**Artificial Intelligence Lab**

**LAB 7 – Unification and Resolution**

**Aditya Kumar**

**RA1911030010100**

**O2 Section**

**Date – 15/3/22**

**Problem Statement :**

Implementation of unification

**Algorithm :**

1. Initialize the substitution set to be empty.
2. Recursively unify atomic sentences:

* Check for Identical expression match.
* If one expression is a variable vi, and the other is a term ti which does not contain variable vi, then:
* Substitute ti / vi in the existing substitutions
* Add ti /vi to the substitution setlist.
* If both the expressions are functions, then function name must be similar, and the number of arguments must be the same in both the expression.

For each pair of the following atomic sentences find the most general unifier (If exist).

**Code:**

def get\_index\_comma(string):

index\_list = list()

par\_count = 0

for i in range(len(string)):

if string[i] == ',' and par\_count == 0:

index\_list.append(i)

elif string[i] == '(':

par\_count += 1

elif string[i] == ')':

par\_count -= 1

return index\_list

def is\_variable(expr):

for i in expr:

if i == '(' or i == ')':

return False

return True

def process\_expression(expr):

expr = expr.replace(' ', '')

index = None

for i in range(len(expr)):

if expr[i] == '(':

index = i

break

predicate\_symbol = expr[:index]

expr = expr.replace(predicate\_symbol, '')

expr = expr[1:len(expr) - 1]

arg\_list = list()

indices = get\_index\_comma(expr)

if len(indices) == 0:

arg\_list.append(expr)

else:

arg\_list.append(expr[:indices[0]])

for i, j in zip(indices, indices[1:]):

arg\_list.append(expr[i + 1:j])

arg\_list.append(expr[indices[len(indices) - 1] + 1:])

return predicate\_symbol, arg\_list

def get\_arg\_list(expr):

\_, arg\_list = process\_expression(expr)

flag = True

while flag:

flag = False

for i in arg\_list:

if not is\_variable(i):

flag = True

\_, tmp = process\_expression(i)

for j in tmp:

if j not in arg\_list:

arg\_list.append(j)

arg\_list.remove(i)

return arg\_list

def check\_occurs(var, expr):

arg\_list = get\_arg\_list(expr)

if var in arg\_list:

return True

return False

def unify(expr1, expr2):

if is\_variable(expr1) and is\_variable(expr2):

if expr1 == expr2:

return 'Null'

else:

return False

elif is\_variable(expr1) and not is\_variable(expr2):

if check\_occurs(expr1, expr2):

return False

else:

tmp = str(expr2) + '/' + str(expr1)

return tmp

elif not is\_variable(expr1) and is\_variable(expr2):

if check\_occurs(expr2, expr1):

return False

else:

tmp = str(expr1) + '/' + str(expr2)

return tmp

else:

predicate\_symbol\_1, arg\_list\_1 = process\_expression(expr1)

predicate\_symbol\_2, arg\_list\_2 = process\_expression(expr2)

# Step 2

if predicate\_symbol\_1 != predicate\_symbol\_2:

return False

# Step 3

elif len(arg\_list\_1) != len(arg\_list\_2):

return False

else:

# Step 4: Create substitution list

sub\_list = list()

# Step 5:

for i in range(len(arg\_list\_1)):

tmp = unify(arg\_list\_1[i], arg\_list\_2[i])

if not tmp:

return False

elif tmp == 'Null':

pass

else:

if type(tmp) == list:

for j in tmp:

sub\_list.append(j)

else:

sub\_list.append(tmp)

# Step 6

return sub\_list

if \_\_name\_\_ == '\_\_main\_\_':

f1 = 'Q(a, g(x, a), f(y))'

f2 = 'Q(a, g(f(b), a), x)'

# f1 = input('f1 : ')

# f2 = input('f2 : ')

result = unify(f1, f2)

if not result:

