

# **Artificial Intelligence**

## **Assignment 4**

### **Report**

#### **Submitted by Group**

- **Anayat: BSCS-2019-52**
  - **Bakht Ullah:-55**
- **Tehreem Fatima: BSCS-2019-26**
  - **Sania Bibi: BSCS-2019-67**
  - **Seemi Shaheen: BSCS-2019-65**
  - **Kainat Bibi: BSCS-2019-64**
- **Jamshed Masood: BSCS-2019-21**

## **Table of contents:**

**Abstract.....**

**1): Introduction.....**

**2): Background.....**

2.1): Artificial Intelligence.....

2.2):Defining\_Khan\_FamilyRelation.....

2.3): Conclusion .....

2.4):Extending\_Khan\_Family\_Relation\_By\_Rule.....

2.5): Conclusion.....

2.6): A recursive rule definition.....

2.7): How Prolog answers the question.....

**3): Pyswip.....**

**8): Testing.....**

**9): Conclusion .....**

**10): References.....**

## **Abstract**

SWI prolog is not the commercial prolog or not the academic enterprises, but it is the community project. Its core system is build according to the current prototypes, mostly for the knowledge –intensive and the interactive system. Community Contributions further added the several interfaces and the libraries. Commercial involvement has the initial garbage collector, added the interfaces and some tools. One is the literature programming documentation system and the other is the unit testing environment.

This Project is the bridge the back-end in prologs (Khan Family) with the front end in python. This project is done in the group of 6 people. We divide our Project work in parts that help us a lot in accomplishing this project in the short time.

**Project manager**   **Anayat**

**Python (fronted)**   **Bakht and Sania**

**Integration**                **Kainat and Tehreem**

**Report**                      **Seemi.**

**Prolog backend**   **Muhammad Jamshed**

is done by the cooperation of all members. In this project we present the SWI prolog as the integrating tool that support our wide ideas and help us to support the foreign links etc.

## **1): Introduction:**

Prolog is the programming language for the symbolic and the non-numeric computation. This language is well suited for the problems that involve the objects

and the relations between the objects. This SWI prolog was develop for the personal understanding. SWI prolog started back in 1986 with the requirement for the Prolog that could handle the recursive interaction with the language.

These days the SWI Prolog is aware of its environment and we need such type of system to support the interactive applications. SWI prolog is based on the simple machine prolog. SWI prolog is very important for the broad point of view in the Artificial Intelligence. Prolog Programming language has its great role in the Artificial Intelligence. Because the Prolog Programming language encourage the programmers toward the main goal. The goal in such a way that it solves and extract the problems for clearing, getting rid of and ungrasping make it possible for the clear and the workable that seems amazingly intelligent.

.

## **2): Background:**

Prolog is the programming of the logic. The idea generally used in 1970 to use the Prolog as the language for the logic. The early developer of this idea includes the Robert Kowalski at Edinburgh and the Alain Colmerauer at Merseilles. But now the popularity of this language is due to the David Warren's. And the implementation of this language is at 1970s at the Edinburgh.

### **2.1): Artificial Intelligence:**

Prolog is the programming language which is based on the small set of mechanism, tree based structure and the automatic backtracking. It is the powerful and the flexible programming language. It is the helpful language in tracking the

data from behind and the forward chain. This type of feature makes the Prolog as the powerful language for the Artificial Intelligence and for the non-numerical programming in general.

