



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

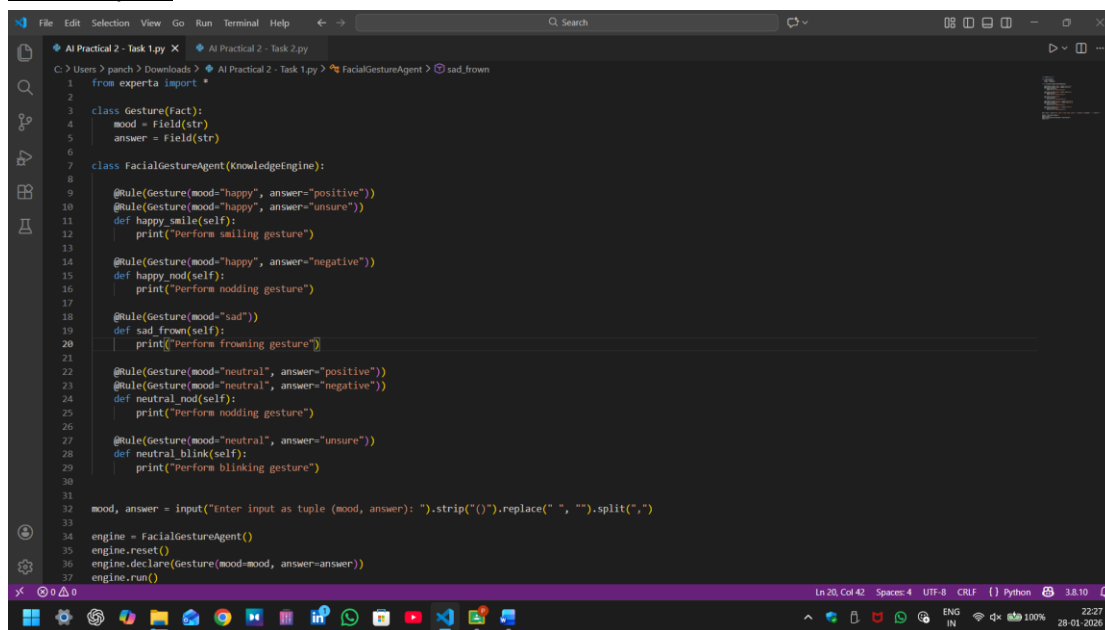
GitHub Link - [Link](#)

❖ **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:



```

1  from experta import *
2
3  class Gesture(Fact):
4      mood = Field(str)
5      answer = Field(str)
6
7  class FacialGestureAgent(KnowledgeEngine):
8
9      @Rule(Gesture(mood="happy", answer="positive"))
10     @Rule(Gesture(mood="happy", answer="unsure"))
11     def happy_smile(self):
12         print("Perform smiling gesture")
13
14     @Rule(Gesture(mood="happy", answer="negative"))
15     def happy_nod(self):
16         print("Perform nodding gesture")
17
18     @Rule(Gesture(mood="sad"))
19     def sad_frown(self):
20         print("Perform frowning gesture")
21
22     @Rule(Gesture(mood="neutral", answer="positive"))
23     @Rule(Gesture(mood="neutral", answer="negative"))
24     def neutral_nod(self):
25         print("Perform nodding gesture")
26
27     @Rule(Gesture(mood="neutral", answer="unsure"))
28     def neutral_blink(self):
29         print("Perform blinking gesture")
30
31
32 mood, answer = input("Enter input as tuple (mood, answer): ").strip(" ").replace(" ", "").split(",")
33
34 engine = FacialGestureAgent()
35 engine.reset()
36 engine.declare(Gesture(mood=mood, answer=answer))
37 engine.run()
  
