First Prolog Programming Assignment Specification

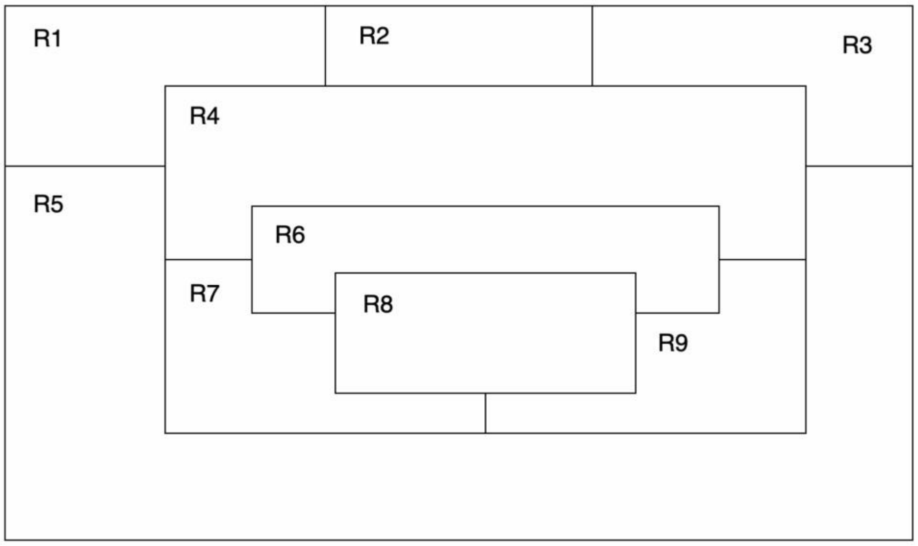
By JIUN KIM

Learning Abstract

This assignment is the first assignment of Prolog programming language. I create the map coloring program based on lecture notes. Task 1 and Task 2 were able to mimic the program through Lesson 4 and Lesson 5 and learn the basic concept of prolog. Task 3 was KB(knowledge based) on Pokemon trading cards with listing their stats. Task 4 is based on lesson 5: List Processing in Prolog but more challenging task than other tasks. It was great experience to make noun phrase and sentence by random predications.

Task 1: Map Coloring

1. An image of the given map, with the regions labelled.



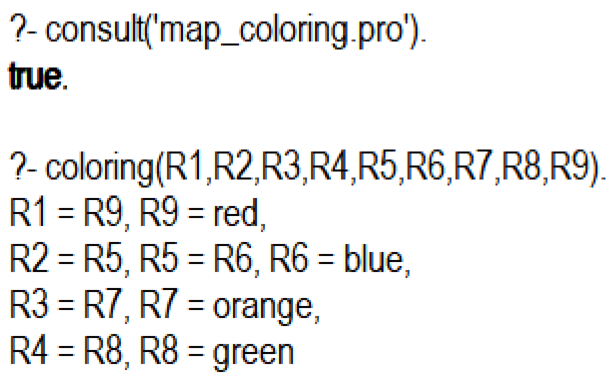
Works for Task 1

Place the following items within the “Task 1: Map Coloring” section of your presentation document:

