Knowledge Representation

Introduction :

Knowledge representation in AI is the method of organizing and storing information so that AI systems can use it to make decisions. It's essential for AI, as it helps systems understand data, reach conclusions, and solve problems.

Purpose of Each File :

1.Logic.py :

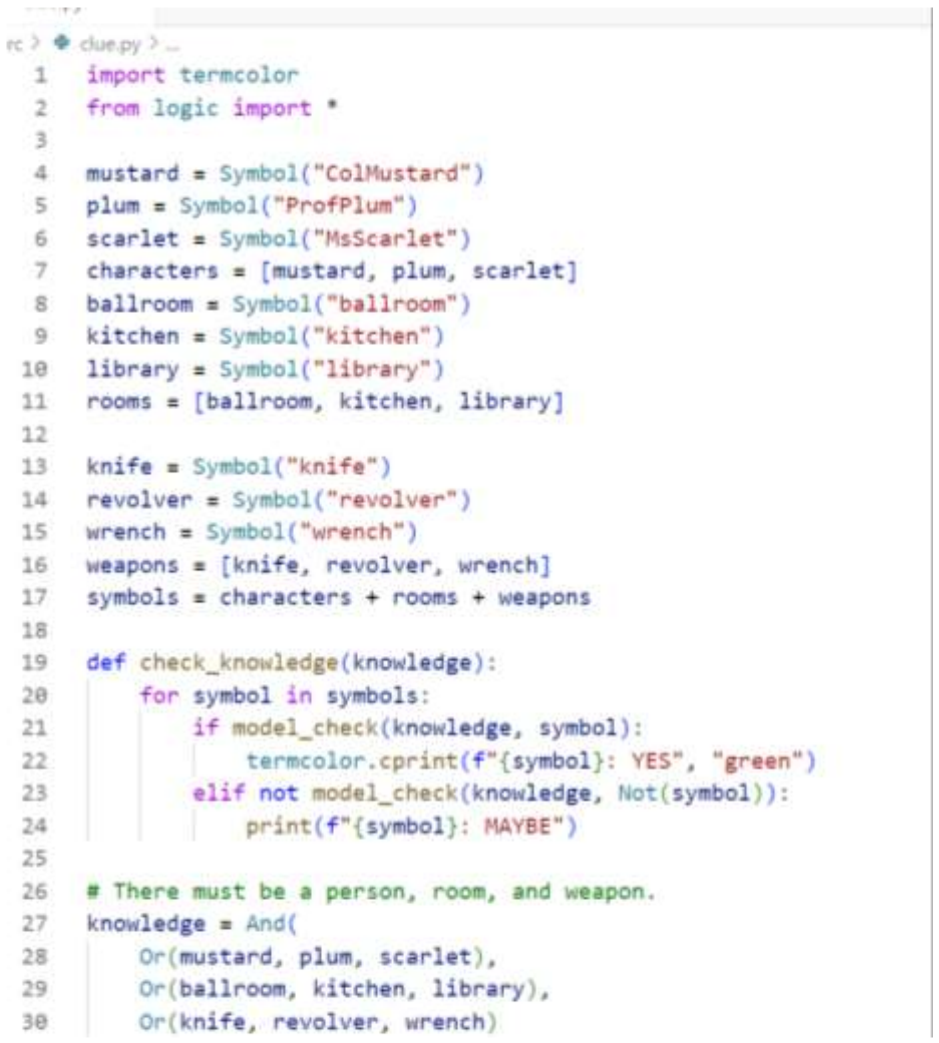
The core library defines the 'Sentence' class and its subclasses to represent logical knowledge as objects. The 'model_check' function is used to evaluate all possible truth assignments and determine logical entailment.

How It Works :
Logical sentences are created using the Sentence class and its subclasses. These sentences can be combined with logical operators. The model_check function is then used to test if a query logically follows from the knowledge base by evaluating all possible truth combinations.This approach allows the program to deduce new facts, solve logical problems, and verify or contradict logical statements efficiently.