```

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")
```

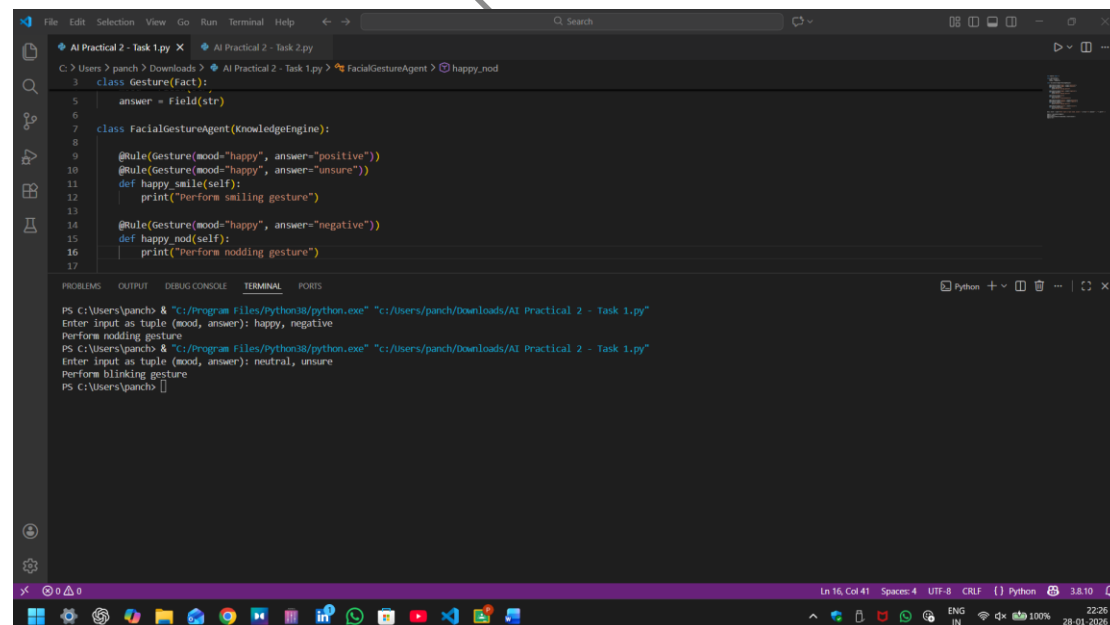
```
@Rule(Gesture(mood="neutral", answer="positive"))
@Rule(Gesture(mood="neutral", answer="negative"))
def neutral_nod(self):
    print("Perform nodding gesture")
```

```
@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()
```

Code Output:



The screenshot shows a Python IDE with a file named 'AI Practical 2 - Task 1.py'. The code in the editor defines a 'Gesture' class with a 'Fact' field and a 'FacialGestureAgent' class that uses a 'KnowledgeEngine'. The agent has three rules: one for 'happy' mood with 'positive' or 'negative' answers (performing a smile or nod), and one for 'neutral' mood with 'unsure' answer (performing a blink). The terminal output shows the execution of the code with the following inputs and outputs:

```
PS C:\Users\pancho> & "c:/Program Files/Python38/python.exe" "c:/Users/pancho/Downloads/AI Practical 2 - Task 1.py"
Enter input as tuple (mood, answer): happy, negative
Perform nodding gesture
PS C:\Users\pancho> & "c:/Program Files/Python38/python.exe" "c:/Users/pancho/Downloads/AI Practical 2 - Task 1.py"
Enter input as tuple (mood, answer): neutral, unsure
Perform blinking gesture
PS C:\Users\pancho>
```

➤ **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

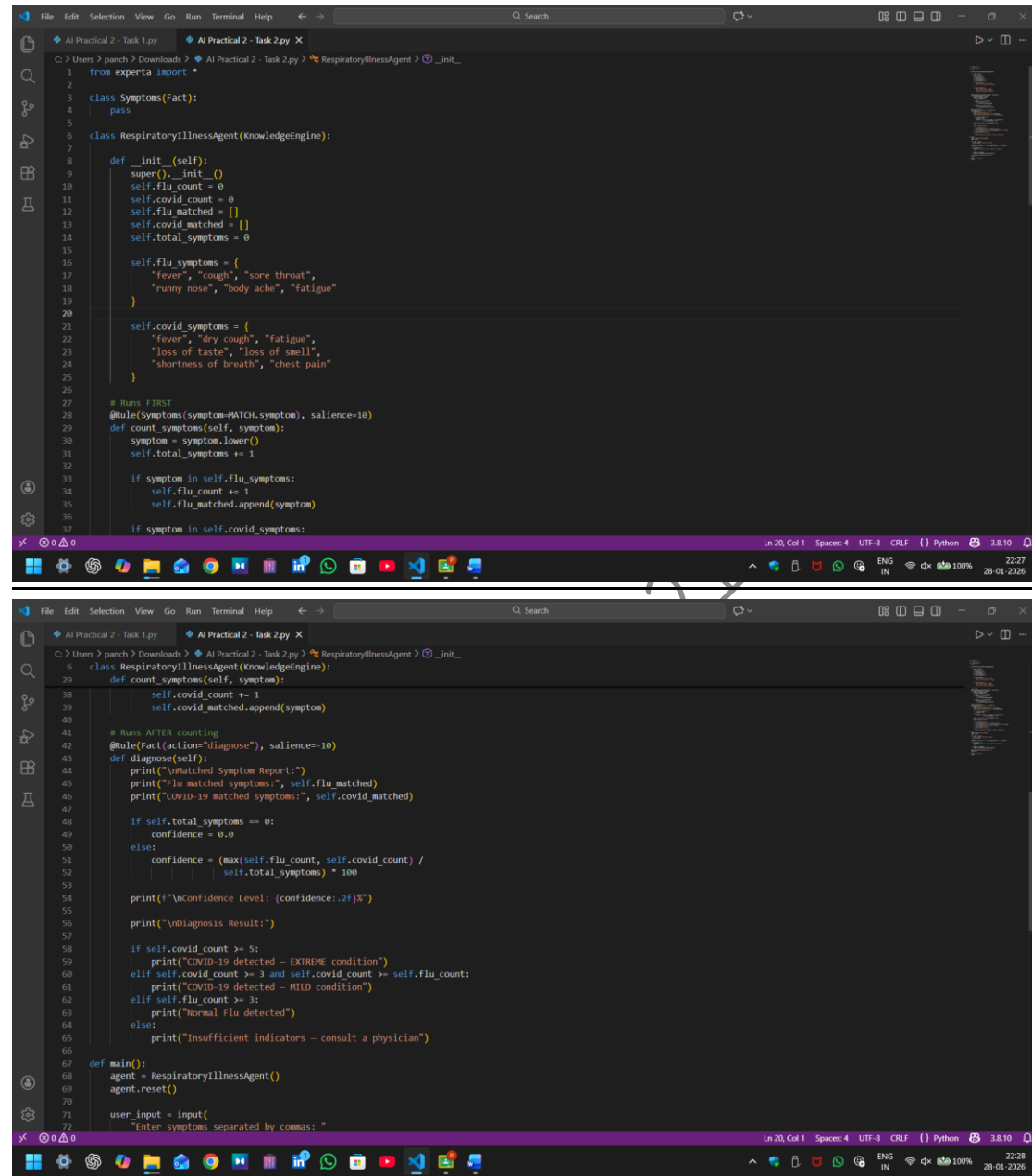
4. S = Sensors:

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

Enrolment No: 22162121007

2. Create a Respiratory illness classification agent.

Code Input:



```

1  from experta import *
2
3  class Symptoms(Fact):
4      pass
5
6  class RespiratoryIllnessAgent(KnowledgeEngine):
7
8      def __init__(self):
9          super().__init__()
10         self.flu_count = 0
11         self.covid_count = 0
12         self.flu_matched = []
13         self.covid_matched = []
14         self.total_symptoms = 0
15
16         self.flu_symptoms = {
17             "fever", "cough", "sore throat",
18             "runny nose", "body ache", "fatigue"
19         }
20
21         self.covid_symptoms = {
22             "fever", "dry cough", "fatigue",
23             "loss of taste", "loss of smell",
24             "shortness of breath", "chest pain"
25         }
26
27         # Runs FIRST
28         @Rule(Symptoms(symptom=MATCH.symptom), salience=10)
29         def count_symptoms(self, symptom):
30             symptom = symptom.lower()
31             self.total_symptoms += 1
32
33             if symptom in self.flu_symptoms:
34                 self.flu_count += 1
35                 self.flu_matched.append(symptom)
36
37             if symptom in self.covid_symptoms:
38                 self.covid_count += 1
39                 self.covid_matched.append(symptom)
40
41         # Runs AFTER counting
42         @Rule(Fact(action="diagnose"), salience=10)
43         def diagnose(self):
44             print("\nMatched Symptom Report:")
45             print("Flu matched symptoms:", self.flu_matched)
46             print("COVID-19 matched symptoms:", self.covid_matched)
47
48             if self.total_symptoms == 0:
49                 confidence = 0.0
50             else:
51                 confidence = (max(self.flu_count, self.covid_count) /
52                             self.total_symptoms) * 100
53
54             print(f"\nConfidence Level: {confidence:.2f}%")
55
56             print("\nDiagnosis Result:")
57
58             if self.covid_count >= 5:
59                 print("COVID-19 detected - EXTREME condition")
60             elif self.covid_count >= 3 and self.covid_count >= self.flu_count:
61                 print("COVID-19 detected - MILD condition")
62             elif self.flu_count >= 3:
63                 print("Normal Flu detected")
64             else:
65                 print("Insufficient indicators - consult a physician")
66
67     def main():
68         agent = RespiratoryIllnessAgent()
69         agent.reset()
70
71         user_input = input(
72             "Enter symptoms separated by commas: "

```