## **2.2): Defining Khan Family Realtion:**

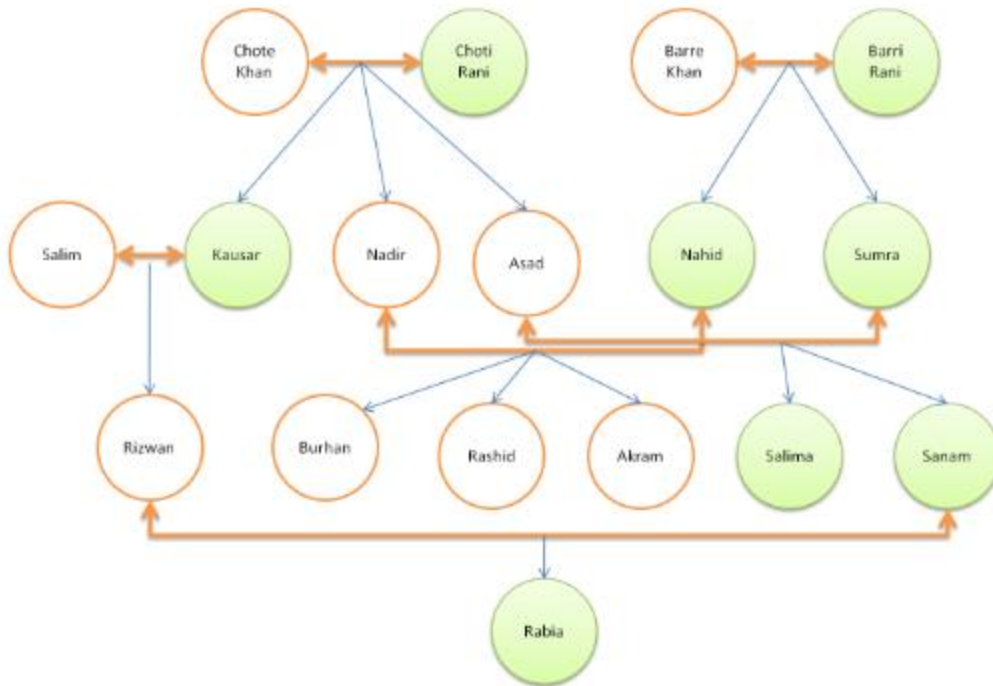
Prolog is the programming language for the symbolic and the non programming language. It is well suited for the problems that involve the objects and the relations between the objects. The relation can be written in prolog language as,

**Mianbewi** (fact1, fact2).

**Parent** (fact1, fact2).

**Gins** (fact1, fact2).

Here the Mianbewi is the name of relation. fact1 and fact2 are its argument. Here we use the following tree for the analysis.



Here the six clauses will be used for the mianbewi that are.

**mianbewi('chotekhan', 'chotiRani').**

**mianbewi('bare Khan','barirani').**

**mianbewi('salim', 'kousar').**

**mianbewi('asad','sumra').**

**mianbewi ('nadir','naheed').**

**mianbewi('rizwan', 'sanam').**

Similarly the seven clauses will use for the gins, that specify the male, or female. And 24 clauses are used for the parents. Each of these clauses declares the fact about the parent, gins and the mianbewi relation.

If we look at the working of Prolog for the clause that

**mianbewi('chotekhan', 'chotiRani').**

## SWI-prolog

Then the prolog answer is in yes or True.

Further if we write that;

**mianbewi('barykhan', 'chotiRani').**

Then the prolog answer is in false.

Because the in program we do not mention that **barykhan** and the **chotiRani** are the mianbewi.

We can also ask the question that;

**Mianbewi(X, ChotiRani).**

Here the Prolog do not answer in the true or false but here it will return the value of X, that is the **X=chotyKhan.**

Here we also ask the question that who is the wife of the chotyKhan. Then

**Mianbewi (ChotyKhan, X).**

Similarly here the X will be chotiRani.

If we want to print the all the list of Mianbewi then we use the.

**Mianbewi(X, Y).**

We can still ask many other complicated questions. Just like that who is the **grandfather** of this child, Similarly, **Chacha** or the **succer** of this one.

Assume that the Y is the parent of kousar and X is the parent of Y .So that queries were written in the prolog is as,

**Parent(Y, akram), Parent(X,Y).**

Here the

**SWI-prolog**

**Y = Nadir**

**X = ChotyKhan.**

Even if we change the requirements then logically they have the same meanings.

**Parent(X,Y), Parent(Y, Akram)**

So in the similar way we ask the question that who are the chotyKhan grandchild.so

**Parent (chotyKhan),parent(X,Y)**

The answer will be ;

X = Nadir

Y = Rashid

X = Nadir

Y = akram

X = Nadir

Y = Burhan.

