# LABORATORY EXERCISE 3

# RULE-BASED INFERENCE

**Learning Objectives**

By the end of this laboratory exercise, students should be able to:

* Understand the fundamental concepts of rule-based systems, including facts, rules, and inference engines.
* Implement a simple forward-chaining inference mechanism in Python.
* Represent knowledge formally using logical rules and facts.
* Analyze the inference process and trace the derivation of conclusions.

**Prerequisite student experiences and knowledge**

Before starting this exercise, students should have:

* Basic knowledge of graph theory (nodes, edges, weights).
* Familiarity with Python programming (lists, dictionaries, classes).
* Understanding of heuristic functions and admissible heuristics.

**Background**

Rule-based inference is a core symbolic Artificial Intelligence (AI) method for knowledge representation and reasoning. A rule-based system consists of a knowledge base (a set of facts and rules) and an inference engine that applies logical rules to the known facts to derive new facts (a process called forward chaining) or to check if a goal is supported by the facts (backward chaining). This approach is powerful for modeling expert knowledge in well-defined domains, such as diagnostic systems, classification problems, and procedural configurators.

**Materials/Resources**

* A PC with a Python 3.x interpreter installed.
* A code editor (e.g., VS Code, PyCharm, or IDLE).
* Internet access for research (optional).
* **Version Control: Git and GitHub.**

**Laboratory Activity**

**Instruction**

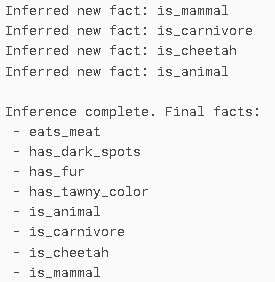
1. You will implement a simple forward-chaining inference engine.
2. The system will use a knowledge base defined within the code, containing:
   * facts: A set of initial true statements (e.g., `"has\_fur"`, `"barks"`).
   * rules: A list of IF-THEN rules. The `IF` part (antecedent) can be a single condition or multiple conditions joined with `AND`. The `THEN` part (consequent) is a single conclusion.
3. The engine should iterate through the rules, and for any rule whose entire antecedent is satisfied by the current facts, it should add the consequent to the facts set. This process should repeat until no new facts can be added (reaching a fixed point).
4. Run the engine and print the final set of inferred facts.

**Started Cide Skeleton:**



Output / Results

* Example Output for Question 1:



* Document the implemented code, screenshots all.
* Link to the GitHub repository.

**QUESTIONS:**

1. What is the conclusion after running the provided code with the initial facts (**has\_fur, eats\_meat, has\_tawny\_color, has\_dark\_spots**)? List all inferred facts.

The inferred facts are:   
Inferred new fact: is\_mammal

Inferred new fact: is\_carnivore

Inferred new fact: is\_cheetah

Inferred new fact: is\_animal

1. Modify the initial facts to represent a tiger. What facts do you need to add or change? Show your code modification and the new output.

system.add\_fact("has\_fur")

system.add\_fact("eats\_meat")

system.add\_fact("has\_tawny\_color")

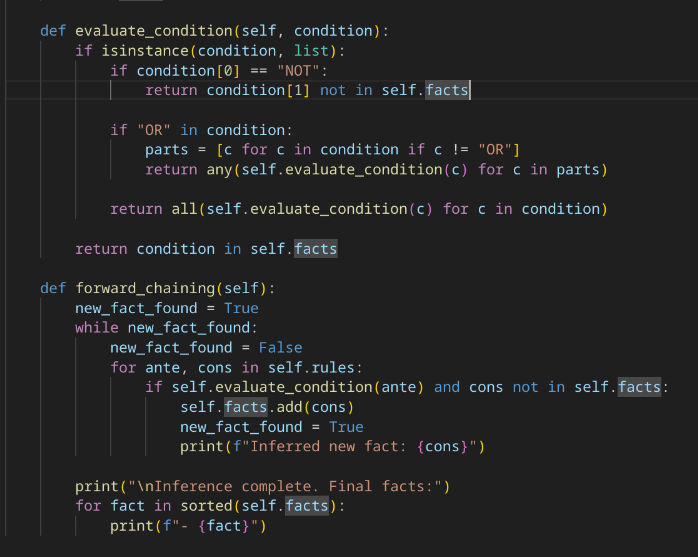
system.add\_fact("has\_black\_stripes")   
  
I changed system.add\_fact(“has\_dark\_spots”) to system.add\_fact(“has\_black\_stripes”)  
so the inferred facts became:  
Inferred new fact: is\_mammal

Inferred new fact: is\_carnivore

Inferred new fact: is\_tiger

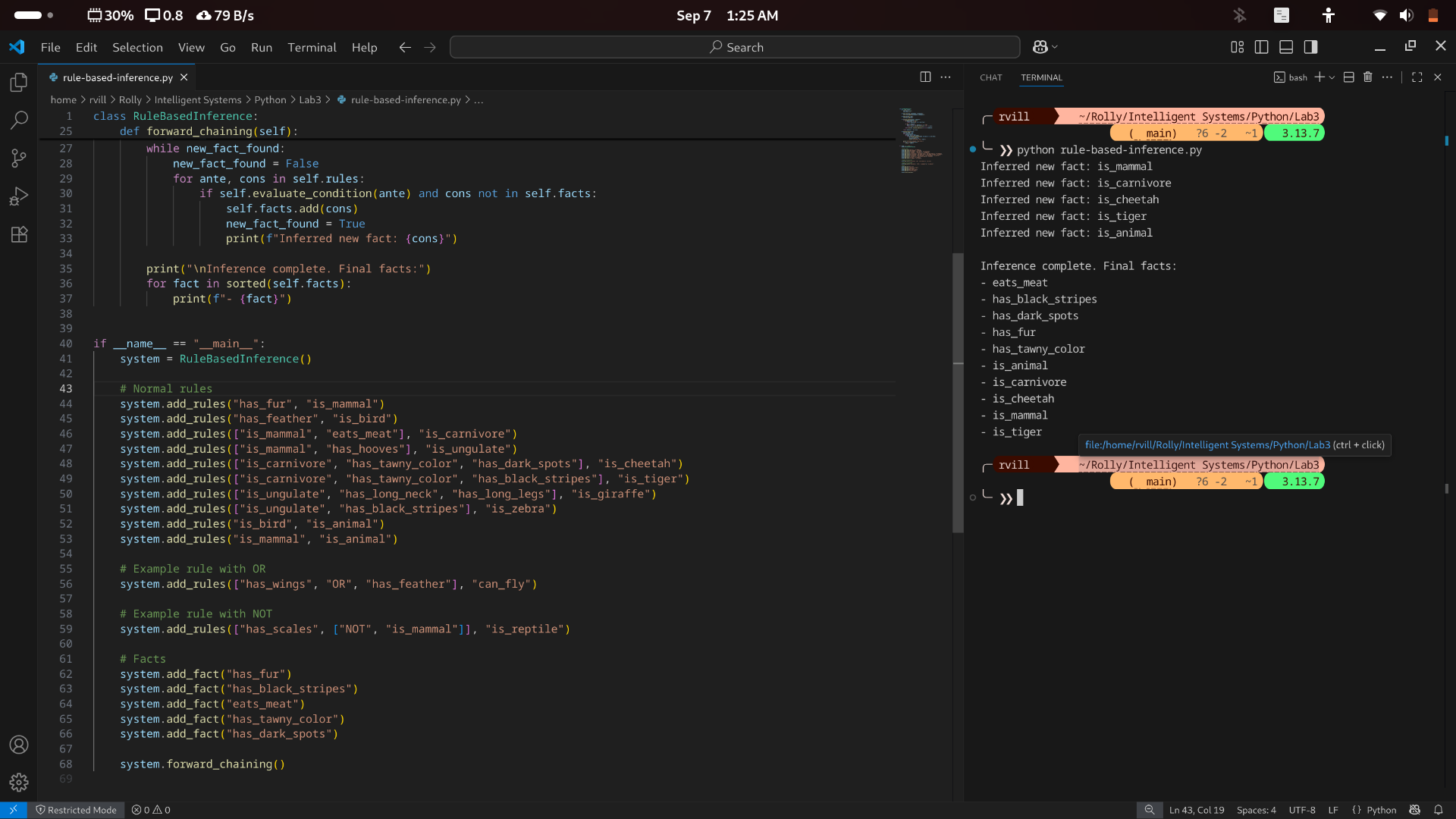
Inferred new fact: is\_animal

1. The current system cannot handle OR conditions or negation (NOT). How would adding these features increase the expressiveness of the knowledge base? Propose a syntax change to the **add\_rule** method to support one of these (e.g., **["has\_wings", OR, "can\_fly"]** or **[NOT, "has\_scales"]**).

****

By the use of AI , I ask it to add a feature that can handle OR and NOT conditions to increase its expressiveness of the knowledge base. How does it work and what is its significance to its performance? By adding a feature that can handle an OR and NOT conditions, it can improve complex and realistic reasoning without needing a huge number of extra rules. What about its process? If the condition is just a fact (like "has\_fur"), it just check if that fact exists in the knowledge base. If the condition is a list that starts with NOT, it checks that the following fact is **not** in the knowledge base, if it is in the list it will return false. If the condition is a list that has “OR”, it checks if **at least one** of the listed conditions is true. Lastly, if the condition is a list without NOT or OR, it treats it as an **AND** condition, meaning **all** items in the list must be true.

**Output / Results**



**Conclusion**

The **RuleBasedInference** system demonstrates how facts and rules can be combined to automatically infer new knowledge through forward chaining. By starting with a small set of initial facts, the system applies rules repeatedly until no new facts can be added. Originally, it could only handle **AND** conditions, which limited its reasoning to situations where all facts must be true. With the addition of **OR** and **NOT**, the system becomes far more expressive: it can now represent alternatives, handle exceptions, and capture more realistic reasoning patterns. This makes the knowledge base more powerful, compact, and closer to how human decision-making works. Overall, the system shows how rule-based inference engines can form the foundation of simple expert systems that mimic logical reasoning of a human.