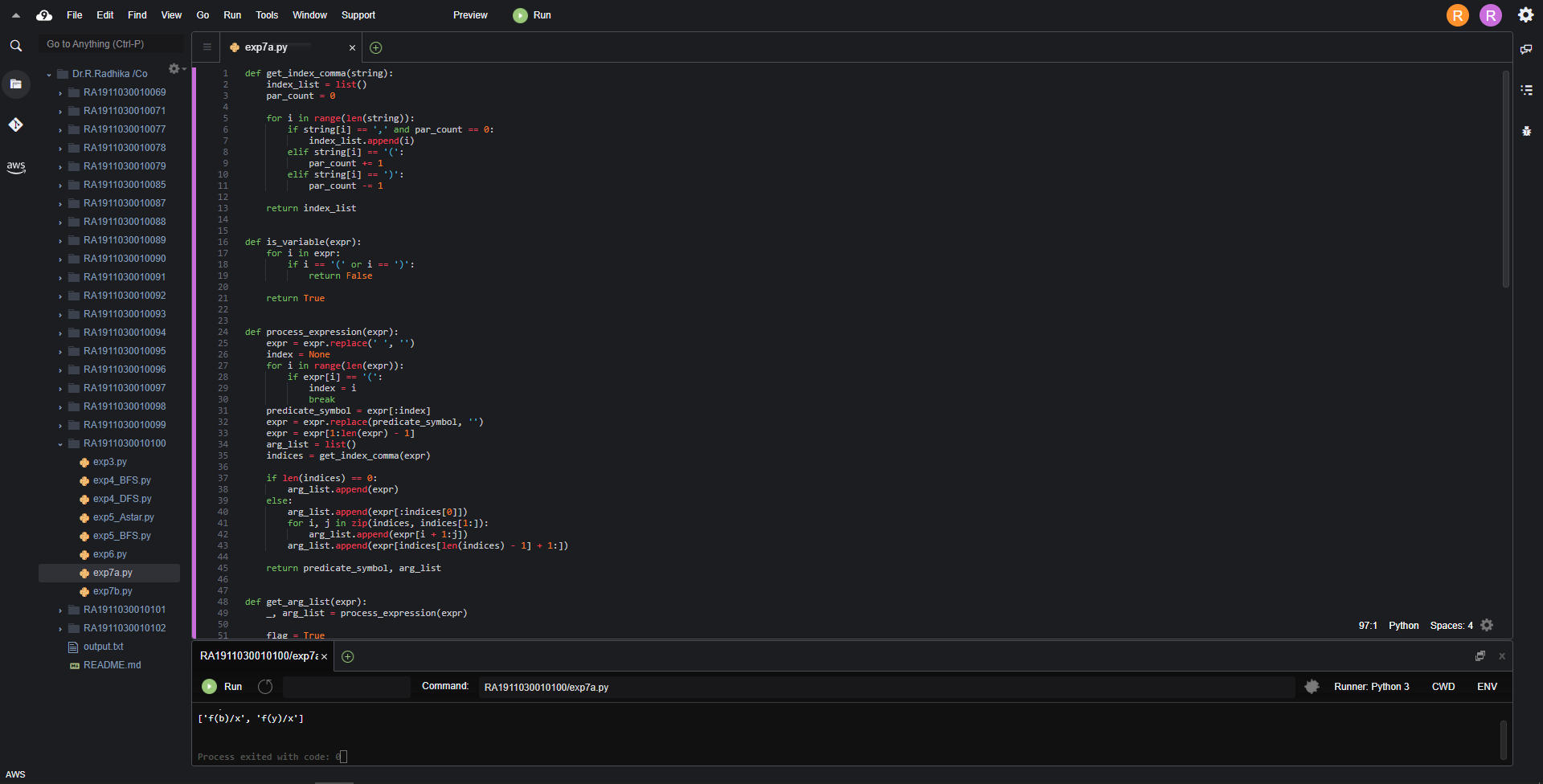
print('The process of Unification failed!')

