DIP: Prolog Language

Presenter: Dr. Ha Viet Uyen Synh.

Comparing Imperative and Declarative Languages

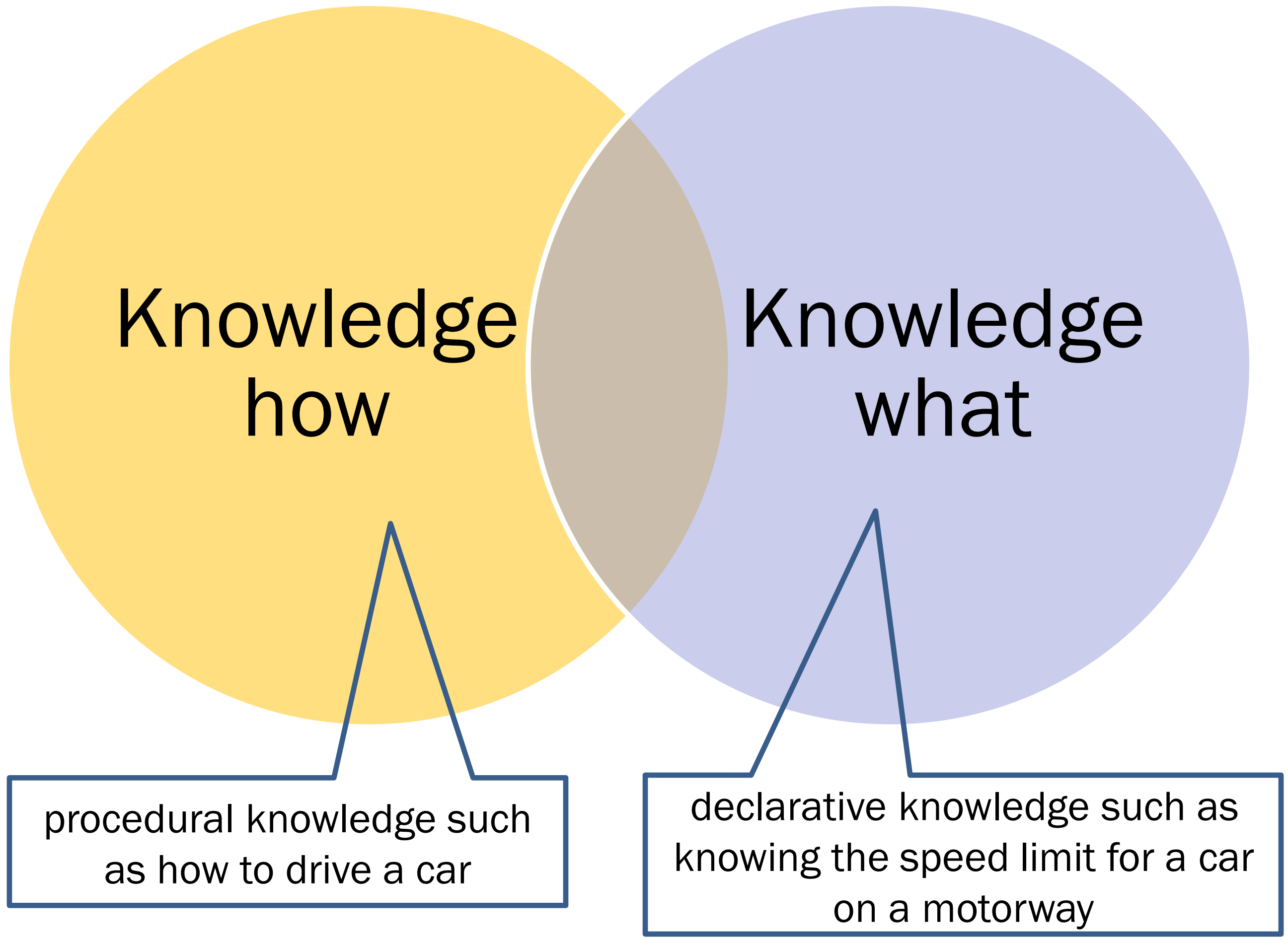
Imperative Languages

- Procedural programming requires that the programmer tell the computer what to do.
- That is, *how* to get the output for the range of required inputs.
- The programmer must know an appropriate algorithm.

