1.

**IF** egg is safe

**IF** temperature is ideal

**AND** temperature is not cool

**AND** humidity is fine

**THEN** incubating is successful

2.

```
#First import the neccessary packages and modules
from utils import *
from logic import *
from notebook import psource

#Next symbolised the Letters needed
(x, T, H, C, S, I) = expr('x, T, H, C, S, I')

#Develop a structure for the KB
hatching_kb = PropKB()
'''
IF T & H -> S
IF C -> !S
IF S -> I
'''
hatching_kb.tell((T & H) | '==>' | S)
hatching_kb.tell(C | '==>' | ~S)
hatching_kb.tell(S | '==>' | I)

hatching_kb.tell((T & H))
hatching_kb.ask_if_true(I)
```

3. I)

```
In [12]: # LAB TEST
         #Next symbolised the Letters needed
         (x, T, H, C, S, I) = expr('x, T, H, C, S, I')

         #Develop a structure for the KB
         hatching_kb = PropKB()
         '''Now we develop the KB of Sunflower based on the rules of thumb

         IF T & H -> S
         IF C -> !S
         IF S -> I
         '''
         # hatching_kb.tell((~M & T) | '==>' | ~S)
         hatching_kb.tell((T & H) | '==>' | S)
         hatching_kb.tell(C | '==>' | ~S)
         hatching_kb.tell(S | '==>' | I)

         hatching_kb.tell(~I)
         hatching_kb.ask_if_true(S)

Out[12]: False
```

ii)

```
In [23]: # LAB TEST
         #Next symbolised the Letters needed
         (x, T, H, C, S, I) = expr('x, T, H, C, S, I')

         #Develop a structure for the KB
         hatching_kb = PropKB()
         '''Now we develop the KB of Sunflower based on the rules of thumb

         IF T & H -> S
         IF C -> !S
         IF S -> I
         '''
         hatching_kb.tell((T & H) | '==>' | S)
         hatching_kb.tell(C | '==>' | ~S)
         hatching_kb.tell(S | '==>' | I)

         hatching_kb.tell(~S)
         hatching_kb.ask_if_true(C)

Out[23]: False
```

The main limitation of using propositional logic in this application is that it is difficult to represent the actual precision of consequents such as temperature and humidity using atomic symbols. Another limitation of using propositional logic in this application is that it is unable to infer possible antecedents based on a given consequent using abductive reasoning if no equivalence rule is used. For example, the application is unable to reason that the incubating is not successful because the temperature might be cool using the rule 'C ➔ ~S'.

4.

```
#First import the neccessary packages and modules
from utils import *
from logic import *
from notebook import psource

clauses = []
clauses.append(expr("(Temperature(Ideal) & Humidity(Fine)) ==> Egg(Safe)"))
clauses.append(expr("Temperature(Cool) ==> Egg(Unsafe)"))
clauses.append(expr("Egg(Safe) ==> Incubating(Success)"))

#Develop a structure for the KB
FolHatching_kb = FolKB(clauses)

#Given the fact now,
FolHatching_kb.tell(expr('Humidity(Fine)'))
FolHatching_kb.tell(expr('Temperature(Ideal)'))
answer = fol_fc_ask(FolHatching_kb, expr('Egg(x)'))
print(list(answer))
```

5.

```
In [12]: #First import the neccessary packages and modules
         from utils import *
         from logic import *
         from notebook import psource

         clauses = []
         clauses.append(expr("(Temperature(Ideal) & Humidity(Fine)) ==> Egg(Safe)"))
         clauses.append(expr("Temperature(Cool) ==> Egg(Unsafe)"))
         clauses.append(expr("Egg(Safe) ==> Incubating(Success)"))

         #Develop a structure for the KB
         FolHatching_kb = FolKB(clauses)

         #Given the fact now,
         FolHatching_kb.tell(expr('Humidity(Fine)'))
         FolHatching_kb.tell(expr('Temperature(Ideal)'))
         answer = fol_fc_ask(FolHatching_kb, expr('Egg(x)'))
         print(list(answer))

         [{x: Safe}]

Out[12]: [((Temperature(Ideal) & Humidity(Fine)) ==> Egg(Safe)),
          (Temperature(Cool) ==> Egg(Unsafe)),
          (Egg(Safe) ==> Incubating(Success)),
          Humidity(Fine),
          Temperature(Ideal),
          Egg(Safe),
          Incubating(Success)]
```

