Institute Of Computer Technology

B. Tech Computer Science & Technology

Sub: Artificial Intelligence (AI)

Practical - 02

Name: Panchal Darshan

Enrolment No.: 22162121007

Subject: AI

Branch: BDA

Batch: 63

Semester: 6

Date of Creation: 25/01/2026

Date of Submission: 28/01/2026

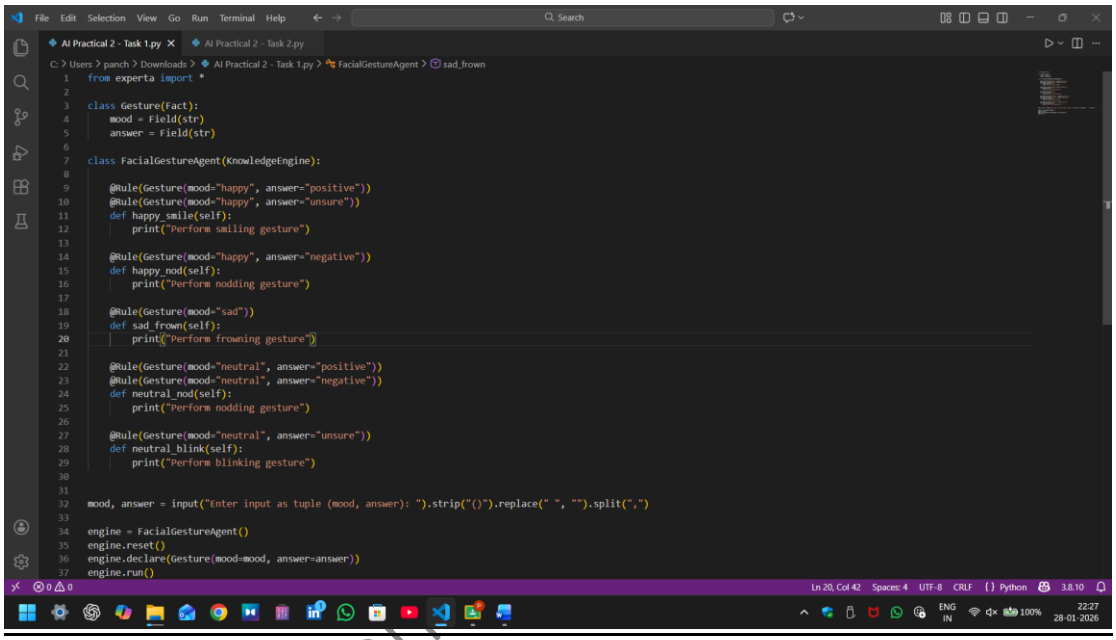
GitHub Link - Link

❖ **AIM - Develop the following two intelligent agents using the language of your choice. Define PEAS for both of them.**