Similarly the user can ask the question that is the Rashid, Akram and Burhan has the common parent or not. Then here the query will be .

**Parent (Rashid),Parent(Burhan),Parent(Akram).**

So the answer will be

X = Nadir

**Conclusion:**



- Through the above example we recognize that defining the relation in the Prolog is easy. Such as the Parent, Mianbewi, by defining the n-tuple of the objects that satisfy the relation.
- The user can easily make the query in the Prolog, about the relations given in the program.
- The argument of the relation may be constant, variable or concrete.
- The question of the system may consist of the one or the combination of the goals. Like:  
**Parent(Y, akram), Parent(X, Burhan).**
- The answer of the question may be True or false. Depend on the defining the condition.

### **2.3): Extending the Khan Family Relation by Rules:**

With the above main clauses of the Khan Family i.e. Mianbewi, Parents. We can extract the other interesting rules .This can be done by adding the some interesting facts in our programs.

```

gins('male','chotekhan').
gins('male','bareKhan').
gins('male','salim').
gins('male','asad').
gins('male','nadir').
gins('male','rizwan').
gins('male','burhan').
gins('male','rashid').
gins('male','akram').
gins('female','chotirani').
gins('female','barirani').
gins('female','kousar').
gins('female','samra').
gins('female','naheed').
gins('female','sanam').
gins('female','salima').
gins('female','rabia').

```

The above relation defines that, that is the male or female. Similarly if we define the child relation that is the inverse of the Parent relation. So we can define it in the similar way as the Parent relation. For example:

### **Beta (Nadir, ChotyKhan).**

The lower clause will have almost the same meaning as the above statement.

**Beta(Y, X):- parent(X, Y).**

This clause can be read as.

**For all X and Y,**

**If X is the Parent of Y then**

**Then Y is the beta of X.**

Prolog clauses such as:

**beta(Y, X):- parent(X, Y).**

Are called rules. But the fact is ;

**parent (ChotyKhan, Nadir).**

So this is the major difference between the fact and the rules. The facts can be unconditionally true, but the rule is true if the condition will satisfy .

Here there is no fact about the **beta**, So here the only way to make the valuable question is to define the rules about the **beta**. The rule in such a way that is applicable to any object such as X, Such as.

**Beta(Nadir,ChoteKhan).**

Here X = Nadir,

And Y = ChoteKhan.

Look at the below clause;

**Beta(Nadir, ChotyKhan):- parent(ChotyKhan, Nadir).**

He condition part will become;

**parent(ChotyKhan, Nadir).**

Here the Prolog will try to find ,if the condition is true or not, starting from.

**Beta(Nadir, ChotyKhan)**

### **Conclusion:**

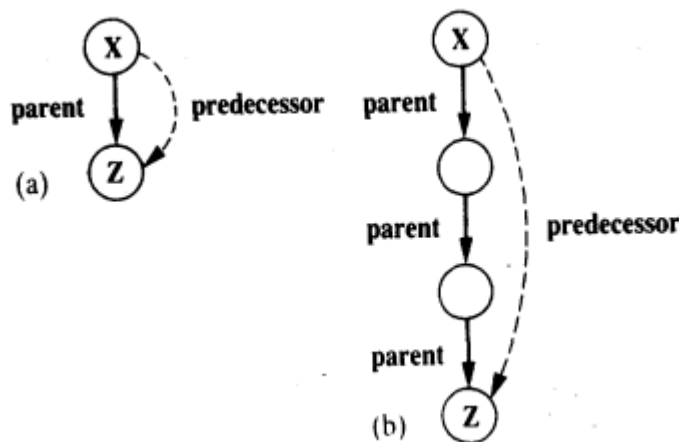
- Prolog Program can be extended by adding the new clause.
- Prolog clauses can be composing of fact, condition and the question.
- Facts are unconditionally true.
- Rules declare the things that are true according to the condition.

- With the help of question ,the user can ask that whether the statement or clause is true or not.

