Algorithme de résolution
Réalisé par ibtissam ech chaibi

1. parse_formula_input:La fonction `parse_formula_input(input_string)` analyse une chaîne de caractères représentant une formule logique, extrayant les clauses en les identifiant entre accolades et en les séparant par des virgules. Chaque clause est ensuite divisée en littéraux, où les espaces sont retirés et les négations sont traitées pour convertir les variables en entiers ASCII, ajustés pour démarrer à partir de 1. Ces littéraux sont ensuite stockés dans une liste de clauses, qui est renvoyée pour permettre des manipulations ultérieures telles que la transformation en forme normale conjonctive et la résolution.

```
import re
def parse_formula_input(input_string):
    clauses_match = re.findall(r'\{(.*?)\}', input_string)
    if not clauses_match:
        raise ValueError("Invalid input format. Please use the format {¬P v ¬Q v R, ¬R, P, ¬T v Q, T}")
    clauses = [re.split(r'\s*\^\s*', clause) for clause in clauses_match[0].split(',')]
    processed_clauses = []
    for clause in clauses:
        processed_clause = []
        for literal in clause:
            literal = literal.strip()
            if literal.startswith("¬"):
                processed_clause.append(-ord(literal[1]) + ord("A") + 1)
            else:
                processed_clause.append(ord(literal) - ord("A") + 1)
        processed_clauses.append(processed_clause)
    return processed_clauses
```

1. Fonction negation(F) : Cette fonction prend une formule F en entrée. Elle vérifie si le premier élément de F est une liste. Si c'est le cas, cela signifie que F est une liste de clauses, donc elle négative chaque littéral dans chaque clause. Si F n'est pas une liste de clauses (c'est-à-dire, c'est un seul littéral), elle négative ce littéral. Elle renvoie la formule négative.

```
def negation(F):
    if isinstance(F[0], list):
        return [[-literal for literal in clause] for clause in F]
    else:
        return [-F]
```

2. Fonction mettre_sous_forme_de_clauses(F) : Cette fonction s'assure que la formule d'entrée F est sous forme d'une liste de clauses. Si le premier élément de F est un entier (un seul littéral), elle l'encadre dans une liste pour créer une formule à une seule clause. Si F est déjà une liste de clauses, elle la renvoie telle quelle.

```
def mettre_sous_forme_de_clauses(F):
    return [F] if isinstance(F[0], int) else F
```

3. Fonction chercher_clauses_resolvantes(clauses) : Cette fonction prend une liste de clauses clauses en entrée. Elle itère à travers chaque paire de clauses dans clauses. Pour chaque paire de clauses, elle vérifie

chaque combinaison de littéraux pour trouver des paires qui sont complémentaires (ont des signes opposés). Si elle trouve des littéraux complémentaires, elle crée un résolvant en supprimant ces littéraux de leurs clauses respectives et en combinant les littéraux restants. Elle ajoute le résolvant à la liste de résultats s'il n'est pas déjà présent. Enfin, elle renvoie la liste des résolvants uniques.

```
def chercher_clauses_resolvantes(clauses):
    result = []
    for i in range(len(clauses)):
        for j in range(i+1, len(clauses)):
            for literal_i in clauses[i]:
                for literal_j in clauses[j]:
                    if abs(literal_i) == abs(literal_j):
                        resolvant = [literal for literal in clauses[i] if literal != literal_i] + \
                                    [literal for literal in clauses[j] if literal != literal_j]
                        if resolvant not in result:
                            result.append(resolvant)
    return result
```

4. Fonction resolution(F) : Cette fonction implémente la méthode de résolution pour vérifier la validité d'une formule F. Elle commence par négativer la formule d'entrée F et la convertir en une liste de clauses. Elle entre dans une boucle où elle recherche à plusieurs reprises des résolvants jusqu'à ce qu'aucun nouveau résolvant ne puisse être trouvé. À l'intérieur de la boucle, elle vérifie s'il y a des nouveaux résolvants. S'il y a de nouveaux résolvants, elle les ajoute à la liste de clauses. Elle vérifie la présence d'une clause vide, ce qui indique que la formule est insatisfaisable. Si aucune clause vide n'est trouvée, elle continue de rechercher de nouveaux résolvants. Si aucun nouveau résolvant ne peut être trouvé, elle renvoie "F est valide", indiquant que la formule est valide.

```
def resolution(F):
    neg_F = negation(F)
    clauses = mettre_sous_forme_de_clauses(neg_F)
    while True:
        new_resolvantes = chercher_clauses_resolvantes(clauses)
        if not new_resolvantes:
            return "F est valide"
        for resolvant in new_resolvantes:
            if all(lit not in clauses for lit in resolvant):
                clauses.append(resolvant)
        if [] in clauses:
            return "F est invalide"
```

5. La fonction test_resolution() permet à l'utilisateur d'entrer une formule logique à tester, puis elle utilise la fonction resolution() pour déterminer si cette formule est valide ou invalide.

```python
def test_resolution():
    formula_input = input("Enter the formula (e.g., {¬P ∨ ¬Q ∨ R, ¬R, P, ¬T ∨ Q}): ")
    try:
        formulas = formula_input.split(';')
        for formula_str in formulas:
            formula = parse_formula_input(formula_str)
            print("Negation:", negation(formula))
            print("Formula:", formula)
            print("Result:", resolution(formula))
            print()
    except ValueError as e:
        print(e)
        return

test_resolution()
```

6. exécution

```
test_resolution()
```

```
Enter the formula (e.g., {¬P ∨ ¬Q ∨ R, ¬R, P, ¬T ∨ Q}):
```

=> formule valide

```
Enter the formula (e.g., {¬P ∨ ¬Q ∨ R, ¬R, P, ¬T ∨ Q}):  {¬P ∨ Q ∨ R, P, ¬Q}
Negation: [[14], [-16], [15]]
Formula: [[-14], [16], [-15]]
Result: F est valide
```

=> formule invalide

```
Enter the formula (e.g., {¬P ∨ ¬Q ∨ R, ¬R, P, ¬T ∨ Q}): {¬P ∨ ¬Q ∨ R, ¬R, P, ¬T }
Negation: [[14], [16], [-16], [18]]
Formula: [[-14], [-16], [16], [-18]]
Result: F est invalide
```