📘 AI Lab – Week 5

Focus: AI in Python by ***Azeem Aslam***

File: `AI_Lab5_YourName_INPUT.ipynb`

Each task now takes **user input** from the keyboard (`input()`), as requested. Press **Run** on a cell and follow the prompts.

What you'll practice today

- Writing clean **functions** and reusing them
- **If/elif** rules to build tiny AI-like **expert systems**
- Simple **keyword intent classification**
- A mini **finite-state machine (FSM)** using Python
- A tiny **rule-based recommender** with scoring

> This lab continues Weeks 3–4 ideas (lists, dictionaries, functions, rule-based logic, NumPy-lite) but stays focused on **AI-in-Python**, not Data Science.

🧭 Instructions

- Run a cell and **type your answers** in the input prompts.
- Type **exit** (or empty input where mentioned) to stop a loop.
- Keep your code **commented** and **readable**.

1. Open **Anaconda** → **Jupyter Notebook**.
2. Save this notebook as **`AI_Lab5_YourName.ipynb`** .
3. For each task:
   - Read the **Description** (markdown).
   - Run the **starter** code.
   - Then complete the **TODO** or review the **Solution (example)**.
4. **Comment** your code using `# ...` to explain steps.
5. Show your work in class (**no email/WhatsApp** submissions unless told).

🎯 Objectives

- Practice AI-like **decision-making** with functions and dictionaries.
- Implement a basic **intent classifier** using **keywords**.
- Build a tiny **expert system** using readable rule blocks.
- Simulate **states & transitions** (FSM) with a simple loop.
- Combine everything into a small **AI Help Desk**.

🔹 Task 1 — Mini Intent Classifier (Keyword Rules)

Goal: Classify a user message into simple intents like: `greeting`, `goodbye`, `thanks`, `food`, `study`, `unknown`.

What you learn:

- Normalize and scan text
- Keyword-to-intent mapping
- Priority ordering (which rule fires first)

Stop: type `exit` to end.

In [ ]:
```python
# --- Task 1 (Interactive) ---
intent_keywords = {
    "greeting": ["hi", "hello", "salam", "assalam", "hey"],
    "goodbye":  ["bye", "goodbye", "see you", "take care"],
    "thanks":   ["thanks", "thank you", "shukriya"],
    "food":     ["eat", "restaurant", "food", "lunch", "dinner"],
    "study":    ["study", "exam", "assignment", "quiz", "lecture"]
}

def classify_intent(text):
    """Return an intent label for the given text using keyword rules.
    Priority: greeting > goodbye > thanks > food > study > unknown
    """
    t = text.lower().strip()
    for intent in ["greeting", "goodbye", "thanks", "food", "study"]:
        for kw in intent_keywords[intent]:
            if kw in t:
                return intent
    return "unknown"

print("Type messages (type 'exit' to stop):")
while True:
    msg = input("You: ").strip()
    if msg.lower() == "exit":
        print("Stopped Task 1.")
        break
    print("Intent:", classify_intent(msg))
```

🔹 Task 2 — Tiny Expert System: Scholarship Eligibility

Goal: Ask user profile inputs and decide: `Full Scholarship`, `Partial Scholarship`, or `Not Eligible`.

Rules (example):

- If **CGPA ≥ 3.7** and **family_income ≤ 60,000 PKR** → `Full Scholarship`
- Else if **CGPA ≥ 3.0** and **family_income ≤ 120,000 PKR** → `Partial Scholarship`
- Else if **CGPA ≥ 3.5** and **achievements include 'national'** → `Partial Scholarship`

- Else → `Not Eligible`

What you learn: Writing readable **if/elif** blocks as expert rules.

