A picture containing text

Description automatically generated

**CZ3005 ARTIFICIAL INTELLIGENCE**

**IMPLEMENTING A TALKING BOX WITH PROLOG**

**ASSIGNMENT 4 REPORT**

Shearman Chua Wei Jie (U1820058D)

LAB GROUP: TSP1

Contents

[Sympathetic Doctor interactive Talking Box 3](#_Toc56626726)

[Scope of interactive Talking Box 3](#_Toc56626727)

[Implementation of Sympathetic Doctor interactive Talking Box 3](#_Toc56626728)

[Starting the Sympathetic Doctor interactive Talking Box 4](#_Toc56626729)

[nextQuestion Predicates Functionality 6](#_Toc56626730)

[answer\_h and answer Predicates Functionality 7](#_Toc56626731)

[human\_symptom and human\_illness Predicates Functionality 8](#_Toc56626732)

[openings and opening Predicates Functionality 9](#_Toc56626733)

[question\_starts, question\_start and human\_diagnose Predicates Functionality 9](#_Toc56626734)

[pain\_level, mood\_level, symptom\_library and illness Predicates Functionality 10](#_Toc56626735)

[reply\_question, reply\_diagnose and response Predicates Functionality 11](#_Toc56626736)

[Demonstration of the Sympathetic Doctor interactive Talking Box 12](#_Toc56626737)

[Diagnosis 12](#_Toc56626738)

[Exiting the Sympathetic Doctor interactive Talking Box Script 13](#_Toc56626739)

Question 4: Patient with a Sympathetic Doctor

# Sympathetic Doctor interactive Talking Box

## Scope of interactive Talking Box

The interactive Talking Box will be programmed in the Prolog language. The Prolog script will take-up the persona of a sympathetic doctor, where he is required to:

1. Converse with a patient who can answer only yes or no.
2. Diagnose the patient’s condition while asking questions sensitively depending upon patient’s pain level and mood level.
3. Able to choose 5 or more different mood considerations (calm, angry, etc.) and 5 or more levels of pain.
4. Able to diagnose five or more diseases, where each disease is characterized by 5 or more symptoms

## Implementation of Sympathetic Doctor interactive Talking Box

As mentioned, the interactive Talking Box will be implemented using the Prolog programming language. The Talking Box will take the form of an automated Chatbot, where the user will answer yes or no to the questions that the Talking Box ask, and at the end of the questioning, provide a diagnosis for the user.

From my implementation of the Sympathetic Doctor interactive Talking Box, the Sympathetic Doctor, named Dr Bot, is able to diagnose these common illnesses and diseases:

1. Allergic reaction to bee sting
2. Moderate reaction to bee sting
3. Bee sting
4. General allergic reaction
5. Eczema
6. Cut
7. Moderate cut
8. Deep cut
9. General injury
10. General severe injury
11. Blunt force injury
12. Severe blunt force injury
13. Fracture
14. Sprain
15. Dengue Fever
16. Food poisoning
17. Cold
18. Flu
19. Severe Flu
20. Headache
21. Migraine
22. Viral fever
23. Heat exhaustion

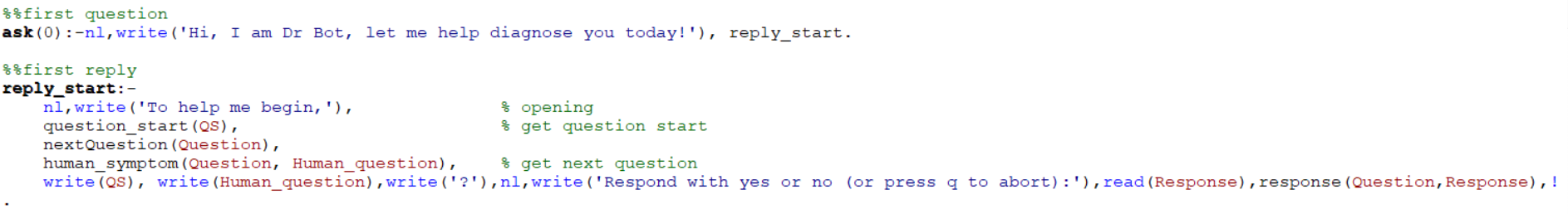
The Sympathetic Doctor interactive Talking Box will ask the user questions regarding the symptoms that he might be having and provide the user with a final diagnosis or diagnoses. The Sympathetic Doctor interactive Talking Box will run continuously for the user to consult the Sympathetic Doctor, Dr Bot, for diagnoses until the letter ‘q’ is entered to the Prolog program to terminate it.

The code uses the assert() function to tell the Knowledge-Based System (KBS) what are questions the user has answered and what are the symptoms the user has. In order to dynamically select subsequent questions to ask, we need to use the “dynamic” command to let Prolog allow such assertions to modify the KBS on the fly:



For our Prolog script, the **answered**, **have\_symptom**, **pain**, **mood**, and **ready\_to\_diagnose** predicates are required to be asserted when the program is being run, therefore, within the Prolog script, we declare these predicates as dynamic.

