Introduction to Prolog

CSCI3180 PRINCIPLES OF PROGRAMMING LANGUAGES

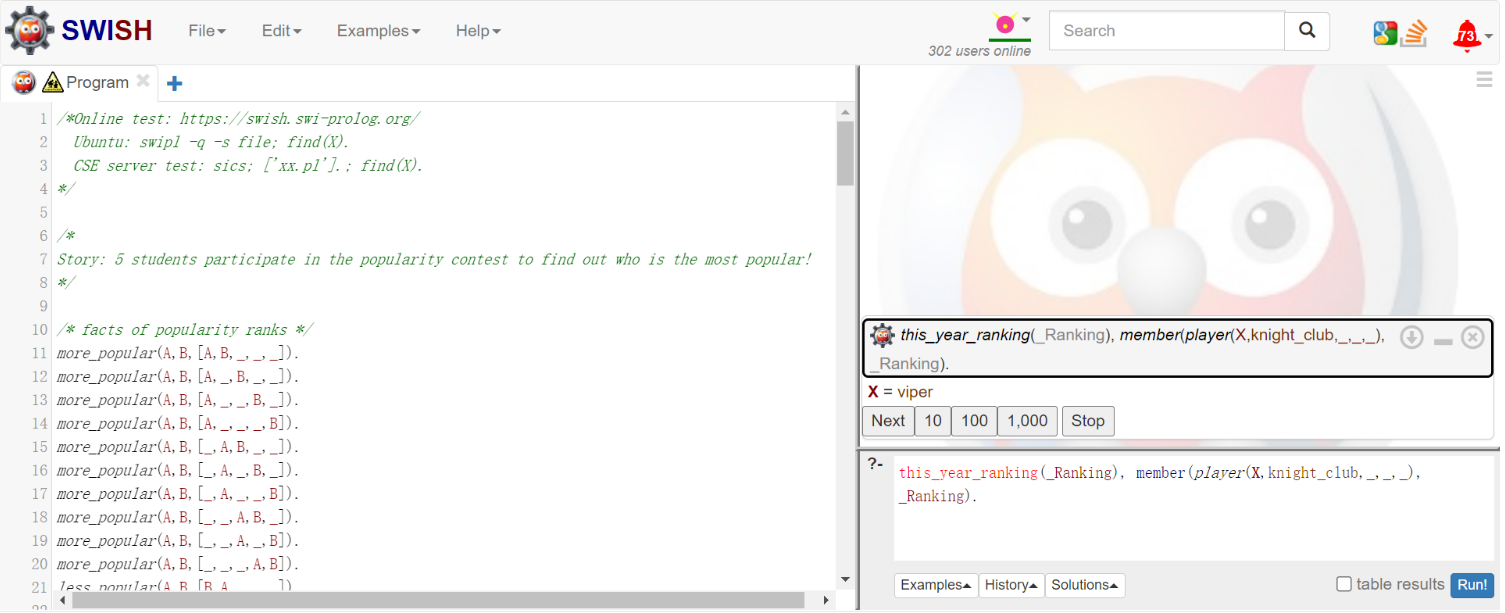
YAOTIAN CUI

Outline

- Programming environment
- Prolog programming
- Assignment 4

Programming environment

- Online: https://swish.swi-prolog.org/
- Ubuntu: swipl -q -s xx.pl; query
- CSE server: sics; ['xx.pl'].; query; ctrl + D to exit
  - This is the official testing environment – but everything covered in this course for Prolog should work just fine in the other two



```
/*Online test: https://swish.swi-prolog.org/
  Ubuntu: swipl -q -s file; find(X).
  CSE server test: sics; ['xx.pl'].; find(X).
*/

/*
Story: 5 students participate in the popularity contest to find out who is the most popular!
*/

/* facts of popularity ranks */
more_popular(A,B,[A,B,_,_,_]).
more_popular(A,B,[A,_,B,_,_]).
more_popular(A,B,[A,_,_,B,_]).
more_popular(A,B,[A,_,_,_,B]).
more_popular(A,B,[_,A,B,_,_]).
more_popular(A,B,[_,A,_,B,_]).
more_popular(A,B,[_,A,_,_,B]).
more_popular(A,B,[_,_,A,B,_]).
more_popular(A,B,[_,_,A,_,B]).
more_popular(A,B,[_,_,_,A,B]).
less_popular(A,B,[B,A,_,_,_]).
```

```
this_year_ranking(_Ranking), member(player(X,knight_club,_,_,_),
_Ranking).
X = viper
```

```
?- this_year_ranking(_Ranking), member(player(X,knight_club,_,_,_),
_Ranking).
```

Prolog programming

- Programming in Logic

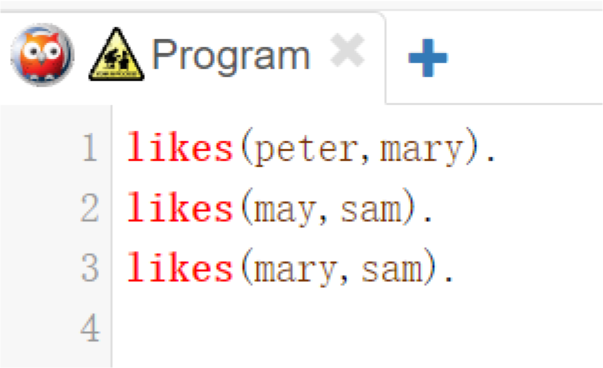
- Given facts and rules, it will automatically analyze the logical relationship, and then allow users to complete complex logical operations through queries

Basic of prolog

- A Prolog program consists of constants, variables, rules and comments
- Constants
  - Constants start with lower case alphabets, e.g. peter, dog, …
- Variables
  - Variables start with an upper case letter, e.g. People, X, Who, …
  - Anonymous Variables begin with an *underscore* '_'
- Rules
  - Head + body
  - Syntax: A :- B_1, B_2, B_3 . (end with dot)
  - A is head, Bi is body
- Fact (also known as unconditional rule/clause): Clause without body
  - Syntax: A . (end with dot)