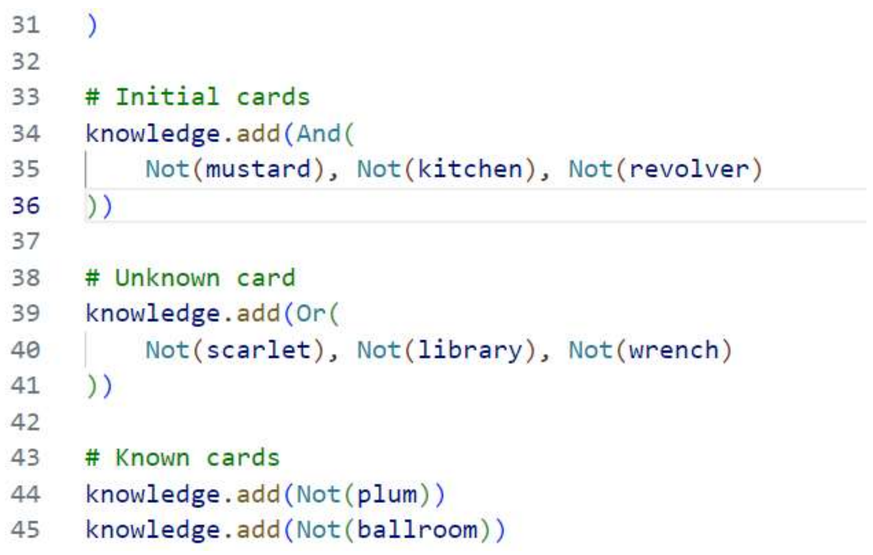
2.clue.py :

The program solves a version of the Clue game by identifying the suspect, weapon, and room involved in the crime. It represents the relationships and constraints between these elements as logical sentences in the knowledge base (KB).

How It Works:
Symbols such as *ColMustard* (suspect), *kitchen* (room), and *knife* (weapon) are used to represent the characters and items.
Rules are created to ensure that only one suspect, one weapon, and one room are correct.
Known facts, like "*ColMustard is not the suspect,*" are added to the KB.
The program analyzes each symbol to determine whether it is true, false, or uncertain, based on the available facts and logical reasoning.

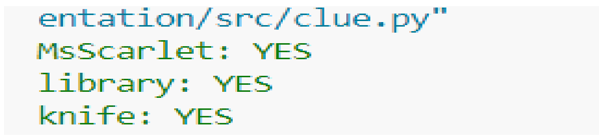


```python
import termcolor
from logic import *

mustard = Symbol("ColMustard")
plum = Symbol("ProfPlum")
scarlet = Symbol("MsScarlet")
characters = [mustard, plum, scarlet]
ballroom = Symbol("ballroom")
kitchen = Symbol("kitchen")
library = Symbol("library")
rooms = [ballroom, kitchen, library]

knife = Symbol("knife")
revolver = Symbol("revolver")
wrench = Symbol("wrench")
weapons = [knife, revolver, wrench]
symbols = characters + rooms + weapons

def check_knowledge(knowledge):
    for symbol in symbols:
        if model_check(knowledge, symbol):
            termcolor.cprint(f"{symbol}: YES", "green")
        elif not model_check(knowledge, Not(symbol)):
            print(f"{symbol}: MAYBE")

# There must be a person, room, and weapon.
knowledge = And(
    Or(mustard, plum, scarlet),
    Or(ballroom, kitchen, library),
    Or(knife, revolver, wrench)
```



```python
)

# Initial cards
knowledge.add(And(
    Not(mustard), Not(kitchen), Not(revolver)
))

# Unknown card
knowledge.add(Or(
    Not(scarlet), Not(library), Not(wrench)
))

# Known cards
knowledge.add(Not(plum))
knowledge.add(Not(ballroom))
```

Result:



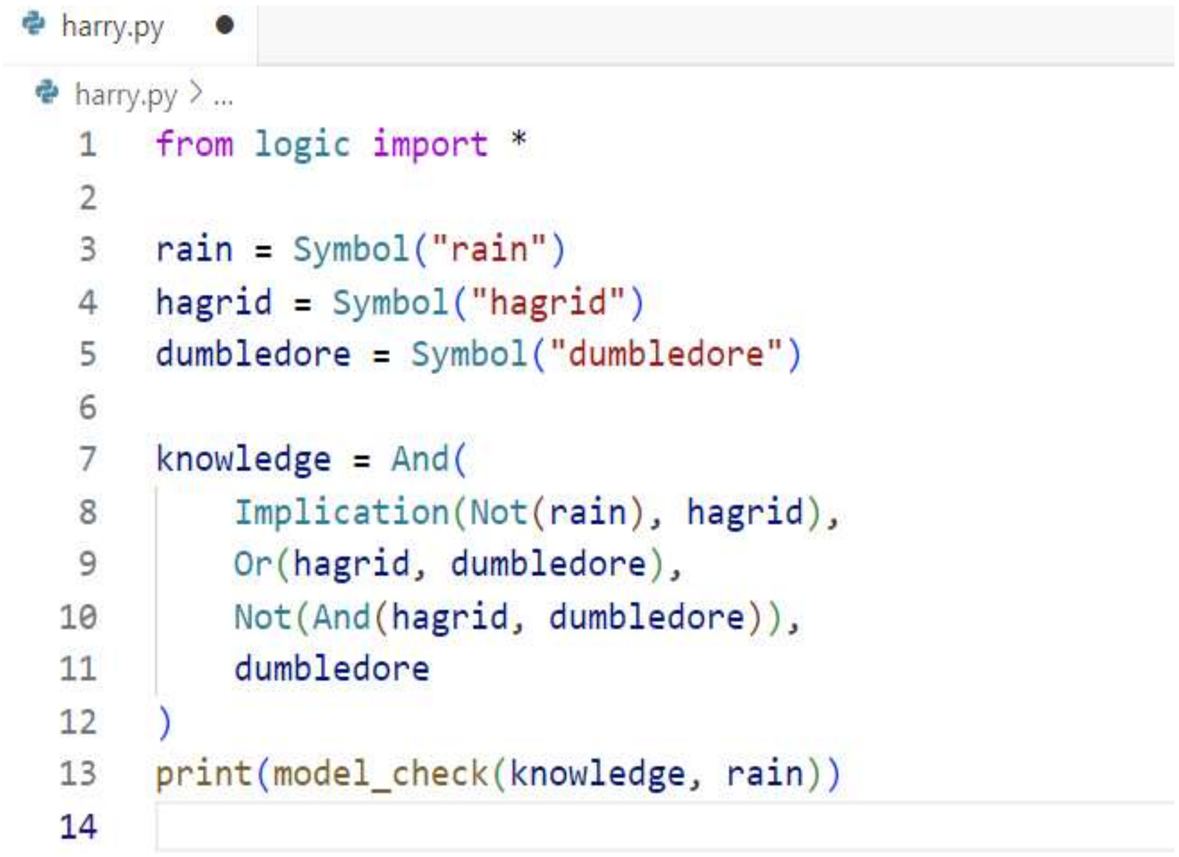
```
entation/src/clue.py"
MsScarlet: YES
library: YES
knife: YES
```

3.harry.py:

The program solves a simple logic puzzle using Harry Potter characters to figure out if it's raining.

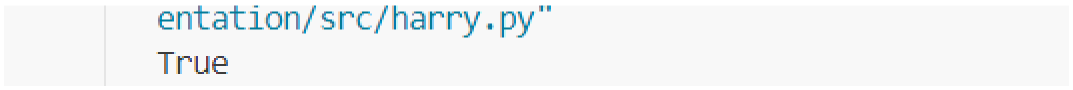
How It Works:

1. Adds rules like: "If it doesn't rain, Hagrid is here."
2. Includes facts like: "Dumbledore is here."
3. Check if these facts mean that it must be raining.



```python
from logic import *

rain = Symbol("rain")
hagrid = Symbol("hagrid")
dumbledore = Symbol("dumbledore")

knowledge = And(
    Implication(Not(rain), hagrid),
    Or(hagrid, dumbledore),
    Not(And(hagrid, dumbledore)),
    dumbledore
)
print(model_check(knowledge, rain))
```

Result :