## **2.4):A recursive rule Definition:**

Let consider we add the predecessor relation. We define this relation in term of the parent relation. We define it into the two rules. The first rule will define the direct

Predecessors and the second rule define the indirect predecessors.



In fig (a), the rule is simple. As

For all X an Z.

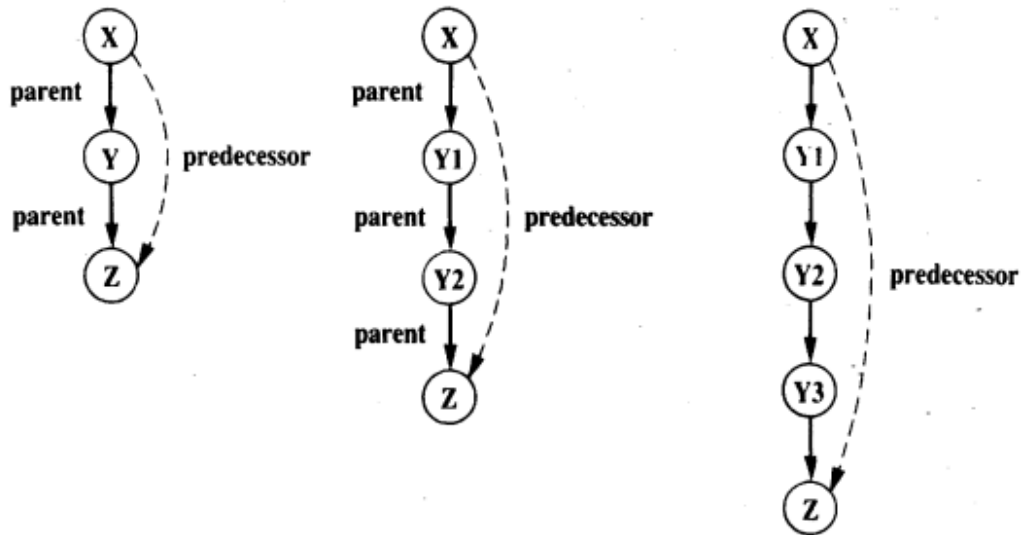
X is the predecessor of Z.

If X is the parent of Z.

If we translate it into the prolog, it will be.

**Predecessor(X, Z):-parent(X, Z)**

The second rule is somehow complicated than the first one. If we define the indirect predecessor, than



Then the relation of the predecessor will be.

```
predecessor(X, Z) :-parent( X, Z).
predecessor(X, Z) :-parent( X, Y),parent( Y, Z).
predecessor( X, Z) :-parent( X, Y1),parent( Y1, Y2),parent( Y2, Z).
predecessor(X, Z) :-parent( X, Y1),parent( Y1, Y2),parent( Y2, Y3),parent( Y3, Z).
```

To define the predecessor relation in term of itself is as.

For all X and Z.

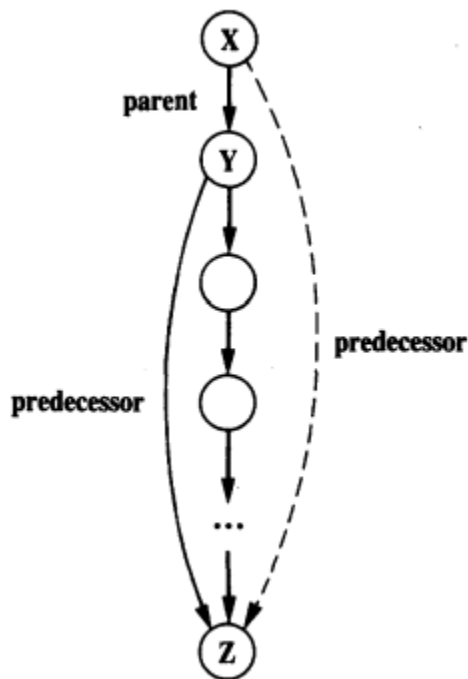
X is the Predecessor of Z, if Y that is.

- X is the parent of Y
- Y is the predecessor of Z.

If we make the prolog clause for it, it will be.

