19/11

10

(3) Creak a knowledge consisting of FOL statement and prove the give query using ~~Fo~~ forward chaining reasoning

Algorithm:-

1) Initialize the Agenda:-
 -> add the query to the agent.

2) while the agent do is not empty :

a) pop a statement from the agenda.

b) if the statement is already known, continues to reject statement

c) If the statement is a law, add it to the set of known fold.

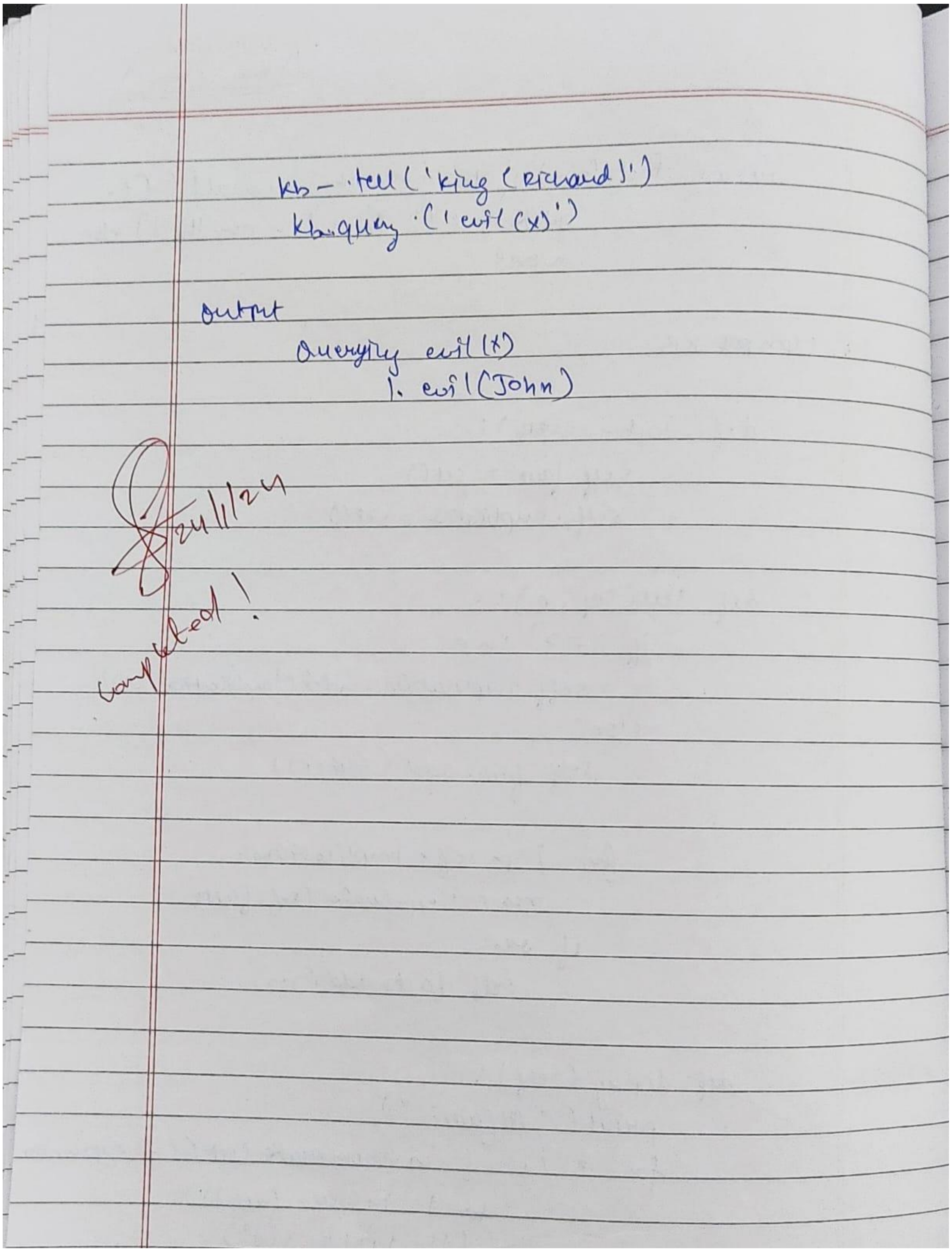
d). If the statement is a rule, apply the rule to the generate ~~the~~ new statements & ~~add~~ then to the agenda.

13) 10) Code for KB consisting FOL & prove the given query using forward reasoning.

```
import re

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

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

def getPredicate(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())
```

))
(x,y)]

```
    def getResult(self):
            return self.result

    def getConstant(self):
        return (None if isVariable(c) else c for c in
                self.params}


Class Implication:
    def __init__(self, expression):

            self.expression = expression
            l = expression.split('=>')
            self.lhs = [fact(f) for f in l[0].split('&')]
            self.rhs = fact(l[1])

def evaluate(self, facts):
        constants = {}

        new_lhs = []
        for fact in facts:
            for val in self.lhs
                if val.predicate == fact.predicate:
                    for i, v, in enumerate(val.getVariable
                        if v:
                            const[v] = fact.getConstants()
                    new_lhs.append(fact)

    for key in constants
            if constants[key]:
            attribute = attributes.replace(key, constant
return fact (expr) if len (new-lhs) and all ([f.
                getResult () for f in new-lhs ]) else
                None

class ~~KB~~ KB

    def _init_ (self):
            self.fact = set()
            self.implication = set()

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

                for i in self.implications:
                        res = i.evaluate (self.facts)
                    if res:
                        self.facts.add (res)

    def display (self):
            print( "All facts: ")
            for ~~i~~ i, f in enumerate (set([f.expression
                            for f in self.facts])):
                print(f'\t {i+1}. {f}')

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)')
```

output

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

[illegible] 24/1/24

Completed!

Output:

```
In [ ]:
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)
```