1. Source Code

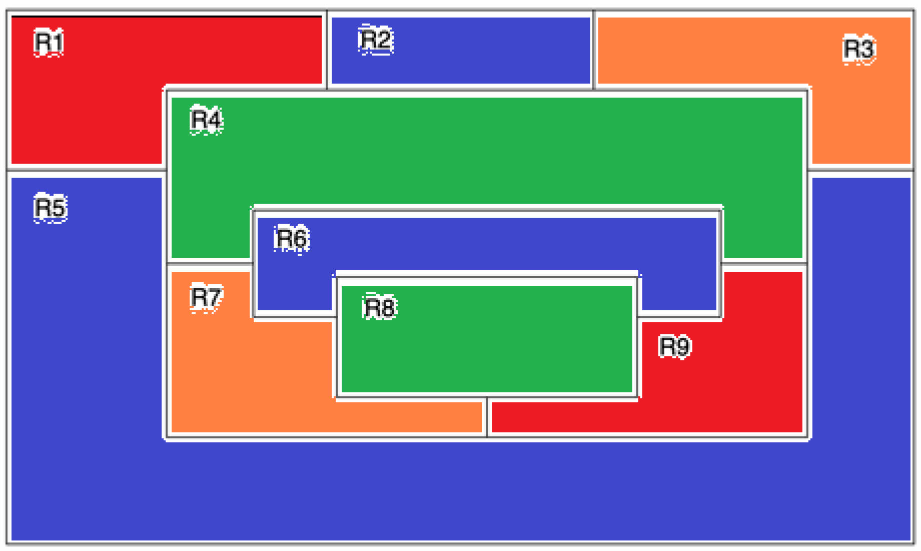
```
% ---------------------------------------------------------------------------
% File: map_coloring.pro
% Line: Program to find a 4 color map rendering for rectangles .
% More: The colors used will be red, blue, green orange.
% More: Rn(number) means each rectnagle.
% ---------------------------------------------------------------------------
% different(X,Y) :: X is not equal to Y
different(red,blue).
different(red,green).
different(red,orange).
different(green,blue).
different(green,orange).
different(green,red).
different(blue,green).
different(blue,orange).
different(blue,red).
different(orange,blue).
different(orange,green).
different(orange,red).
% ---------------------------------------------------------------------------
% coloring(R1,R2,R3,R4,R5,R6,R7,R8,R9) :: each rectangle's colo
% so that none of the rectnagles sharing a border are the same color.
coloring(R1,R2,R3,R4,R5,R6,R7,R8,R9) :-
different(R1,R2),
different(R1,R4),
different(R1,R5),
different(R2,R3),
different(R2,R4),
different(R3,R4),
different(R3,R5),
different(R4,R5),
different(R4,R6),

different(R4,R7),
different(R4,R9),
different(R5,R7),
different(R5,R9),
different(R6,R7),
different(R6,R8),
different(R6,R9),
different(R7,R8),
different(R7,R9),
different(R8,R9).
```

2. The demo of your program.



```
?- consult('map_coloring.pro').
true.

?- coloring(R1,R2,R3,R4,R5,R6,R7,R8,R9).
R1 = R9, R9 = red,
R2 = R5, R5 = R6, R6 = blue,
R3 = R7, R7 = orange,
R4 = R8, R8 = green
```

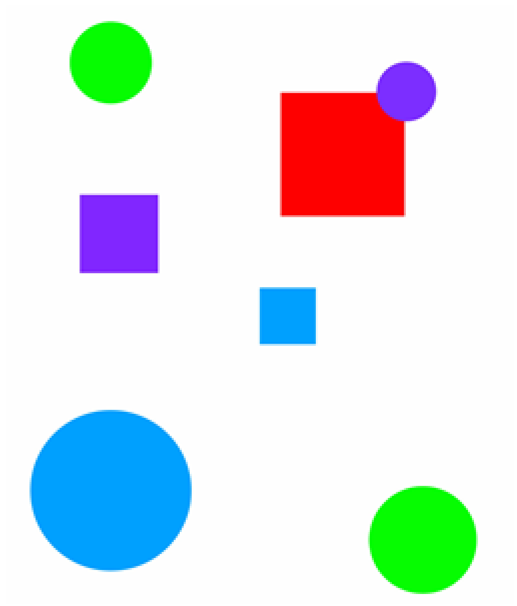
3. Colored Map



.