In [ ]:
```python
# --- Task 2 (Interactive) ---
def decide_scholarship(cgpa, family_income, achievements):
    ach = [a.strip().lower() for a in achievements]
    has_national = any("national" in a for a in ach)

    if cgpa >= 3.7 and family_income <= 60000:
        return "Full Scholarship"
    elif cgpa >= 3.0 and family_income <= 120000:
        return "Partial Scholarship"
    elif cgpa >= 3.5 and has_national:
        return "Partial Scholarship"
    else:
        return "Not Eligible"

def read_float(prompt, lo=None, hi=None):
    while True:
        try:
            x = float(input(prompt))
            if lo is not None and x < lo:
                print(f"Value must be >= {lo}."); continue
            if hi is not None and x > hi:
                print(f"Value must be <= {hi}."); continue
            return x
        except ValueError:
            print("Please enter a number.")

def read_int(prompt, lo=None, hi=None):
    while True:
        try:
            x = int(input(prompt))
            if lo is not None and x < lo:
                print(f"Value must be >= {lo}."); continue
            if hi is not None and x > hi:
                print(f"Value must be <= {hi}."); continue
            return x
        except ValueError:
            print("Please enter an integer.")

while True:
    print("\nEnter candidate info (or type 'exit' to stop)")
    raw = input("Continue? (yes/exit): ").strip().lower()
    if raw == "exit":
        print("Stopped Task 2.")
        break

    cgpa = read_float("CGPA (0.0-4.0): ", 0.0, 4.0)
    income = read_int("Family income PKR (per month): ", 0)
    ach_str = input("Achievements (comma-separated): ").strip()
    achievements = [s for s in ach_str.split(",")] if ach_str else []

    decision = decide_scholarship(cgpa, income, achievements)
    print(f"Decision: {decision}")
```

◆ Task 3 — Rule-Based Recommender (Weighted Scoring)

Goal: Recommend an activity (`'read'` , `'walk'` , `'nap'` , `'music'`) based on how the user feels and the time of day.

Idea: Compute a simple **score** per activity using rules. Highest score wins.

Hints:

- Feelings: `tired` , `stressed` , `bored` , `energetic`
- Time: `morning` , `afternoon` , `evening` , `night`

In [ ]:

```
# --- Task 3 (Interactive) ---
def activity_recommender(feeling, time_of_day):
    feeling = feeling.lower().strip()
    time_of_day = time_of_day.lower().strip()
    scores = { "read": 0, "walk": 0, "nap": 0, "music": 0 }

    # Based on feeling
    if feeling == "tired":
        scores["nap"] += 3; scores["music"] += 1
    elif feeling == "stressed":
        scores["walk"] += 2; scores["music"] += 2
    elif feeling == "bored":
        scores["read"] += 2; scores["walk"] += 1
    elif feeling == "energetic":
        scores["walk"] += 3; scores["read"] += 1

    # Based on time
    if time_of_day in ("morning", "afternoon"):
        scores["walk"] += 1; scores["read"] += 1
    elif time_of_day in ("evening",):
        scores["music"] += 2; scores["walk"] += 1
    elif time_of_day in ("night",):
        scores["nap"] += 2; scores["read"] += 1

    best = max(scores, key=scores.get)
    return best, scores

def read_choice(prompt, choices):
    low = [c.lower() for c in choices]
    while True:
        s = input(f"{prompt} {choices}: ").strip().lower()
        if s in low:
            return s
        print(f"Please choose one of {choices}.")

while True:
    print("\nRecommend an activity (type 'exit' to stop)")
    nxt = input("Continue? (yes/exit): ").strip().lower()
    if nxt == "exit":
        print("Stopped Task 3.")
        break
    feeling = read_choice("Your feeling?", ["tired","stressed","bored","energeti
    time_of_day = read_choice("Time of day?", ["morning","afternoon","evening","
    rec, sc = activity_recommender(feeling, time_of_day)
    print("Recommendation:", rec, "| Scores:", sc)
```

🔹 Task 4 — Mini Finite-State Machine (FSM): Traffic Light

Goal: Simulate a traffic light cycling through states with delays.

States: `RED → GREEN → YELLOW → RED → ...`

What you learn: Represent states with a dictionary and simulate transitions.

