SWISH PROLOG

SWI-Prolog is a versatile implementation of the Prolog language. Although SWI-Prolog gained its popularity primarily in education, its development is mostly driven by the needs for application development. This is facilitated by a rich interface to other IT components by supporting many document types and (network) protocols as well as a comprehensive low-level interface to C that is the basis for high-level interfaces to C++, Java (bundled), C#, Python, Rust, etc. (externally available). Data type extensions such as dicts and strings as well as full support for Unicode and unbounded integers simplify smooth exchange of data with other components. Although Prolog is widely recognised as a special purpose language for tasks such as rule evaluation we consider it primarily a platform that is suitable to be used as glue between various components. The main reason for this is that data is at the core of many modern applications while there is a large variety in which data is structured and stored. Classical query languages such as SQL, SPARQL, XPATH, etc. can each deal with one such format only, while Prolog can provide a concise and natural query language for each of these formats that can either be executed directly or be compiled into dedicated query language expressions. Prolog's relational paradigm fits well with tabular data (RDBMS), while optimized support for recursive code fits well with tree and graph shaped data (RDF). SWI-Prolog is equipped with an extensive web server (HTTP) framework that can be used both for providing (REST) services and end-user applications based on HTML5+CSS+JavaScript. Pengines (Prolog engines) allow clients to run queries against a client-provided program on a remote server using a generic API. Such programs can be executed in a sandbox.

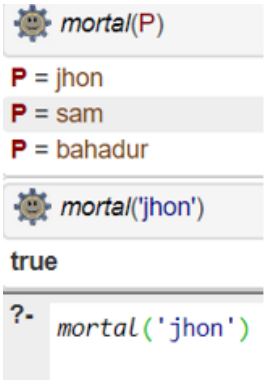
Activity 1

Question :- Find a man is mortal or not ?

code :

```
man('jhon').
man('sam').
man('bahadur').
mortal(x):- man(x).
```

Output :



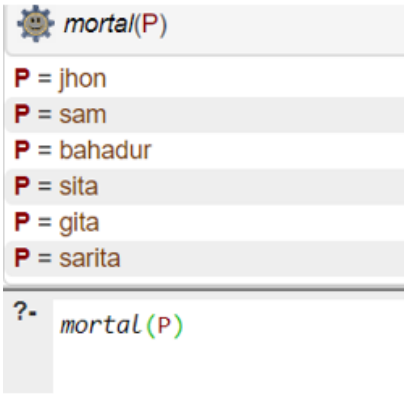
Activity 2

Question :- Find which person are mortal or not ?

code :

```
man('jhon').
man('sam').
man('bahadur').
women('gita').
women('sita').
women('sarita').
person(X):-man(x).
person(X):-women(x).
mortal(x):- person(x).
```

Output :



Activity 3

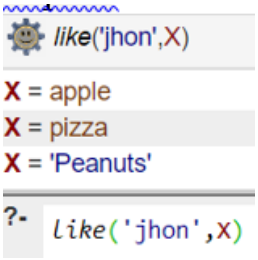
Question :- 12. Consider the following sentences:

1. John likes all kinds of food.
2. Apples are food.
3. Pizza is a food.
4. Anything anyone eats and isn't killed by is a food.
5. Bill eats Peanuts and is still alive.
6. David eats everything Bill eats.
   Prove that "John likes Peanuts".

code :

```
like(jhon,X):-food(X).
food(apple).
food(pizza).
food(X):-alive(Y),eat(Y,X).
eat(bill,peanuts)
eat(david,Q):-eat(bill,Q)
alive(bill)
```

Output :



4. Write a prolog program to find the rules for parent, child, son, daughter, brother, sister, uncle, aunt, ancestor given the facts about man, woman, father and wife only.

code :

```
% Facts
man(dashrat).
man(ram).
man(laxman).
man(bharat).
man(luv).
man(kush).

woman(kaushaliya).
woman(urmila).
woman(sita).

father(dashrat, ram). % dashrat is father of ram
father(dashrat, laxman).
father(dashrat, bharat).
father(ram, luv).
father(ram, kush).

wife(kaushaliya, dashrat). % kaushaliya is wife of dashrat
wife(sita, ram).
wife(urmila, laxman).


% Rules
parent(Parent, Child) :- father(Parent, Child).
parent(Parent, Child) :- father(Father, Child), wife(Parent, Father).

child(Child, Parent) :- parent(Parent, Child).

son(Son, Parent) :- man(Son), parent(Parent, Son).

daughter(Daughter, Parent) :- woman(Daughter), parent(Parent, Daughter).

brother(Brother, Sibling) :- man(Brother), father(Father, Brother), father(Fa
ther, Sibling), Brother \= Sibling.

sister(Sister, Sibling) :- woman(Sister), father(Father, Sister), father(Fath
er, Sibling), Sister \= Sibling.

uncle(Uncle, Child) :- brother(Uncle, Parent), parent(Parent, Child).

aunt(Aunt, Child) :- woman(Aunt), sister(Aunt, Parent), parent(Parent, Child)
```

.

```
ancestor(Ancestor, Descendant) :- parent(Ancestor, Descendant).
ancestor(Ancestor, Descendant) :- parent(Ancestor, Intermediate), ancestor(In
termediate, Descendant).
```

Output: