Explanation for the code:

The folder has 5 python files , among then one file is used for declaring the AI knowledge which is logic.py and rest of them uses the logic.py to get a solution for a logical problem like correct matching with two different types of things.  
  
**Logic.py:**

This code is about creating a framework for symbolic logic, where sentences (logical statements) can be combined and evaluated. It represents logical statements in a structured way, such as symbols (like P or Q), and then combines them with logical operations like NOT, AND, OR, IMPLIES (implication), and IFF (biconditional). These operations let you build complex logical expressions and then check whether these expressions are true or false under different conditions.

Here’s a breakdown of the main components in the code:

**Basic Structure:** The Sentence class is an abstract base class. It doesn't do much on its own but provides the structure for other types of sentences. Each type of sentence—like Symbol, Not, And, Or, Implication, and Biconditional—inherits from Sentence and represents a specific kind of logical operation. **For example:**

Symbol represents a single logical variable like P or Q.

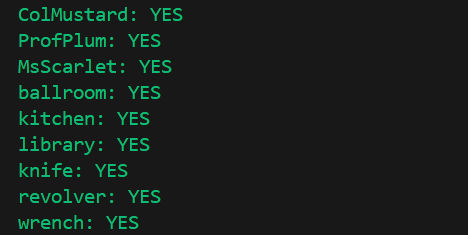
Not represents the negation (opposite) of another sentence.

And, Or, Implication, and Biconditional represent their respective logical operations.

**Evaluating Sentences:** Each class has an evaluate method, which checks if the sentence is true or false based on a provided "model" (a dictionary where each symbol is assigned a true/false value). For instance, if P is True in the model, then evaluating P would return True. Compound statements like And or Or evaluate all their parts. For example, an And sentence will only be True if all parts of it are True.

**Model Checking:** The model\_check function checks if a given knowledge base (a collection of known true statements) entails a query (another statement). It uses recursive logic to try every possible way of assigning true/false values to all symbols in the knowledge base and query. If the knowledge base implies the truth of the query in every possible model, the function returns True.

In short, this code is like a small logic engine that lets you define logical statements, combine them, and test if certain statements logically follow from others. This structure is very helpful in areas like artificial intelligence, where we need to simulate reasoning by checking if certain conclusions follow from a set of known facts.



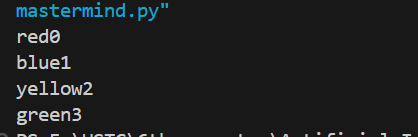
This is an output of clue.py file . This code is a logic-based program that simulates a deduction process, similar to the game Clue, where we try to determine a correct combination of a character, room, and weapon. It starts by defining symbols to represent each character (ColMustard, ProfPlum, and MsScarlet), room (ballroom, kitchen, and library), and weapon (knife, revolver, and wrench). These symbols are combined in a knowledge base (knowledge), which consists of logical constraints. The main constraints include that there must be one character, one room, and one weapon involved, and some initial information, such as the exclusion of certain symbols, which helps narrow down possible solutions.

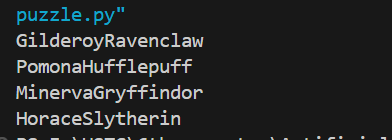
The check\_knowledge function then evaluates each symbol using model\_check, which checks if a symbol is definitively True (included in the solution) based on the knowledge. If a symbol is true, it prints "YES"; otherwise, if neither it nor its negation can be confirmed, it prints "MAYBE." This function iterates over each symbol, applying the logical rules set in knowledge to deduce which symbols must be true. The final output shows which symbols satisfy the constraints, representing a possible solution combination.

**Harry.py:**

This code implements a logical inference system in Python that evaluates whether a query is entailed by a knowledge base using logical sentences. It defines classes for various logical constructs—like Symbol, Not, And, Or, Implication, and Biconditional—each with methods for evaluating expressions, generating string representations, and extracting symbols.

The model\_check function checks if the knowledge base entails a query by examining all possible truth assignments for the symbols involved. It uses a recursive helper function, check\_all, to evaluate the knowledge and query against these models, ensuring that if the knowledge is true in a model, the query must also be true. In the example provided, the code sets up a knowledge base related to rain, Hagrid, and Dumbledore and checks if the knowledge entails the truth of rain, returning a boolean result for entailment.



The code is a logical representation for solving a Mastermind game with four colors (red, blue, green, yellow) in four positions. It defines symbols for each color in each position and establishes constraints in a knowledge base to ensure that each color occupies at least one position, that no color occupies more than one position, and that only one color can be assigned to a specific position. Additional constraints specify valid combinations and explicitly disallow certain placements, such as not allowing blue0, red1, green2, or yellow3. The output shows the valid configuration red0, blue1, yellow2, green3, indicating that this combination satisfies all the defined constraints.  
  


The code implements a logic puzzle where four individuals—Gilderoy, Pomona, Minerva, and Horace—must be assigned to four distinct houses: Gryffindor, Hufflepuff, Ravenclaw, and Slytherin. It establishes rules ensuring that each person belongs to one house only and that no two individuals can share the same house. Specific conditions are added, such as Gilderoy being able to be in either Gryffindor or Ravenclaw, Pomona not being in Slytherin, and Minerva being assigned to Gryffindor.

The output reveals valid house assignments that adhere to these constraints: Gilderoy is assigned to Ravenclaw, Pomona to Hufflepuff, Minerva to Gryffindor, and Horace to Slytherin. These assignments fulfill the conditions laid out in the logic statements, ensuring that each person is in a unique house while satisfying the additional rules specified for certain characters.