19/1/2024 A 19-1-24

→ Forward chaining

1) Input the knowledge base and the query

```
2) for i in KB:
       if i == query return True
       if '=>' in i:
           split lhs and rhs part
           if lhs ~~and rhs~~ in KB:
               add rhs to KB
   return false
```

```
3) To remove variables
     ~~for~~ if i.lower():
         ~~i.replace~~ replace the variable with constants
```

Example:

KB

king(x) & greedy(x) => evil(x)
king(John)
greedy(John)
king(Richard)

Query
evil(x)

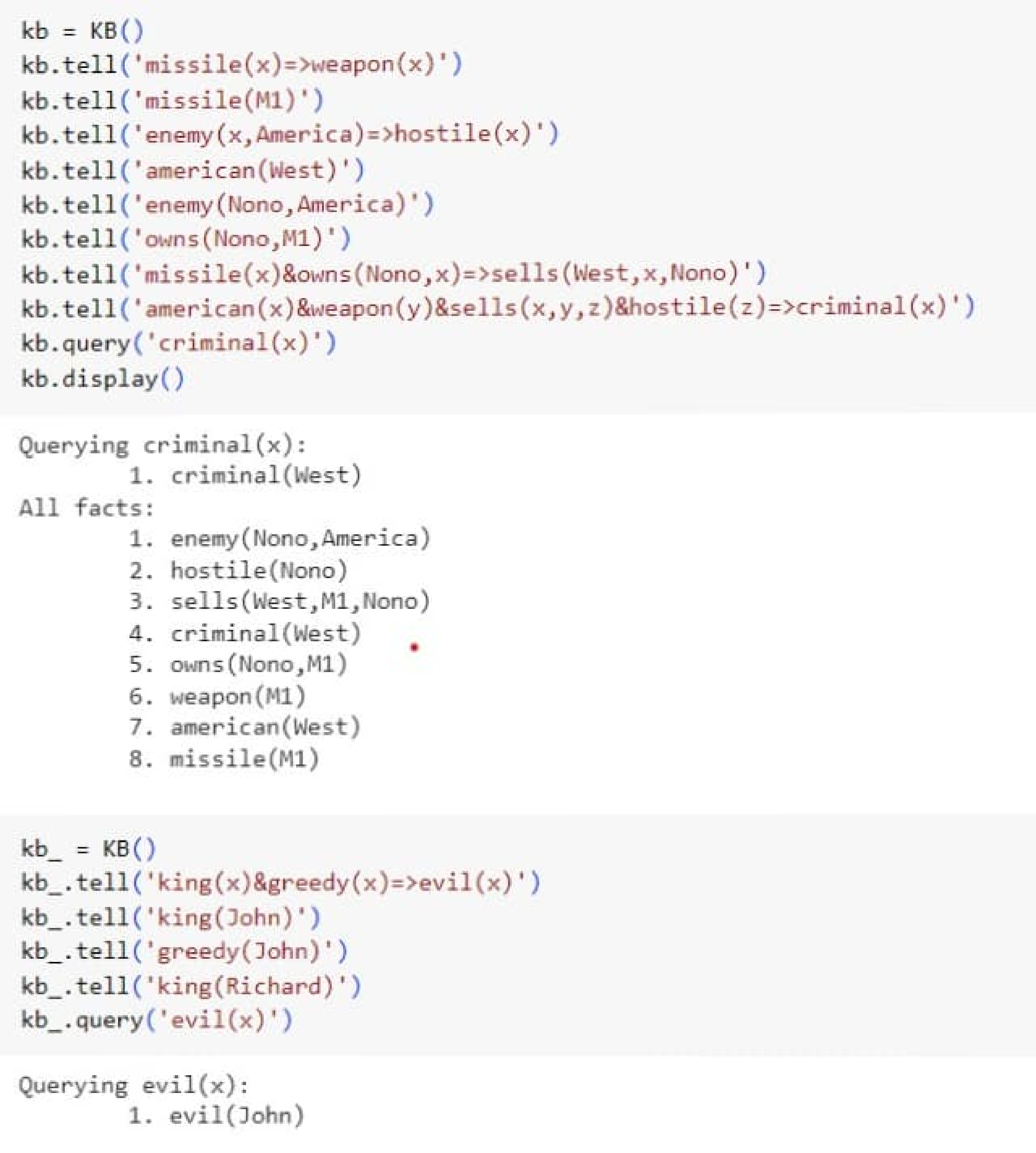
Code:

```
import re

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

def getAttributes(string):
    expr = '\([^)]+\)'
    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].strip('()').split
                                                            (','))
        return [predicate, params]

    def getResult(self):
        return self.result

    def getConstants(self):
        return [None if isVariable(c) else c for c in self.params]

    def getVariables(self):
        return [v if isVariable(v) else None for v in
                        self.params]

    def substitute(self, constants):
        c = constants.copy()
        f = f"{self.predicate}
    ( f',' .join([constants.pop(0) if is Variable(p) else p for
        p in self. params ])})"
    return  Fact ( f)

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. getVariables()
                        if  v:
                            constants[v] = fact. getConstants()[i
                    . newlhs. append (fact)
        predicate, attributes = getPredicates (self. rhs. expression
        str (getAttributes (self. rhs. expression)[0] )
            for  key  in  constants :
                if  Constants[key]:
                    attributes = attributes .replace (key,
                        Constants [key])
                expr = f' {predicate} {attributes}'
            return  Fact (expr)  if len(new_lhs) and
        all ([f. getResults() for  f in new_lhs])
        else
            None
```

```
class KB:
    def __init__(self):
        self. facts = set()
        self. implications = set()

    def tell (self, e):
        if '=>' in e:
            self. implications. 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 query (self, e):
        facts = set ( [f. expression for f in self. facts])
        i=1
        print ( f'Querying {e}:')
        for f in facts:
            if Fact(f). predicate = Fact(e) .predicate
            print( f'\t {i}. {f}')

            i+=1

        def display (self):
            print ("All facts")
            for 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)
```



```python
kb = KB()
kb.tell('missile(x)=>weapon(x)')
kb.tell('missile(M1)')
kb.tell('enemy(x,America)=>hostile(x)')
kb.tell('american(West)')
kb.tell('enemy(Nono,America)')
kb.tell('owns(Nono,M1)')
kb.tell('missile(x)&owns(Nono,x)=>sells(West,x,Nono)')
kb.tell('american(x)&weapon(y)&sells(x,y,z)&hostile(z)=>criminal(x)')
kb.query('criminal(x)')
kb.display()
```

```
Querying criminal(x):
        1. criminal(West)
All facts:
        1. enemy(Nono,America)
        2. hostile(Nono)
        3. sells(West,M1,Nono)
        4. criminal(West)
        5. owns(Nono,M1)
        6. weapon(M1)
        7. american(West)
        8. missile(M1)
```

```python
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)
```