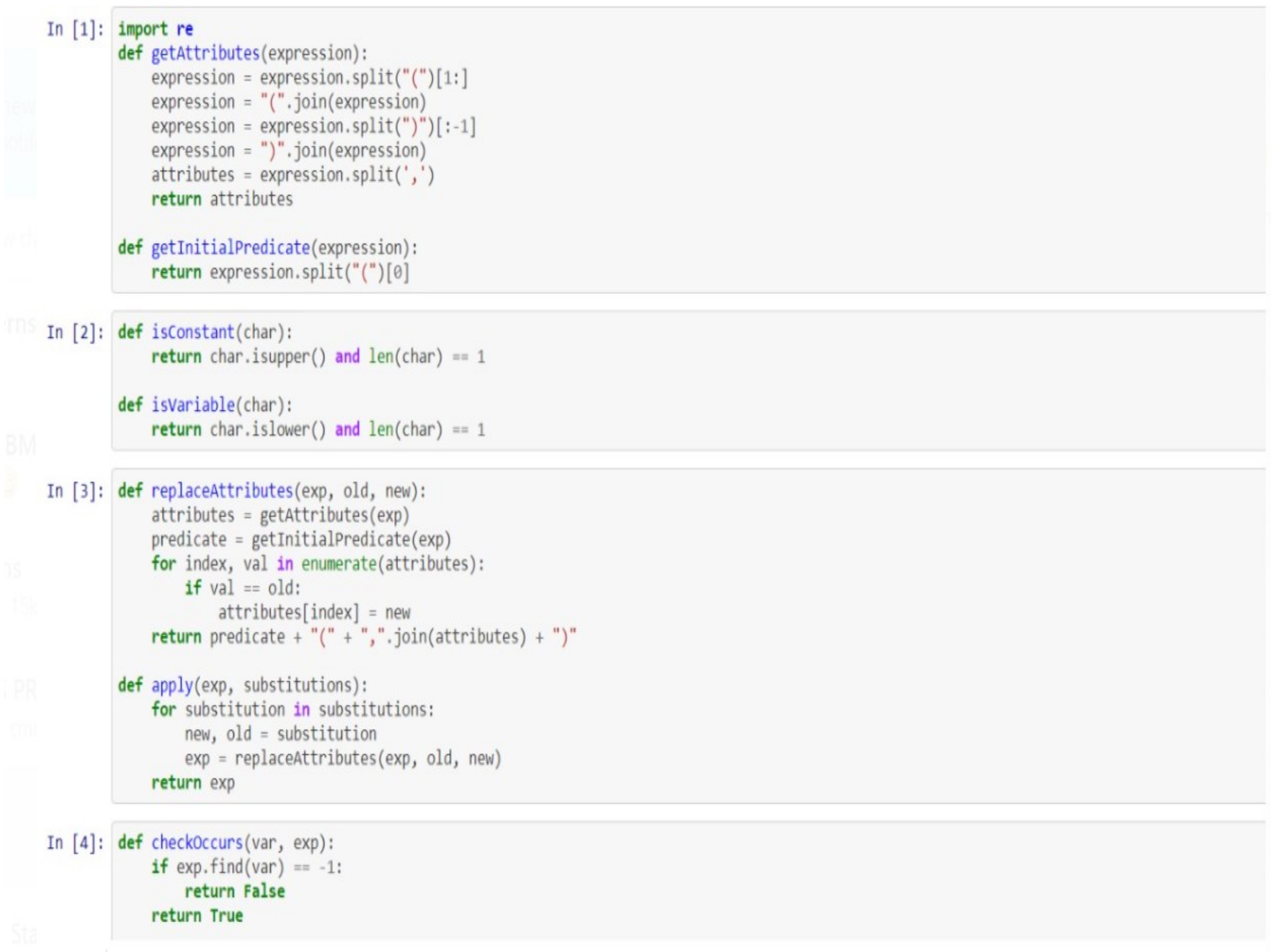
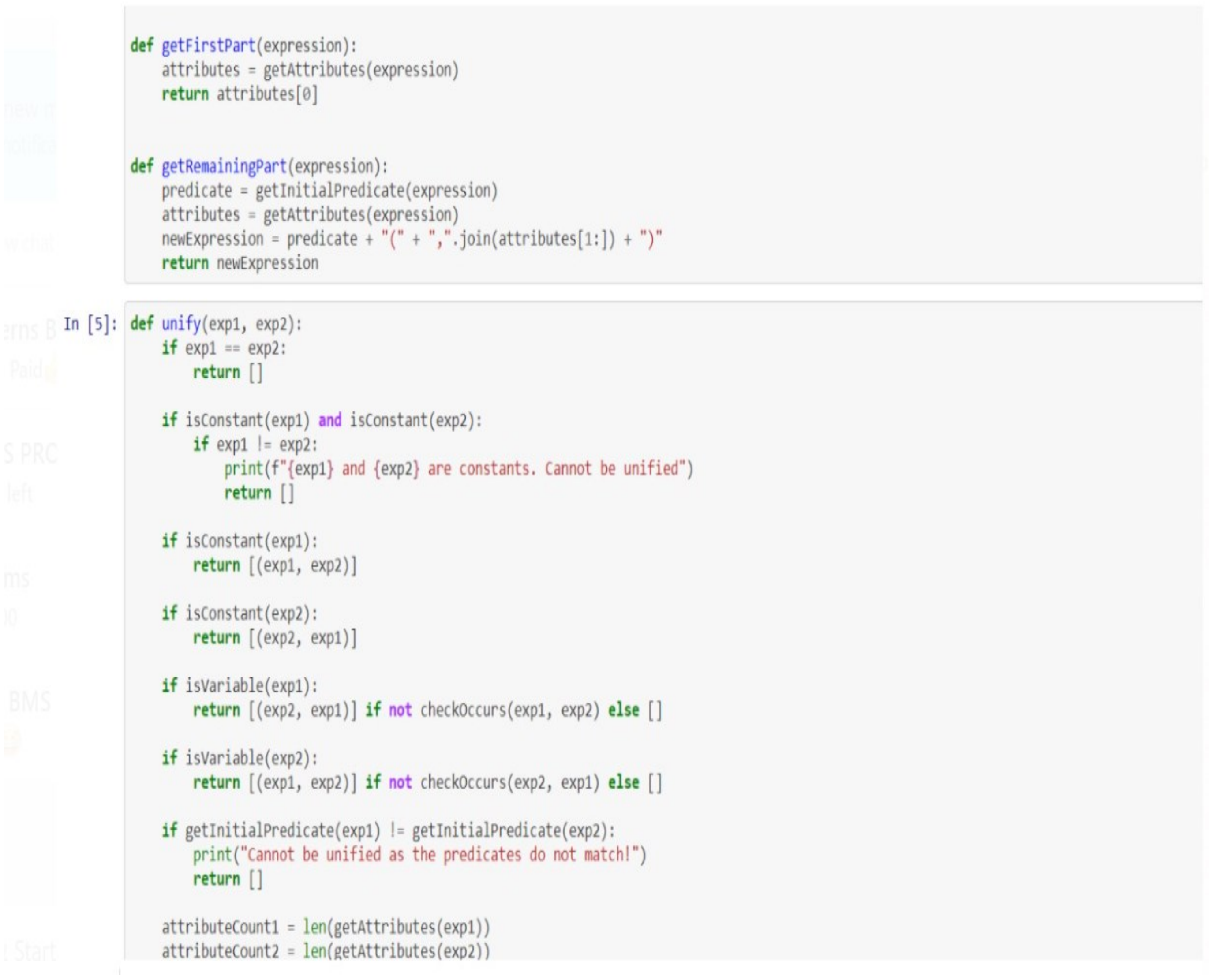
CODE IMPLEMENTATION

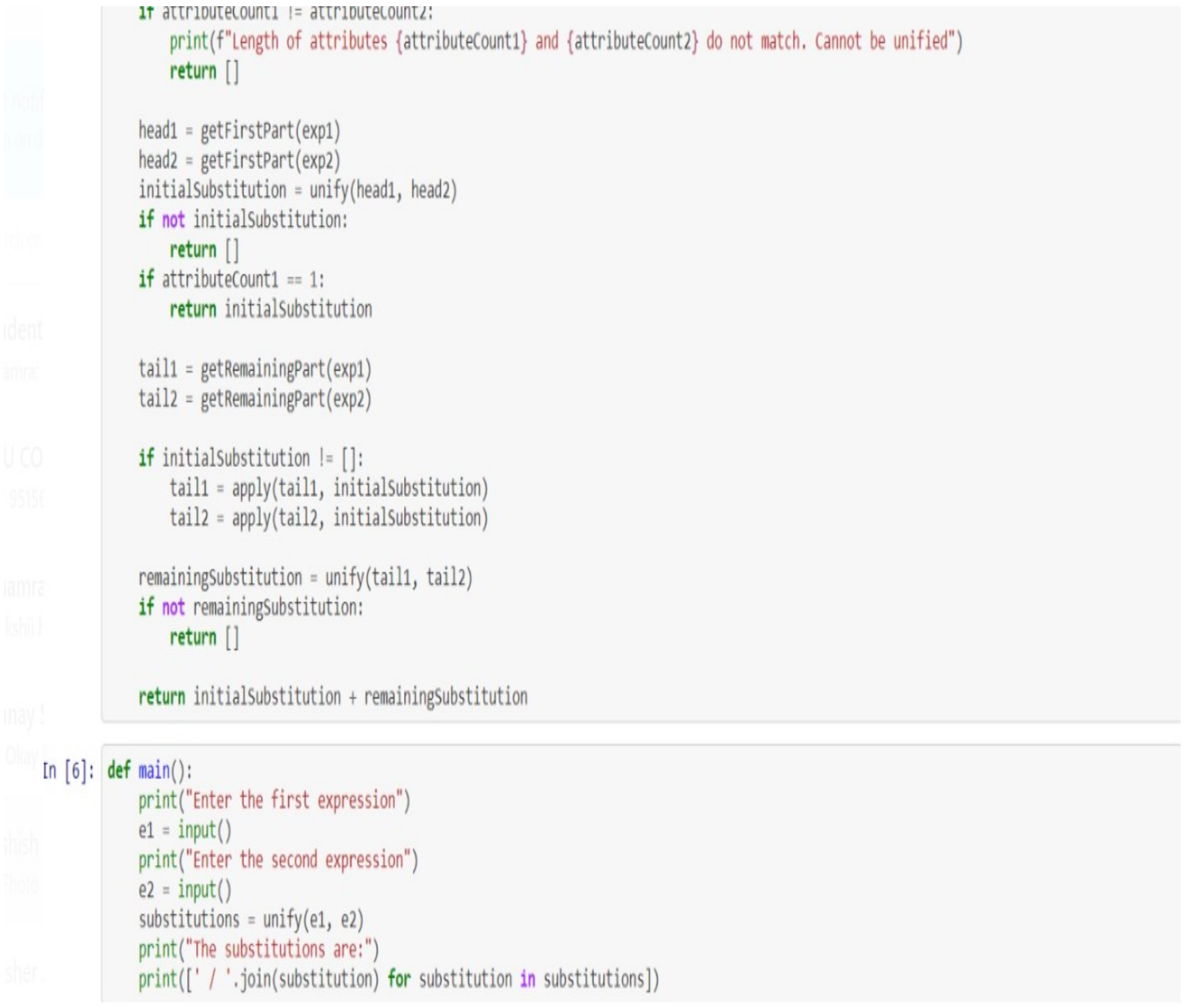


```
In [1]: import re
        def getAttributes(expression):
            expression = expression.split("(")[1:]
            expression = "(".join(expression)
            expression = expression.split(")")[:-1]
            expression = ")".join(expression)
            attributes = expression.split(',')
            return attributes

        def getInitialPredicate(expression):
            return expression.split("(")[0]

In [2]: def isConstant(char):
            return char.isupper() and len(char) == 1

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

In [3]: def replaceAttributes(exp, old, new):
            attributes = getAttributes(exp)
            predicate = getInitialPredicate(exp)