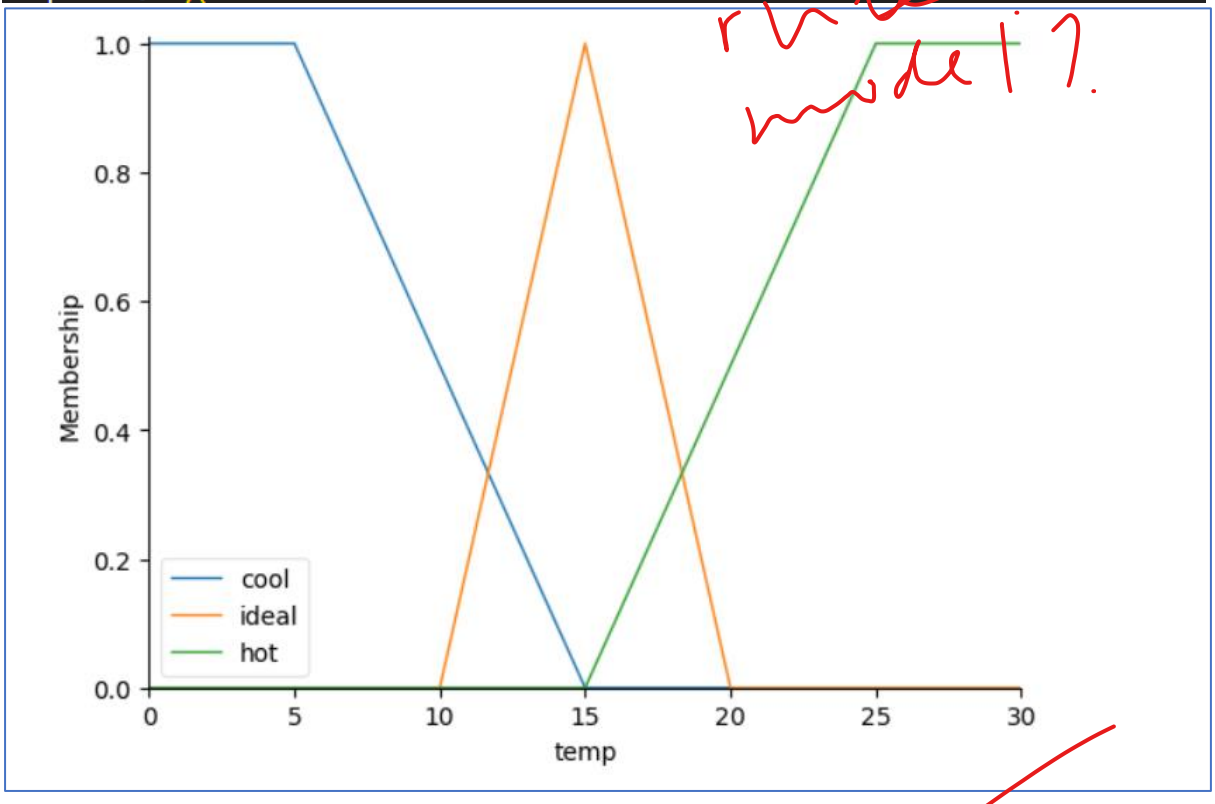
This is forward chaining, because all antecedents are assessed and reached the final consequent which is Incubating(Success).