➢ **I have used Experta Library to solve these two problems!!**

1. Create a facial gesture selection Agent.

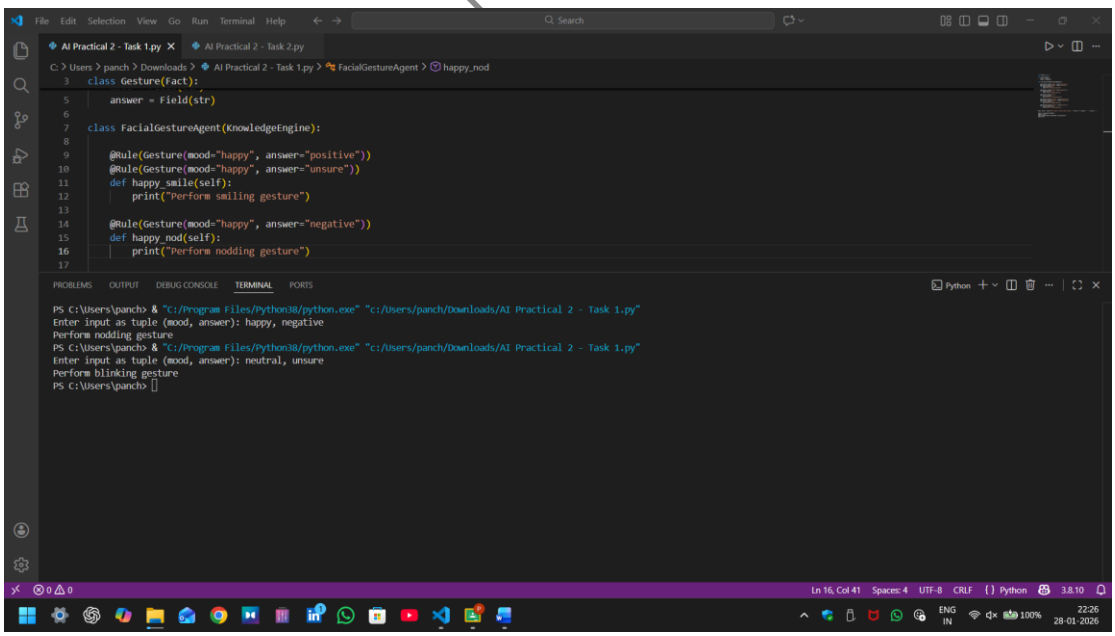
Code Input:



```
from experta import *

class Gesture(Fact):
    mood = Field(str)
    answer = Field(str)

class FacialGestureAgent(KnowledgeEngine):

    @Rule(Gesture(mood="happy", answer="positive"))
    @Rule(Gesture(mood="happy", answer="unsure"))
    def happy_smile(self):
        print("Perform smiling gesture")

    @Rule(Gesture(mood="happy", answer="negative"))
    def happy_nod(self):
        print("Perform nodding gesture")

    @Rule(Gesture(mood="sad"))
    def sad_frown(self):
        print("Perform frowning gesture")

    @Rule(Gesture(mood="neutral", answer="positive"))
    @Rule(Gesture(mood="neutral", answer="negative"))
    def neutral_nod(self):
        print("Perform nodding gesture")

    @Rule(Gesture(mood="neutral", answer="unsure"))
    def neutral_blink(self):
        print("Perform blinking gesture")


mood, answer = input("Enter input as tuple (mood, answer): ").strip("()").replace(" ", "").split(",")

engine = FacialGestureAgent()
engine.reset()
engine.declare(Gesture(mood=mood, answer=answer))
engine.run()
```

```
    @Rule(Gesture(mood="sad"))
    def sad_frown(self):
        print("Perform frowning gesture")

    @Rule(Gesture(mood="neutral", answer="positive"))
    @Rule(Gesture(mood="neutral", answer="negative"))
    def neutral_nod(self):
        print("Perform nodding gesture")

    @Rule(Gesture(mood="neutral", answer="unsure"))
    def neutral_blink(self):
        print("Perform blinking gesture")


mood, answer = input("Enter input as tuple (mood, answer): ").strip("()").replace(" ",
"").split(",")

engine = FacialGestureAgent()
engine.reset()
engine.declare(Gesture(mood=mood, answer=answer))
engine.run()
```


Code Output:



➢ PEAS Definition – Facial Gesture Selection Agent

1. P = Performance Measure:

- ❖ Correct selection of facial gesture based on mood and answer
- ❖ Natural and context-appropriate gesture output.
- ❖ Zero ambiguity in rule execution

2. E = Environment:

- ❖ Interactive, user-driven environment
- ❖ Non-real-time, software-based system
- ❖ Input provided as a tuple: (mood, answer)

3. A = Actuators:

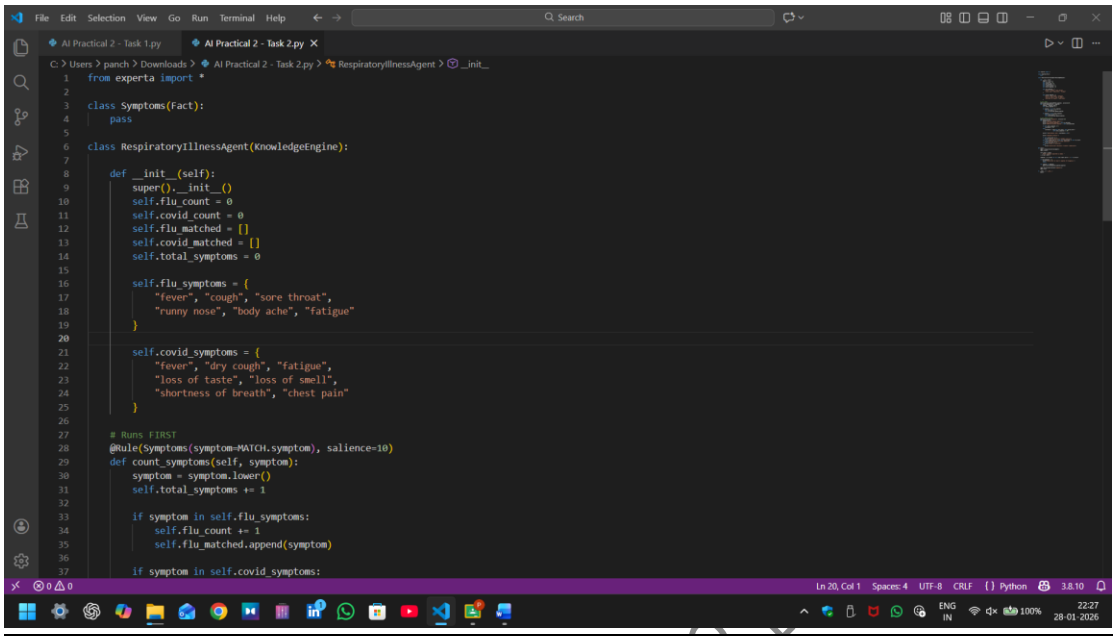
- ❖ Perform Smile
- ❖ Perform Nod
- ❖ Perform Blink
- ❖ Perform Frown