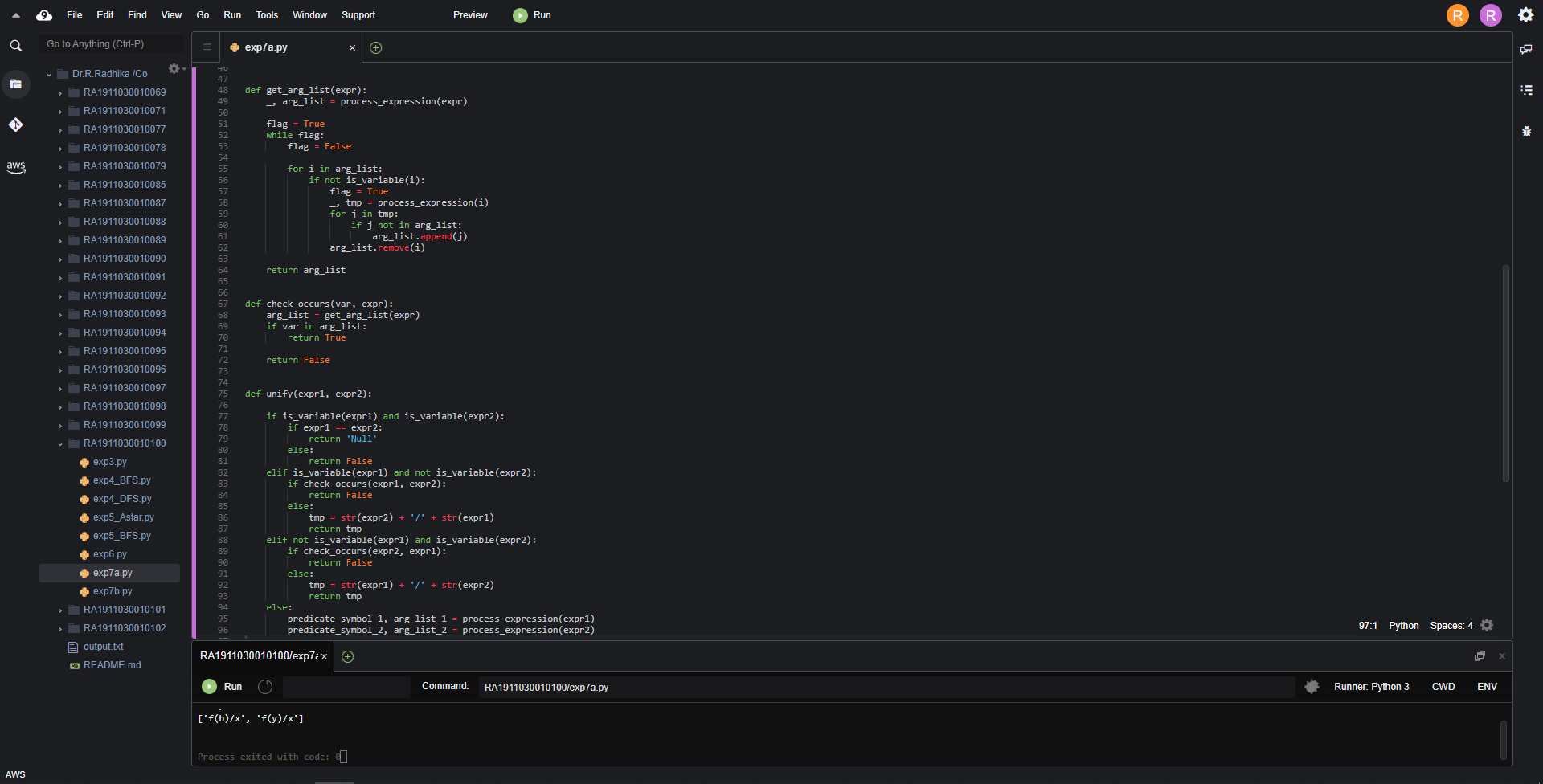
else:

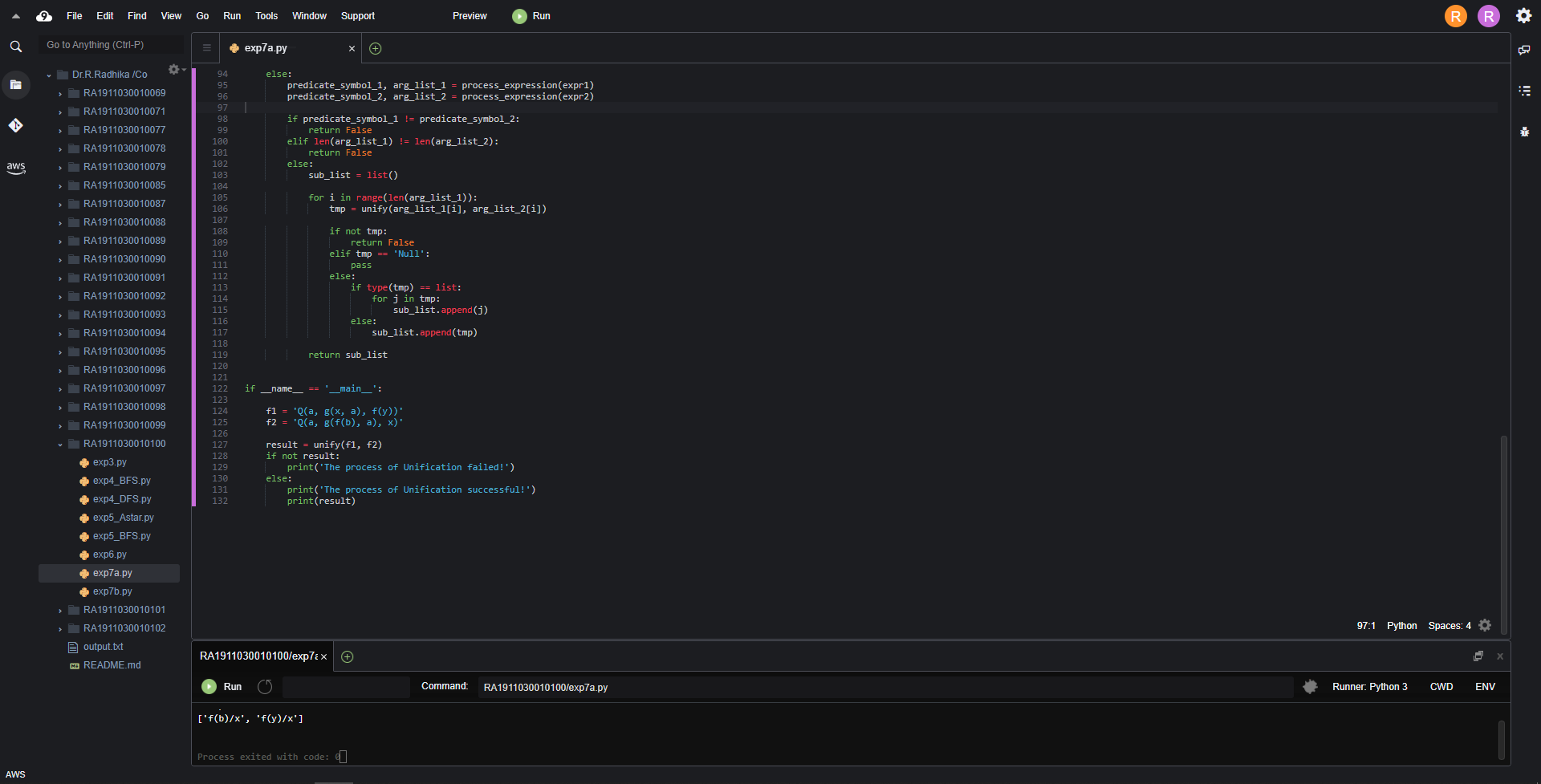
print('The process of Unification successful!')

print(result)

**Output:**







**Result:**

Hence we performed unification.