            for index, val in enumerate(attributes):
                if val == old:
                    attributes[index] = new
            return predicate + "(" + ",".join(attributes) + ")"

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

In [4]: def checkOccurs(var, exp):
            if exp.find(var) == -1:
                return False
            return True
```



```python
def getFirstPart(expression):
    attributes = getAttributes(expression)
    return attributes[0]


def getRemainingPart(expression):
    predicate = getInitialPredicate(expression)
    attributes = getAttributes(expression)
    newExpression = predicate + "(" + ",".join(attributes[1:]) + ")"
    return newExpression
```

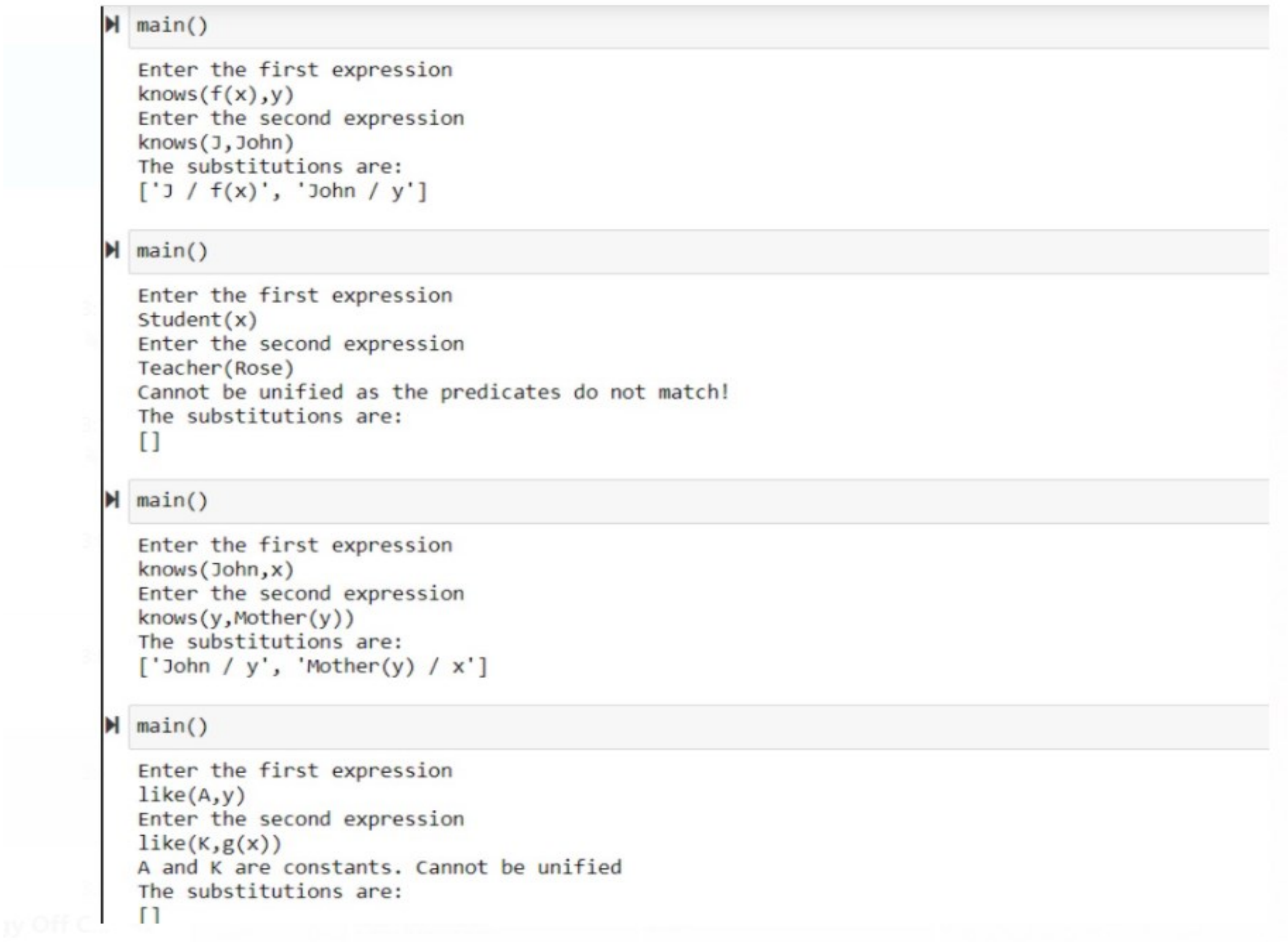
```python
In [5]: def unify(exp1, exp2):
            if exp1 == exp2:
                return []

            if isConstant(exp1) and isConstant(exp2):
                if exp1 != exp2:
                    print(f"{exp1} and {exp2} are constants. Cannot be unified")
                    return []

            if isConstant(exp1):
                return [(exp1, exp2)]

            if isConstant(exp2):
                return [(exp2, exp1)]

            if isVariable(exp1):
                return [(exp2, exp1)] if not checkOccurs(exp1, exp2) else []

            if isVariable(exp2):
                return [(exp1, exp2)] if not checkOccurs(exp2, exp1) else []

            if getInitialPredicate(exp1) != getInitialPredicate(exp2):
                print("Cannot be unified as the predicates do not match!")
                return []

            attributeCount1 = len(getAttributes(exp1))
            attributeCount2 = len(getAttributes(exp2))
```



```
    if attributeCount1 != attributeCount2:
        print(f"Length of attributes {attributeCount1} and {attributeCount2} do not match. Cannot be unified")
        return []

    head1 = getFirstPart(exp1)
    head2 = getFirstPart(exp2)
    initialSubstitution = unify(head1, head2)
    if not initialSubstitution:
        return []
    if attributeCount1 == 1:
        return initialSubstitution

    tail1 = getRemainingPart(exp1)
    tail2 = getRemainingPart(exp2)

    if initialSubstitution != []:
        tail1 = apply(tail1, initialSubstitution)
        tail2 = apply(tail2, initialSubstitution)

    remainingSubstitution = unify(tail1, tail2)
    if not remainingSubstitution:
        return []

    return initialSubstitution + remainingSubstitution
```

```
In [6]: def main():
            print("Enter the first expression")
            e1 = input()
            print("Enter the second expression")
            e2 = input()
            substitutions = unify(e1, e2)
            print("The substitutions are:")
            print([' / '.join(substitution) for substitution in substitutions])
```

OUTPUT



```
main()

Enter the first expression
knows(f(x),y)
Enter the second expression
knows(J,John)
The substitutions are:
['J / f(x)', 'John / y']
```

```
main()

Enter the first expression
Student(x)
Enter the second expression
Teacher(Rose)
Cannot be unified as the predicates do not match!
The substitutions are:
[]
```

```
main()

Enter the first expression
knows(John,x)
Enter the second expression
knows(y,Mother(y))
The substitutions are:
['John / y', 'Mother(y) / x']
```

```
main()

Enter the first expression
like(A,y)
Enter the second expression
like(K,g(x))
A and K are constants. Cannot be unified
The substitutions are:
[]
```