4. S = Sensors:

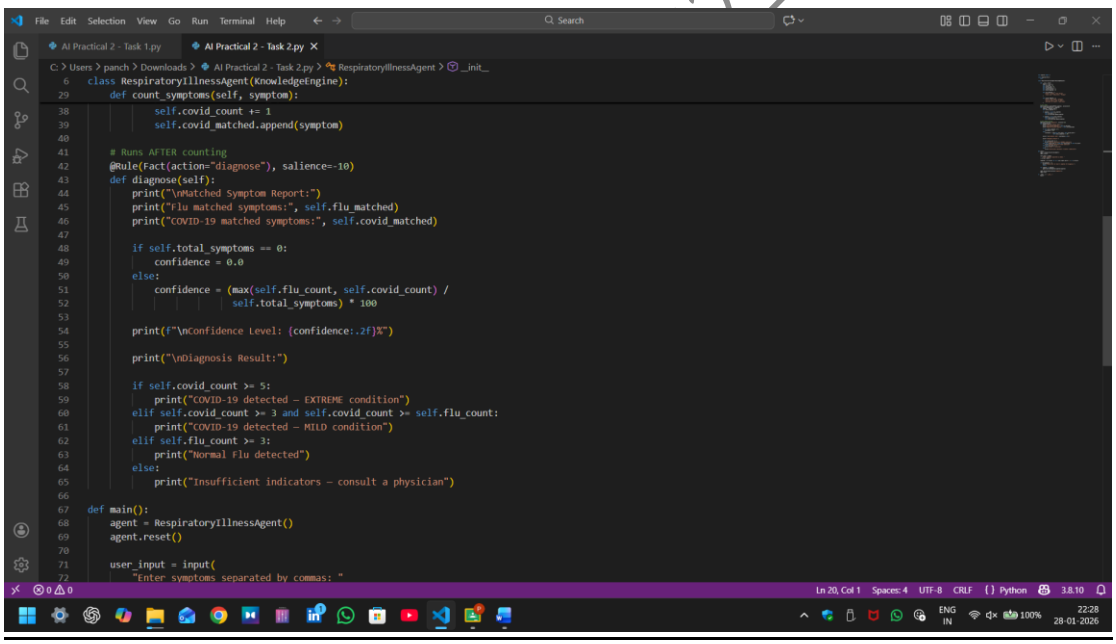
- ❖ Mood input: happy, sad, neutral
- ❖ Answer input: positive, negative, unsure