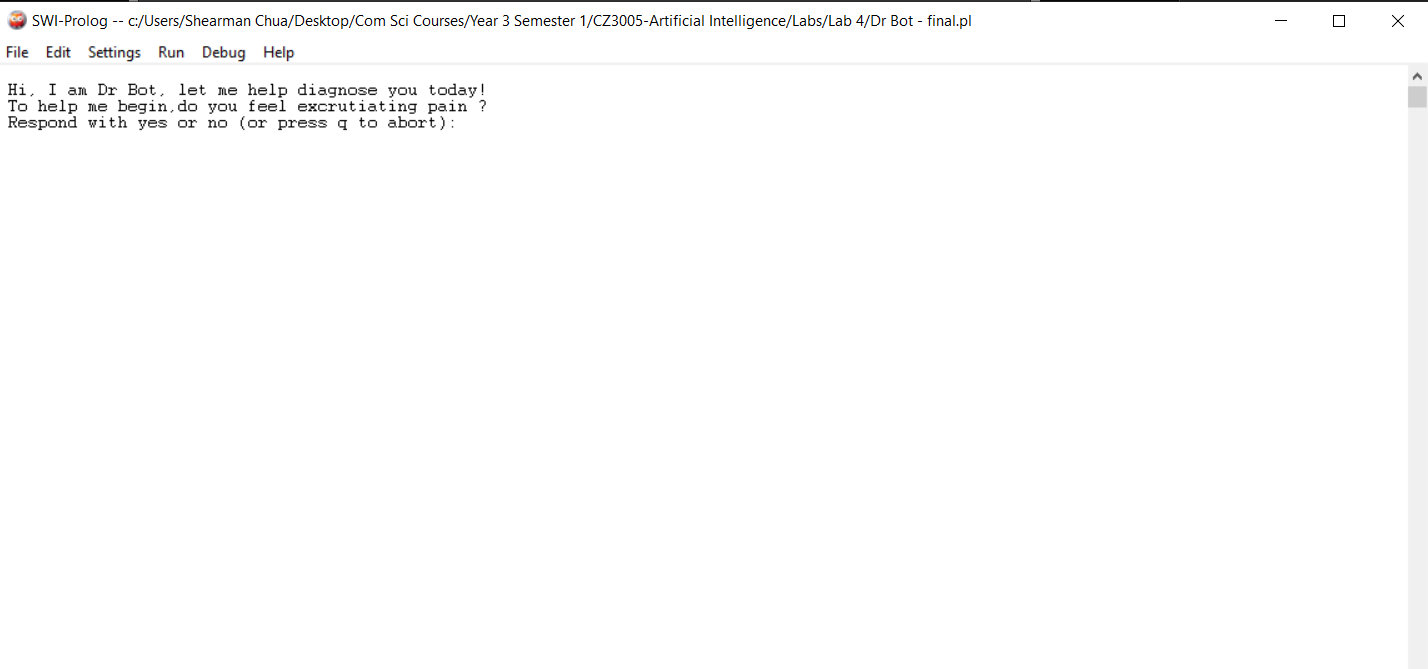
### Starting the Sympathetic Doctor interactive Talking Box





The Sympathetic Doctor interactive Talking Box Prolog script is able automatically started asking the user questions when the Prolog file is opened in Prolog. This is due to the fact that the line **?-ask(0)** is declared in the script which will automatically run the first function **ask(0)**, when the script is opened.

In the Prolog script, we make use of the **write()** predicate that is built-in to Prolog to display messages to the users to tell them what to do. The **ask(0)** predicate writes the introductory greeting to the user, as the persona of Dr Bot, a Sympathetic Doctor who helps to diagnose a user’s illness. The **ask(0)** function then calls the **reply\_start** function to start asking the user questions. In the **reply\_start** function, we call the **question\_start** function to get the head of the question, **nextQuestion** function to get the question to be asked, **human\_symptom** function to convert the question to be asked into a string and finally write it onto the Prolog console using the **write()** predicate. The **read()** predicate is then used to read the user’s response and then the response is passed to the **response()** predicate to handle the user’s response.



**Display in Prolog when Prolog Script is started**

As can be seen from the figure above, when the Sympathetic Doctor interactive Talking Box Prolog script is opened in Prolog, the Prolog displays the initial greeting and begins asking the user questions. The first question the Sympathetic Doctor interactive Talking Box will ask is regarding the level of pain the user is feeling. There are 5 levels of pain in the Sympathetic Doctor Prolog script, namely:

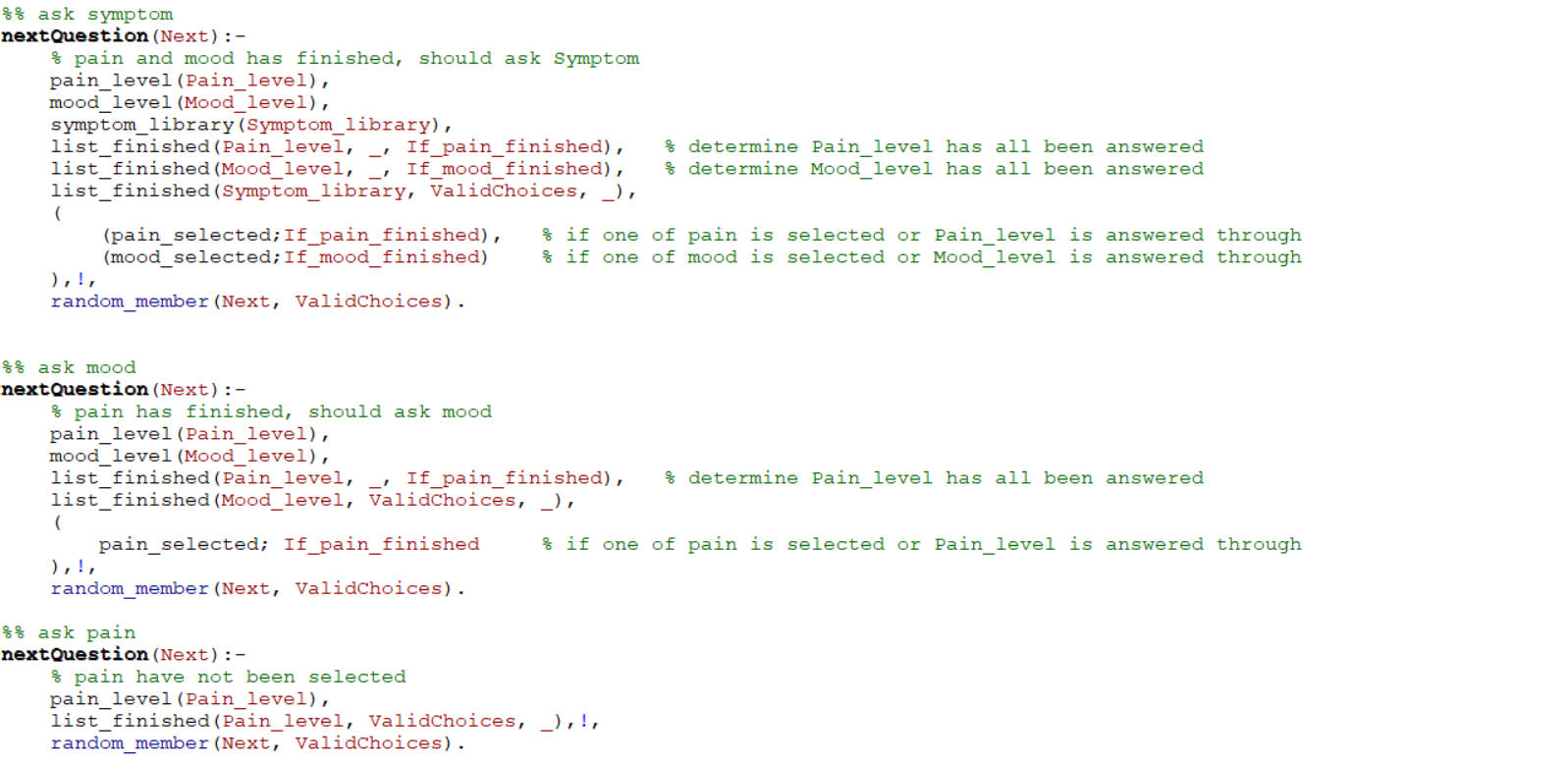
1. no\_pain (No pain)
2. mild\_pain (Mild pain)
3. moderate\_pain (Moderate pain)
4. severe\_pain (Severe pain)
5. excruciating\_pain (Excruciating pain)