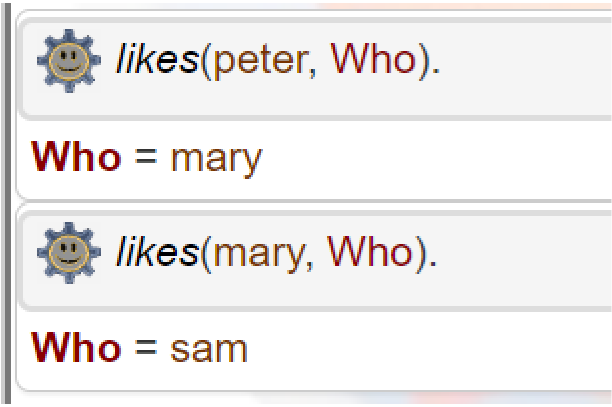
Example

- facts

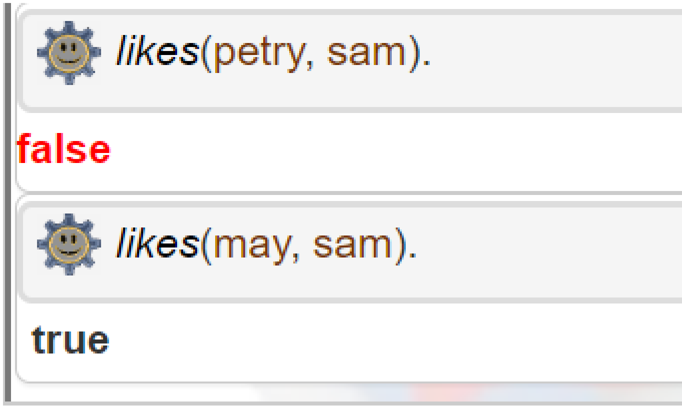


```
likes(peter,mary).
likes(may, sam).
likes(mary, sam).
```

- Queries



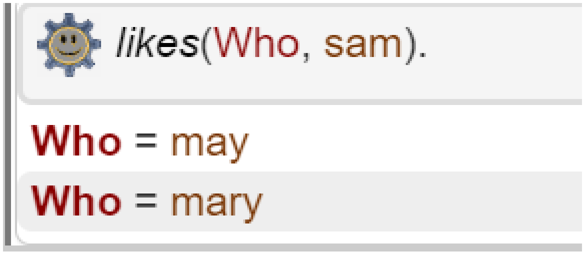
```
likes(peter, Who).
Who = mary
likes(mary, Who).
Who = sam
```



```
likes(petry, sam).
false
likes(may, sam).
true
```

Example

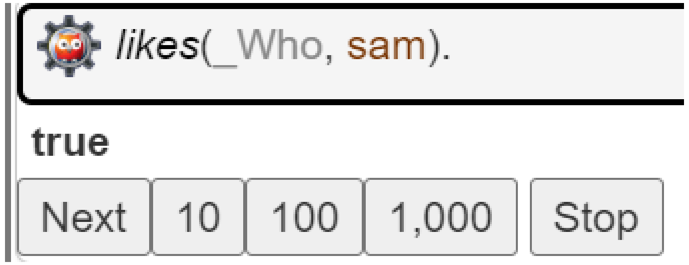
- Variables (Who likes sam?)



```
likes(Who, sam).
```

```
Who = may
Who = mary
```

- Anonymous Variables (a variable that starts with an underscore) (Anyone likes sam?)

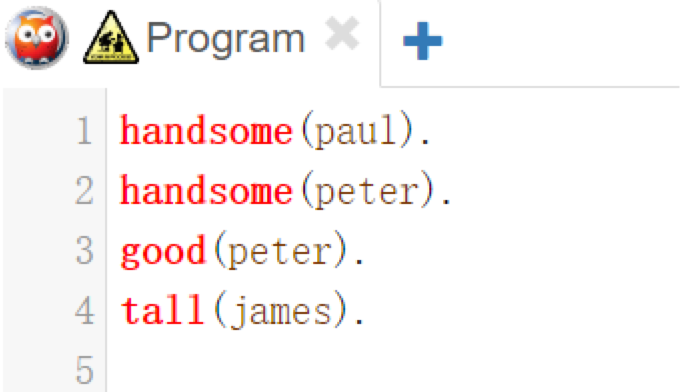


```
likes(_Who, sam).
```

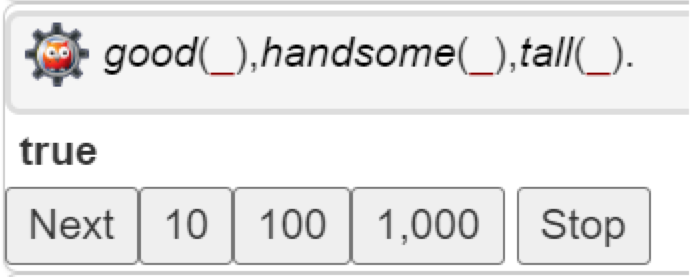
```
true
```

Example- _ , _X and _x

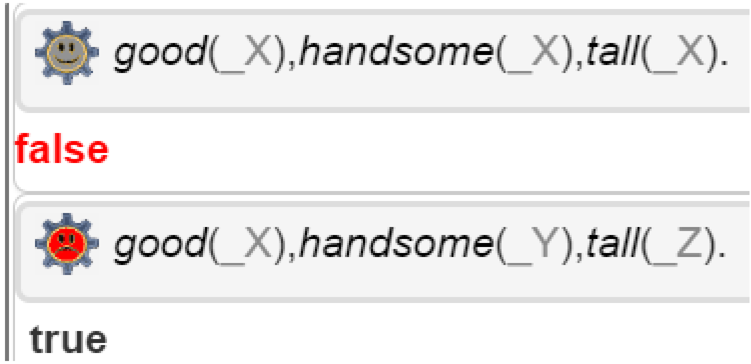
- anonymous variable: _
  - matches anything: e.g: someone good, someone handsome, someone tall
  - Output is true/false
- Named singleton variables: _X
  - Named singletons start with a double underscore (__) or a single underscore followed by an uppercase letter, e.g., __X or _X;
  - matches to the same person
- Normal variables
  - All other variables are ‘normal' variables. Note this makes _x a normal variable



```
handsome(paul).
handsome(peter).
good(peter).
tall(james).
```