Declarative Languages

- Declarative programming requires a more descriptive style.
- The programmer must know *what* relationship s hold between various entities.
- Prolog provides a search strategy for free

How do we represent what we know?



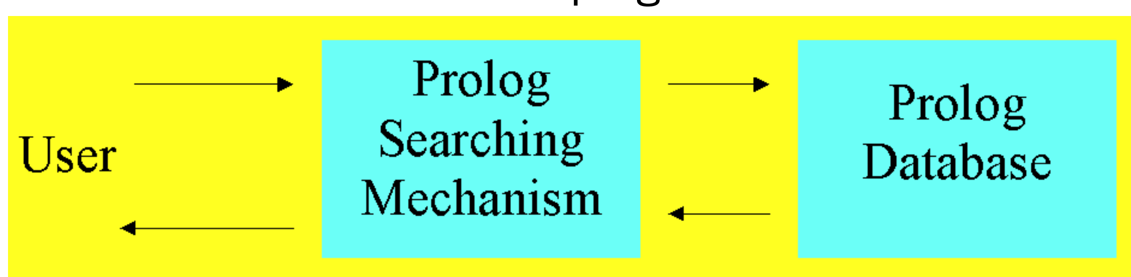
What is a Prolog program?

Programming in Prolog is very different from programming in a traditional procedural language. In Prolog you don't say how the program will work.

Prolog can be separated in two parts :

The Program: Database is a text file (*.pl) that contain the facts and rules that will be used by the user of the program. It contains all the relations that make this program.

The Query: Search Mechanism. When you launch a program you are in query mode. This mode is represented by the sign ? - at the beginning of the line. In query mode you ask questions about relations described in the program.



Part A

DATABASE

Facts

Facts either consist of a particular term or a relation between terms.

- Simple facts (propositions).
- Facts with arguments.

A query in Prolog is the action of asking the program about information contained within its data base.

Ex:

```
1.  sunny. /* It's sunny*/
?- sunny.

 2. eats(fred,oranges).      /* 'Fred eats oranges' */
    eats(tony,apple).        /* 'Tony eats apple'   */
    eats(john,apple).        /* 'John eats apple'   */
?- eats(fred,oranges).
    yes
?- eats(john,apple).
    yes
?- eats(mike,apple).
    no
```

Terms

Atoms: Start with non-capital letters or are enclosed in single quotes.

harry, nimbus2000, 'Professor Dumbledore', auntpetunia

Numbers: 3, 6, 2957, 8.34, ...

Variables: Start with a capital letter or an underscore.

Harry, _harry

Complex terms: An atom (the functor) is followed by a comma separated sequence of Prolog terms enclosed in parenthesis (the arguments).

like(harry, X), np(det(the),n(potion))

Rules

Rules are of the form Head :- Body.

Like facts and queries, they have to be followed by a full stop.

Head is a complex term.

Body is complex term or a sequence of complex terms separated by commas.

Example:

```
happy(auntpetunia) :- happy(dudley).
happy(unclevernon) :- happy(dudley),
unhappy(harry).
```

Rules

Rules allow us to make conditional statements about our world.

Each rule can have several variations, called clauses. These clauses give us different choices about how to perform inference about our world.

Ex:

```
mortal(X) :-person(X). /*'All people are mortal'*/
person(socrates).
?- mortal(socrates).
Yes
?- mortal(P).
P = socrates
yes
```

The clause can be read in two ways:

- The declarative interpretation is "For a given X, X is mortal if X is a person."
- The procedural interpretation is "To prove the main goal that X is mortal, prove the subgoal that X is a person."

Rules without variable(s)

```
a :- b.            % a if b

a :- b, c.         % a if b and c.

a :- b; c.         % a if b or c.

a :- \++ b.        % a if b is not provable

a :- not b.        % a if b fails

a :- b -> c; d.    % a if (if b then c else d)
```

Rules with parameter(s)

The parameter starts with upper case letter.

```
isFood(X) :- isEdible(X).

isEdible(apple).
isEdible(tomato).
isEdible(potato).


?- isFood(X).
X = apple;
X = tomato;
X = potato.
```

Rules with multiple statements

```
isApple(X) :-
isFruit(X),
isRed(X).

isRed(object1).
isFruit(object1).
isFruit(object2).

?- isApple(object1).
true.

?- isApple(X).
X = object1.
?- isApple(object2).
false.
```

Rules with "return" values

```
add(A,B,Sum) :-
Sum is A + B.

?- add(10,5,X).
X = 15.
add(10,5,15).
Yes

multiply(A,X,Y,Z) :-
X is A*2,
Y is A*3,
Z is A*4.

?- multiply(5,X,Y,Z).
X = 10,
Y = 15,
Z = 20.
```

Variables and Unification

In order to match arguments with items, we must use a Variable.