Only when the user answers ‘yes’ to one of the pain levels, or when the 5 levels of pain are asked finished and the user answers ‘no’ to all of them, then the Prolog script proceeds to ask the user regarding the mood the user is feeling. There are 5 different mood considerations in the Sympathetic Doctor Prolog script, namely:

1. angry
2. upset
3. frustrated
4. fatigued
5. calm

Only when the user answers ‘yes’ to one of the mood considerations, or when the 5 mood considerations are asked finished and the user answers ‘no’ to all of them, then the Prolog script proceeds to ask the user about the symptoms the user have.

### nextQuestion Predicates Functionality



The figure above shows the 3 **nextQuestion** Predicates that are used to decide the next question to be asked to the user.

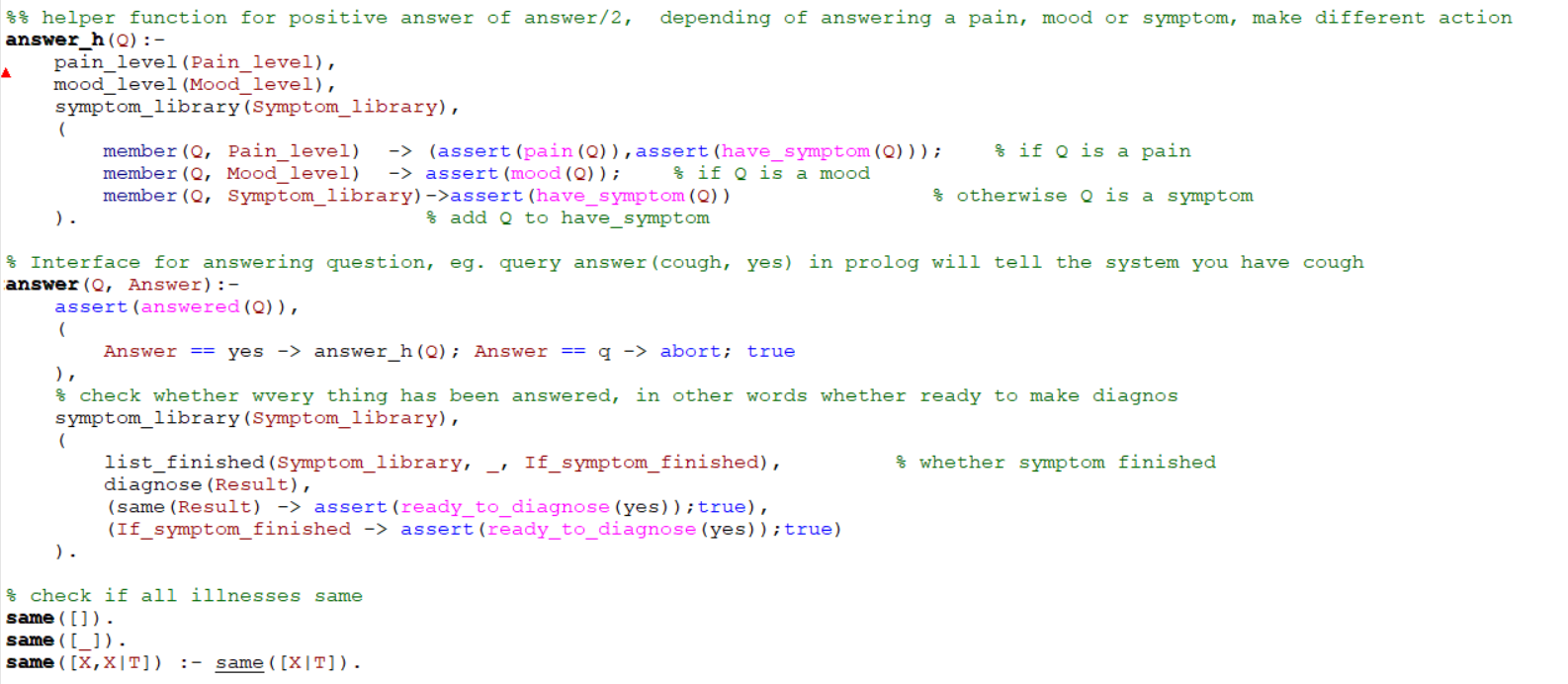
The very first **nextQuestion** predicate first retrieves the 5 levels of pain, retrieves the 5 mood considerations, and retrieves the available. Next, the **nextQuestion** predicate checks if one of the pain level is selected or if the 5 levels of pain are asked finished, as well as if one of the mood is selected or if the 5 mood considerations are asked finished, if yes, the **nextQuestion** predicate proceed to choose one of the symptoms to ask the user from the list of symptoms that had not been asked in **ValidChoices** by calling the **random\_member** predicate to choose one of the symptoms from **ValidChoices**, if not we proceed to the second **nextQuestion** predicate.