**Predecessor(X, Z):- parent(X,Y), predecessor(Y,Z).**

The key formulation of the above clause is the use of the predecessor in its own definition. If we want to define the something, that in further we need, but it will not be completely used. Then this type of definition is known as the **recursive**. Recursive programming is one of the fundamental principles of the Prolog Programming language. The recursive formulation of the predecessor relation is as follow.



The clauses of the tree will be:-

---

```

mianbiwi('chotekhan','chotiRani').
mianbiwi('bareKhan','barirani').
mianbiwi('salim','kousar').
mianbiwi('asad','sumra').
mianbiwi('nadir','naheed').
mianbiwi('rizwan','sanam').
gins('male','asad').
gins('male','nadir').
gins('male','rizwan').
gins('male','burhan').
gins('male','rashid').
gins('male','akram').
gins('female','chotirani').
gins('female','barirani').
gins('female','kousar').
gins('female','samra').
gins('female','naheed').
gins('female','sanam').
gins('female','salima').
gins('female','rabia').
parent('chotekhan','kausar').
parent('choterani','kausar').
parent('chotekhan','nadir').
parent('choterani','nadir').
parent('chotekhan','asad').
parent('choterani','asad').
parent('barekhan','nahid').
parent('brherani','nahid').
baap(X,Y):- parent(X,Y),gins('male',X).
beti(X,Y):- parent(Y,X),gins('female',X).
beti(X,Y):- parent(Y,X),gins('male',X).
beta(X,Y):- parent(Y,X),gins('male',X).
dada(X,Y):- parent(Z,Y),parent(X,Z),gins('male',X).

```

▲

## 2.5): How Prolog answer the Question:

The question that we ask in the prolog is always the combination of the one or more goal. The prolog will answer if it satisfies the goal. To satisfy the goal mean is that it answers in the form of true or false. In other word to satisfy the goal

mean, that the goal logically follow the rules and the facts. If we are using the variable, then it's means that the Prolog specify the object to the variable, to answer whether the goal is true or false. We know that **parent (chotyKhan, nadir)** is the fact, but **dada (ChoteKhan, Burhan)** is the derived fact from the **parent (chotyKhan, nadir)**. So through the above statement the prolog identify that the answer is in true. But similarly if we use the derived rule in the inverse form then it will be as **dada (Burhan, ChotyKhan)**. It will definitely turn the false answer. Similarly if Prolog Programming uses the large goal, It can extract its meaning in its programming language. The first rule may be definitely true, because it will match with the fact, but the other two will be backtrack by the Prolog Programming language in order to extract the meaning.

### **3): Pyswip:**

Pyswip is the python, the SWI prolog bridge enabling to query SWI prolog in our python program. The Pyswip does not use any compilation installion, but it uses the SWI Prolog as the shared library.

### **Feature Extraction:**

In our assignment we try to connect our SWI-prolog with the python. We want to fetch data from prolog in our python for which we import prolog from pyswip library. And to import .pl file of prolog we use "prolog. Consult". We define different functions in which we fetch the data such as parents, mianbiwi and gins. We try to find indirect relationship between the families. First we have tried to design this manually which means when we run our program it will ask us to enter a number about which relation we want to know. So it was displaying us



information but after taking some information from us. Then we try to convert it into a better design that we just enter the name or a relation it fetches the data by itself.