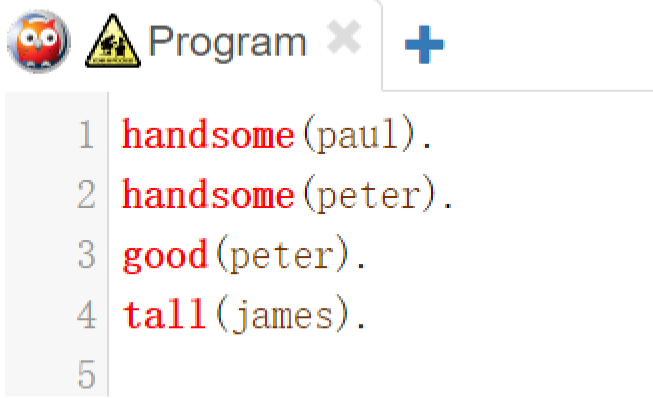
```
good(_),handsome(_),tall(_).

true
```



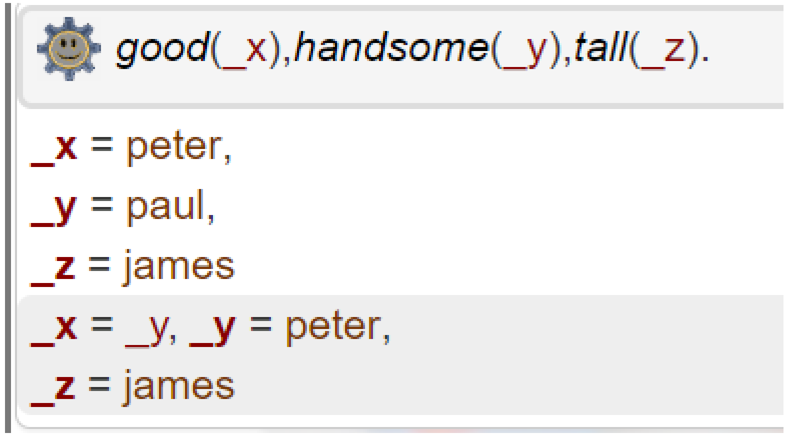
```
good(_X),handsome(_X),tall(_X).

false

good(_X),handsome(_Y),tall(_Z).

true
```

Example- _ , _X and _x

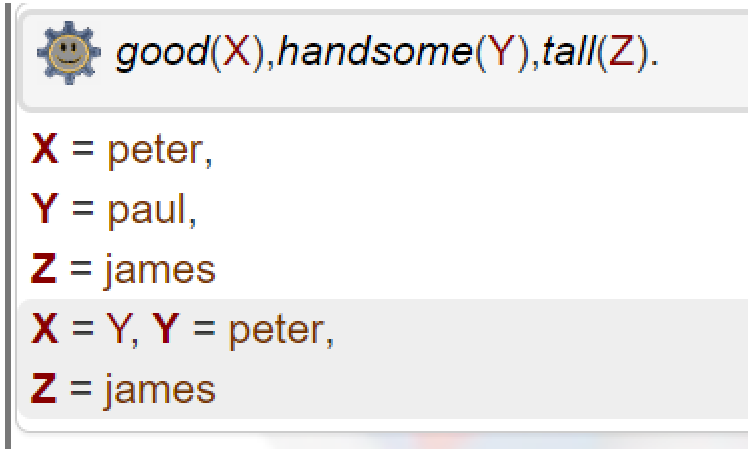
- Normal variables- X and _x



```
handsome(paul).
handsome(peter).
good(peter).
tall(james).

```



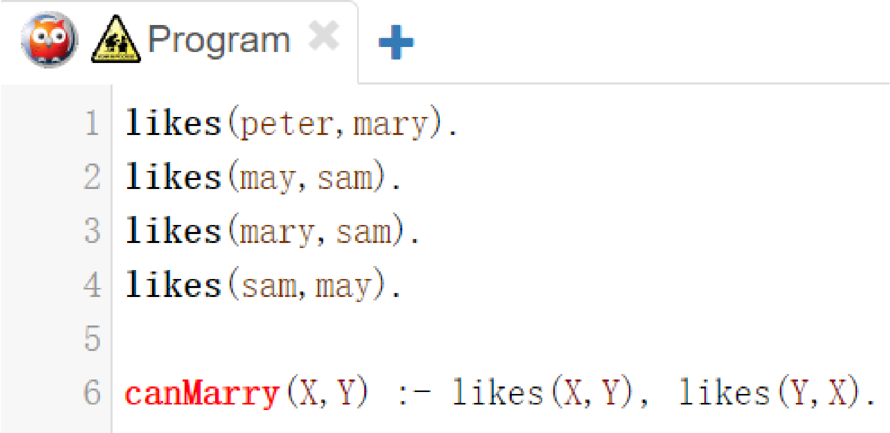
```
good(_x),handsome(_y),tall(_z).

_x = peter,
_y = paul,
_z = james
_x = _y, _y = peter,
_z = james
```



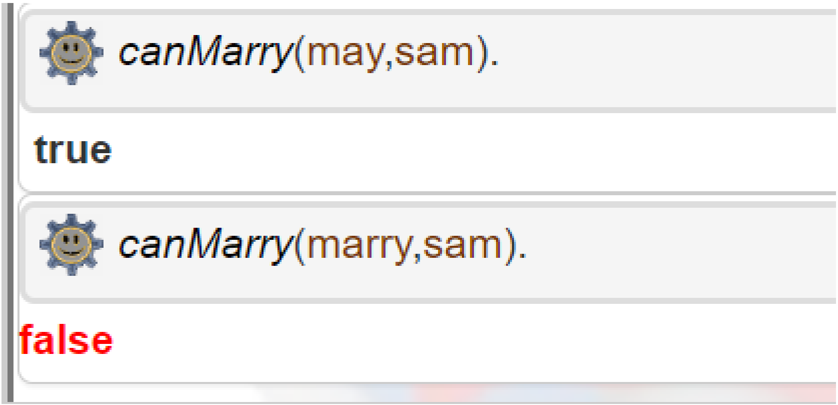
```
good(X),handsome(Y),tall(Z).

X = peter,
Y = paul,
Z = james
X = Y, Y = peter,
Z = james
```

Example

- Add a rule
  - X and Y can marry if X likes Y and Y like X



```
likes(peter,mary).
likes(may,sam).
likes(mary,sam).
likes(sam,may).

canMarry(X,Y) :- likes(X,Y), likes(Y,X).
```



```
canMarry(may,sam).
true
canMarry(marry,sam).
false
```

Prolog Programs

- Rules
  - Syntax: A :- A_1, A_2.