The second **nextQuestion** predicate first retrieves the 5 levels of pain and retrieves the 5 mood considerations. Next, the **nextQuestion** predicate checks if one of the pain level is selected or if the 5 levels of pain are asked finished, , if yes, the **nextQuestion** predicate proceed to choose one of the moods to ask the user from the list of moods that had not been asked in **ValidChoices** by calling the **random\_member** predicate to choose one of the moods from **ValidChoices**, if not we proceed to the third **nextQuestion** predicate.

The third **nextQuestion** predicate first retrieves the 5 levels of pain. Then, the **nextQuestion** predicate proceed to choose one of the pain level to ask the user from the list of pain that had not been asked in **ValidChoices** by calling the **random\_member** predicate to choose one of the pain level from **ValidChoices**.

The 3 **nextQuestion** Predicates helps ensure that the doctor has determined the pain level and the mood of the patient before moving on to ask the user about his symptoms.

### answer\_h and answer Predicates Functionality

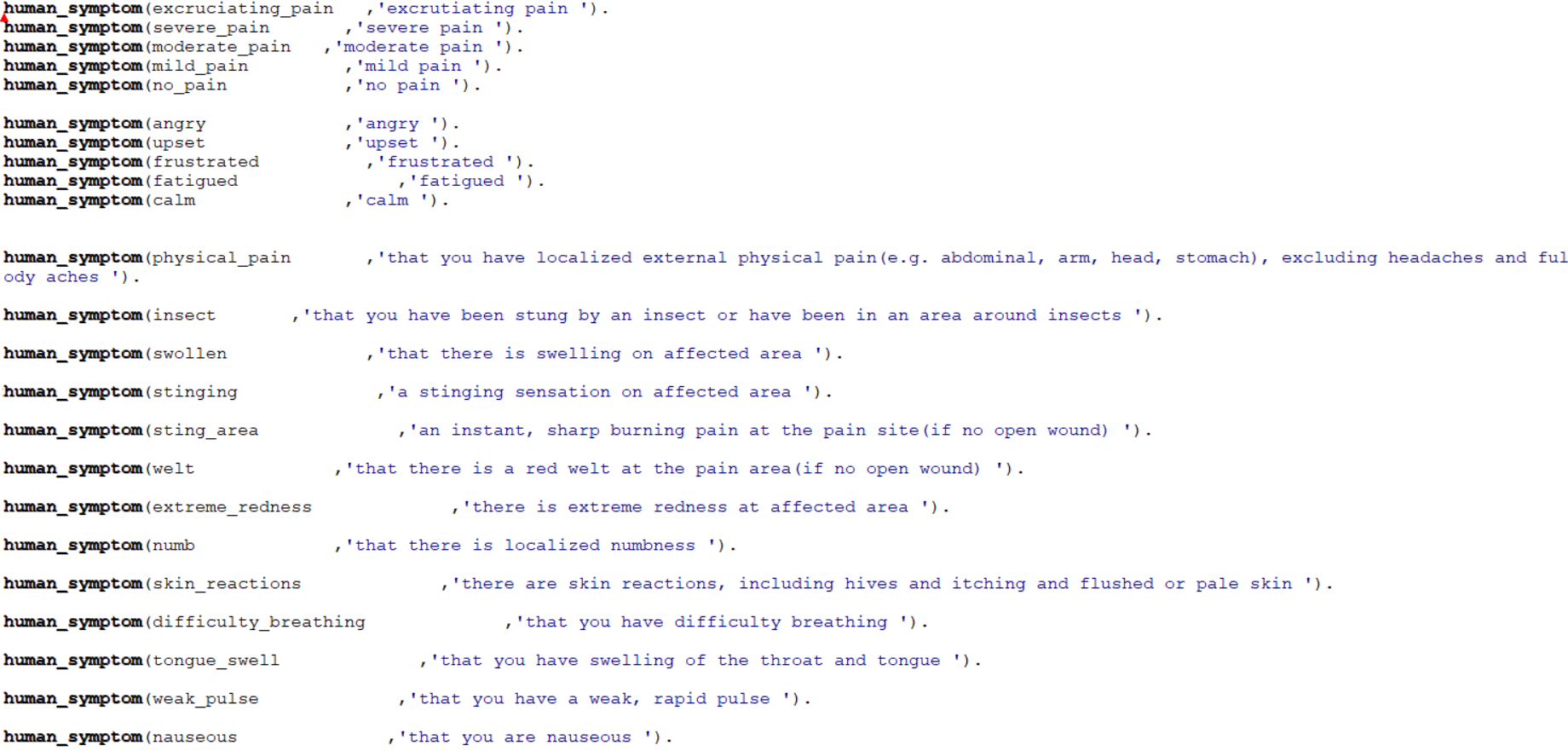


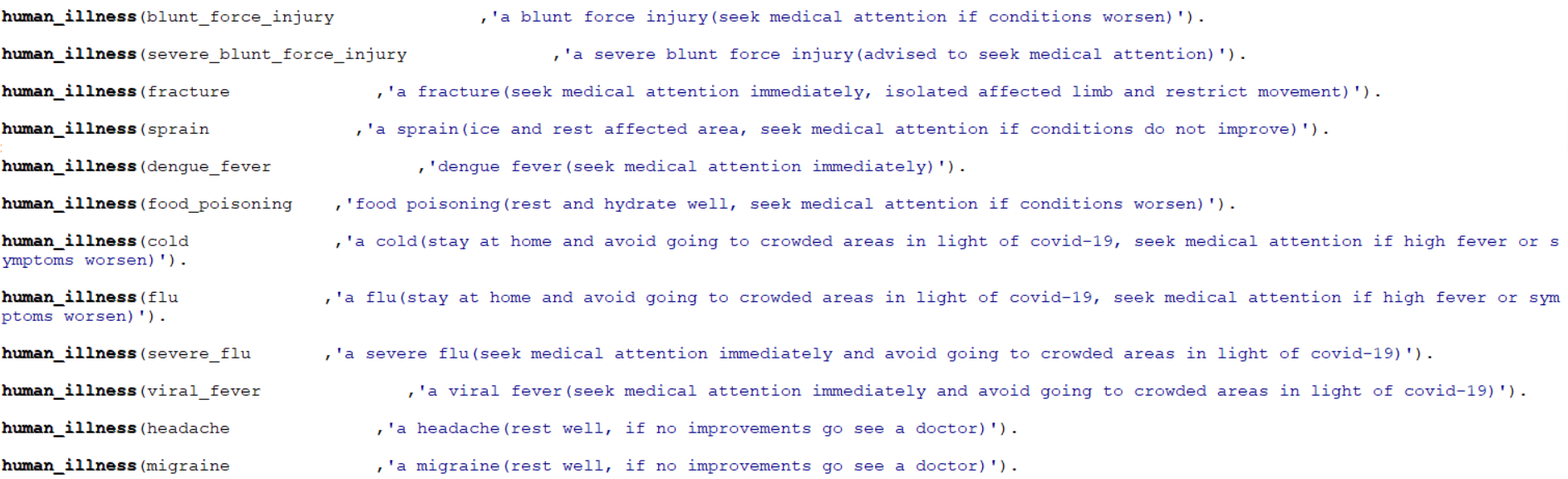