The process of matching items with variables is known as unification.

Variables are distinguished by starting with a capital letter.

X /* a capital letter */

VaRiAbLe /* a word - it be made up or either case of letters */

My_name /* we can link words together via '_' (underscore) */

Examples

```
   eats(fred,apple).
   eats(fred,oranges).
?- eats(fred,What).
   What=apple ;
   What=oranges ;
   no


   book(1,title1,author1).
   book(2,title2,author1).
   book(3,title3,author2).
   book(4,title4,author3).

?- book(_,_,author2).
   yes

?- book(_,X,author1).
X=title1 ;
X=title2 ;
```

Types

Prolog is a typeless programming language.

Prolog provides for numbers, atoms, lists, tuples, and patterns.

Simple Types

TYPE	VALUES
boolean	true, fail
integer	integers
real	floating point numbers
variable	variables
atom	character sequences

Type Predicates

PREDICATE	CHECKS IF
var(V)	V is a variable
nonvar(NV)	NV is not a variable
atom(A)	A is an atom
integer(I)	I is an integer
real(R)	R is a floating point number
number(N)	N is an integer or real
atomic(A)	A is an atom or a number
functor(T,F,A)	T is a term with functor F and arity A
T =..L	T is a term, L is a list.
clause(H,T)	H :- T is a rule in the program

Examples

```
?- functor(t(a,b,c),F,A).
F = t
A = 3
Yes

?- t(a,b,c) =..L.
L = [t,a,b,c]
yes
?- T =..[t,a,b,c].
T = t(a,b,c)
yes
```

Expressions

Arithmetic expressions are evaluated with the built in predicate is which is used as an infix operator in the following form

variable is expression

?- X is 3*4.

X = 12

Yes

SYMBOL	OPERATION
+	addition
-	subtraction
*	multiplication
/	real division
//	integer division
mod	modulus
**	power

Boolean Predicates

SYMBOL	OPERATION	ACTION
A ?= B	unifiable	A and B are unifiable but does not unify A and B
A = B	unify	unifys A and B if possible
A \+= B	not unifiable	
A == B	identical	does not unify A and B
A \+== B	not identical	
A =:= B	equal (value)	evaluates A and B to determine if equal
A =\+= B	not equal (value)	
A < B	less than (numeric)	
A =< B	less or equal (numeric)	
A > B	greater than (numeric)	
A >= B	greater or equal (numeric)	
A @< B	less than (terms)	
A @=< B	less or equal (terms)	
A @> B	greater than (terms)	
A @>= B	greater or equal (terms)	

Functions

Prolog does not provide for a function type therefore, functions must be defined as relations (rules).

Examples:

fac(0,1).

fac(N,F) :- N > 0, M is N - 1, fac(M,Fm), F is N * Fm.

fib(0,1).

fib(1,1).

fib(N,F) :- N > 1, N1 is N - 1, N2 is N - 2, fib(N1,F1), fib(N2,F2), F is F1+ F2.

Part B

SEARCH MECHANISM

Backtracking

KB:

```
eating(dudley).

happy(aunt_petunia) :- happy(dudley).

happy(uncle_vernon) :- happy(dudley),unhappy(harry).

happy(dudley) :- kicking(dudley,harry).

happy(dudley) :- eating(dudley).
```