Task 2: The Floating Shapes World

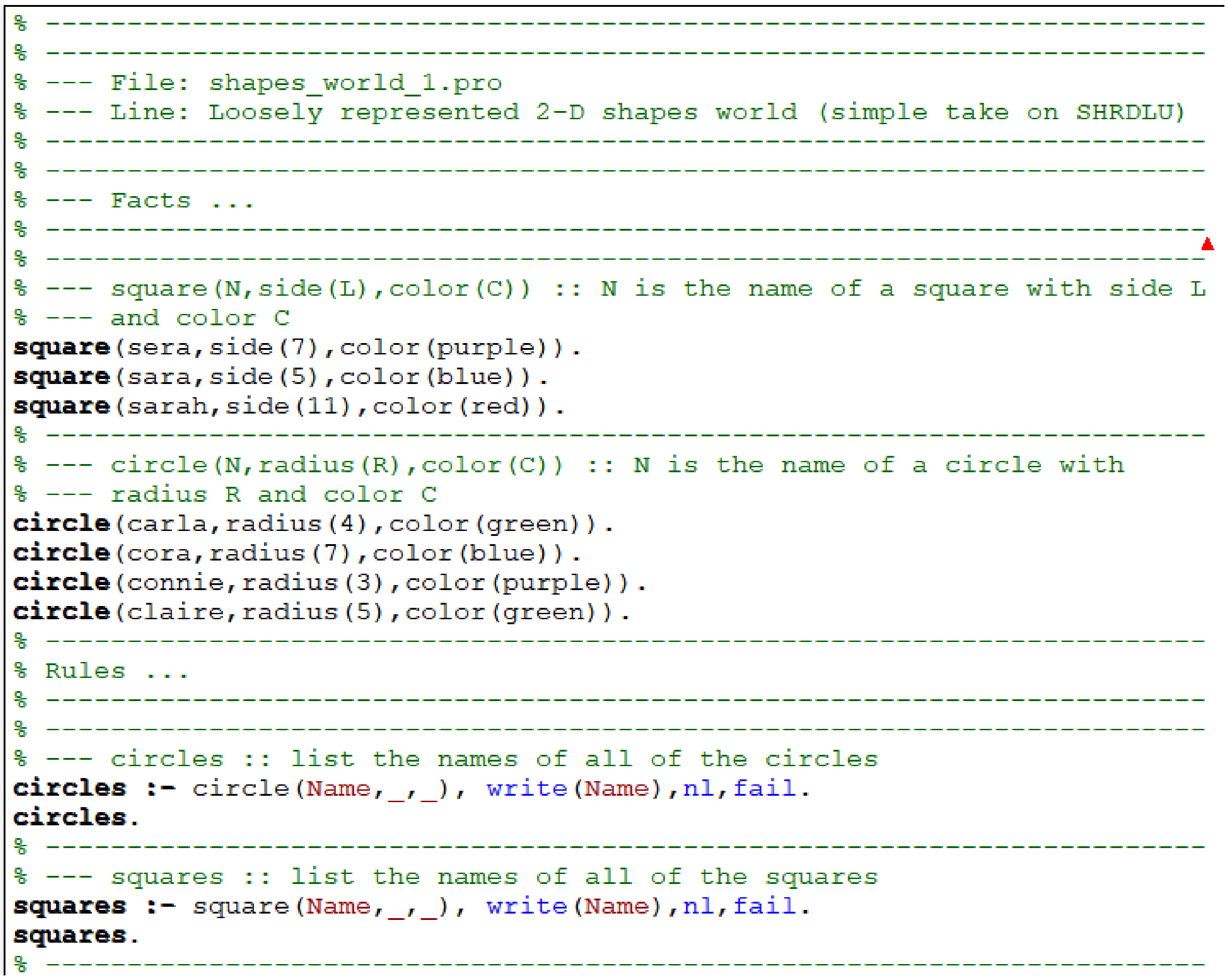
Original Image.