In the figure above, we have the **answer\_h()** and **answer()** predicates. The **answer()** predicate first asserts the answered question by calling **assert(answered(Q))** which tells the Knowledge-Based System (KBS) what questions have been answered by the user. Next, if the user has answered ‘yes’ for that question, the **answer\_h()** predicate is called.

The **answer\_h()** predicate then helps to determine if the question answered is a **pain**, **mood** or a **symptom** and uses the assert function to assert the **pain**, **mood** or **have\_symptom** predicates, respectively.

We then go back to the **answer()** predicate to check if either, there is only one possible diagnosis the Sympathetic Doctor interactive Talking Box Prolog script has narrowed down to by first calling the **diagnose()** predicate to get the possible diagnoses based on the symptoms answered ‘yes’ and check if the diagnoses that were returned are the same illness, or if all of the symptoms have been asked. If either condition is true, the **ready\_to\_diagnose** predicate is set to yes to inform the Prolog script to write out the diagnosis to the user.

### human\_symptom and human\_illness Predicates Functionality

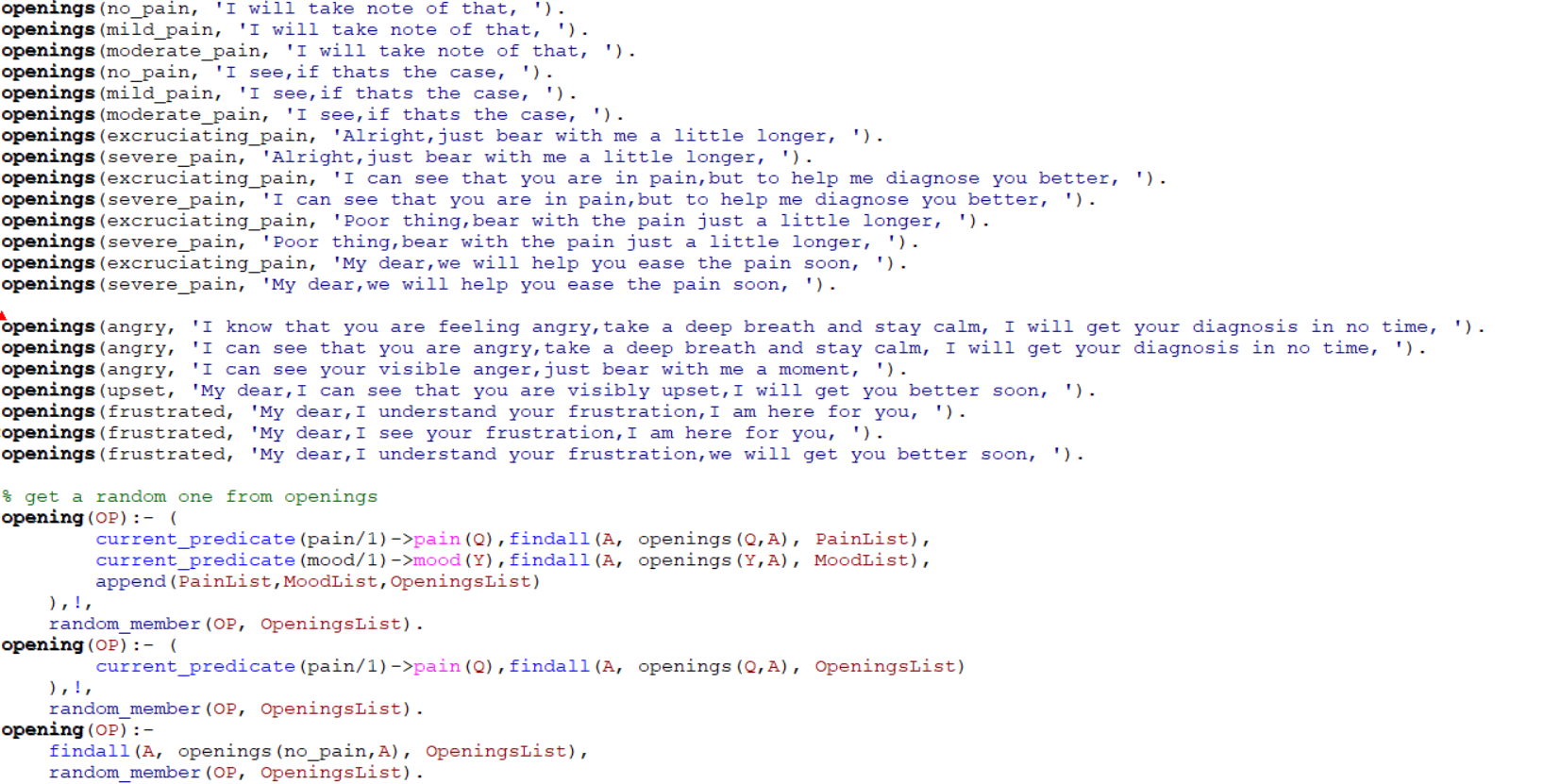




The **human\_symptom** and **human\_illness** predicates stores the sentence string for each of the symptoms and illnesses in the Prolog Knowledge Base to be written out to the console for the user to see and understand in a more human readable form.