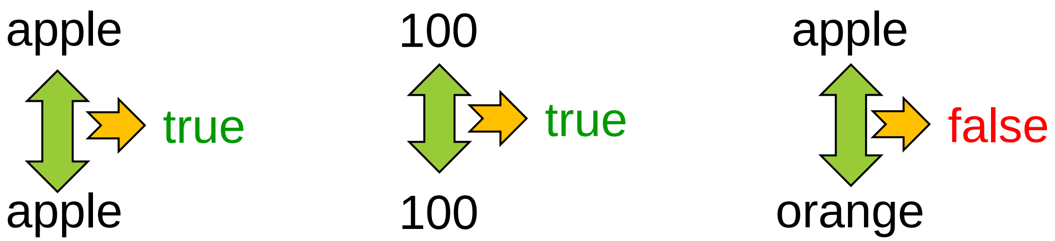
A is true if A_1 and A_2 is true.

- Rule Ordering
  - Execute sequentially, from top to down
  - A executed first and then B
  - A :- A_1, A_2 .
  - B :- B_1, B_2 .

- Goal Ordering
  - Ordering of terms within the body of a rule
  - Execute sequentially, from left to right
  - A_1 executed first and then A_2
  - A :- A_1, A_2 .

Pattern Matching

- Matching constants

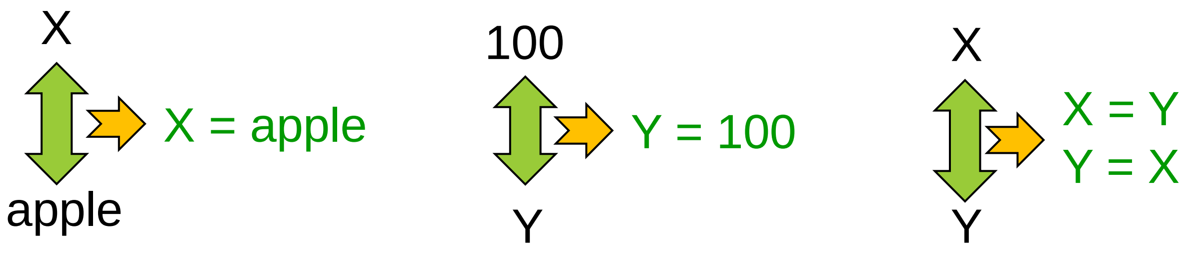


apple
true
apple

100
true
100

apple
false
orange

- Matching variables



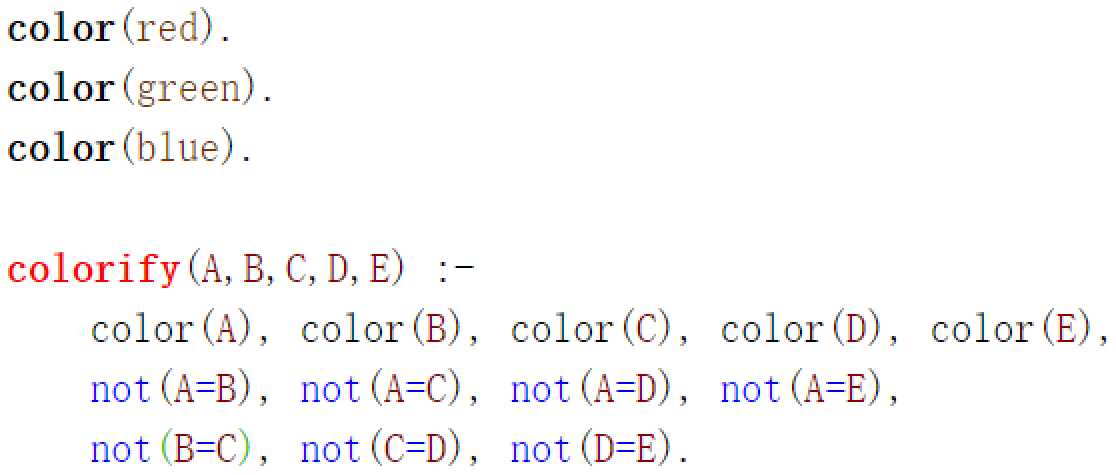
X
X = apple
apple

100
Y = 100
Y

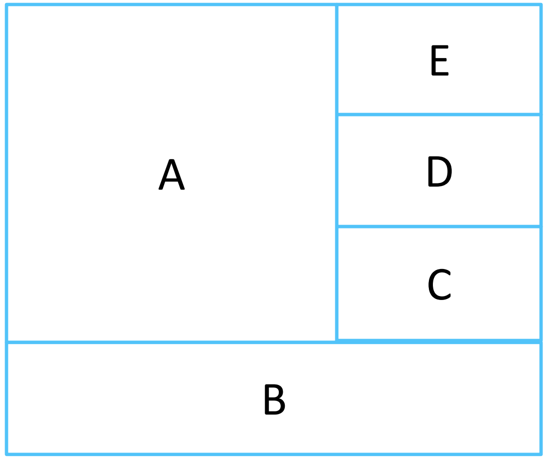
X
X = Y
Y = X
Y

Example: Graph Coloring

- No adjacent area can have the same color
- 3 colors: red, green, blue

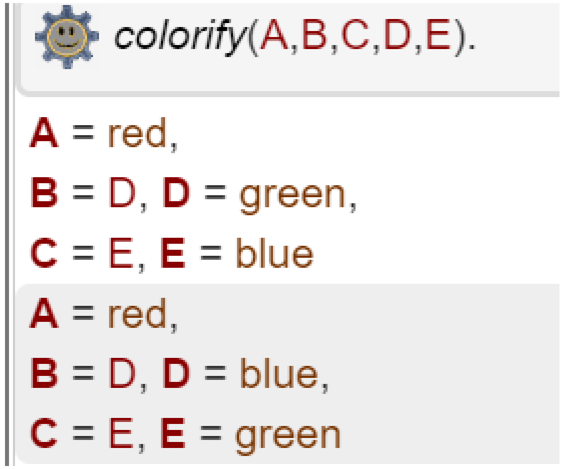


```
color(red).
color(green).
color(blue).

colorify(A,B,C,D,E) :-
    color(A), color(B), color(C), color(D), color(E),
    not(A=B), not(A=C), not(A=D), not(A=E),
    not(B=C), not(C=D), not(D=E).
```

