# Lab :- 8

~~19~~ 19/1/23

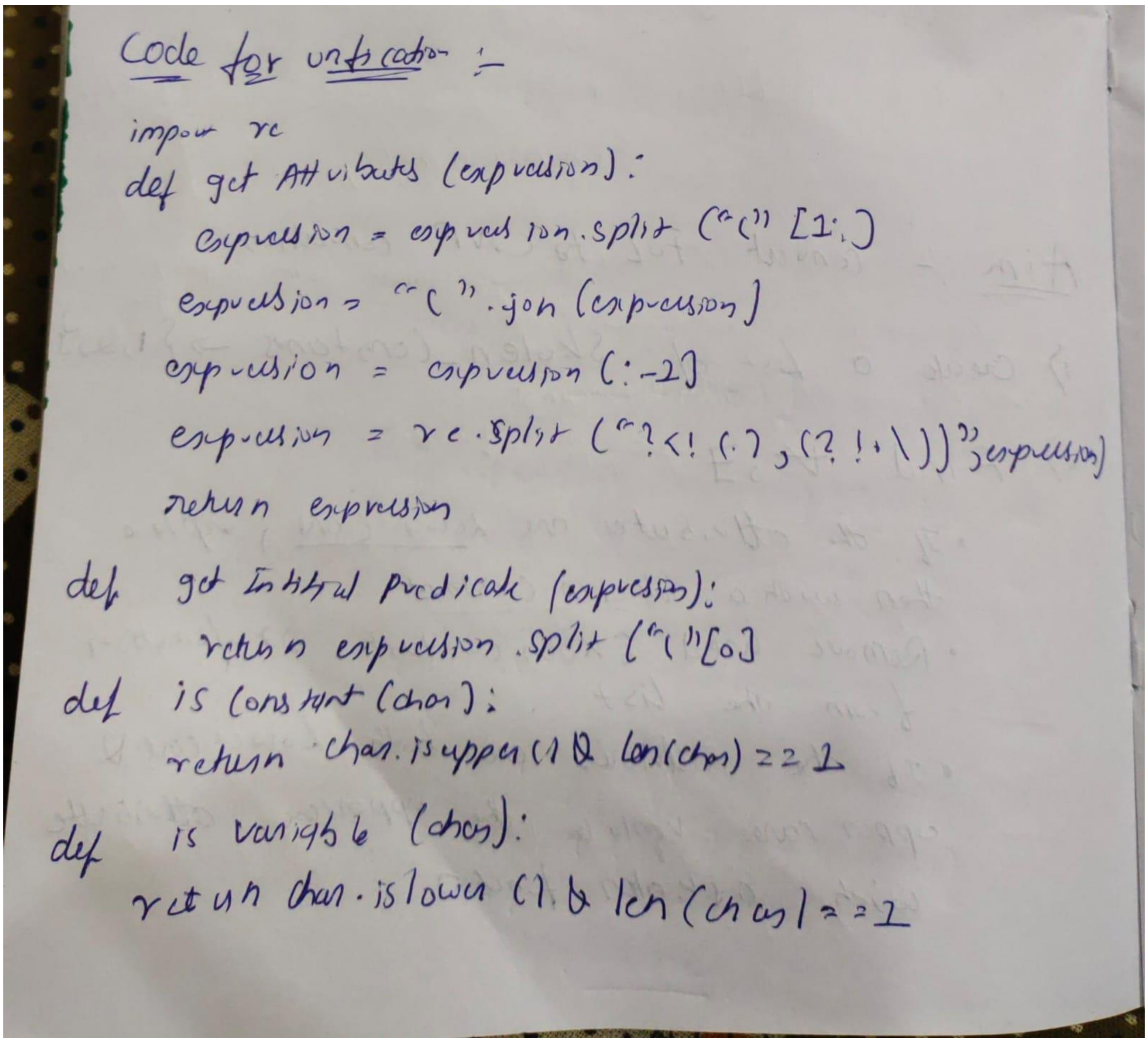
URA 19-1-23

**Aim** :- Implement unification in first order Logic

eg :- Knows (John, x) Knows (John, Jane)

{x / Jane}

```
Step 1 :- If term1 ~~&~~ or term 2 is a variable or
          constant then:
          a) Term 1 or Term2 are identical return
             NIL.
          b) Else if: term1 is a variable
                 if term1 occurs in term 2
                     return Fail
                 else
                     return {(term2 / term1)}
          c) else if term 2 is a variable
                 if term2 occurs in term1
                     return Fail
                 else
                     return {(term 1 / term2)}
          d) else return fail
Step2 :- If predicate (term1) ≠ Predicate (term2)
          return fail
Step3 :- Number of arguments ≠
             return fail
Step 4 :- Set (subst) to Nil
Step 5 :- For i=1 to the Number of elements in
          term 1
          a) call unify (i th term1, i th term2)
               put result into S
```