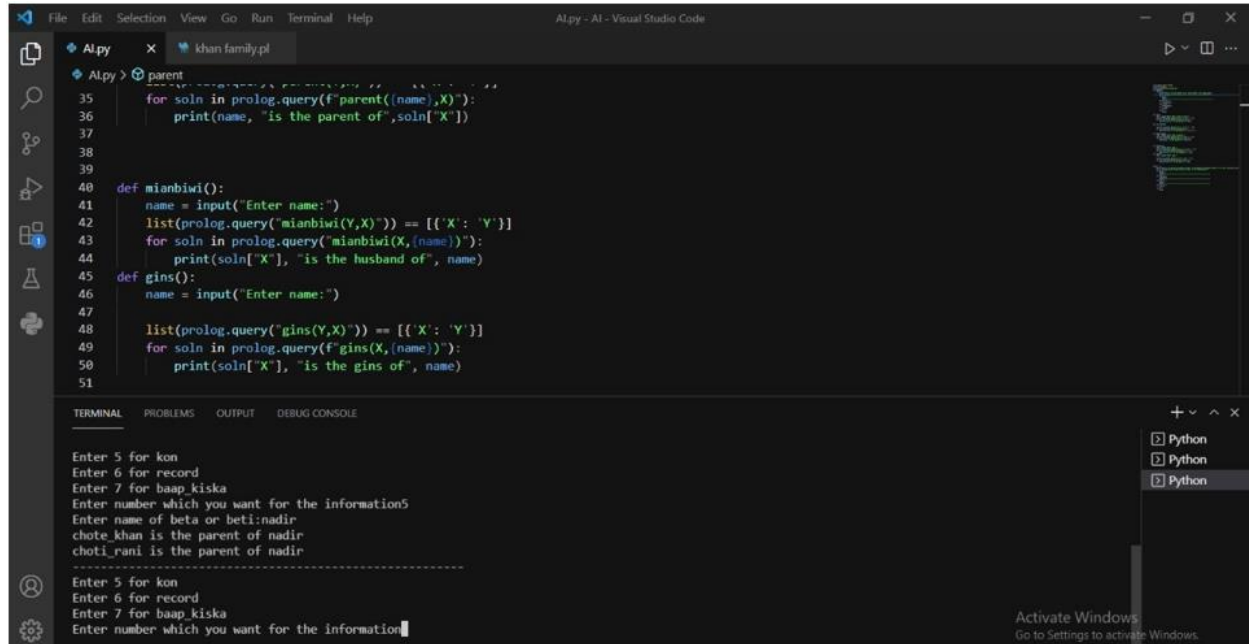
### **Data and Implementation:**

Data is meant to be the real data which is extracted after reading

### **Testing:**

we have test the code for khan family tree and ensure that all the possible rules which is defined from the tree through three different fact one of son and parent the other is mianbewi and the third one is gins which defined each person sex.so the rules for each unknown identity worked properly let suppose when we enter 1 then the member names of khan family is shown and then we have to choose the suitable number for the for the given printed rules and then we have to find out the desired output.