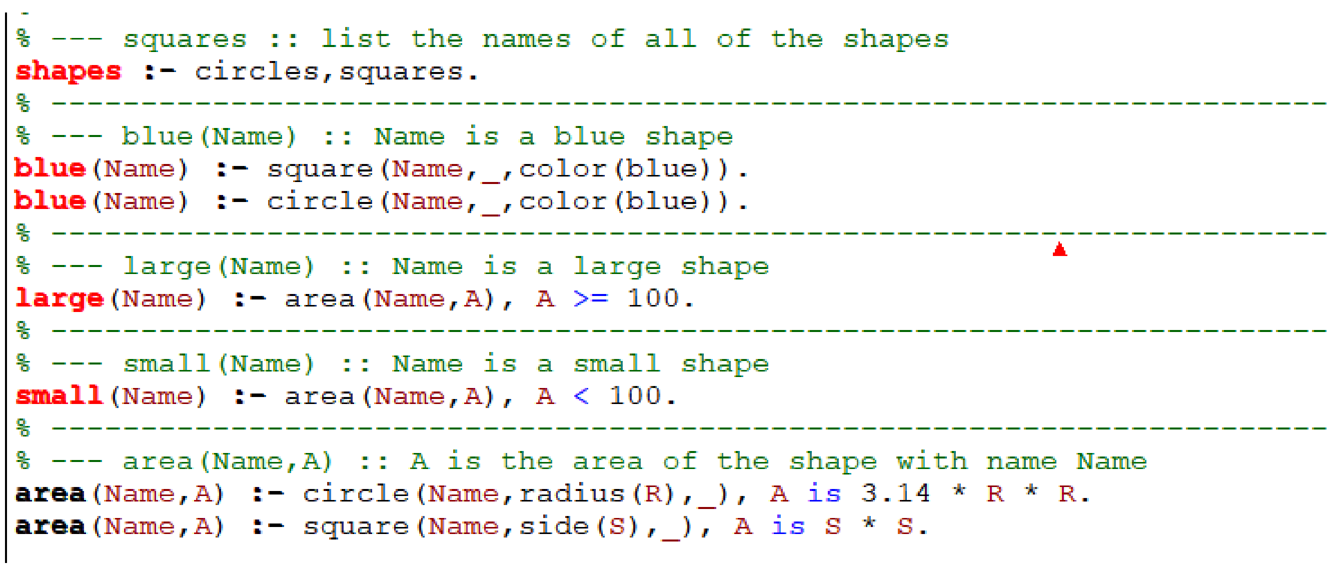
Works for Task 2

.

1. The Prolog KB



```
% ----------------------------------------------------------------------
% ----------------------------------------------------------------------
% --- File: shapes_world_1.pro
% --- Line: Loosely represented 2-D shapes world (simple take on SHRDLU)
% ----------------------------------------------------------------------
% ----------------------------------------------------------------------
% --- Facts ...
% ----------------------------------------------------------------------
% ----------------------------------------------------------------------
% --- square(N,side(L),color(C)) :: N is the name of a square with side L
% --- and color C
square(sera,side(7),color(purple)).
square(sara,side(5),color(blue)).
square(sarah,side(11),color(red)).
% ----------------------------------------------------------------------
% --- circle(N,radius(R),color(C)) :: N is the name of a circle with
% --- radius R and color C
circle(carla,radius(4),color(green)).
circle(cora,radius(7),color(blue)).
circle(connie,radius(3),color(purple)).
circle(claire,radius(5),color(green)).
% ----------------------------------------------------------------------
% Rules ...
% ----------------------------------------------------------------------
% ----------------------------------------------------------------------
% --- circles :: list the names of all of the circles
circles :- circle(Name,_,_), write(Name),nl,fail.
circles.
% ----------------------------------------------------------------------
% --- squares :: list the names of all of the squares
squares :- square(Name,_,_), write(Name),nl,fail.
squares.
% ----------------------------------------------------------------------
```

