## Are **Python** and **Python Logic Framework** the same?

| **Feature** | **Regular Python** | **Python Logic Framework** |
| --- | --- | --- |
| Example | Using +, -, \*, / directly in code | Using libraries like kanren to define logic rules |
| Programming style | Procedural / Functional | Declarative / Logic-based |
| Code style | result = a + b | Define a rule add(a, b, result) and query it |
| Common use | Basic programs, apps, scripts | AI, logic-based reasoning, rule inference |
| Similar to | C, Java, etc. | Prolog |

**Title: Representing Family Relationships Using Python Logic Framework**

## Tools Required

We use a Python library called **kanren** (formerly called logic programming in Python) to perform this.

📌 Install this library first:

pip install kanren

# Step 1: Import necessary tools from kanren  
from kanren import Relation, facts, run, var, conde

**Explanation:**

* Relation lets us define a new logical relationship (like parent, male, female).
* facts is used to add known truths into those relationships.
* run is used to ask logical questions (queries) and get answers.
* var() creates logic variables (like X, Y) that we want to find.
* conde allows us to write logic rules (like if someone is female and a parent, then she is a mother).

# Step 2: Define relationship containers  
parent = Relation()  
male = Relation()  
female = Relation()

**Explanation:**

* These are like empty folders where we will store information (facts).
* parent will hold parent-child pairs.
* male and female will hold names of male and female people respectively.

# Step 3: Add known facts to the relationships  
facts(parent,  
 ('John', 'Mary'),  
 ('John', 'Tom'),  
 ('Linda', 'Mary'),  
 ('Linda', 'Tom'))  
  
facts(male, 'John', 'Tom')  
facts(female, 'Linda', 'Mary')

**Explanation:**

* We are stating real-world facts:
  + John is the parent of Mary and Tom
  + Linda is the parent of Mary and Tom
  + John and Tom are male
  + Linda and Mary are female

# Step 4: Define logical rules using conde  
  
def mother(x, y):  
 return conde((female(x), parent(x, y)))  
  
def father(x, y):  
 return conde((male(x), parent(x, y)))

**Explanation:**

* We are creating two logic rules:
  + mother(x, y) means: x is the mother of y if x is a female AND x is a parent of y.
  + father(x, y) means: x is the father of y if x is a male AND x is a parent of y.
* conde is used to express logical AND conditions.
* This is similar to rule definition in Prolog.

# Step 5: Declare a logic variable  
x = var()

**Explanation:**

* x is a logic variable, like a blank space.
* We will use x to ask questions like: Who is the father of Mary?

# Step 6: Ask logic questions (queries) and print answers  
  
print("Mother of Tom:", run(1, x, mother(x, 'Tom')))  
print("Father of Mary:", run(1, x, father(x, 'Mary')))  
print("Children of John:", run(2, x, parent('John', x)))  
print("All females in the family:", run(10, x, female(x)))

**Explanation:**

* run(n, x, condition) is used to perform a logic query.
  + n is the number of results you want (1 = first answer, 10 = all possible answers)
  + x is the variable you’re solving for
  + condition is the rule or fact to use in the query

**Examples:**

* mother(x, 'Tom') → Who is the mother of Tom?
* parent('John', x) → What are the children of John?
* female(x) → Who is female?

**Output:**

Mother of Tom: ('Linda',)  
Father of Mary: ('John',)  
Children of John: ('Mary', 'Tom')  
All females in the family: ('Linda', 'Mary')

What Learn from This:

* How to write facts using facts()
* How to define logic rules with conde()
* How to create variables using var()
* How to run logic queries using run()

### Summary Table:

| Concept | Tool Used | Example |
| --- | --- | --- |
| Fact | facts() | facts(parent, ('A', 'B')) |
| Rule | conde() | mother(x, y) rule |
| Variable | var() | x = var() |
| Query | run() | run(1, x, mother(x, 'Tom')) |

# family\_relationship.py

# This program uses the 'kanren' logic programming library to represent and query family relationships.

# Step 1: Import necessary functions from the kanren library

from kanren import Relation, facts, run, var, conde

# Step 2: Create empty relationship containers

# These will store facts about parent-child, male, and female relationships

parent = Relation() # Parent-child relationship

male = Relation() # Male people

female = Relation() # Female people

# Step 3: Add facts into each relationship

# These facts represent known family relationships

# parent('Parent', 'Child')

facts(parent,

('John', 'Mary'),

('John', 'Tom'),

('Linda', 'Mary'),

('Linda', 'Tom'))

# List of males

facts(male, 'John', 'Tom')

# List of females

facts(female, 'Linda', 'Mary')

# Step 4: Define rules using conde (logical AND conditions)

# Rule for mother:

# A person X is the mother of Y if X is female AND X is a parent of Y

def mother(x, y):

return conde((female(x), parent(x, y)))

# Rule for father:

# A person X is the father of Y if X is male AND X is a parent of Y

def father(x, y):

return conde((male(x), parent(x, y)))

# Step 5: Create a logic variable

# This variable will be used in queries to find answers

x = var()

# Step 6: Ask logic queries and print the results

# Who is the mother of Tom?

print("Mother of Tom:", run(1, x, mother(x, 'Tom')))

# Who is the father of Mary?

print("Father of Mary:", run(1, x, father(x, 'Mary')))

# Who are the children of John?

print("Children of John:", run(2, x, parent('John', x)))

# List all females in the family

print("All females in the family:", run(10, x, female(x)))

**Output**

Mother of Tom: ('Linda',)

Father of Mary: ('John',)

Children of John: ('Mary', 'Tom')

All females in the family: ('Linda', 'Mary')

1. Add a new person (e.g., ‘Mary’ is parent of ‘Sara’)
2. Define a new rule for sibling (x, y)
3. Write queries to find grandparents