Query:

```
?- happy(aunt_petunia).
yes
```

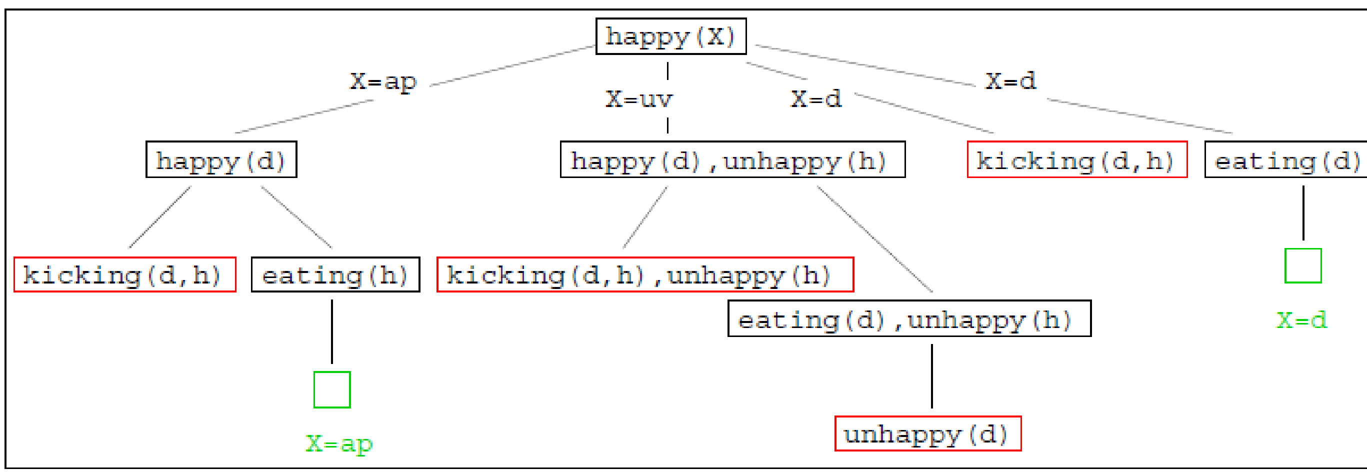
- Check for a fact or a rule's head that match the query.
- If you find a fact, you're done.
- If you find a rule, prove all goals specified in the body of the rule.

Backtracking

KB:

```
eating(dudley).
happy(aunt_petunia):-happy(dudley).
happy(uncle_vernon):-happy(dudley),unhappy(harry).
happy(dudley):-kicking(dudley,harry).
happy(dudley):-eating(dudley).
```