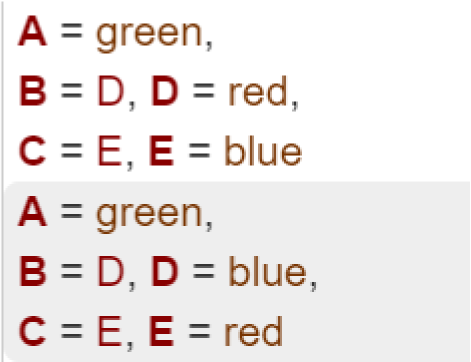


Example: Graph Coloring

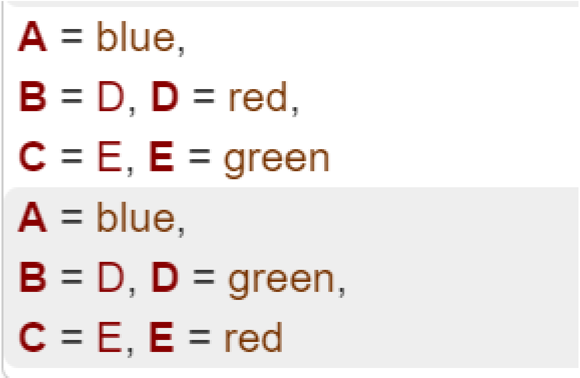
- 6 possible answers



```
colorify(A,B,C,D,E).

A = red,
B = D, D = green,
C = E, E = blue
A = red,
B = D, D = blue,
C = E, E = green
```



```
A = green,
B = D, D = red,
C = E, E = blue
A = green,
B = D, D = blue,
C = E, E = red
```



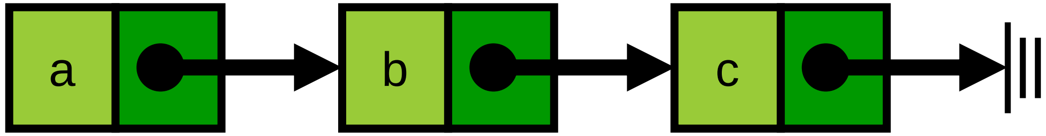
```
A = blue,
B = D, D = red,
C = E, E = green
A = blue,
B = D, D = green,
C = E, E = red
```

Lists as Compound Terms

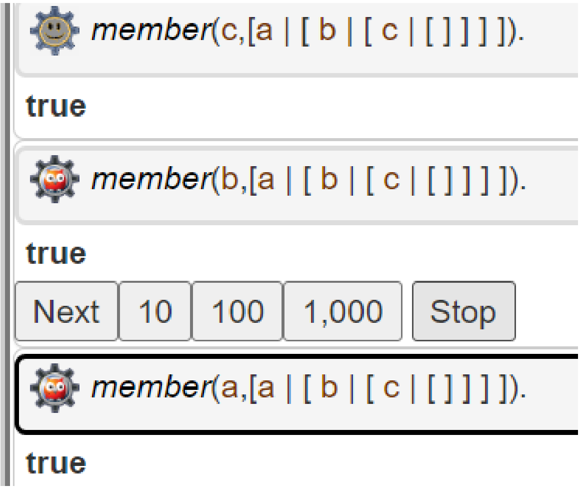
- List as a compound term for a variable-length linear sequence
  - (a, b, c, d, e)
  - (x, y, z)
  - …
- Functor: l/2
  - l(*content*, *rest of items in the list*)
- A list (a, b, c) can be represented as
  - l(a, l(b, l(c, nil)))
  - where nil represent an empty list

Lists as Compound Terms

- l(a, l(b, l(c, nil)))
- Such usage of compound terms resembles a *linked list*

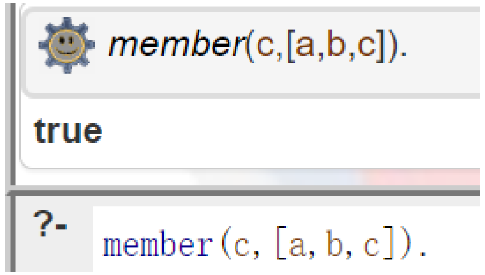


- Lists
  - Prolog's built-in list representation
  - [] denote empty list
  - [X|Y], X is the head and Y is the tail of list
  - Example: a list (a, b, c)
    - [a | [b | [c | [ ]]]]
    - [a,b,c]
    - [a,b|[c]]
    - [a|[b,c]]

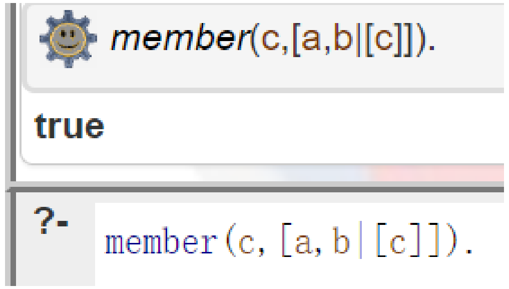


```
member(c,[a | [ b | [ c | [ ] ] ] ]).
true
member(b,[a | [ b | [ c | [ ] ] ] ]).
true
```

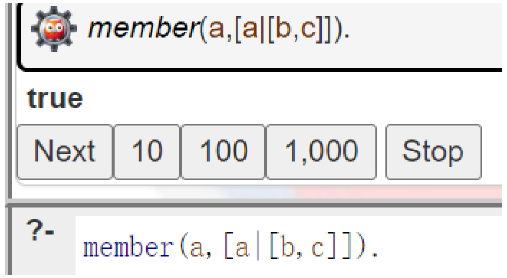
```
member(a,[a | [ b | [ c | [ ] ] ] ]).
true
```



```
member(c,[a,b,c]).
true
?- member(c, [a, b, c]).
```



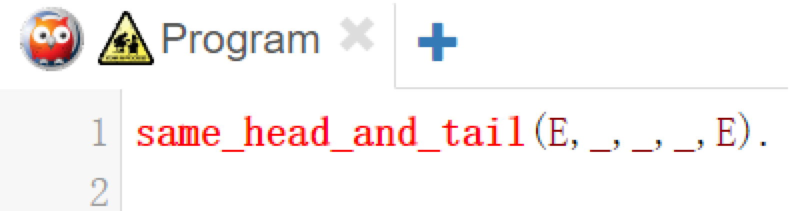
```
member(c,[a,b|[c]]).
true
?- member(c, [a, b|[c]]).
```