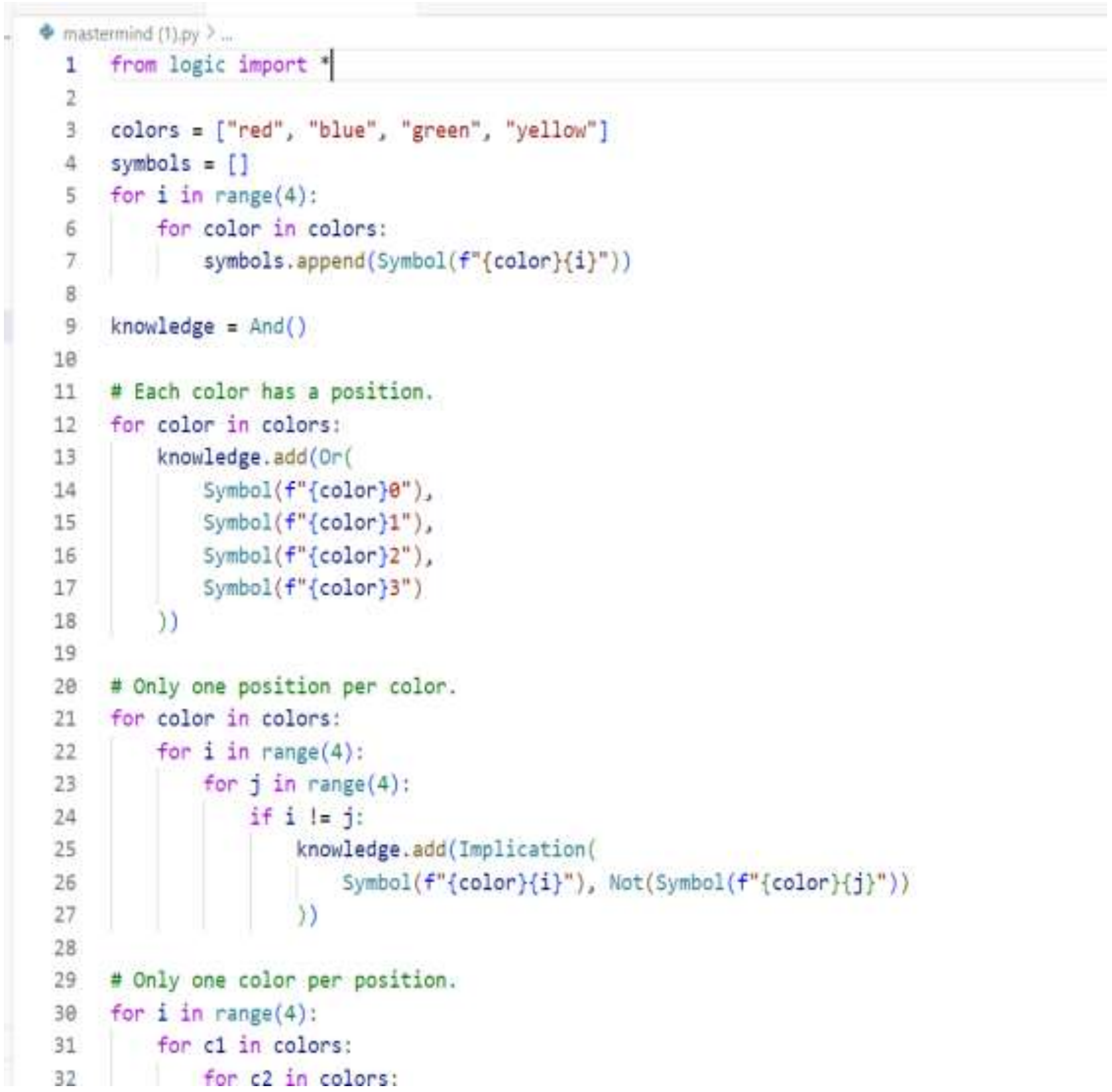
```
entation/src/harry.py"
True
```

4.mastermind.py :

