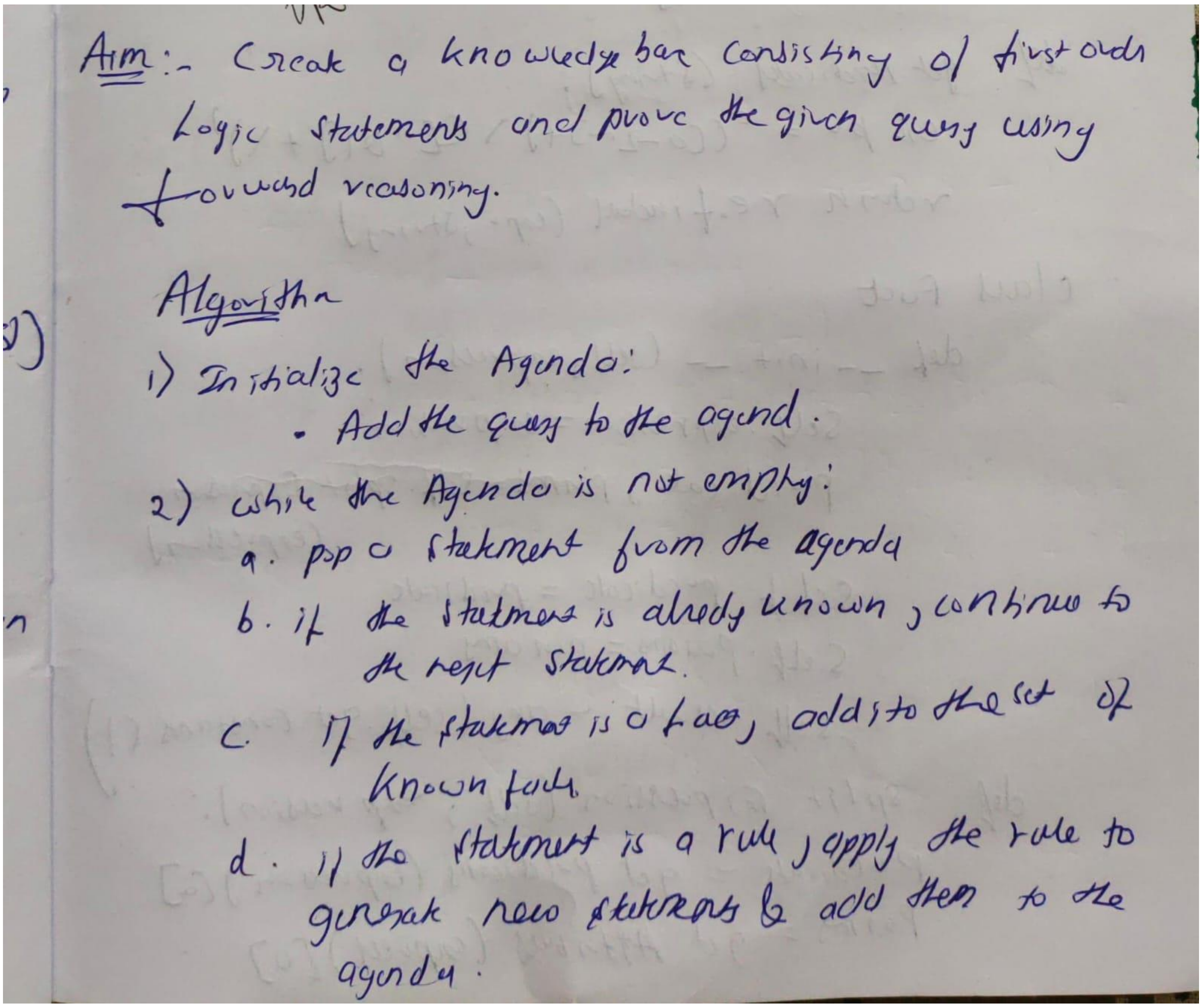
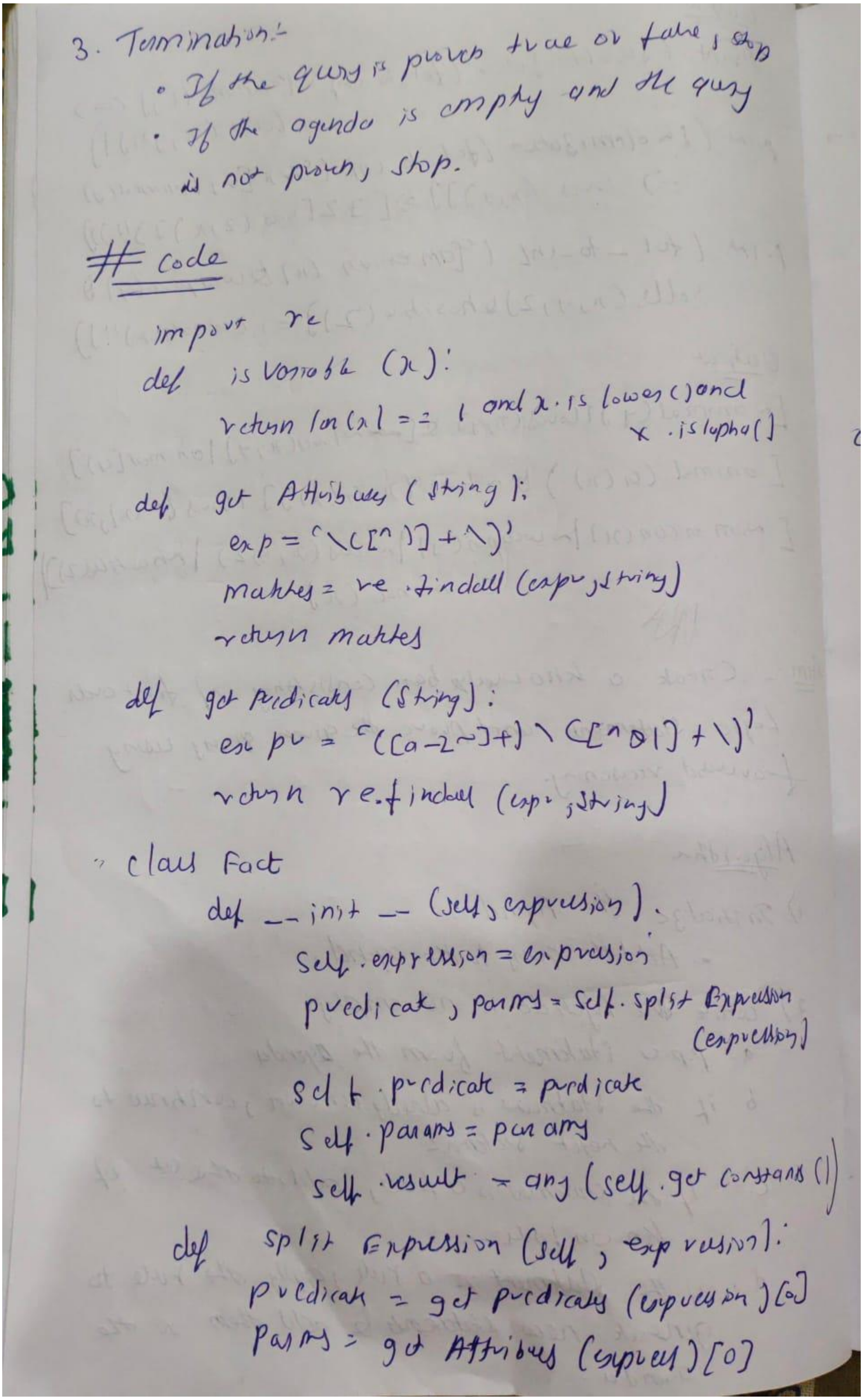