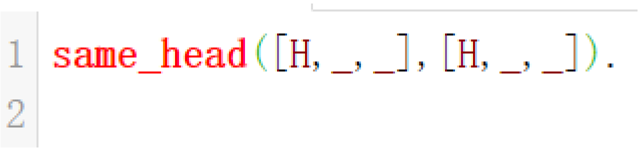
```
member(a,[a|[b,c]]).
true
```

```
?- member(a, [a|[b, c]]).
```

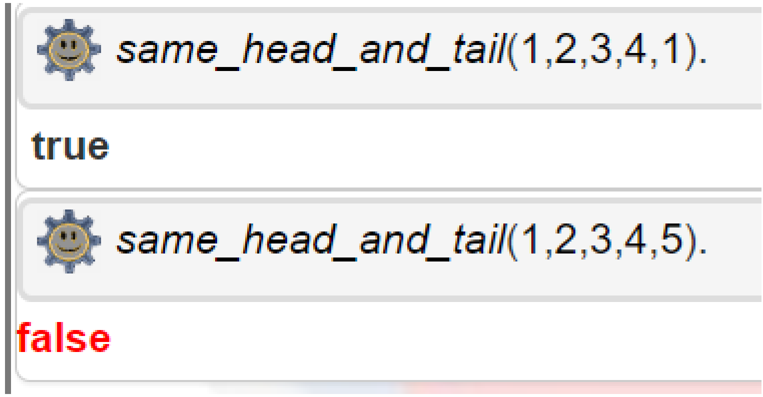
Example-list matching



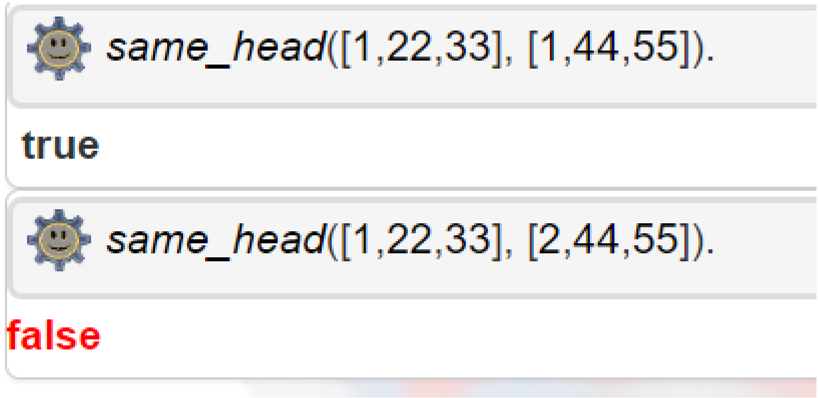
```
same_head_and_tail(E,_,_,_,E).
```



```
same_head([H,_,_],[H,_,_]).
```



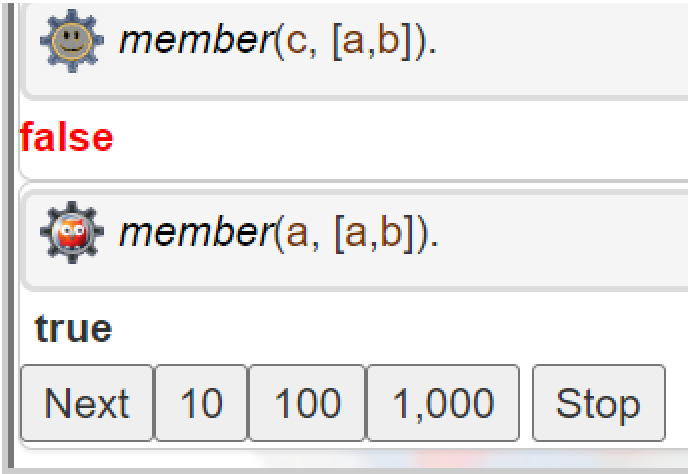
```
same_head_and_tail(1,2,3,4,1).
true
same_head_and_tail(1,2,3,4,5).
false
```



```
same_head([1,22,33], [1,44,55]).
true
same_head([1,22,33], [2,44,55]).
false
```

Example: member/2

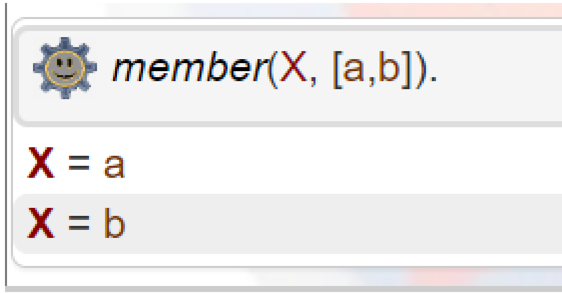
- membership of list
  - member(X,[X|_]).
    - % Fact: X is member of list begin with X
  - member(X,[_|L]) :- member(X,L).
    - /* Rule: X is member of list if it is member of tail */
- Queries



```
member(c, [a,b]).
false
member(a, [a,b]).
true
```

Example: member/2

- Query for members of a list
  - |?- member(X, [a,b]).
  - X = a ? ;
  - X = b ? ;
  - no

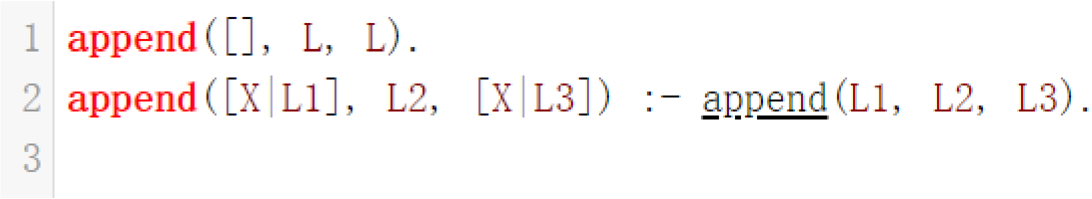


```
member(X, [a,b]).
X = a
X = b
```

- Typing a semicolon (;) for more answers
- Pressed Enter to stop the query

append/3

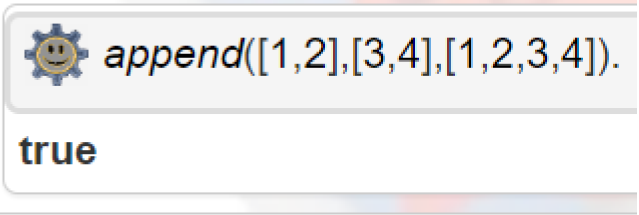
- append one list L2 to another list L1
  - append([],L,L).
    - Fact: Appending L2 to an empty list results in itself
  - append([H|X],Y,[H|Z]) :-append(X,Y,Z).
    - Rule: L3 is the result of appending L2 to L1 if L1 and L3 are non-empty list, they both have the same head, and the tail of L3 is the result of appending L2 to the tail of L1.