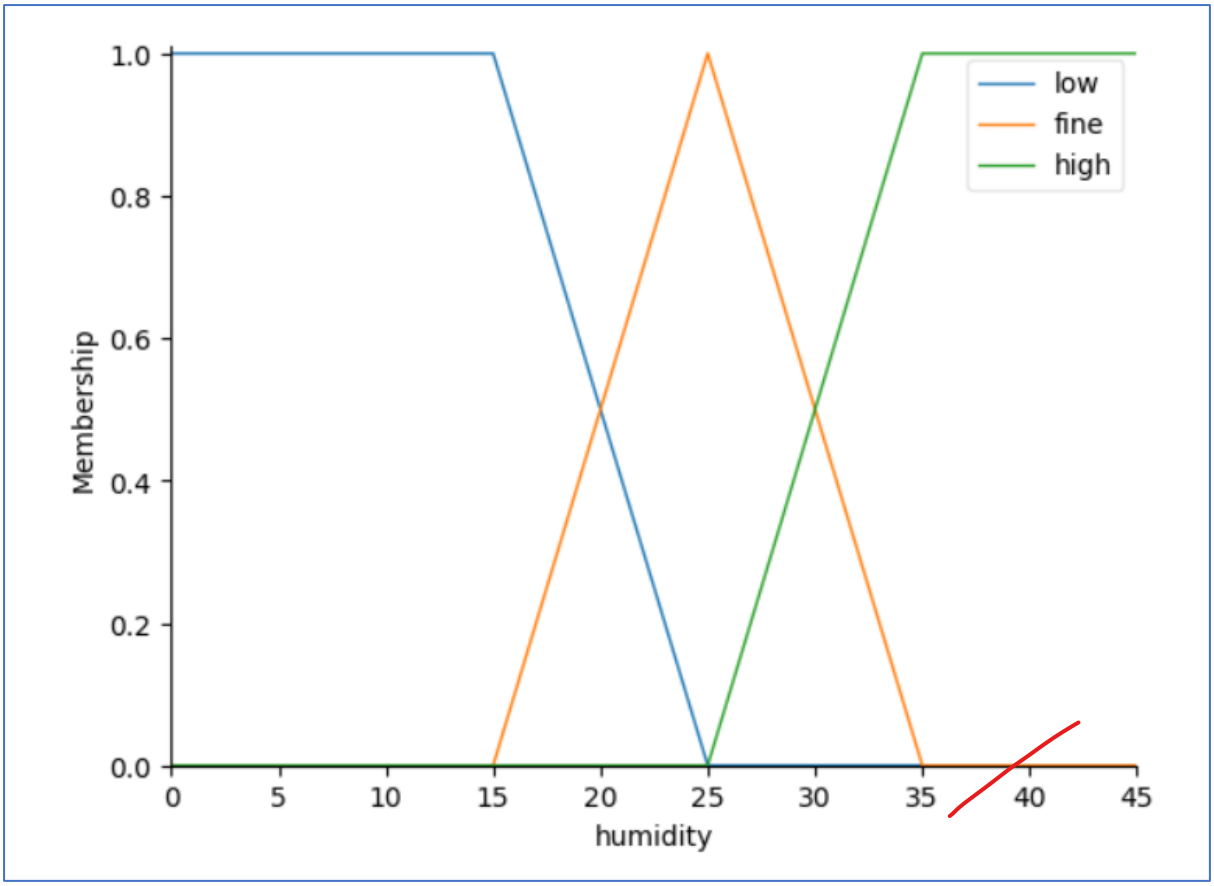
6. The main advantages of using FOL on this application compared to propositional logic is because predicates are used to represent rules in a more human understandable way. FOL is also able to represent rules that contains semantic meaning which cannot be represented using propositional logic.

