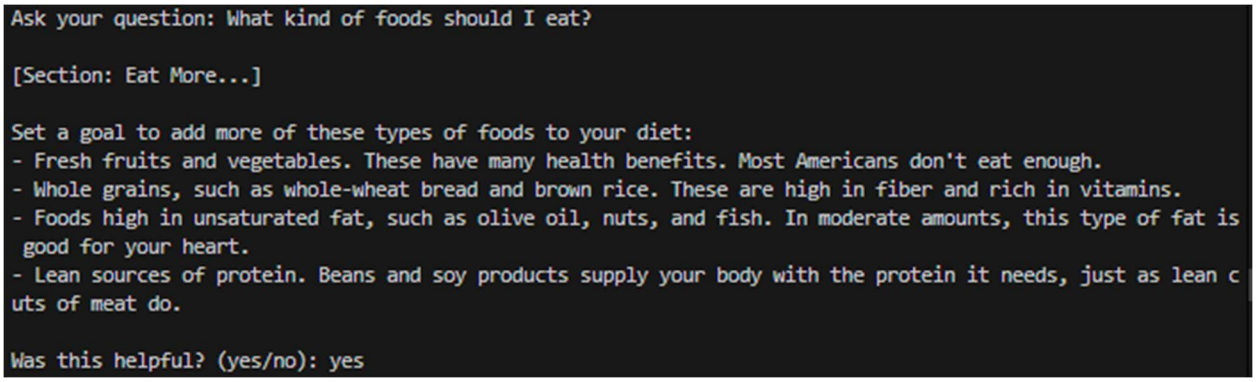
After that, I wrote another script named section_keywords_counter.py. This script went through the JSON file and found important words (keywords) from each section. It ignored common, less important words, which helped my program know exactly what to look for.

How the AI Finds Answers

After getting all the information set up, the next step was to build the actual brain of the assistant. My goal was to do something that could take a person's question and find the best answer from the booklet information I had stored. For this part, I used some of the ideas from class about intelligent search and expert systems. You could say my program is like a simple expert system. It has all the expert knowledge from the booklet, and it just uses some basic rules I created to connect the user to the right piece of information.

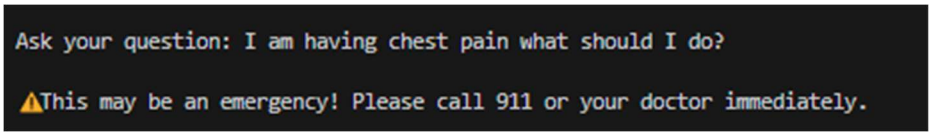
The search method I built is a ranked keyword search. When a user asks a question, like “what activities do I need to avoid,” the program first cleans up the question. It removes common words like “what,” “do,” and “I,” which don’t carry much meaning. Then, it looks at the important keywords that are left. My program compares these keywords to a list of important words I already identified for each section of the booklet using another script, keyword_counter.py.

The program gives each section a score based on how many keywords it shares with the user’s question. Sections that have matching words in their title get a higher score. After scoring all the sections, the program creates a ranked list from highest score to lowest. This ranked list is a great example of an uninformed search strategy. The program doesn't know the "right" answer ahead of time, but it uses a simple rule, the keyword score, to intelligently guess which sections are most likely to be helpful.

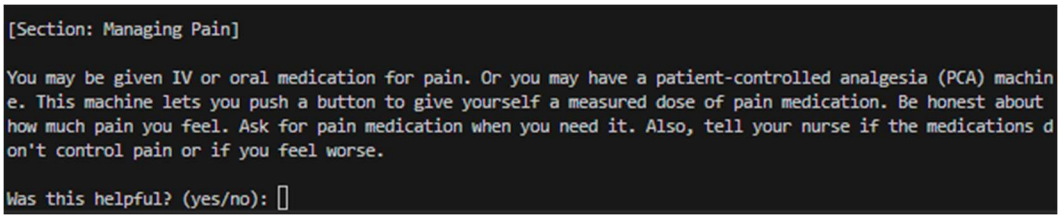


For example, when I asked the program, “What kind of foods should I eat,” it returned the correct section on nutrition as the very first result, which was a perfect match. However, the search doesn't always get it right on the first try, which is where the ranking comes in handy. In another test, when I asked, “What activities do I need to avoid?” the program needed a bit more guidance. The first answer wasn't quite right, and it took four "no" responses from me before the program presented the correct section on the fifth try. This shows how the search method keeps trying until it finds what the user needs.