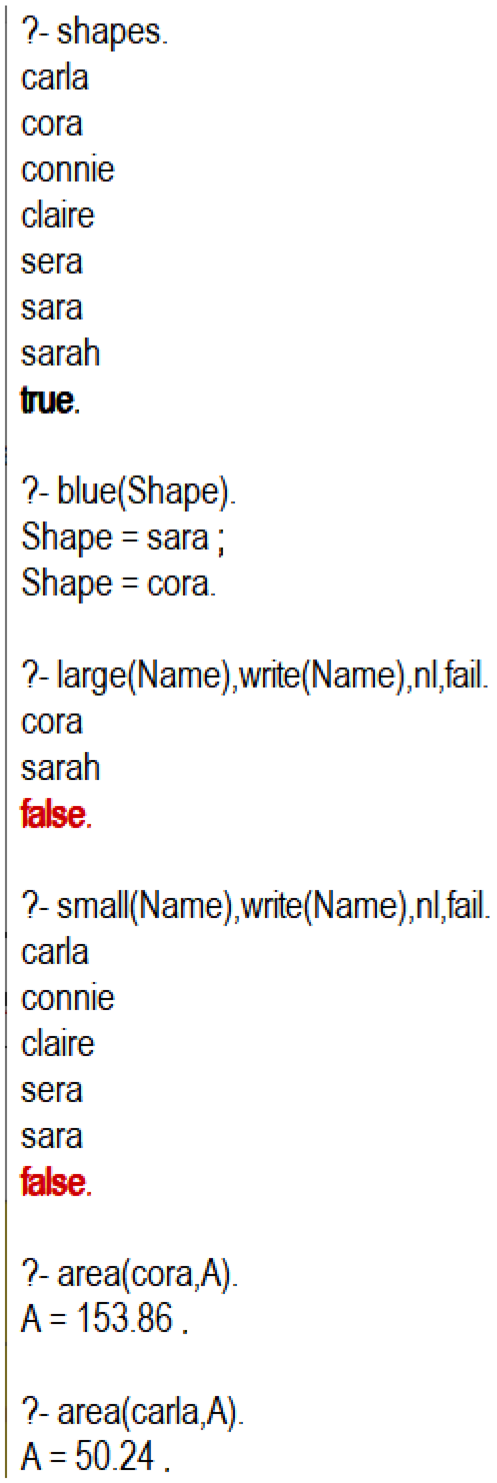


```
-
% --- squares :: list the names of all of the shapes
shapes :- circles,squares.
% ----------------------------------------------------------------------
% --- blue(Name) :: Name is a blue shape
blue(Name) :- square(Name,_,color(blue)).
blue(Name) :- circle(Name,_,color(blue)).
% ----------------------------------------------------------------------
% --- large(Name) :: Name is a large shape
large(Name) :- area(Name,A), A >= 100.
% ----------------------------------------------------------------------
% --- small(Name) :: Name is a small shape
small(Name) :- area(Name,A), A < 100.
% ----------------------------------------------------------------------
% --- area(Name,A) :: A is the area of the shape with name Name
area(Name,A) :- circle(Name,radius(R),_), A is 3.14 * R * R.
area(Name,A) :- square(Name,side(S),_), A is S * S.
```

2. The demo that you generate (corresponding to that presented the lesson).

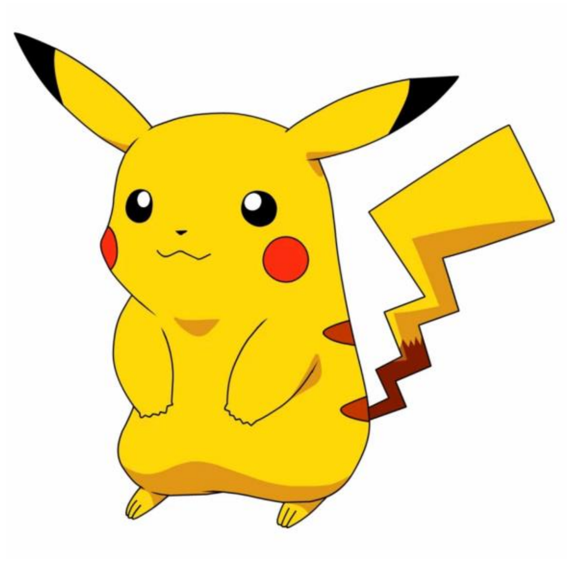
```
?- consult('shapes_world_1.pro').
true.

?- listing(squares).
squares :-
    square(Name, _, _),
    write(Name),
    nl,
    fail.
squares.

true.

?- squares
|
|    .
sera
sara
sarah
true.

?- listing(circles).
circles :-
    circle(Name, _, _),
    write(Name),
    nl,
    fail.
circles.

true.
```

```
?- circles.
carla
cora
connie
claire
true.

?- listing(shapes).
shapes :-
    circles,
    squares.

true.
```



```
?- shapes.
carla
cora
connie
claire
sera
sara
sarah
true.

?- blue(Shape).
Shape = sara ;
Shape = cora.

?- large(Name),write(Name),nl,fail.
cora
sarah
false.

?- small(Name),write(Name),nl,fail.
carla
connie
claire
sera
sara
false.

?- area(cora,A).
A = 153.86 .

?- area(carla,A).
A = 50.24 .
```

Task 3: Pokemon KB Interaction and Programming



Preliminary Note

For this task, you will need to incorporate, into your computational world, the knowledge base on pokemon trading cards that I am providing as a sibling document to the one that you are now reading.