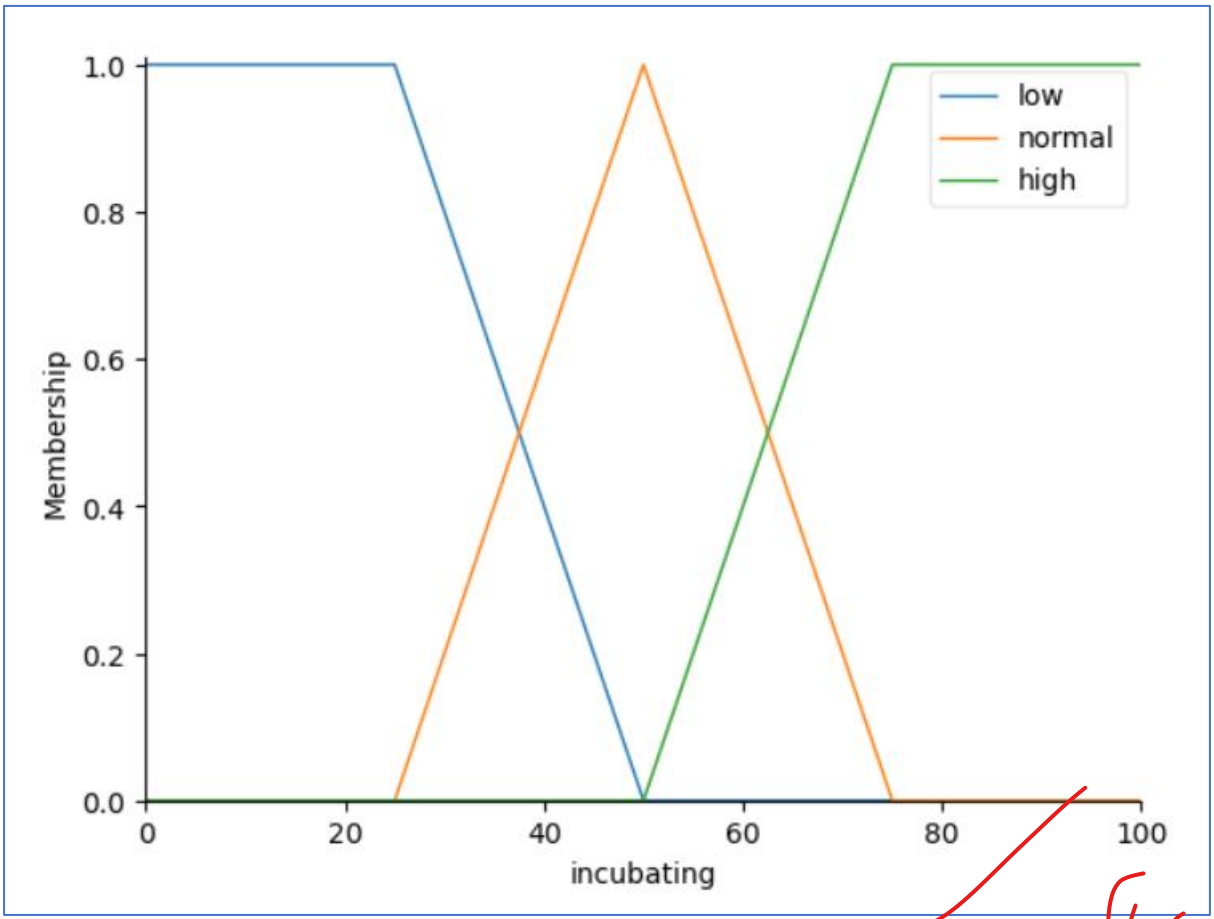
7.

```python
import numpy as np
import skfuzzy as fuzz
import matplotlib.pyplot as plt
from skfuzzy import control as ctrl

# New Antecedent/Consequent objects hold universe variables and membership
# functions
temp = ctrl.Antecedent(np.arange(0, 31, 1), 'temp') # 20C as origin (0), 50C
humidity = ctrl.Antecedent(np.arange(0, 46, 1), 'humidity') # 50% as 0, 95% a
incubating = ctrl.Consequent(np.arange(0, 101, 1), 'incubating')

temp['cool'] = fuzz.trapmf(temp.universe, [0, 0, 5, 15])
temp['ideal'] = fuzz.trimf(temp.universe, [10, 15, 20])
temp['hot'] = fuzz.trapmf(temp.universe, [15, 25, 30, 30])

# Custom membership functions can be built interactively with a familiar,
# Pythonic API
humidity['low'] = fuzz.trapmf(humidity.universe, [0, 0, 15, 25])
humidity['fine'] = fuzz.trimf(humidity.universe, [15, 25, 35])
humidity['high'] = fuzz.trapmf(humidity.universe, [25, 35, 45, 45])

# Custom membership functions can be built interactively with a familiar,
# Pythonic API
incubating['low'] = fuzz.trapmf(incubating.universe, [0, 0, 25, 50])
incubating['normal'] = fuzz.trimf(incubating.universe, [25, 50, 75])
incubating['high'] = fuzz.trapmf(incubating.universe, [50, 75, 100, 100])
```

8.




1.0
0.8
0.6
0.4
0.2
0.0
Membership
0
5
10
15
20
25
30
35
40
45
humidity
low
fine
high


1.0
0.8
0.6
0.4
0.2
0.0
Membership
0
20
40
60
80
100
incubating
low
normal
high

result ?

9. The main advantages of using fuzzy logic for this application compared to FOL and propositional logic is the degree of truth between 2 compatible states of a antecedent can be represented and decided upon. Using fuzzy logic, humans can also propose real life values for antecedents as opposed to using the actual numerical data.

X 10