Planning the Conversation



My program also uses a simple form of symbolic planning to guide the conversation with the user. This plan is for the flow of interaction. The plan is a series of steps the program follows. First, it gets the user's question. Second, it checks for any emergency keywords like “chest pain” or “stroke.” If it finds any, its plan immediately changes to show an emergency warning and advise the user to call 911. This is a critical safety rule.



If there is no emergency, the program follows its search plan: find the best section and show it to the user. Then, it asks for feedback. Based on the user's "yes" or "no" answer, the plan decides what to do next. A "yes" ends that search, while a "no" tells the program to continue down its ranked list of sections. This whole process, from checking for emergencies to showing answers in order, is a symbolic plan that helps the program solve the user's need for information in a structured way. The general idea for creating this kind of rule based chatbot was inspired by guides on how to build them using Python (w3resource, n.d.; Code-B.dev, 2024).

Tools and Conclusion

To build this project, I used the Python programming language and a few key libraries. These included python-docx to read the booklet (python-docx, n.d.), the re module for working with text (Python Software Foundation, n.d.-b), and the json module to store the knowledge base (Python Software Foundation, n.d.-a). It is important to note that my program does not use deep learning or complex neural networks. It is a symbolic AI that relies on clear rules and logic. This approach makes it very predictable and reliable for the specific task it was designed for.

In the end, I created a fully working AI program that solves a real world problem. It provides patients with an easy way to get answers to their questions about cardiac recovery, directly from a trusted source. I was happy that my program successfully used knowledge representation, intelligent search, and symbolic planning to support users in their decision making. While it will never replace a healthcare professional, it serves as a valuable and accessible tool to help people on their journey to better health.

Future Improvements

While I'm proud of what I built, there are ways to make it even better in the future. The biggest improvement would be to make the search smarter. Right now, it just looks for keywords. It would be much more powerful if it could understand the meaning behind a person's question, like a semantic search, even if they don't use the exact right words.

Also, A simple, clean webpage or a mobile app would make it much easier for people to use, especially for those who aren't comfortable with computers. Finally, it would be great if the program could learn from its conversations. When a user says "no" to an answer, the program could remember that and be less likely to show that same answer for a similar question in the future. These additions would make the assistant an even more effective and user-friendly tool.

References

Code-B.dev. (2024, March 6). *Detailed guide on how to develop a chatbot in Python*. https://code-b.dev/blog/detailed-guide-on-how-to-develop-a-chatbot

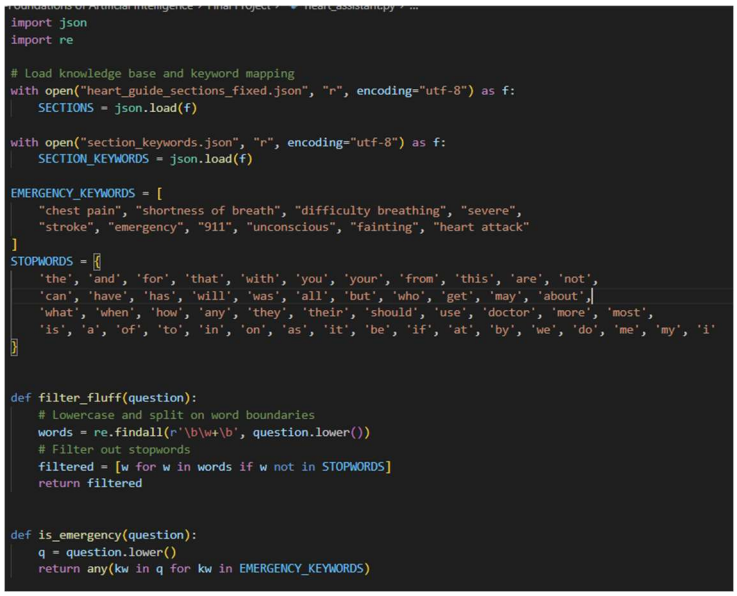
Krames. (2021). *A Guide to Cardiac Rehabilitation*. Krames.