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

```

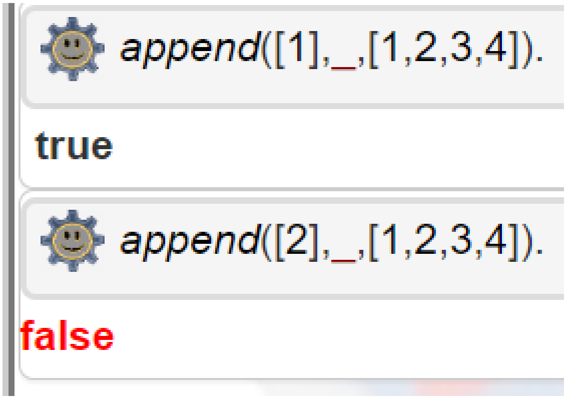
Example: append/3

- Check concatenation



```
append([1,2],[3,4],[1,2,3,4]).
true
```

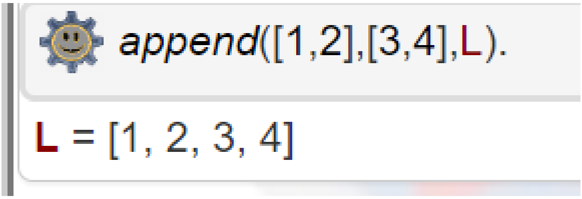
- Check head element



```
append([1],_,[1,2,3,4]).
true
append([2],_,[1,2,3,4]).
false
```

Example: append/3

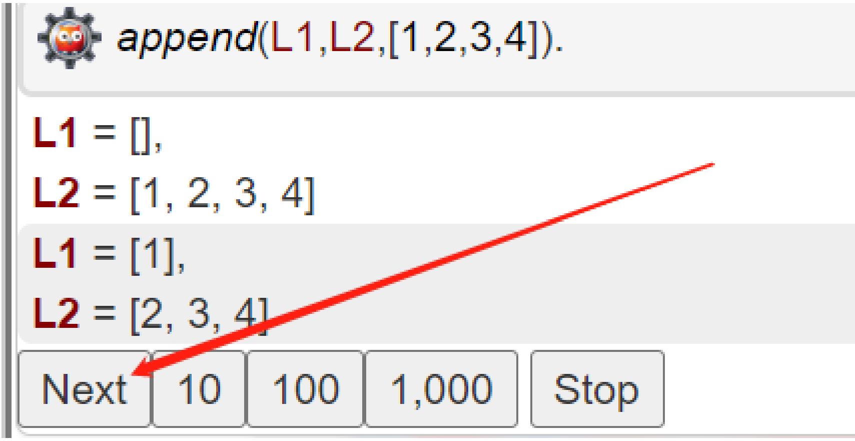
- Concatenate 2 lists



```
append([1,2],[3,4],L).
```

```
L = [1, 2, 3, 4]
```

- Decompose a list

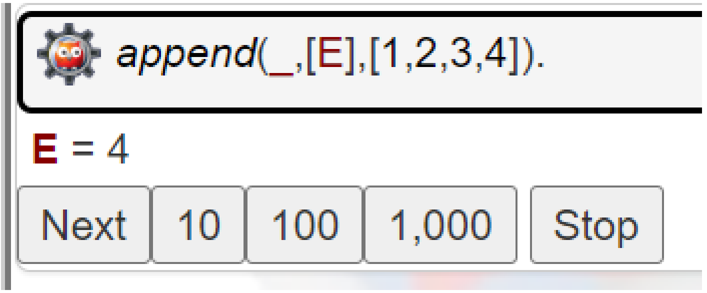


```
append(L1,L2,[1,2,3,4]).
```

```
L1 = [],
L2 = [1, 2, 3, 4]
L1 = [1],
L2 = [2, 3, 4]
```

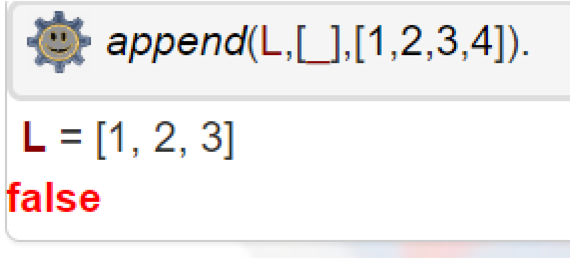
Example: append/3

- Generate last element



```
append(_,[E],[1,2,3,4]).
E = 4
```

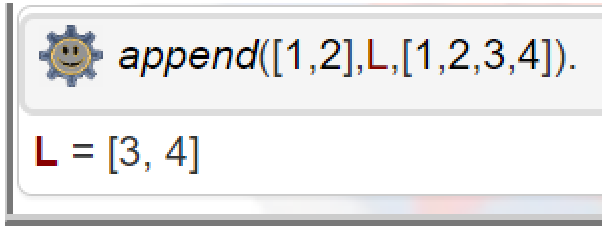
- Delete last element



```
append(L,[_],[1,2,3,4]).
L = [1, 2, 3]
false
```

Example: append/3

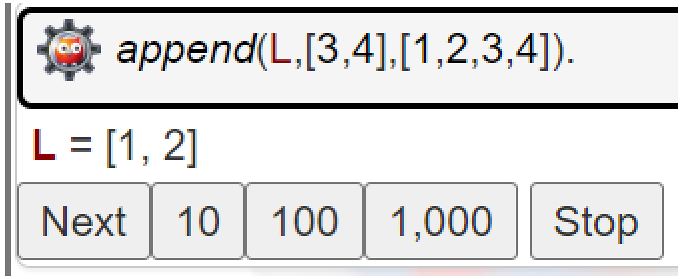
- subtract first part of a list