```

File Edit Selection View Go Run Terminal Help
C:\Users\punch> Downloads > AI Practical 2 - Task 1.py AI Practical 2 - Task 2.py RespiratoryIllnessAgent > __init__
6 class RespiratoryIllnessAgent(KnowledgeEngine):
43     def diagnose(self):
64         else:
65             print("Insufficient indicators - consult a physician")
66
67 def main():
68     agent = RespiratoryIllnessAgent()
69     agent.reset()
70
71     user_input = input(
72         "Enter symptoms separated by commas: "
73     ).strip().lower()
74
75     symptoms = [s.strip() for s in user_input.split(",") if s.strip()]
76
77     if len(symptoms) < 3:
78         print("Please enter at least 3 symptoms for diagnosis.")
79         return
80
81     for symptom in symptoms:
82         agent.declare(symptoms(symptom=symptom))
83
84     agent.declare(Fact(action="diagnose"))
85     agent.run()
86
87 if __name__ == "__main__":
88     main()
89
Ln 20, Col 1 Spaces: 4 UTF-8 CRLF Python 3.8.10
22:28 28-01-2026

```

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

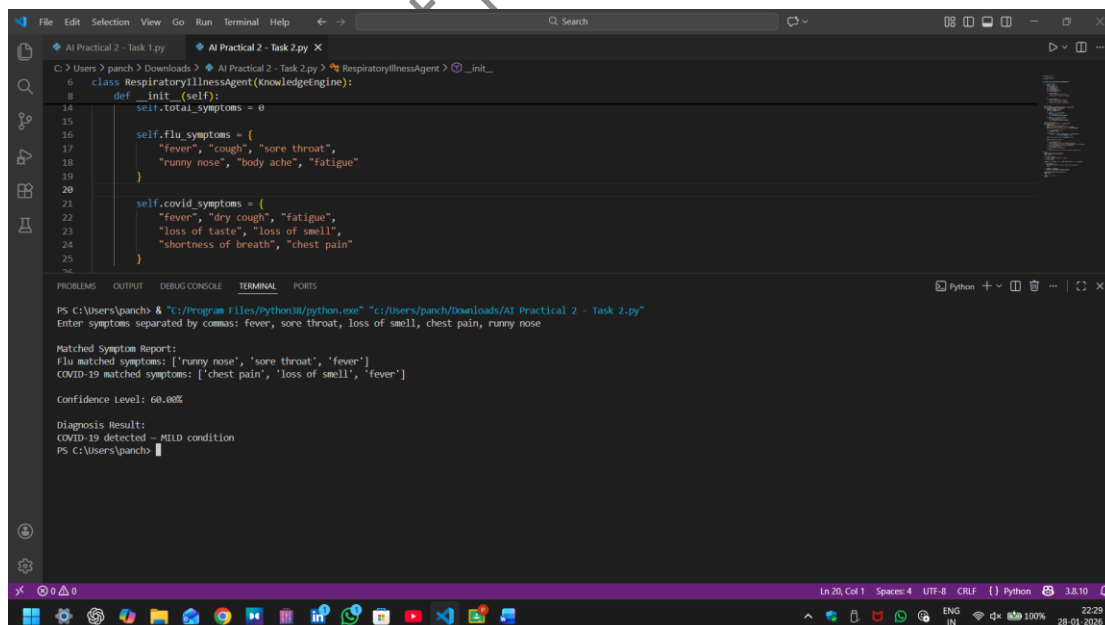
for symptom in symptoms:
    agent.declare(Symptoms(symptom=symptom))

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

if __name__ == "__main__":
    main()

```

Code Output:



The screenshot shows a Python IDE with a file named 'AI Practical 2 - Task 2.py'. The code defines a class 'RespiratoryIllnessAgent' with methods for declaring symptoms and running a diagnosis. The terminal output shows the user input 'fever, sore throat, loss of smell, chest pain, runny nose' and the resulting diagnosis report.

```

PS C:\Users\pancho> "C:/Program Files/Python38/python.exe" "C:/Users/pancho/Downloads/AI Practical 2 - Task 2.py"
Enter symptoms separated by commas: fever, sore throat, loss of smell, chest pain, runny nose

Matched Symptom Report:
Flu matched symptoms: ['runny nose', 'sore throat', 'fever']
COVID-19 matched symptoms: ['chest pain', 'loss of smell', 'fever']

Confidence Level: 60.00%

Diagnosis Result:
COVID-19 detected - MILD condition
PS C:\Users\pancho>

```


➤ **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