python-docx. (n.d.). *python-docx Documentation*. Retrieved July 5, 2025, from https://python-docx.readthedocs.io/en/latest/

Python Software Foundation. (n.d.-a). *json — JSON encoder and decoder*. Python 3 documentation. Retrieved July 5, 2025, from https://docs.python.org/3/library/json.html

Python Software Foundation. (n.d.-b). *re — Regular expression operations*. Python 3 documentation. Retrieved July 5, 2025, from https://docs.python.org/3/library/re.html

w3resource. (n.d.). *Develop a rule-based chatbot using Python and regex*. https://www.w3resource.com/python-exercises/advanced/develop-a-rule-based-chatbot-using-python-and-regex.php



```python
import json
import re

# Load knowledge base and keyword mapping
with open("heart_guide_sections_fixed.json", "r", encoding="utf-8") as f:
    SECTIONS = json.load(f)

with open("section_keywords.json", "r", encoding="utf-8") as f:
    SECTION_KEYWORDS = json.load(f)

EMERGENCY_KEYWORDS = [
    "chest pain", "shortness of breath", "difficulty breathing", "severe",
    "stroke", "emergency", "911", "unconscious", "fainting", "heart attack"
]
STOPWORDS = {
    'the', 'and', 'for', 'that', 'with', 'you', 'your', 'from', 'this', 'are', 'not',
    'can', 'have', 'has', 'will', 'was', 'all', 'but', 'who', 'get', 'may', 'about',
    'what', 'when', 'how', 'any', 'they', 'their', 'should', 'use', 'doctor', 'more', 'most',
    'is', 'a', 'of', 'to', 'in', 'on', 'as', 'it', 'be', 'if', 'at', 'by', 'we', 'do', 'me', 'my', 'i'
}


def filter_fluff(question):
    # Lowercase and split on word boundaries
    words = re.findall(r'\b\w+\b', question.lower())
    # Filter out stopwords
    filtered = [w for w in words if w not in STOPWORDS]
    return filtered


def is_emergency(question):
    q = question.lower()
    return any(kw in q for kw in EMERGENCY_KEYWORDS)
```

```python
def ranked_sections(question, top_n=5):
    important_words = set(filter_fluff(question))
    scored = []

    for section, content_keywords in SECTION_KEYWORDS.items():
        # Get the words directly from the section title string
        title_words = set(filter_fluff(section))

        # Score title matches with a higher weight (e.g., 3 points)
        title_score = sum(word in important_words for word in title_words) * 3

        # Score content matches with the base weight (1 point)
        content_score = sum(word in important_words for word in content_keywords)

        total_score = title_score + content_score

        if total_score > 0:
            scored.append((total_score, section))

    # Sort total score
    scored.sort(reverse=True)
    return [section for score, section in scored[:top_n]]


def ai_assistant():
    print("Welcome to the Heart Recovery Q&A Assistant!")
    print("Ask your question, or type 'quit' to exit.\n")

while True:
    question = input("Ask your question: ").strip()
    if question.lower() in ("quit", "exit"):
        print("Thank you for using the assistant. Stay healthy!")
        break

    if is_emergency(question):
        print("\n⚠ This may be an emergency! Please call 911 or your doctor immediately.\n")
```

```python
    section_list = ranked_sections(question, top_n=6)
    if not section_list:
        print("\nSorry, I couldn't find an answer. Please consult your recovery booklet or doctor.")
        print("\nDISCLAIMER: This is general information, not medical advice. Always consult your healthcare provider.\n")
        continue

    shown = set()
    for section in section_list:
        if section not in SECTIONS or section in shown:
            continue
        print(f"\n[Section: {section}]\n")
        print(SECTIONS[section][:2000])
        shown.add(section)
        resp = input("\nWas this helpful? (yes/no): ").strip().lower()
        if resp in ("yes", "y"):
            print("\nGlad I could help! If you have another question, just ask.")
            break
        elif resp in ("no", "n"):
            continue
        else:
            print("I'll take that as a no. Let's keep looking.")
    else:
        print("\nSorry, I've shown all the most likely sections. Please check your booklet or consult your healthcare provider.")

    print("\nDISCLAIMER: This is general information, not medical advice. Always consult your healthcare provider.\n")


if __name__ == "__main__":
    ai_assistant()
```