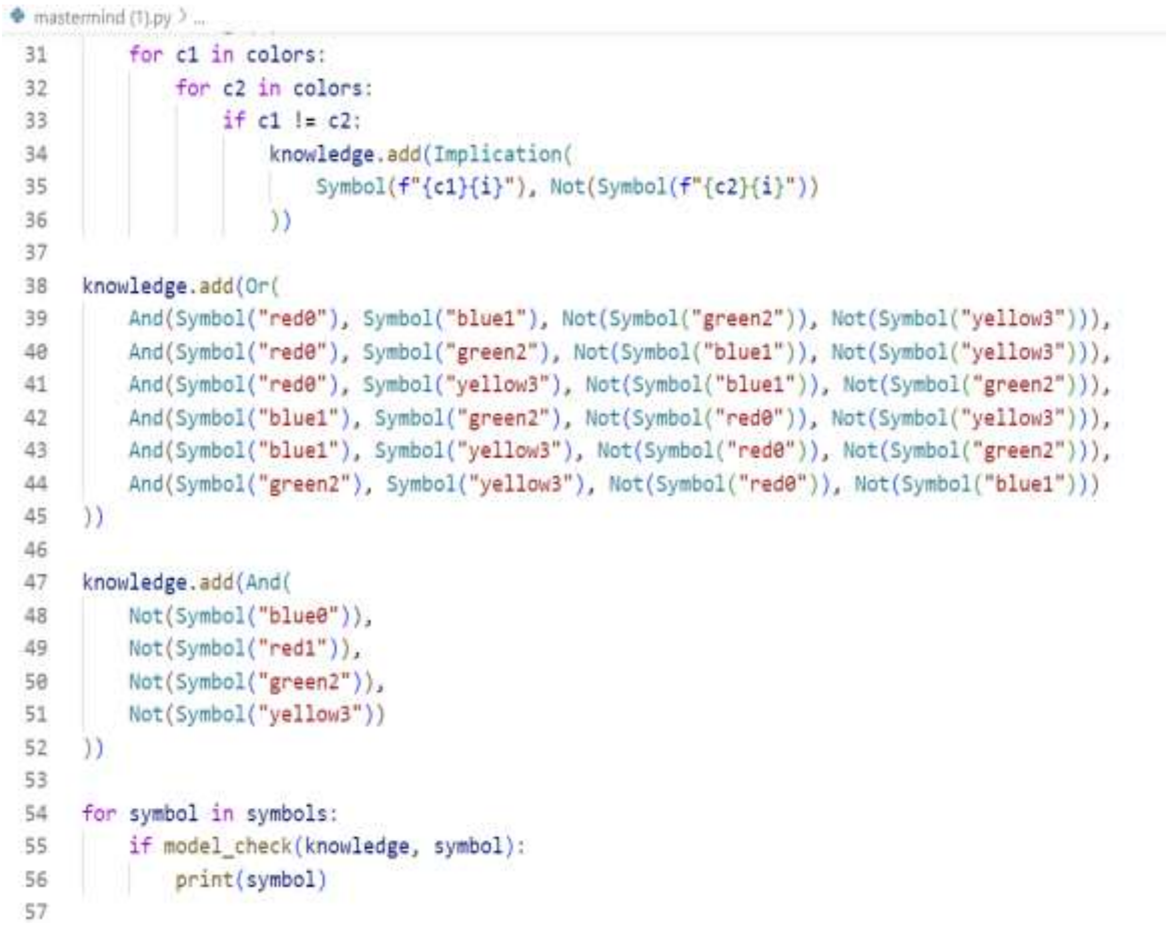
This program solves a simplified version of the Mastermind game by using logical deduction to find the correct sequence of colors.

How it works:

1. Color-Position Representation: Each color is linked with a specific position, like red0 (where red is in position 0).
2. Unique Color Usage: The program ensures that each color is used only once and occupies a single position in the sequence.
3. Clues and Restrictions: Clues such as "blue is not in position 0" are provided to narrow down possible combinations.
4. Logical Analysis: The program analyzes these clues, restrictions, and previous guesses to eliminate incorrect possibilities and deduce the correct color order.