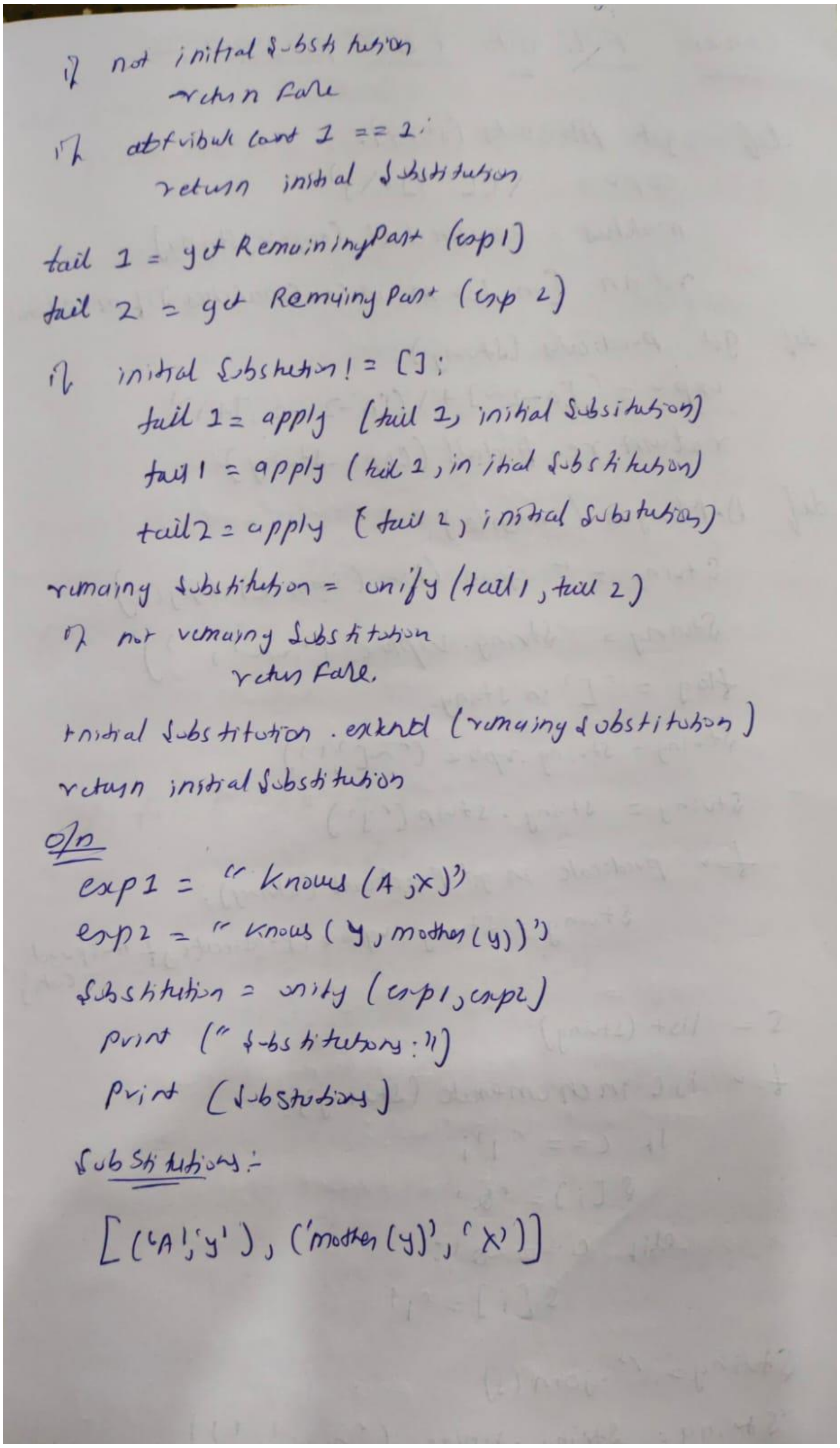


Code for unification :-

```
impor re
def get Attributes (expression):
    expression = expression.split ("(" [1:]
    expression = "(".join (expression)
    expression = expression [:-2]
    expression = re.split ("?<!(.),(?!.\))", expression)
    return expression
def get Initial Predicate (expression):
    return expression.split ("(")[0]
def is Constant (char):
    return char.isupper() & len(char) == 1
def is variable (char):
    return char.islower (). & len (char) == 1
```

```
def replace Attributes (exp, old, new):
        attributes = get Attributes(exp)
        for index, val in enumerate (attributes):
            if val == old:
                    attributes (index) = new
        predicate = get Initial predicate (exp)
        return predicate + "(" + ", ".join (attributes) + ")"

def apply (exp, substitutions):
    for substitution in substitutions:
            new, old = substitution
            exp = replace Attributes (exp, old, new)
        return exp

def check Occurs (var, exp):
        if exp.find (var) == -1:
            return False
        return True

def get Firstpart (expression):
    attributes = get Attributes (expression)
    return attributes (0)

def get RemainingPart (expression):
    predicate = get Initial predicate (expression]
    attributes = get Attributes (expression)
    new Expression = predicate + "(" + ".join (attributes[1:])
    return new Expression
```

```
def unify (exp1, exp2):
    if exp1 == exp2
        return[]
    if is Constant (exp1) & isConstant (exp2):
        if exp1 != exp2:
            return False.
    if is Constant (exp1):
        return [(exp1,exp2)]
    if is Constant (exp2):
        return [(exp2,exp1)]

    if isvariable (exp1):
        if check occurs (exp1,exp2):
