



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:

The screenshot shows a Jupyter Notebook interface with the following code:

```
File Edit Selection View Go Run Terminal Help ← → Q. Search
AI Practical 2 - Task 1.py X AI Practical 2 - Task 2.py
C:\Users\panch>Downloads> AI Practical 2 - Task 1.py > FacialGestureAgent > sad_frown
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 mood, answer = input("Enter input as tuple (mood, answer): ").strip("()").replace(" ", "").split(",")
32
33 engine = FacialGestureAgent()
34 engine.reset()
35 engine.declare(Gesture(mood=mood, answer=answer))
36 engine.run()
37
```

```
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 Jupyter Notebook interface with two open files: `AI Practical 2 - Task 1.py` and `AI Practical 2 - Task 2.py`. The terminal below shows the execution of the code from `Task 1.py`, which defines a `FacialGestureAgent` class with methods for smiling and nodding based on mood input.

```
File Edit Selection View Go Run Terminal Help ↺ → ○ Search

AI Practical 2 - Task 1.py × AI Practical 2 - Task 2.py
C:\Users\panch> Downloads > AI Practical 2 - Task 1.py > FacialGestureAgent > happy_nod
3   class Gesture(Fact):
4     answer = Field(str)
5
6   class FacialgestureAgent(KnowledgeEngine):
7
8     @rule(Gesture(mood="happy", answer="positive"))
9     @rule(Gesture(mood="happy", answer="unsure"))
10    def happy_smile(self):
11      print("Perform smiling gesture")
12
13    @rule(Gesture(mood="happy", answer="negative"))
14    def happy_nod(self):
15      print("Perform nodding gesture")
16
17

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS

PS C:\Users\panch> & "C:\Program Files\python38\python.exe" "c:/Users/panch/Downloads/AI Practical 2 - Task 1.py"
Enter input as tuple (mood, answer): happy, negative
Perform nodding gesture
PS C:\Users\panch> & "C:\Program Files\python38\python.exe" "c:/Users/panch/Downloads/AI Practical 2 - Task 1.py"
Enter input as tuple (mood, answer): neutral, unsure
Perform blinking gesture
PS C:\Users\panch> ]
```

➤ **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:

The screenshot shows the PyCharm IDE interface with the following details:

- File Menu:** File, Edit, Selection, View, Go, Run, Terminal, Help.
- Search Bar:** Q: Search.
- Toolbars:** Standard toolbar with icons for file operations.
- Code Editor:** The main window displays Python code for an AI practical task. The code defines a `Symptoms` class and a `RespiratoryIllnessAgent` class that inherits from `KnowledgeEngine`. It includes lists of symptoms for flu and COVID-19, and methods for counting symptoms and matching them against input.
- Status Bar:** Shows line 20, column 4, spaces: 4, encoding: UTF-8, CRLF, Python, 3.8.10, and a timestamp: 28-01-2026.
- Bottom Icons:** Taskbar icons for various applications like File Explorer, Task Manager, and Start.

```
File Edit Selection View Go Run Terminal Help < > Q: Search
AI Practical 2 - Task 2.py ×
C:\Users\panch\Downloads> AI Practical 2 - Task 2.py
1  from experts 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
39
40
41
42
43
44
45
46
47
48
49
50
51
52
53
54
55
56
57
58
59
59
```

```
File Edit Selection View Go Run Terminal Help < > Q. Search

AI Practical 2 - Task 1.py AI Practical 2 - Task 2.py x

C:\> Users\panch> Downloads > AI Practical 2 - Task2.py > RespiratoryIllnessAgent > __init__.py

1 class RespiratoryIllnessAgent(KnowledgeEngine):
2     def count_symptoms(self, symptom):
3         self.flu_count += 1
4         self.covid_matched.append(symptom)
5
6     # Runs AFTER counting
7     @Rule(Fact(action="diagnose"), salience=-10)
8     def diagnose(self):
9         print("\nMatched Symptom Report:")
10        print("Flu matched symptoms: ", self.flu.matched)
11        print("COVID-19 matched symptoms: ", self.covid.matched)
12
13        if self.total_symptoms == 0:
14            confidence = 0.0
15        else:
16            confidence = (max(self.flu_count, self.covid_count) /
17                           self.total_symptoms) * 100
18
19        print(f"\nConfidence Level: {confidence:.2f}%")
20
21        print("\nDiagnosis Result:")
22
23        if self.covid_count >= 5:
24            print("COVID-19 detected - EXTREME condition")
25        elif self.covid_count >= 3 and self.covid_count <= self.flu_count:
26            print("COVID-19 Detected - MILD condition")
27        elif self.flu_count >= 3:
28            print("Normal Flu detected")
29        else:
30            print("Insufficient indicators - consult a physician")
31
32    def main():
33        agent = RespiratoryIllnessAgent()
34        agent.reset()
35
36        user_input = input(
37            "Enter symptoms separated by commas: "
38
39
40
41
42
43
44
45
46
47
48
49
50
51
52
53
54
55
56
57
58
59
60
61
62
63
64
65
66
67
68
69
70
71
72
73
```

```

File Edit Selection View Go Run Terminal Help ↻ →
C > Users > panch > Downloads > AI Practical 2 - Task 2.py > RespiratoryIllnessAgent > __init__.py
43     def diagnose(self):
44         else:
45             print("Insufficient indicators - consult a physician")
46
47     def main():
48         agent = RespiratoryIllnessAgent()
49         agent.reset()
50
51         user_input = input(
52             "Enter symptoms separated by commas: "
53         ).strip().lower()
54
55         symptoms = [s.strip() for s in user_input.split(",") if s.strip()]
56
57         if len(symptoms) < 3:
58             print("Please enter at least 3 symptoms for diagnosis.")
59             return
60
61         for symptom in symptoms:
62             agent.declare(Symptoms(symptom=symptom))
63
64         agent.declare(Fact(action="diagnose"))
65         agent.run()
66
67     if __name__ == "__main__":
68     main()

```

Ln 20, Col 1 Spaces:4 UTF-8 CRLF Python 3.8.10 22:28
ENG IN 100% 28-01-2026

```

from experts 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