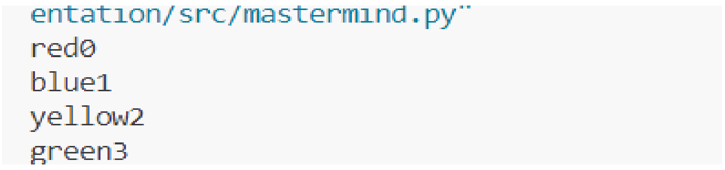

mastermind (1).py > ...

```python
from logic import *

colors = ["red", "blue", "green", "yellow"]
symbols = []
for i in range(4):
    for color in colors:
        symbols.append(Symbol(f"{color}{i}"))

knowledge = And()

# Each color has a position.
for color in colors:
    knowledge.add(Or(
        Symbol(f"{color}0"),
        Symbol(f"{color}1"),
        Symbol(f"{color}2"),
        Symbol(f"{color}3")
    ))

# Only one position per color.
for color in colors:
    for i in range(4):
        for j in range(4):
            if i != j:
                knowledge.add(Implication(
                    Symbol(f"{color}{i}"), Not(Symbol(f"{color}{j}"))
                ))

# Only one color per position.
for i in range(4):
    for c1 in colors:
        for c2 in colors:
```



```python
    for c1 in colors:
        for c2 in colors:
            if c1 != c2:
                knowledge.add(Implication(
                    Symbol(f"{c1}{i}"), Not(Symbol(f"{c2}{i}"))
                ))

knowledge.add(Or(
    And(Symbol("red0"), Symbol("blue1"), Not(Symbol("green2")), Not(Symbol("yellow3"))),
    And(Symbol("red0"), Symbol("green2"), Not(Symbol("blue1")), Not(Symbol("yellow3"))),
    And(Symbol("red0"), Symbol("yellow3"), Not(Symbol("blue1")), Not(Symbol("green2"))),
    And(Symbol("blue1"), Symbol("green2"), Not(Symbol("red0")), Not(Symbol("yellow3"))),
    And(Symbol("blue1"), Symbol("yellow3"), Not(Symbol("red0")), Not(Symbol("green2"))),
    And(Symbol("green2"), Symbol("yellow3"), Not(Symbol("red0")), Not(Symbol("blue1")))
))

knowledge.add(And(
    Not(Symbol("blue0")),
    Not(Symbol("red1")),
    Not(Symbol("green2")),
    Not(Symbol("yellow3"))
))

for symbol in symbols:
    if model_check(knowledge, symbol):
        print(symbol)
```

Result :



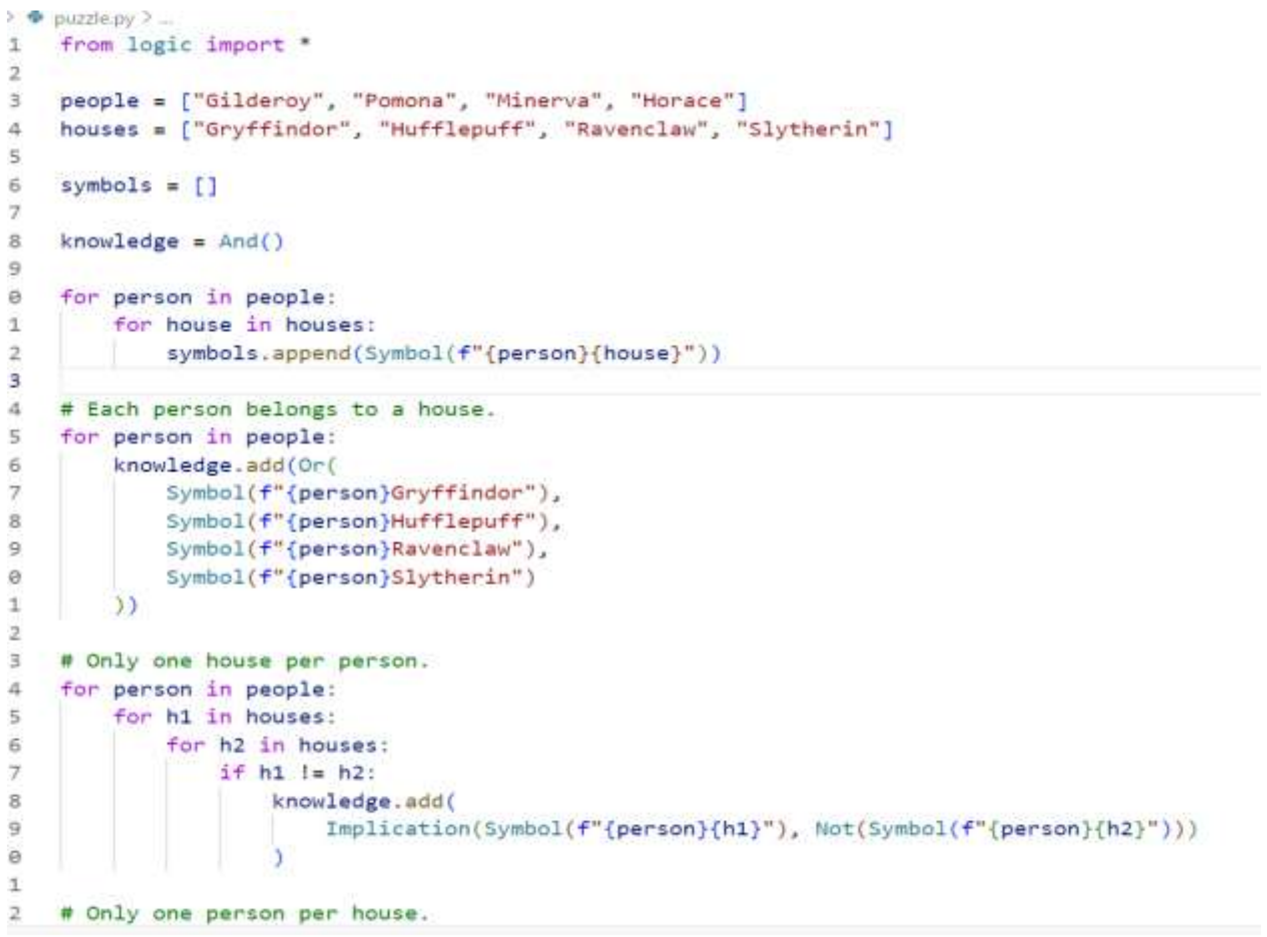
```
entation/src/mastermind.py"
red0
blue1
yellow2
green3
```