2. Create a Respiratory illness classification agent.

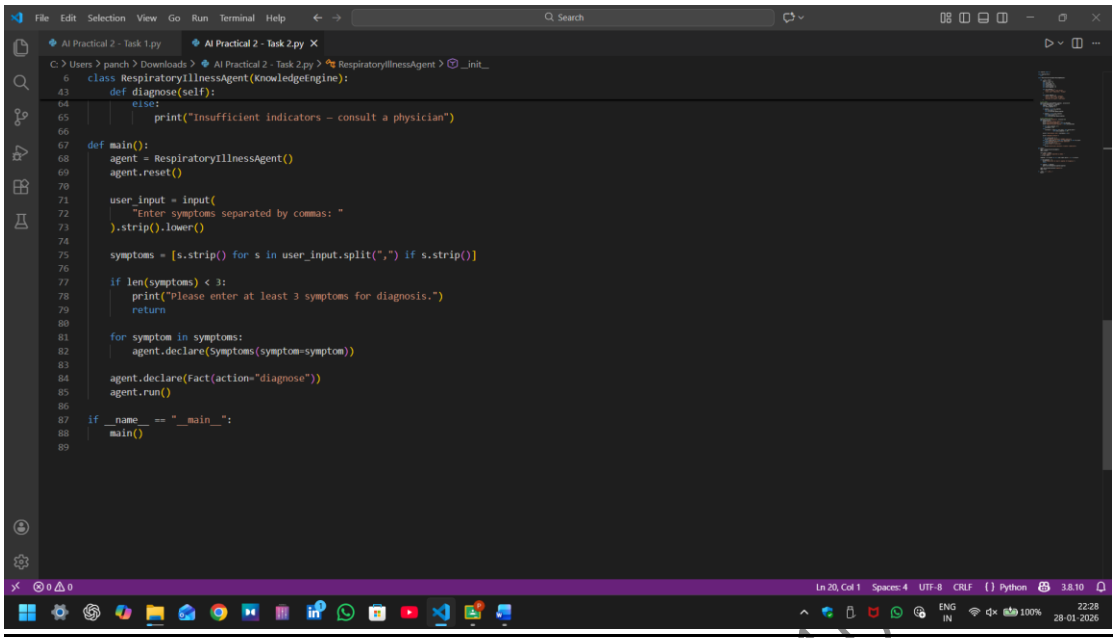
Code Input:



```
from experta import *

class Symptoms(Fact):
    pass

class RespiratoryIllnessAgent(KnowledgeEngine):

    def __init__(self):
        super().__init__()
        self.flu_count = 0
        self.covid_count = 0
        self.flu_matched = []
        self.covid_matched = []
        self.total_symptoms = 0

        self.flu_symptoms = {
            "fever", "cough", "sore throat",
            "runny nose", "body ache", "fatigue"
        }

        self.covid_symptoms = {
            "fever", "dry cough", "fatigue",
            "loss of taste", "loss of smell",
            "shortness of breath", "chest pain"
        }

    # Runs FIRST
    @Rule(Symptoms(symptom=MATCH.symptom), salience=10)
    def count_symptoms(self, symptom):
        symptom = symptom.lower()
        self.total_symptoms += 1

        if symptom in self.flu_symptoms:
            self.flu_count += 1
            self.flu_matched.append(symptom)

        if symptom in self.covid_symptoms:
```



```
class RespiratoryIllnessAgent(KnowledgeEngine):
    def count_symptoms(self, symptom):
            self.covid_count += 1
            self.covid_matched.append(symptom)

    # Runs AFTER counting
    @Rule(Fact(action="diagnose"), salience=-10)
    def diagnose(self):
        print("\nMatched Symptom Report:")
        print("Flu matched symptoms:", self.flu_matched)
        print("COVID-19 matched symptoms:", self.covid_matched)

        if self.total_symptoms == 0:
            confidence = 0.0
        else:
            confidence = (max(self.flu_count, self.covid_count) /
                          self.total_symptoms) * 100

        print(f"\nConfidence Level: {confidence:.2f}%")

        print("\nDiagnosis Result:")

        if self.covid_count >= 5:
            print("COVID-19 detected – EXTREME condition")
        elif self.covid_count >= 3 and self.covid_count >= self.flu_count:
            print("COVID-19 detected – MILD condition")
        elif self.flu_count >= 3:
            print("Normal Flu detected")
        else:
            print("Insufficient indicators – consult a physician")

def main():
    agent = RespiratoryIllnessAgent()
    agent.reset()

    user_input = input(
        "Enter symptoms separated by commas: "
```