You should probably just copy and paste the pokemon code, look it over, and then make sure that it loads into Prolog. It works for me, so if it doesn't work for you, that is probably because of an "error in transmission" that you will need to sort out.

Work for Task3 Part1

1.Demo

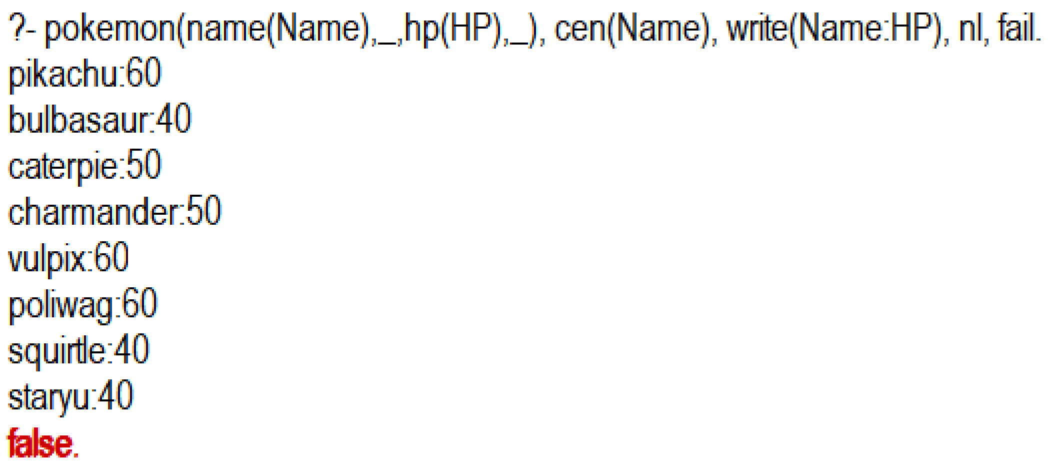
```
?- consult('pokemon.pro').
true.

?- cen(pikachu).
true.

?- cen(raichu).
false.

?- cen(Name).
Name = pikachu ;
Name = bulbasaur ;
Name = caterpie ;
Name = charmander ;
Name = vulpix ;
Name = poliwag ;
Name = squirtle ;
Name = staryu.

?- cen(Name), write(Name), nl,fail.
pikachu
bulbasaur
caterpie
charmander
vulpix
poliwag
squirtle
staryu
false.

?- evolves(squirtle,wartortle).
true.

?- evolves(wartortle,squirtle).
false.

?- evolves(squirtle,blastoise).
false.
```

```
?- evolves(A,B), evolves(B,C).
A = bulbasaur,
B = ivysaur,
C = venusaur ;
A = caterpie,
B = metapod,
C = butterfree ;
A = charmander,
B = charmeleon,
C = charizard ;
A = poliwag,
B = poliwhirl,
C = poliwrath ;
A = squirtle,
B = wartortle,
C = blastoise ;
false.
```

```
?- evloves(A,B), evolves(B,C), write(A-->C), nl, fail.
Correct to: "evolves(A,B)"? yes
bulbasaur-->venusaur
caterpie-->butterfree
charmander-->charizard
poliwag-->poliwrath
squirtle-->blastoise
false.

?- pokemon(name(Name),_,_,_), write(Name),nl,fail.
pikachu
raichu
bulbasaur
ivysaur
venusaur
caterpie
metapod
butterfree
charmander
charmeleon
charizard
vulpix
ninetails
poliwag
poliwhirl
poliwrath
squirtle
wartortle
blastoise
staryu
starmie
false.

?- pokemon(name(Name),fire,_,_), write(Name),nl,fail.
charmander
charmeleon
charizard
vulpix
ninetails
false.