            return False
    else
        return [(exp2,exp1)]
    if is variable (exp2):
        if check occurss (exp2,exp1):
            return False
        else
            return[(exp1, exp2)]

    if get Initial predicate (exp1) != get Initial predicate (exp2)
        print ("predicates do not match. cannot be
                                             unified")
        return False.
att ribute Count 1 = len (getAttributes (exp1))
attribute count 2 = len (get Attributes(exp2))
    if attribute count 1 != attribute count 2:
        return False
```

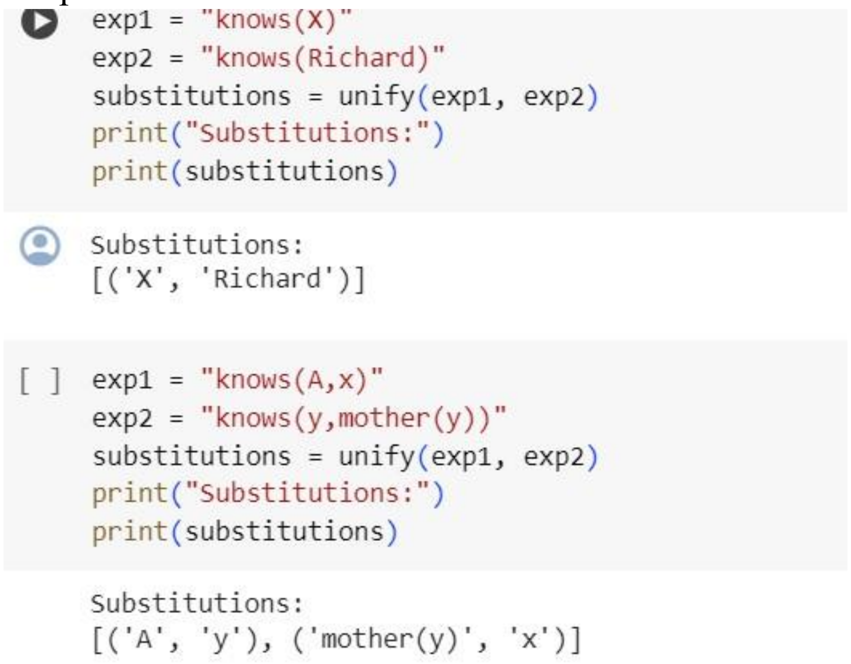


```
if not initial substitution
        return false
if attribute count 1 == 2:
        return initial substitution

tail 1 = get RemainingPart (exp1)
tail 2 = get RemainingPart (exp 2)

if initial substitution != []:
    tail 1 = apply (tail 1, initial Substitution)
    tail 1 = apply (tail 1, initial substitution)
    tail2 = apply (tail 2, initial substitution)
remaining substitution = unify (tail1, tail 2)
 if not remaining substitution
          return false.
initial substitution .extend (remaining substitution)
return initial substitution
```

o/p

```
 exp1 = " knows (A ,x)"
 exp2 = " knows ( y , mother (y))"
 substitution = unify (exp1, exp2)
  print (" substitutions :")
  print (substitutions)

 substitutions :-

 [('A','y'), ('mother (y)', 'x')]
```

Output:



```
exp1 = "knows(X)"
exp2 = "knows(Richard)"
substitutions = unify(exp1, exp2)
print("Substitutions:")
print(substitutions)
```

```
Substitutions:
[('X', 'Richard')]
```

```
[ ] exp1 = "knows(A,x)"
    exp2 = "knows(y,mother(y))"
    substitutions = unify(exp1, exp2)
    print("Substitutions:")
    print(substitutions)
```

```
Substitutions:
[('A', 'y'), ('mother(y)', 'x')]
```