```
append([1,2],L,[1,2,3,4]).
```

```
L = [3, 4]
```

- subtract last part of a list



```
append(L,[3,4],[1,2,3,4]).
```

```
L = [1, 2]
```

Try by yourself

- find head element
  - append([H],_,[1,2,3,4]).
- check tail of a list
  - append(_,[2,3,4],[1,2,3,4]).
  - append(_,[1,3,4],[1,2,3,4]).
- check last element
  - append(_,[4],[1,2,3,4]).
  - append(_,[1],[1,2,3,4]).
- test element
  - append(_,[2|_],[1,2,3,4]).
  - append(_,[7|_],[1,2,3,4]).
- generate an element
  - append(_,[E|_],[1,2,3,4]).

Try by yourself

- delete any element (L is answer)
  - append(L1,[3|L2],[1,2,3,4]), append(L1,L2,L).
- delete any element (L is answer)
  - append(L1,[_|L2],[1,2,3,4]), append(L1,L2,L).
- check sublist
  - append(_,[2,3],L), append(L,_,[1,2,3,4]).
  - append(_,[1,3],L), append(L,_,[1,2,3,4]).
- generate sublist (X is answer)
  - append(_,X,L), append(L,_,[1,2,3,4]).

Try by yourself

- check rotate left
  - append([H],T,[1,2,3,4]), append(T,[H],[2,3,4,1]).
  - append([H],T,[1,2,3,4]), append(T,[H],[2,3,4,5]).
- generate rotate left (L is answer)
  - append([H],T,[1,2,3,4]), append(T,[H],L).
- check rotate right (L is answer)
  - append(L1,[E],[1,2,3,4]), append([E],L1,[4,1,2,3]).
  - append(L1,[E],[1,2,3,4]), append([E],L1,[4,3,2,1]).
- generate rotate right (L is answer)
  - append(L1,[E],[1,2,3,4]), append([E],L1,L).

Try by yourself

- computing triplets (L is answer)
  - append([1,2],[1,2],L1), append(L1,[1,2],L).
- Note the independence of goal order
  - append(L1,[1,2],L), append([1,2],[1,2],L1).

Assignment 4

- In a certain university, MOBA (multiplayer online battle arena) is a very popular game genre among all the students.
- The top players are skillful and also charming.
- 5 top MOBA players (the same 5 persons) participate in a popularity contest this year and last year.
- They are ranked rank 1 to rank 5 by popularity in each of the two contests.
- Each player also is a member of some club (yes, they manage to have social lives other than playing MOBA games) and has their own favorite food, sport and music.
- You will create facts and rules based on the information we give you.
- Then, based on those, you can use Prolog queries to find out who is the most popular, least popular, likes chicken, etc..

Basic information

- 5 player names
  - rookie, jack, ning, viper, scout
- 5 popularity ranks
  - In the Prolog facts and rules, the ranking is represented by a list of 5 players
  - The first player in the list is the most popular one (i.e. rank 1)
  - The second player in the list is the almost most popular one (i.e. rank 2)
  - The third player in the list is the medium popular one (i.e. rank 3)
  - The fourth player in the list is the almost least popular one (i.e. rank 4)
  - The last player in the list is the least popular one (i.e. rank 5)
- 5 clubs
  - royal club, killer club, elf club, knight club, magic club
- 5 favorite food
  - chicken, hamburger, hotpot, chips, bread
- 5 favorite sport
  - basketball, swim, baseball, football, running
- 5 favorite music
  - jazz, blues, pop music, rock music, classical music

Rules for this year

1)jack likes chicken, rookie comes from killer_club, jack is more popular than rookie

2)scout likes playing baseball, viper like jazz_music, scout is more popular than viper

3)The almost most popular player likes pop_music and he is more popular than the one who likes hamburger and playing baseball

4)ning likes hot_pot, he is less popular than the one who comes from royal_club and likes rock_music

5)The player comes from magic_club is less popular than the one who likes swimming and jazz_music

6)The most popular player likes playing football

7)The medium popular player likes blues

8)The least popular player likes running and classical_music

9)The almost least popular player likes chips and he is more popular than the one who comes from magic_club and likes hot_pot

10)The player who likes hamburger is less popular than the one who likes bread

11)The player who comes from royal_club and likes rock_music is a rival of the one who comes from killer_club, and is also more popular than that rival

12)The player who likes playing baseball and blues is a rival of the one who likes bread and playing basketball, and is also less popular than that rival

13)The player who likes chips and swimming is a rival of the one who comes from elf_club, and is also less popular

14)The player who comes from knight_club and likes chips is a rival of the one who likes running and classical_music, and is also more popular

Rules for last year

1)viper is medium popular in last year

2)ning is the least popular

3)scout is the rival with jack and scout is more popular than jack

4)jack is more popular than viper and they are also rivals

5)viper is the rival with rookie and rookie is less popular than viper

6)rookie is rival with ning and rookie is more popular than ning

Implementations

- A reference .pl file as answer sheet
- Facts
  - more_popular(A,B, [A,B,_,_,_])…
  - less_popular(A,B, [A,B,_,_,_])…
  - most_popular(), least_popular(), medium_popular(), almost_most_popular(), almost_least_popular()
  - rivals(A,B, [A,B,_,_,_]), rivals(A,B, [_,B,A,_,_])
- Rules
  - Translate rules with compound terms

Testcases example

- Query (who likes Chicken?)
  - this_year_ranking(_Ranking), member(player(X,_,chicken,_,_), _Ranking).
  - Answer: X=jack
- Query (who comes from knight_club?)
  - this_year_ranking(_Ranking), member(player(X,knight_club,_,_,_), _Ranking).
  - Answer: X=viper
- Query (who are the rival(s) of viper?):
  - this_year_ranking(_Ranking), rivals(player(viper,_,_,_,_),player(X,_,_,_,_),_Ranking).
  - Answer: X=ning; X=scout
- query (who are the rival(s) of the student who likes classical music?):
  - this_year_ranking(_Ranking), rivals(player(_,_,_,_,classical_music),player(X,_,_,_,_),_Ranking).
  - Answer: X=viper;
- global answer
  - this_year_ranking(Ranking).

Testcases example

- Query (Who is most popular?)
  - last_year_ranking(_Ranking),most_popular(player(X,_,_,_,_),_Ranking).
  - Answer: X=scout
- Query (Who is almost most popular?)
  - last_year_ranking(_Ranking),almost_most_popular(player(X,_,_,_,_),_Ranking).
  - Answer: X=jack
- Query (Who is almost least popular?)
  - last_year_ranking(_Ranking),almost_least_popular(player(X,_,_,_,_),_Ranking).
  - Answer: X=rookie