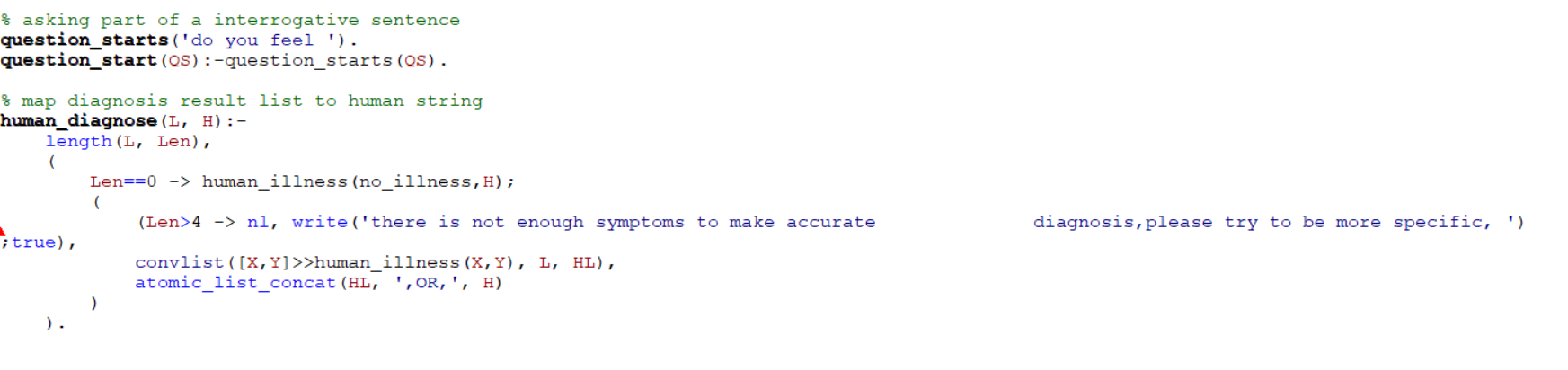
The **human\_symptom** predicate stores the symptoms to be asked to the user and the sentence string allows the user to better understand the symptoms to be able to answer the questions more accurately. The **human\_illness** predicate stores the possible illnesses sentence strings in the Prolog knowledge base and when a diagnosis is made, the diagnosed result/results are written to the console in human readable form based on the **human\_illness** predicate sentence strings as well as the different advise for the various illnesses.

### openings and opening Predicates Functionality



The **openings** predicate stores the sentence string for each of the possible openings for the questions asked by Dr Bot for the Sympathetic Doctor interactive Talking Box. In the scope, we mentioned that the interactive Talking Box should be able to diagnose the patient’s condition while asking questions sensitively depending upon patient’s pain level and mood level. Therefore, each **openings** predicate stores the sentence string of an opening to a question for a particular pain level or mood, and the Prolog script will choose one of sentence string corresponding to the patient’s mood or pain level.

### question\_starts, question\_start and human\_diagnose Predicates Functionality



The **question\_start** predicate chooses one of the possible starts to a question from the possible **question\_starts** predicates after the opening of the question which is decided by the **opening** predicate. In our case, there is only one possible **question\_starts** predicate to choose from. Next, the **human\_diagnose** predicate is used to concatenate all the possible diagnoses obtained from the **diagnose()** predicate into a string to be written to the console to tell the user the possible diagnoses the Sympathetic Doctor interactive Talking Box has derived.

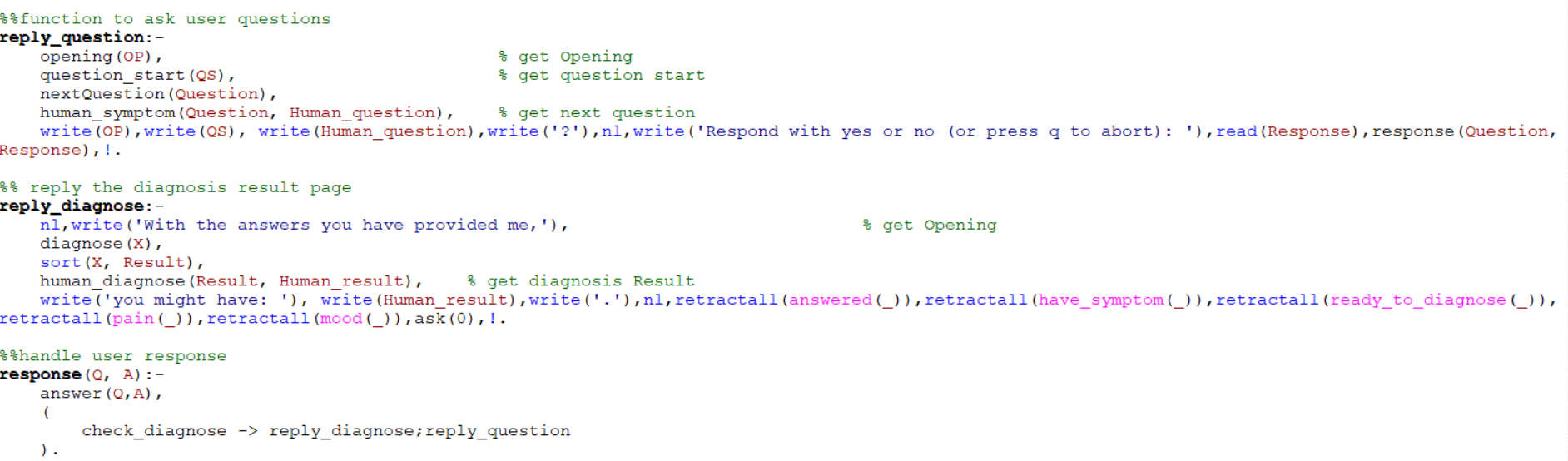
### pain\_level, mood\_level, symptom\_library and illness Predicates Functionality



The **pain\_level** predicate stores the list of the 5 levels of pain, and the **mood\_level** predicate stores the 5 mood considerations that are available in the Prolog Knowledge-Based System (KBS). The **symptom\_library** predicate then stores all the defined symptoms in the Prolog Knowledge-Based System (KBS).