Aim:- Create a knowledge base consisting of first order Logic statements and prove the given query using forward reasoning.

Algorithm

```
1) Initialize the Agenda:
     • Add the query to the agend.
2) while the Agenda is not empty:
     a. pop a statement from the agenda
     b. if the statement is already known, continue to
           the next statement.
     c. If the statement is a fact, add it to the set of
           known facts.
     d. if the statement is a rule, apply the rule to
        generate new statements & add them to the
          agenda.
```

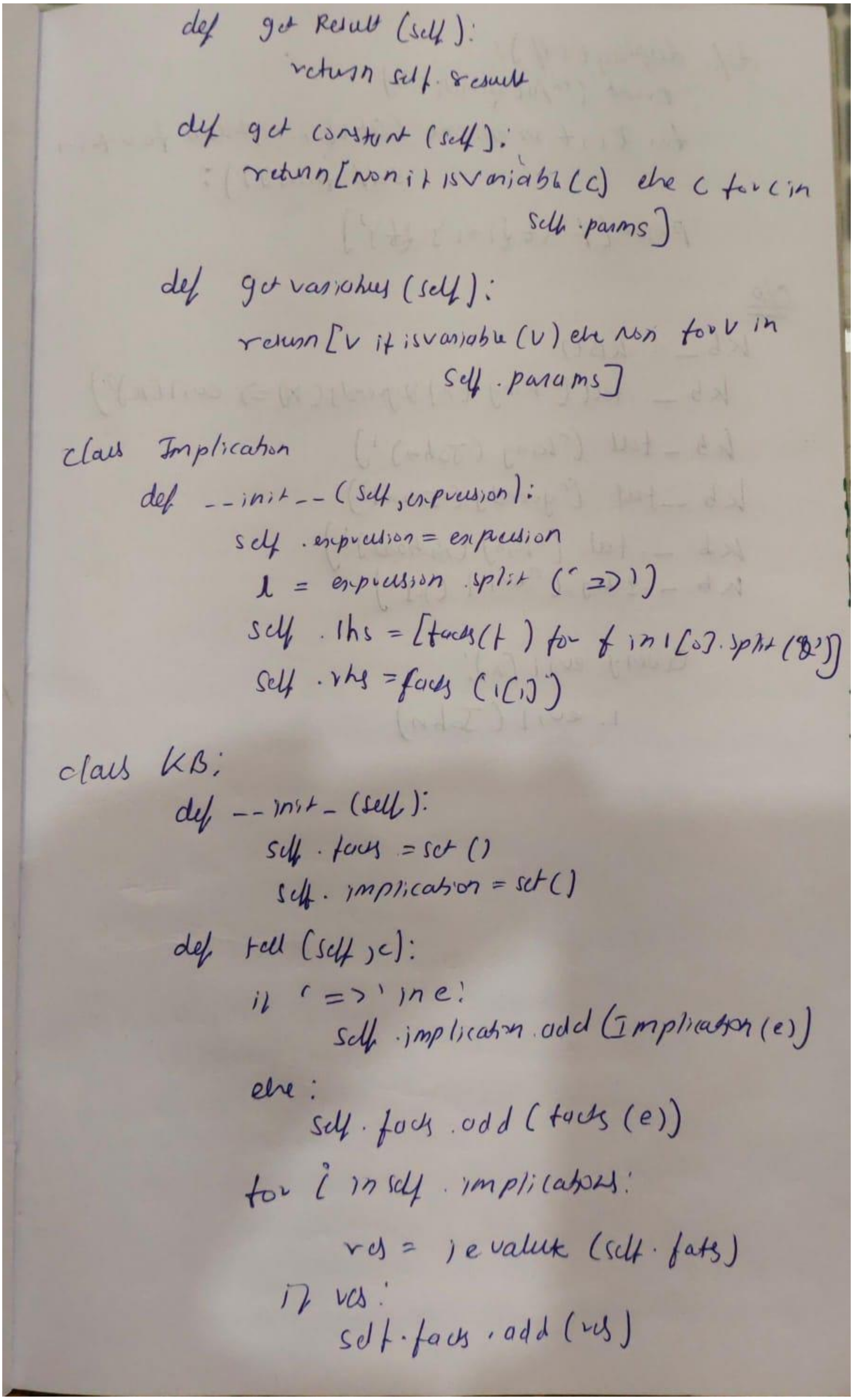


3. Termination:-

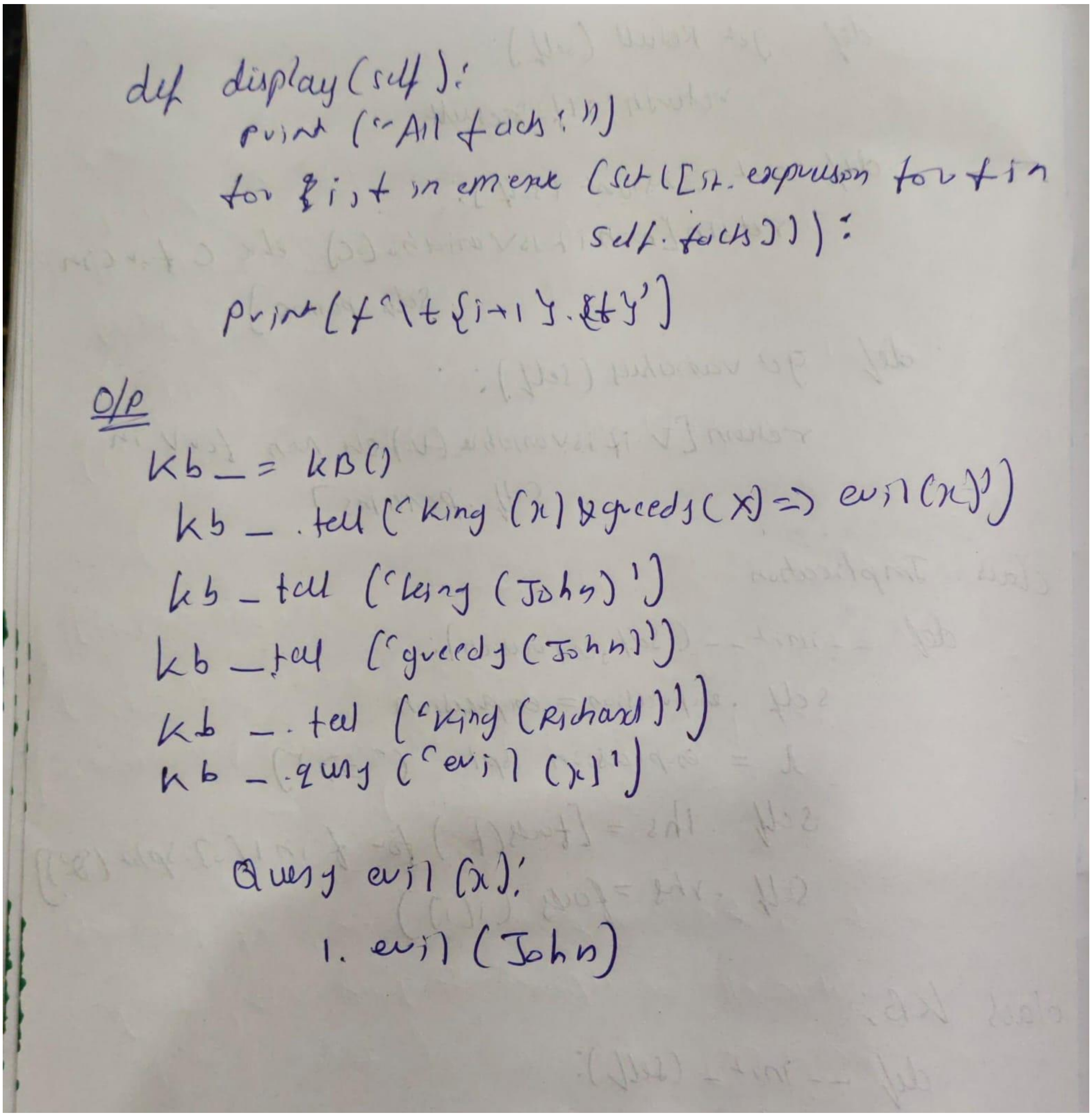
- If the query is proved true or false, stop
- If the agenda is empty and the query is not proven, stop.

code

```
import re
def isVariable (x):
    return len(x) == 1 and x.islower() and
                                        x.isalpha()

def getAttributes (string):
    exp = '\([^)]+\)'
    matches = re.findall (expr, string)
    return matches

def getPredicates (String):
    expr = '([a-z~]+)\([^&|]+\)'
    return re.findall (expr, String)

class Fact
    def __init__ (self, expression).
        self.expression = expression
        predicate, params = self.splitExpression
                                        (expression)
        self.predicate = predicate
        self.params = params
        self.result = any (self.getConstants())
    def splitExpression (self, expression):
        predicate = getPredicates (expression)[0]
        params = getAttributes (expression)[0]
```



```
    def get Result (self):
          return self. result

    def get constant (self):
        return[non if isvariable(c) else c for c in
                                  self.params]

    def get variables (self):
        return[v if isvariable (v) else non for v in
                            self . params]

class Implication
    def __init__(self,expression):
        self .expression = expression
        l = expression .split ('=>')
        self .lhs = [facts(f ) for f in l[0].split ('&')]
        self .rhs =facts (l[1])

class KB:
    def __init_ (self):
        self . facts =set ()
        self. implication = set()

    def tell (self,e):
        if '=>' in e:
            self .implication .add (Implication(e))

        else :
            self. facts .add ( facts (e))

        for i in self . implications:

              res = i.evaluate (self . fats)
          if res:
            self.facts .add (res)
```

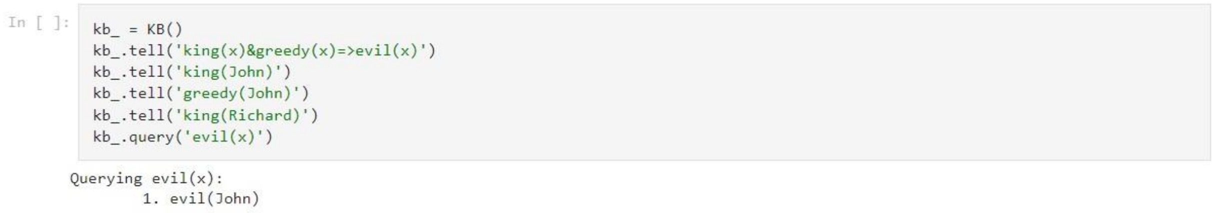


```
def display(self):
    print("All facts:")
    for i,t in enumerate(set([st.expression for t in
                              self.facts])):

    print(f'\t{i+1}.{t}')
```

O/P

```
kb_ = KB()
 kb_.tell('King(x) & greedy(X) => evil(x)')
 kb_tell('King(John)')
kb_tell('greedy(John)')
kb_.teal('King(Richard)')
kb_.query('evil(x)')

    Query evil(x):
        1. evil(John)
```

Output:



```
kb_ = KB()
kb_.tell('king(x)&greedy(x)=>evil(x)')
kb_.tell('king(John)')
kb_.tell('greedy(John)')
kb_.tell('king(Richard)')
kb_.query('evil(x)')
```

```
Querying evil(x):
        1. evil(John)
```