```

```
?- pokemon(name(Name),Kind,_,_), write(nks(Name,kind(Kind))),nl,fail.
nks(pikachu,kind(electric))
nks(raichu,kind(electric))
nks(bulbasaur,kind(grass))
nks(ivysaur,kind(grass))
nks(venusaur,kind(grass))
nks(caterpie,kind(grass))
nks(metapod,kind(grass))
nks(butterfree,kind(grass))
nks(charmander,kind(fire))
nks(charmeleon,kind(fire))
nks(charizard,kind(fire))
nks(vulpix,kind(fire))
nks(ninetails,kind(fire))
nks(poliwag,kind(water))
nks(poliwhirl,kind(water))
nks(poliwrath,kind(water))
nks(squirtle,kind(water))
nks(wartortle,kind(water))
nks(blastoise,kind(water))
nks(staryu,kind(water))
nks(starmie,kind(water))
false.

?- pokemon(name(N),_,_,attack(waterfall,_)).
N = wartortle .

?- pokemon(name(N),_,_,attack(poison-powder,_)).
N = venusaur .
```

```
?- pokemon(_,water,_,attack(Attack,_)), write(Attack),nl,fail.
water-gun
amnesia
dashing-punch
bubble
waterfall
hydro-pump
slap
star-freeze
false.

?- pokemon(name(poliwhirl),_hp(HP),_).
ERROR: Syntax error: Operator expected
ERROR: pokemon(name(poliwhirl),_h
ERROR: ** here **
ERROR: p(HP),_) .
?- pokemon(name(poliwhirl),_,hp(HP),_).
HP = 80.

?- pokemon(name(butterfree),_,hp(HP),_).
HP = 130.

?- pokemon(name(Name),_,hp(HP),_), HP  > 85, write(Name), nl, fail.
raichu
venusaur
butterfree
charizard
ninetails
poliwrath
blastoise
false.

?- pokemon(_,_,_,attack(N,Damage)), Damage >60,  write(N), nl, fail.
thunder-shock
poison-powder
whirlwind
royal-blaze
fire-blast
false.
```



```
?- pokemon(name(Name),_,hp(HP),_), cen(Name), write(Name:HP), nl, fail.
pikachu:60
bulbasaur:40
caterpie:50
charmander:50
vulpix:60
poliwag:60
squirtle:40
staryu:40
false.
```