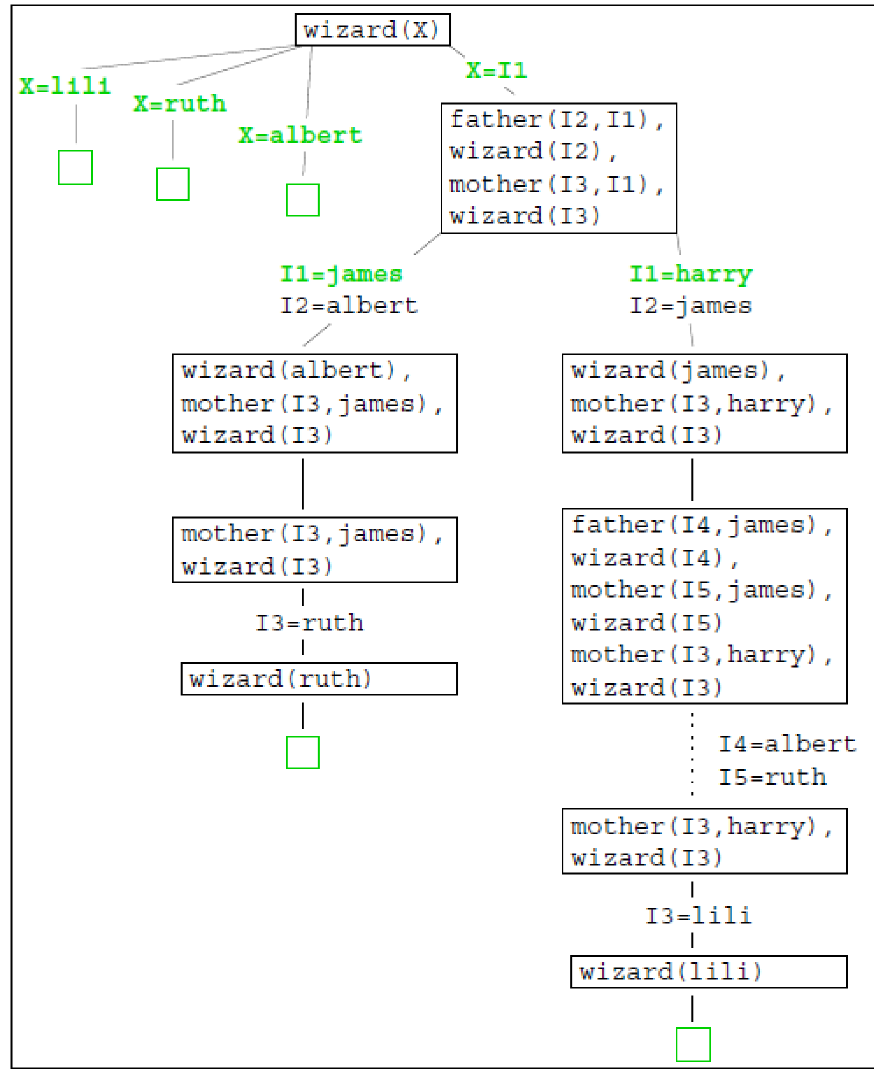
Query: `?- happy(X).`



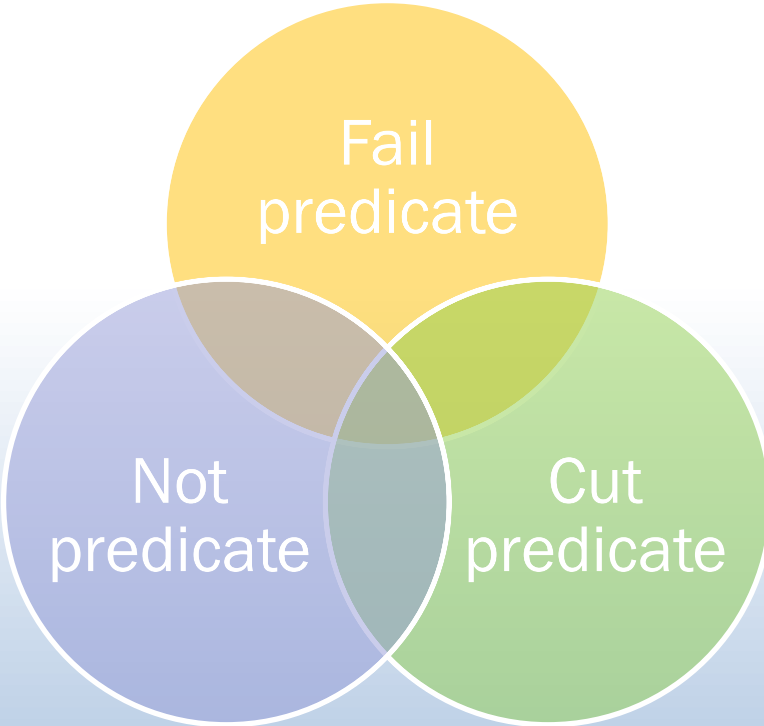
Backtracking

```
father(albert,james).
father(james,harry).
mother(ruth,james).
mother(lili,harry).

wizard(lili).
wizard(ruth).
wizard(albert).
wizard(X) :-
        father(Y,X),
        wizard(Y),
        mother(Z,X),
        wizard(Z).
```



Backtracking



Fail predicate

When it is called, it causes the failure of the rule.

Example:

```
goal(X) :- failure(X),!,fail.
goal(X).
```

Cut symbol/predicate

Cut predicate is used to turn off backtracking.

Cut predicate represented by an exclamation point (!)

Example:

```
data(pentiumIII).
data(athlon).
compare_cut_1(X) :-   data(X).
compare_cut_1('last chip').
?- compare_cut_1(X), write(X), nl, fail.
    pentiumIII
    athlon
    last chip
    no

compare_cut_2(X) :-       data(X),       !.
compare_cut_2('last chip').
?- compare_cut_2(X), write(X), nl, fail.
    pentiumIII
    no
```

Cut symbol

```
max(A,B,M) :- A <  B, M = B.
max(A,B,M) :- A >= B, M = A.
```

The code may be simplified by dropping the conditions on the second rule.

```
max(A,B,B) :- A <  B.
max(A,B,A).
```

```
?- max(3,4,M).
M = 4;
M = 3
```

To prevent backtracking to the second rule the cut symbol is inserted into the first rule.

```
max(A,B,B) :- A < B.!.
max(A,B,A).
```

Not predicate

it is always possible to rewrite the predicates without the cut. When using the cut, the order of the rules become important.