In [ ]:

```
# --- Task 4 (Interactive) ---
import time

fsm = {
    "RED":    {"next": "GREEN",  "duration": 1},
    "GREEN":  {"next": "YELLOW", "duration": 1},
    "YELLOW": {"next": "RED",    "duration": 1},
}

def run_traffic_light(cycles=2, real_delay=False):
    state = "RED"
    history = []
    for _ in range(cycles * len(fsm)):
        info = fsm[state]
        print(f"State: {state} (wait {info['duration']}s)")
        history.append(state)
        time.sleep(info["duration"] if real_delay else 0)
        state = info["next"]
    return history

while True:
    print("\nTraffic light simulation (type 'exit' to stop)")
    cont = input("Continue? (yes/exit): ").strip().lower()
    if cont == "exit":
        print("Stopped Task 4.")
        break
    while True:
        try:
            cycles = int(input("Number of cycles (e.g., 2): "))
            if cycles <= 0:
                print("Enter a positive integer."); continue
            break
        except ValueError:
            print("Please enter an integer.")
    delay_ans = input("Use real delays? (y/n): ").strip().lower()
    history = run_traffic_light(cycles, real_delay=(delay_ans == 'y'))
    print("Visited states:", history)
```

🔹 Task 5 — Robust Input Helpers (Re-usable Functions)

Goal: Write tiny helper functions that safely read and validate user input.

Functions to write:

- `read_nonempty(prompt)` → returns a nonempty string
- `read_choice(prompt, choices)` → returns a validated choice from a list
- `read_float_range(prompt, lo, hi)` → returns a float within `[lo, hi]`

Then **use them** to ask a user for: **name**, **preferred activity**, **CGPA**.

In [ ]:
```python
# --- Task 5 (Interactive) ---
def read_nonempty(prompt):
    while True:
        s = input(prompt).strip()
        if s:
            return s
        print("Input cannot be empty, try again.")

def read_choice(prompt, choices):
    choices_lower = [c.lower() for c in choices]
    while True:
        s = input(f"{prompt} {choices}: ").strip().lower()
        if s in choices_lower:
            return s
        print(f"Please enter one of {choices}.")

def read_float_range(prompt, lo, hi):
    while True:
        try:
            x = float(input(prompt))
            if lo <= x <= hi:
                return x
            else:
                print(f"Value must be between {lo} and {hi}.")
        except ValueError:
            print("Please enter a number.")

print("Fill the form below:")
name = read_nonempty("Your name: ")
activity = read_choice("Preferred activity?", ["read","walk","nap","music"])
cgpa = read_float_range("Your CGPA (0.0-4.0): ", 0.0, 4.0)
print(f"Summary -> Name: {name}, Activity: {activity}, CGPA: {cgpa}")
```

✅ Wrap-Up Challenge — AI Help Desk (Everything Together)

Goal: Build a tiny interactive helper that:

1. Reads a user message.
2. Classifies intent with **Task 1**.
3. If intent is `study`, suggest an activity using **Task 3**.
4. If intent is `greeting` or `thanks` or `goodbye`, reply politely.

5. Keep conversation history (list) and show a **summary** at the end.

> Tip: Keep it **simple** and **readable**. You may reuse functions from above.

In [ ]:
```python
# --- Wrap-Up (Interactive) ---
history = []

def bot_reply(user_msg):
    intent = classify_intent(user_msg)
    if intent == "greeting":
        reply = "Hello! How can I help you today?"
    elif intent == "thanks":
        reply = "You're welcome! Happy to help."
    elif intent == "goodbye":
        reply = "Goodbye! See you soon."
    elif intent == "study":
        rec, _ = activity_recommender("stressed", "evening")
        reply = f"Study tip: set a 25-minute focus timer. Also try a quick {rec}
    elif intent == "food":
        reply = "Try a light, healthy meal so your energy stays stable while stu
    else:
        reply = "I'm not sure yet, but I can help with study tips, greetings, fo
    return intent, reply

print("Chat with AI Help Desk (type 'exit' to finish).")
while True:
    u = input("You: ").strip()
    if u.lower() == "exit":
        print("Session ended. Summary below.")
        break
    it, resp = bot_reply(u)
    history.append((u, it, resp))
    print("Bot:", resp)

print("\nConversation Summary:")
for i, (u, it, r) in enumerate(history, 1):
    print(f"{i}. Intent={it:8s} | User='{u}' | Reply='{r}'")
```

📥 Submission Checklist

- Saved as `AI_Lab5_YourName.ipynb`
- Each task has:
  - Clear **Description** (markdown)
  - **Commented** code
  - A demo **output**
- Be ready to explain your code in class.