Next, we have the **illness** predicates. Each of the illness predicate stores an illness that can be diagnosed by the Sympathetic Doctor interactive Talking Box, followed by the list of symptoms that the illness presents.

### reply\_question, reply\_diagnose and response Predicates Functionality



Finally, we come to the last 3 predicates that ensure that the Sympathetic Doctor interactive Talking Box Prolog script is able to run continuously and ask the user questions.

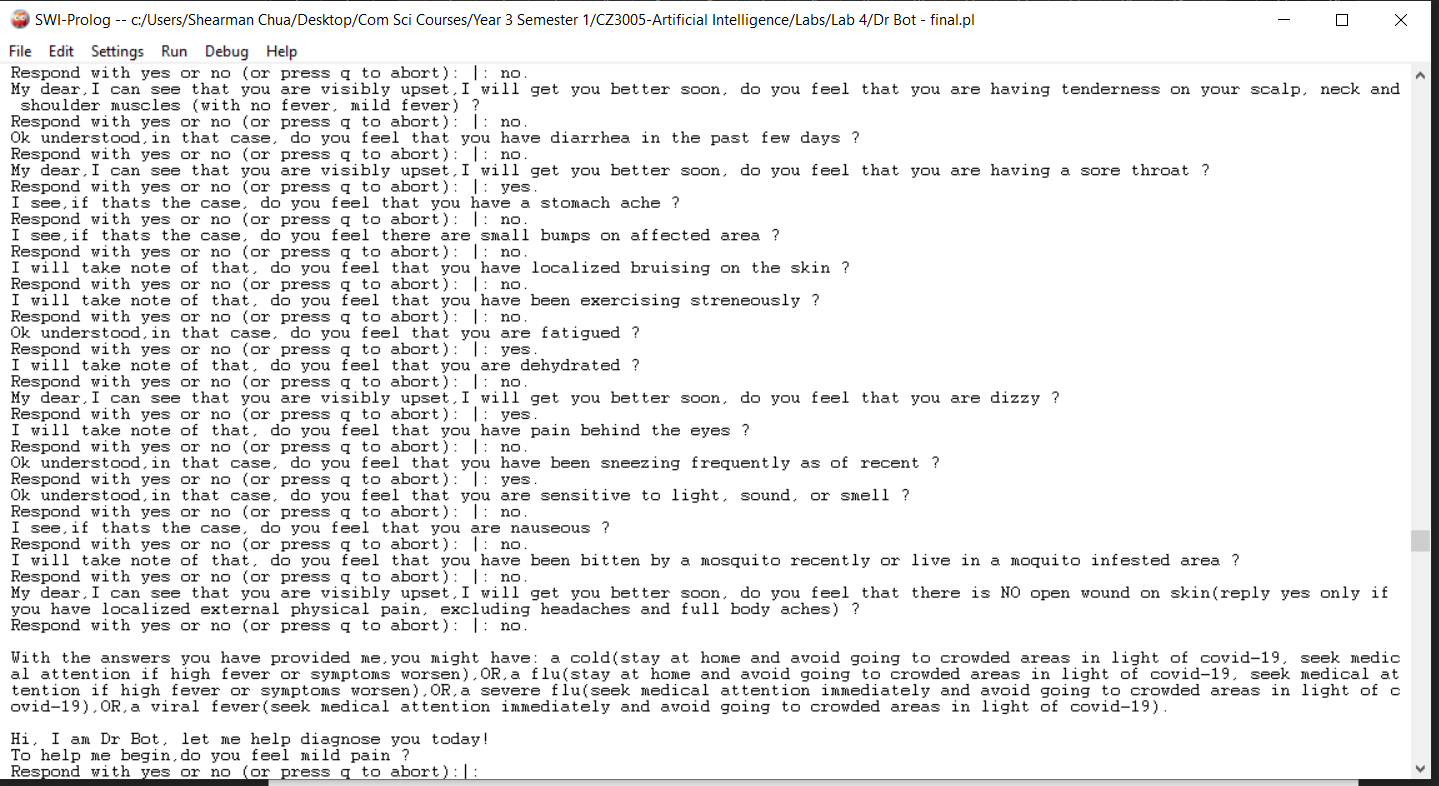
The **reply\_question** predicate is used to write the question onto the Prolog console for the user to view and reply to. The **reply\_question** predicate first calls the **opening**, **question\_start** and **nextQuestion** predicates to retrieve the opening, the question header as well as the question to be asked to the user from the Prolog knowledge base. Next, because the **nextQuestion** predicate returns the symptom to be asked in a not so human readable form, we use the **human\_symptom** predicate to get the human readable sentence string to be asked to the user. Then, finally, we use the **write()** predicate to write the question to the user on the Prolog console. When the user provides a response in the form of ‘yes.’ or ‘no.’, we will read the response given by using the **read()** predicate and then pass the response and the corresponding question into the **response** predicate to be handled.

The **response** predicate first calls the **answer** predicate to log the user’s response into the Prolog knowledge base. Then, the **response** predicate calls the **check\_diagnose** predicate to check if the Sympathetic Doctor interactive Talking Box Prolog script is ready to make a diagnosis based on the symptoms provided by the user so far, if yes, we call the **reply\_diagnose** predicate to write the diagnosis to the user, if not, the **reply\_question** predicate is called to ask the user another question. The **response** predicate iteratively calls the **reply\_question** predicate is called to ask the user questions until the Sympathetic Doctor interactive Talking Box Prolog script is ready to make a diagnosis which makes the script runs automatically and iteratively.