## SWI-prolog

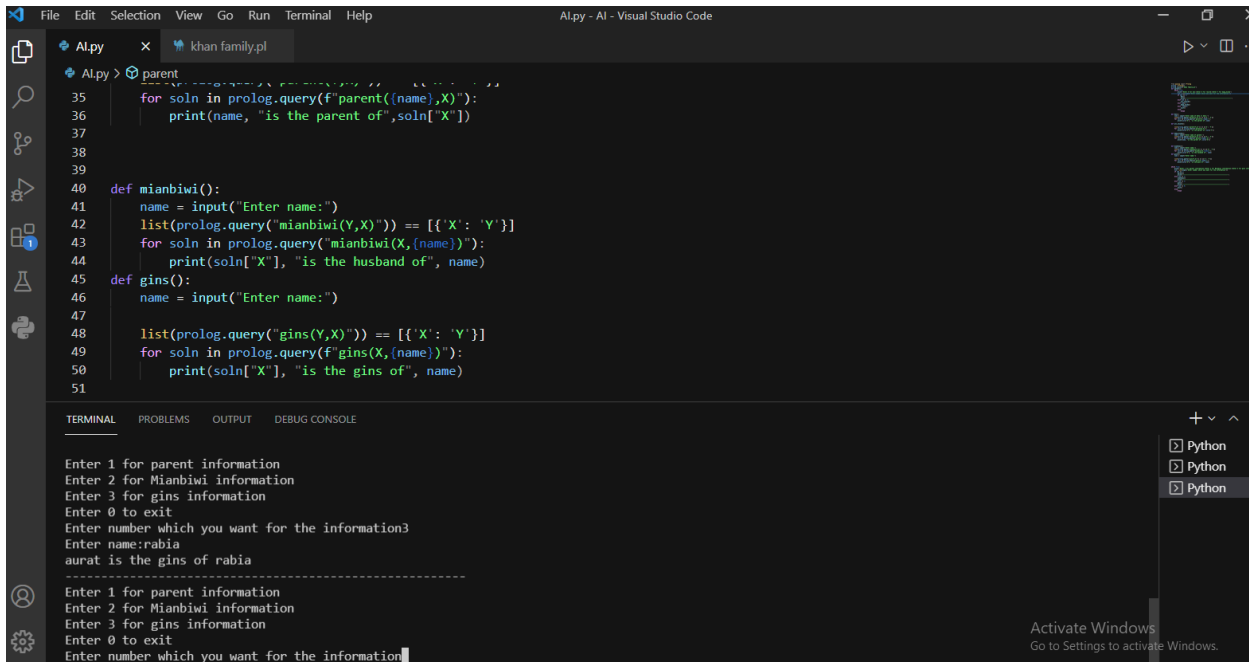


```
File Edit Selection View Go Run Terminal Help
Alpy - AI - Visual Studio Code

Alpy > parent
35 for soln in prolog.query(f"parent((name),X)":
36     print(name, "is the parent of", soln["X"])
37
38
39
40 def mianbiwi():
41     name = input("Enter name:")
42     list(prolog.query("mianbiwi(Y,X)") == [{X: 'Y'}])
43     for soln in prolog.query("mianbiwi(X,(name))"):
44         print(soln["X"], "is the husband of", name)
45
46 def gins():
47     name = input("Enter name:")
48
49     list(prolog.query("gins(Y,X)") == [{X: 'Y'}])
50     for soln in prolog.query(f"gins(X,(name))"):
51         print(soln["X"], "is the gins of", name)

TERMINAL PROBLEMS OUTPUT DEBUG CONSOLE
+ ^ v x
Python
Python
Python

Enter 5 for kon
Enter 6 for record
Enter 7 for baap_kiska
Enter number which you want for the information5
Enter name of beta or beti:nadir
chote_khan is the parent of nadir
choti_rani is the parent of nadir
-----
Enter 5 for kon
Enter 6 for record
Enter 7 for baap_kiska
Enter number which you want for the information
```



```
File Edit Selection View Go Run Terminal Help
Alpy - AI - Visual Studio Code

Alpy > parent
35 for soln in prolog.query(f"parent((name),X)":
36     print(name, "is the parent of", soln["X"])
37
38
39
40 def mianbiwi():
41     name = input("Enter name:")
42     list(prolog.query("mianbiwi(Y,X)") == [{X: 'Y'}])
43     for soln in prolog.query("mianbiwi(X,(name))"):
44         print(soln["X"], "is the husband of", name)
45
46 def gins():
47     name = input("Enter name:")
48
49     list(prolog.query("gins(Y,X)") == [{X: 'Y'}])
50     for soln in prolog.query(f"gins(X,(name))"):
51         print(soln["X"], "is the gins of", name)

TERMINAL PROBLEMS OUTPUT DEBUG CONSOLE
+ ^ v x
Python
Python
Python

Enter 1 for parent information
Enter 2 for Mianbiwi information
Enter 3 for gins information
Enter 0 to exit
Enter number which you want for the information3
Enter name:rabia
aurat is the gins of rabia
-----
Enter 1 for parent information
Enter 2 for Mianbiwi information
Enter 3 for gins information
Enter 0 to exit
Enter number which you want for the information
```

## Conclusion:

SWI prolog helps to answer those entire questions that we have to ask. This is the easy and convenient way to build our logical expression. With the help of SWI prolog, we will exactly tell that how the relations are indirectly connected with each other in the chain. Similarly we look that how python help us to fetch our data of the SWI prolog in the implementation.

### **11): References:**

- <https://silp.iiita.ac.in/wp-content/uploads/PROLOG.pdf>
- [http://intelligency.org/ai\\_prolog.php](http://intelligency.org/ai_prolog.php)

\*\*\*\*\*