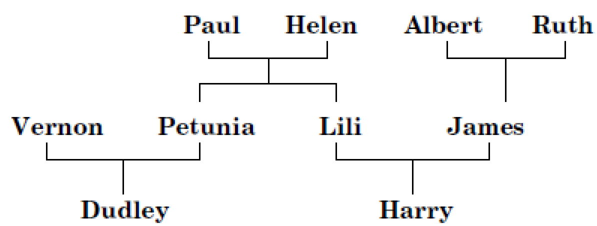
Example:

```
not_2(X) :- X = pentiumIII.
not_2(X) :- not(X = pentiumIII).
```

Recursion

In Prolog, recursion appears when a predicate contain a goal that refers to itself

Example:




```
parent_of(paul,petunia).
parent_of(helen,petunia).
parent_of(paul,lili).
parent_of(helen,lili).
parent_of(albert,james).
parent_of(ruth,james).
parent_of(petunia,dudley).
parent_of(vernon,dudley).
parent_of(lili,harry).
parent_of(james,harry).
```

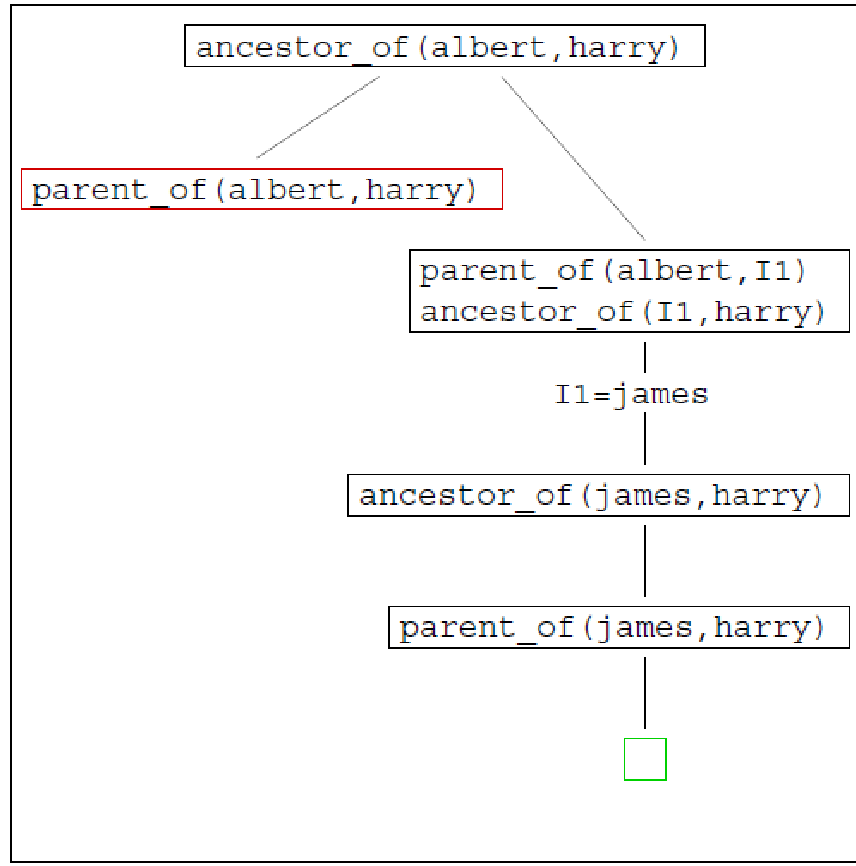
Task: Define a predicate `ancestor_of(X,Y)` which is true if `X` is an ancestor of `Y`.

```
ancestor(X,Y) :- parent(X,Y). /* If X is a parent of Y, then X is an ancestor of Y */
ancestor(X,Y) :- parent(X,Z),  ancestor(Z,Y).
/* if Y is an ancestor of Z and Z is a parent of X, then Y is an ancestor of X */
?- ancestor(albert,harry).
```