5.puzzle.py:

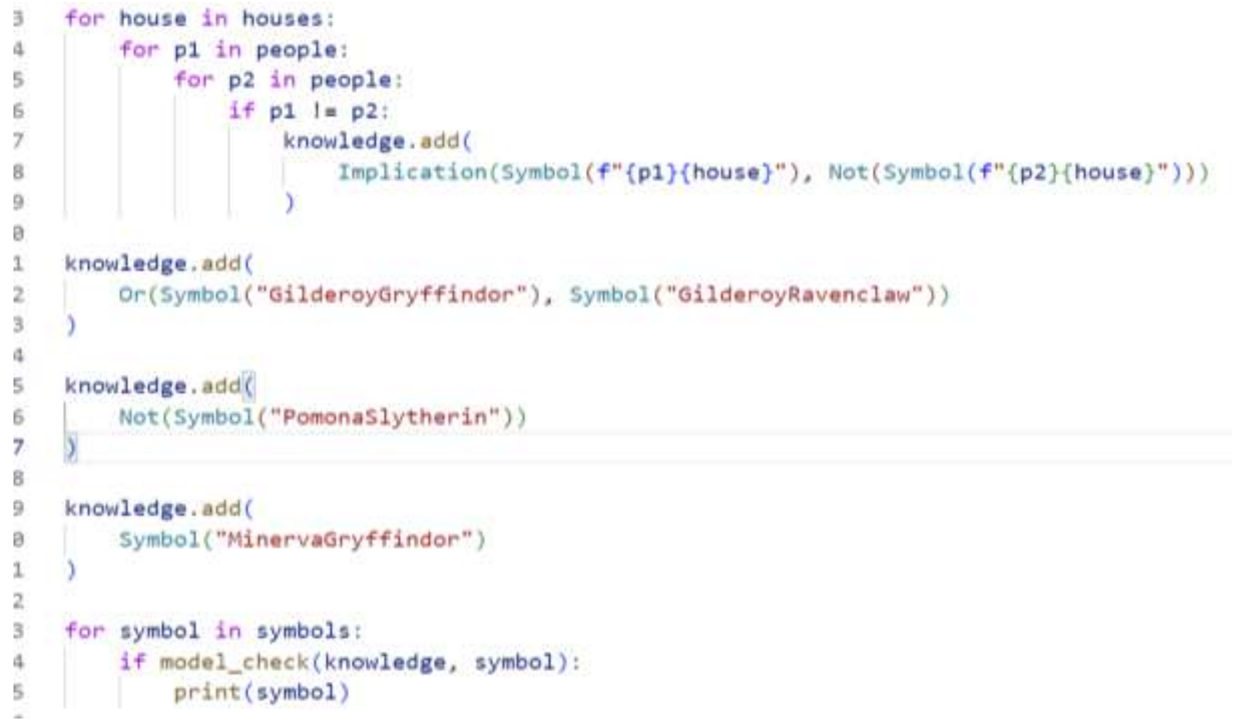
This program solves a logic puzzle where the goal is to assign people to specific houses, like determining who belongs in Gryffindor.

How it works:

1. **Person-House Representation**: It uses symbols, such as GilderoyGryffindor, to represent the potential assignments of people to houses.
2. **Rules of Assignment**: The program enforces rules that ensure each person is assigned to only one house and that each house can only have one person.
3. **Known Facts**: The program includes predefined facts and constraints about who belongs where.
4. **Logical Deduction**: By analyzing the facts and applying the rules, the program deduces the correct house assignments for each person.

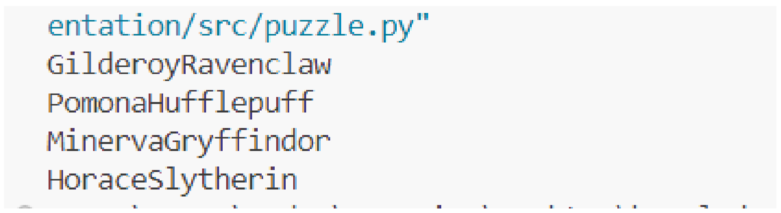


```python
from logic import *

people = ["Gilderoy", "Pomona", "Minerva", "Horace"]
houses = ["Gryffindor", "Hufflepuff", "Ravenclaw", "Slytherin"]

symbols = []

knowledge = And()

for person in people:
    for house in houses:
        symbols.append(Symbol(f"{person}{house}"))

# Each person belongs to a house.
for person in people:
    knowledge.add(Or(
        Symbol(f"{person}Gryffindor"),
        Symbol(f"{person}Hufflepuff"),
        Symbol(f"{person}Ravenclaw"),
        Symbol(f"{person}Slytherin")
    ))

# Only one house per person.
for person in people:
    for h1 in houses:
        for h2 in houses:
            if h1 != h2:
                knowledge.add(
                    Implication(Symbol(f"{person}{h1}"), Not(Symbol(f"{person}{h2}")))
                )

# Only one person per house.
```



```python
for house in houses:
    for p1 in people:
        for p2 in people:
            if p1 != p2:
                knowledge.add(
                    Implication(Symbol(f"{p1}{house}"), Not(Symbol(f"{p2}{house}")))
                )

knowledge.add(
    Or(Symbol("GilderoyGryffindor"), Symbol("GilderoyRavenclaw"))
)

knowledge.add(
    Not(Symbol("PomonaSlytherin"))
)

knowledge.add(
    Symbol("MinervaGryffindor")
)

for symbol in symbols:
    if model_check(knowledge, symbol):
        print(symbol)
```

Result :



```
entation/src/puzzle.py"
GilderoyRavenclaw
PomonaHufflepuff
MinervaGryffindor
HoraceSlytherin
```

How the program works:

- **Symbols:** It uses symbols to represent different facts, such as ColMustard for a suspect or red0 for the red color in position 0.
- **Logic Sentences:** The program combines these symbols with logical rules, like "If A, then B" or "A and B must both be true."
- **Knowledge Base:** This is a collection of rules and established facts that the program uses to draw conclusions.
- **Model Checking:** The program tests different possibilities by checking how the rules and facts align, determining, for instance, if ColMustard could be the culprit.

Simplified Example:

- Suppose we know:
  - "If it rains, the grass is wet."
  - "The grass is wet."
- The program represents these statements as symbols and uses logic to determine whether it must be raining.

Key Points :

- **Translation of Real-World Problems**: These programs convert real-world scenarios into logical rules and symbols, making it easier to reason about them.
- **Model Checking**: The program automates the process of testing what is true or possible by applying the given rules and facts.
- **Advanced Logic**: For more complex problems, First-Order Logic is used, allowing for the inclusion of variables and relationships to model more intricate scenarios.