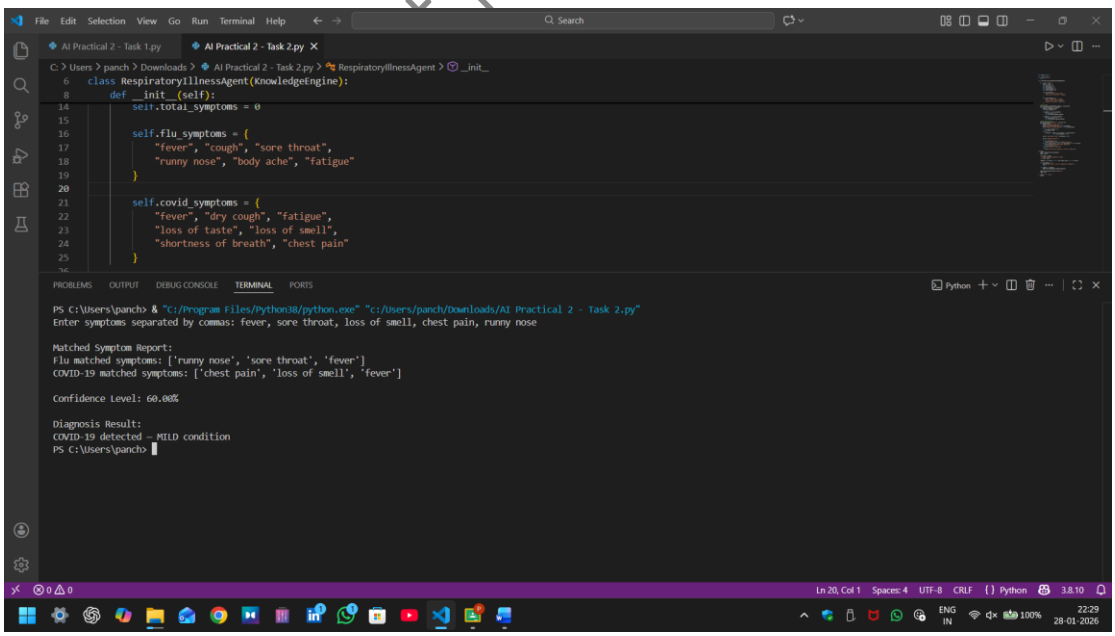



```
from experta import *

class Symptoms(Fact):
   pass

class RespiratoryIllnessAgent(KnowledgeEngine):

   def __init__(self):
      super().__init__()
      self.flu_count = 0
      self.covid_count = 0
      self.flu_matched = []
      self.covid_matched = []
      self.total_symptoms = 0

      self.flu_symptoms = {
         "fever", "cough", "sore throat",
         "runny nose", "body ache", "fatigue"
      }

      self.covid_symptoms = {
         "fever", "dry cough", "fatigue",
         "loss of taste", "loss of smell",
         "shortness of breath", "chest pain"
      }
```

```
    # Runs FIRST
    @Rule(Symptoms(symptom=MATCH.symptom), salience=10)
    def count_symptoms(self, symptom):
        symptom = symptom.lower()
        self.total_symptoms += 1

        if symptom in self.flu_symptoms:
            self.flu_count += 1
            self.flu_matched.append(symptom)

        if symptom in self.covid_symptoms:
            self.covid_count += 1
            self.covid_matched.append(symptom)

    # Runs AFTER counting
    @Rule(Fact(action="diagnose"), salience=-10)
    def diagnose(self):
        print("\nMatched Symptom Report:")
        print("Flu matched symptoms:", self.flu_matched)
        print("COVID-19 matched symptoms:", self.covid_matched)

        if self.total_symptoms == 0:
            confidence = 0.0
        else:
            confidence = (max(self.flu_count, self.covid_count) /
                          self.total_symptoms) * 100

        print(f"\nConfidence Level: {confidence:.2f}%")

        print("\nDiagnosis Result:")

        if self.covid_count >= 5:
            print("COVID-19 detected — EXTREME condition")
        elif self.covid_count >= 3 and self.covid_count >= self.flu_count:
            print("COVID-19 detected — MILD condition")
        elif self.flu_count >= 3:
            print("Normal Flu detected")
        else:
            print("Insufficient indicators — consult a physician")

def main():
    agent = RespiratoryIllnessAgent()
```

```
    agent.reset()

    user_input = input(
        "Enter symptoms separated by commas: "
    ).strip().lower()

    symptoms = [s.strip() for s in user_input.split(",") if s.strip()]

    if len(symptoms) < 3:
        print("Please enter at least 3 symptoms for diagnosis.")
        return

    for symptom in symptoms:
        agent.declare(Symptoms(symptom=symptom))

    agent.declare(Fact(action="diagnose"))
    agent.run()

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


Code Output:



➢ PEAS Definition – Respiratory Illness Classification Agent

1. P = Performance Measure:
   - ❖ Accurate classification between Normal Flu and COVID-19
   - ❖ Correct identification of COVID-19 severity (Mild or Extreme)
   - ❖ Prevention of diagnosis when fewer than 3 symptoms are provided
   - ❖ Correct comparison of Flu and COVID symptom counters
2. E = Environment:
   - ❖ User-interactive, non-clinical decision support environment.
   - ❖ Static and deterministic
   - ❖ Text-based input system
3. A = Actuators:
   - ❖ Display diagnosis result
   - ❖ Prompt user to enter additional symptoms when input is insufficient
   - ❖ Display advisory messages
4. S = Sensors:
   - ❖ User-entered symptom list
   - ❖ Count of symptoms matching Flu
   - ❖ Count of symptoms matching COVID-19

Thank You Sir, for your Time & Consideration.

Yours Sincerely,

Panchal Darshan

Enrolment Number: 22162121007