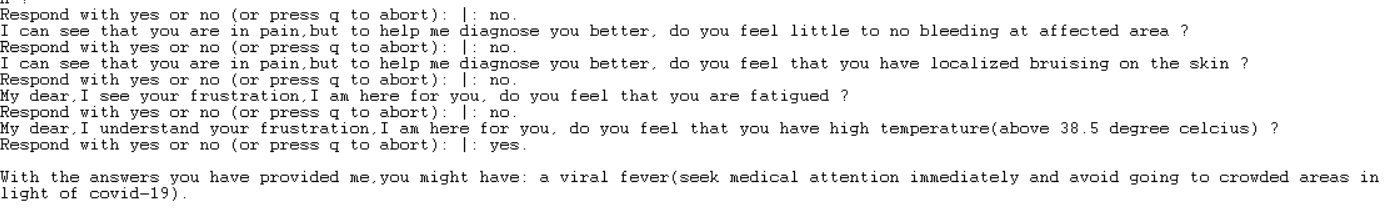
The **reply\_diagnose** predicate is used to write the diagnosis derived by the Sympathetic Doctor interactive Talking Box Prolog script onto the Prolog console for the user. The predicate first retrieves the possible illnesses diagnosed by the Prolog script by calling the **diagnose** predicate. Next, the sort predicate is used to sort the list of diagnosed illnesses and remove any duplicate diagnosis, we then call the **human\_diagnose** predicate to retrieve the human readable sentence string of the concatenated list of diagnosed illnesses which is then written onto the Prolog console.

# Demonstration of the Sympathetic Doctor interactive Talking Box

## Diagnosis

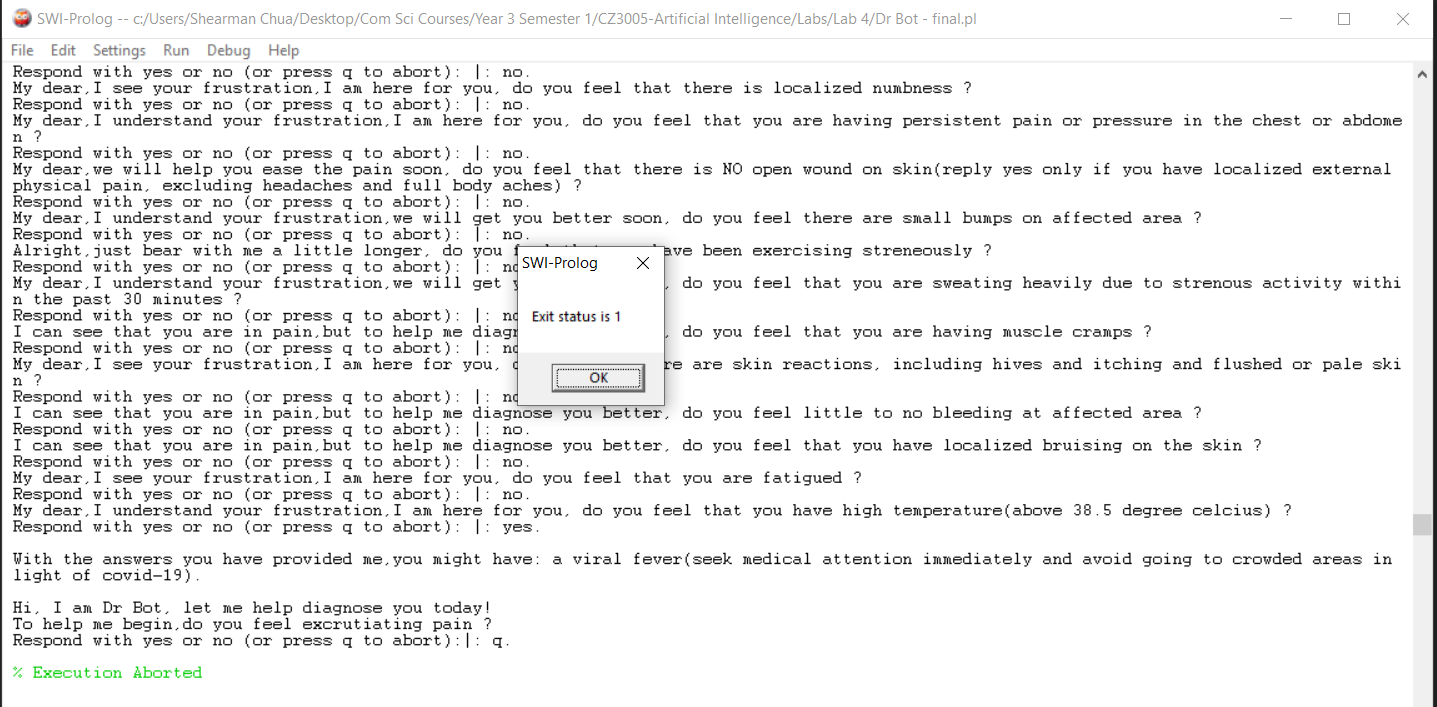


After asking the user questions, when **the ready\_to\_diagnose** predicate is set, the Sympathetic Doctor interactive Talking Box displays the diagnosis results on the Prolog console. In the example above, as the user is presenting flu like symptoms, those flu-related illnesses are diagnosed to the user. This is due to the fact that our Sympathetic Doctor interactive Talking Box implementation uses the elimination method to reduce the possible illnesses that the user may have to a small set and presents to the user the possible illnesses the user may have. After the diagnosis results are presented to the user, the script continues to run again and the user can get another diagnosis.



In the example above, as the user has accurately described his symptoms, the Sympathetic Doctor interactive Talking Box is able to give a single illness diagnosis to the user. Therefore, the more specific the symptoms the user presents, the lesser illnesses are being diagnosed to the user by the Sympathetic Doctor.

## Exiting the Sympathetic Doctor interactive Talking Box Script



To quit the Sympathetic Doctor interactive Talking Box Prolog script, the user just has to enter ‘q.’ to the Prolog console and the Prolog script will be exited.