Recursion

```
parent_of(paul,petunia).
parent_of(helen,petunia).
parent_of(paul,lili).
parent_of(helen,lili).
parent_of(albert,james).
parent_of(ruth,james).
parent_of(petunia,dudley).
parent_of(vernon,dudley).
parent_of(lili,harry).
parent_of(james,harry).

ancestor_of(X,Y) :-
        parent_of(X,Y).
ancestor_of(X,Y) :-
        parent_of(X,Z),
        ancestor_of(Z,Y).
```



Iteration

```
length(List,LenghtofList) :- length(List,0,LengthofList).
length([],LenghtofPrefix,LengthofPrefix).
length([Element|List],LengthofPrefix,LengthofList) :-
PrefixPlus1 is LengthofPrefix + 1, length(List,PrefixPlus1,LengthofList).

reverse(List,RList) :- reverse(List,[],RList).
reverse([],RL,RL).
reverse([Element|List],RevPrefix,RL) :-
reverse(List,[Element|RevPrefix],RL).

sum([ ],0).
sum([X|L],Sum) :- sum(L,SL), Sum is X + SL.

product([ ],1).
product([X|L],Prod) :- product(L,PL), Prod is X * PL.

append([ ],L,L).
append([X1|L1],L2, [X1|L3]) :- append(L1,L2,L3).
```

Tail Recursion

```
fib(0,1).
fib(1,1).
fib(N,F) :- N > 1, N1 is N - 1, N2 is N - 2, fib(N1,F1), fib(N2,F2), F is F1+ F2.
```

```
fib(0,1).
fib(1,1).
fib(N,F) :- N > 1, fib(N,1,1,F).

fib(2,F1,F2,F) :- F is F1 + F2.
fib(N,F1,F2,F) :- N > 2, N1 is N - 1, NF1 is F1 + F2,  fib(N1,NF1,F1,F).
```

List

Lists are powerful data structures for holding and manipulating groups of things.

In Prolog, a list is simply a collection of terms. The terms can be any Prolog data types, including structures and other lists.

Syntactically, a list is denoted by square brackets ([]) with the terms separated by commas.

The empty list is represented by a set of empty brackets [].

Example:

```
 list_where([Tequila,Whisky,Vodka], bathroom).
 list_where([Martini,Muscat], kitchen).
 list_where([Malibu,Soho], under_the_bed).
 list_where([Epita], everywhere).
?- list_where(X, under_the_bed).
    X = [Malibu,Soho]
```

List

The special notation for list structures [X | Y]. X is bound to the first element of the list, called the head. Y is bound to the list of remaining elements, called the tail.

Example:

```
?- [X|Y] = [a, b, c, d, e].
   X = a
   Y = [b, c, d, e]

?- [X|Y] = [].
   no

?- [First, Second | Q] = [water,gin,tequila,whisky].
   First = water
   Second = gin
   Q = [tequila,whisky]
```

List

```
length([],0).
length([H|T],N) :- length(T,M), N is M+1.

member(X,[X|List).
member(X,[Element|List]) :- member(X,List).

prefix([],List).
prefix([X|Prefix],[X|List]) :- prefix(Prefix,List).

suffix(Suffix,Suffix).
suffix(Suffix,[X|List]) :- suffix(Suffix,List).

append([],List,List).
append([Element|List1],List2,[Element|List1List2]) :-
append(List1,List2,List1List2).
```

Read/Write

read(X): Read the term from the active input and unifie X with it.

write(Y): Write the term Y on the active output

Example: Calculating the cube of an integer

```
 cube :-  read(X), calc(X).  /* read X then query calc(X). */
 calc(stop) :- !.                 /* if X = stop then it ends   */
 calc(X) :- C is X * X * X, write(C),cube.
/* calculate C and write it then ask again cube. */
```

Questions? More Information?

hvusynh@hcmiu.edu.vn

Quiz

1. Write a function to find the minimum between 2 numbers.

2. Write a function to implement the Ackermann function

$$A(m,n) = \begin{cases} n+1 & \text{if } m = 0 \\ A(m-1, 1) & \text{if } m > 0 \text{ and } n = 0 \\ A(m-1, A(m, n-1)) & \text{if } m > 0 \text{ and } n > 0. \end{cases}$$

3. Write a program to find the last element of a list

?- my_last(X,[a,b,c,d]).
X = d