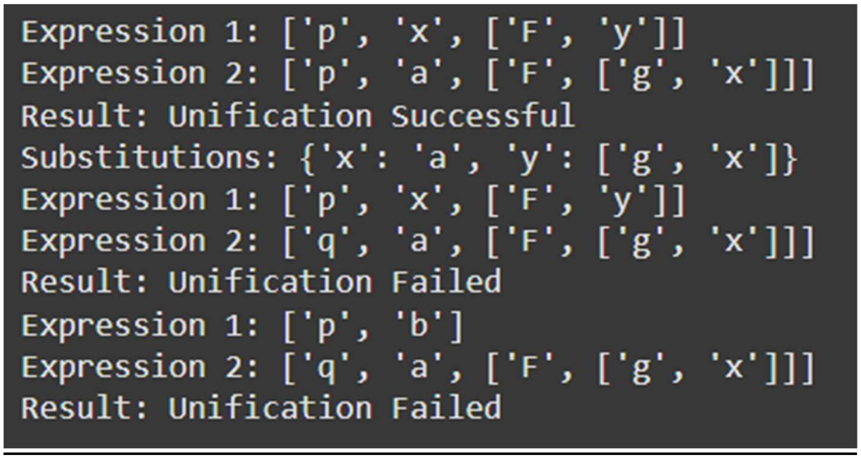
Lab-7: Unification using FOL:

```
def unify(x, y, subst=None):
    """
    Unifies two expressions x and y and returns the substitution set if they can be unified.
    Returns 'FAILURE' if unification is not possible.
    """
    if subst is None:
        subst = {}  # Initialize an empty substitution set

    # Step 1: Handle cases where x or y is a variable or constant
    if x == y:  # If x and y are identical
        return subst
    elif isinstance(x, str) and x.islower():  # If x is a variable
        return unify_var(x, y, subst)
    elif isinstance(y, str) and y.islower():  # If y is a variable
        return unify_var(y, x, subst)
    elif isinstance(x, list) and isinstance(y, list):  # If x and y are compound expressions (lists)
        if len(x) != len(y):  # Step 3: Different number of arguments
            return "FAILURE"

        # Step 2: Check if the functors (first elements) are the same
        if x[0] != y[0]:
            return "FAILURE"

        # Step 5: Recursively unify each element (skip the first element, as it's the functor)
        for xi, yi in zip(x[1:], y[1:]):
            subst = unify(xi, yi, subst)
            if subst == "FAILURE":
                return "FAILURE"

        return subst
    else:  # If x and y are different constants or non-unifiable structures
        return "FAILURE"


def unify_var(var, x, subst):
    """
    Handles unification of a variable with another term.
    """
    if var in subst:  # If var is already substituted
        return unify(subst[var], x, subst)
    elif isinstance(x, (list, tuple)) and tuple(x) in subst:  # Handle compound expressions
        return unify(var, subst[tuple(x)], subst)
    elif occurs_check(var, x):  # Check for circular references
        return "FAILURE"
    else:
        # Add the substitution to the set (convert list to tuple for hashability)
        subst[var] = tuple(x) if isinstance(x, list) else x
        return subst


def occurs_check(var, x):
    """
    Checks if var occurs in x (to prevent circular substitutions).
    """
    if var == x:
        return True
    elif isinstance(x, list):  # If x is a compound expression
        return any(occurs_check(var, xi) for xi in x)
    return False


# Helper function to perform unification and return a result status
def unify_and_check(expr1, expr2):
    """
    Attempts to unify two expressions and returns a tuple:
    (is_unified: bool, substitutions: dict or None)
    """
    result = unify(expr1, expr2)
    if result == "FAILURE":
        return False, None
    return True, result


# Helper function to display results
def display_result(expr1, expr2, is_unified, subst):
    print("Expression 1:", expr1)
    print("Expression 2:", expr2)
    if not is_unified:
        print("Result: Unification Failed")
    else:
        print("Result: Unification Successful")
        print("Substitutions:", {k: list(v) if isinstance(v, tuple) else v for k, v in subst.items()})


# Example usage
if __name__ == "__main__":
    # Correct representation of the expressions
    expr1 = ["p", "x", ["F", "y"]]  # Represents p(a, F(y))
    expr2 = ["p", "a", ["F", ["g", "x"]]]  # Represents p(a, F(g(x)))
    # Perform unification
    is_unified, result = unify_and_check(expr1, expr2)

    # Display the results
    display_result(expr1, expr2, is_unified, result)

    # Test with a case where the functors don't match
    expr1 = ["p", "x", ["F", "y"]]  # Represents p(x, F(y))
    expr2 = ["q", "a", ["F", ["g", "x"]]]  # Represents q(a, F(g(x)))

    # Perform unification
    is_unified, result = unify_and_check(expr1, expr2)

    # Display the results
    display_result(expr1, expr2, is_unified, result)

    # Test with a case where the functors don't match
    expr1 = ["p","b"]  # Represents p(x, F(y))
    expr2 = ["q", "a", ["F", ["g", "x"]]]  # Represents q(a, F(g(x)))

    # Perform unification
    is_unified, result = unify_and_check(expr1, expr2)

    # Display the results
    display_result(expr1, expr2, is_unified, result)Output:
```

Output:



```
Expression 1: ['p', 'x', ['F', 'y']]
Expression 2: ['p', 'a', ['F', ['g', 'x']]]
Result: Unification Successful
Substitutions: {'x': 'a', 'y': ['g', 'x']}
Expression 1: ['p', 'x', ['F', 'y']]
Expression 2: ['q', 'a', ['F', ['g', 'x']]]
Result: Unification Failed
Expression 1: ['p', 'b']
Expression 2: ['q', 'a', ['F', ['g', 'x']]]
Result: Unification Failed
```