Part 2: Programs

1. KB for Task 3

```
% ---------------------------------------------------------------------------
% ---------------------------------------------------------------------------
% --- File: pokemon.pro
% --- Line: Loosely represented 2-D shapes world (simple take on SHRDLU)
% ---------------------------------------------------------------------------

% ---------------------------------------------------------------------------
% --- cen(P) :: Pokemon P was "creatio ex nihilo"

cen(pikachu).
cen(bulbasaur).
cen(caterpie).
cen(charmander).
cen(vulpix).
cen(poliwag).
cen(squirtle).
cen(staryu).

% ---------------------------------------------------------------------------
% --- evolves(P,Q) :: Pokemon P directly evolves to pokemon Q

evolves(pikachu,raichu).
evolves(bulbasaur,ivysaur).
evolves(ivysaur,venusaur).
evolves(caterpie,metapod).
evolves(metapod,butterfree).
evolves(charmander,charmeleon).
evolves(charmeleon,charizard).
evolves(vulpix,ninetails).
evolves(poliwag,poliwhirl).
evolves(poliwhirl,poliwrath).
evolves(squirtle,wartortle).
evolves(wartortle,blastoise).
evolves(staryu,starmie).

% ---------------------------------------------------------------------------
% --- pokemon(name(N),T,hp(H),attach(A,D)) :: There is a pokemon with
% --- name N, type T, hit point value H, and attach named A that does
% --- damage D.

pokemon(name(pikachu), electric, hp(60), attack(gnaw, 10)).
pokemon(name(raichu), electric, hp(90), attack(thunder-shock, 90)).

pokemon(name(bulbasaur), grass, hp(40), attack(leech-seed, 20)).
pokemon(name(ivysaur), grass, hp(60), attack(vine-whip, 30)).
pokemon(name(venusaur), grass, hp(140), attack(poison-powder, 70)).

pokemon(name(caterpie), grass, hp(50), attack(gnaw, 20)).
pokemon(name(metapod), grass, hp(70), attack(stun-spore, 20)).
pokemon(name(butterfree), grass, hp(130), attack(whirlwind, 80)).
pokemon(name(charmander), fire, hp(50), attack(scratch, 10)).
pokemon(name(charmeleon), fire, hp(80), attack(slash, 50)).
pokemon(name(charizard), fire, hp(170), attack(royal-blaze, 100)).

pokemon(name(vulpix), fire, hp(60), attack(confuse-ray, 20)).
pokemon(name(ninetails), fire, hp(100), attack(fire-blast, 120)).

pokemon(name(poliwag), water, hp(60), attack(water-gun, 30)).
pokemon(name(poliwhirl), water, hp(80), attack(amnesia, 30)).
pokemon(name(poliwrath), water, hp(140), attack(dashing-punch, 50)).

pokemon(name(squirtle), water, hp(40), attack(bubble, 10)).
pokemon(name(wartortle), water, hp(80), attack(waterfall, 60)).
pokemon(name(blastoise), water, hp(140), attack(hydro-pump, 60)).

pokemon(name(staryu), water, hp(40), attack(slap, 20)).
pokemon(name(starmie), water, hp(60), attack(star-freeze, 20)).

% --- display_names :: list of all pokemon.
display_names :- pokemon(name(Name),_,_,_), write(Name),nl,fail.
display_names.

% --- display_attacks :: list of all atacks.
display_attacks :- pokemon(_,_,_,attack(N,_)), write(N),nl,fail.
display_attacks.

% --- powerful(Name) :: powerful pokemon, pokemon attack with more than
% --- 55 units damage.
powerful(Name) :-
pokemon(name(Name),_,_,attack(_,U)), U>55.

% --- tough(Name) :: pokemon can absorb more than 100 units of damage.
tough(Name) :-
pokemon(name(Name),_,hp(U),_), U>=100.

% --- type(Name,Type) :: the name and type of pokemon, which succeeds
% --- only if the named pokemon is of the specified type.
type(Name,Type) :-
pokemon(name(Name),Type,_,_).

% --- dump_kind(Type) :: display all of specified type of pokemon.
dump_kind(Type) :-
listing(pokemon(_,Type,_,_)), nl,fail.

% --- display_cen :: display all names of the "creatio ex nihilo" pokemon.
display_cen:-
cen(Name),write(Name),nl,fail.
display_cen.

% --- family(CEN) :: predicate, presumed to be a "creatio ex nihilo" pokemon,
% --- which displays the "evolutionary family" of the specified pokemon.
```

```
family(Cen) :- evolves(Cen,A), write(Cen), write(' '), write(A),
evolves(A,B), write(' '), write(B).

% --- families :: display all of the evolitonary pokemon families.
families:-
cen(Cen), evolves(Cen,A), nl, write(Cen), write(' '), write(A), evolves(A,B),
write(' '), write(B), fail.
familes.

% --- lineage(Name) :: display all of the information for the pokemon and
% --- for each subsequent pokemon in the evolutionaly lineage of pokemon.
lineage(Name):-
pokemon(name(Name),Type,hp(HP),attack(Attack,Damage)),
write(pokemon(name(Name),Type,hp(HP),attack(Attack,Damage))),nl,

evolves(Name,A),
pokemon(name(A),Type,hp(HP_A),attack(Attack_A,Damage_A)),
write(pokemon(name(A),Type,hp(HP_A),attack(Attack_A,Damage_A))),nl,

evolves(A,B),
pokemon(name(B),Type,hp(HP_B),attack(Attack_B,Damage_B)),
write(pokemon(name(B),Type,hp(HP_B),attack(Attack_B,Damage_B))).
```

2. Demo

```
?- consult('pokemon.pro').
true.

?- display_names.
pikachu
raichu
bulbasaur
ivysaur
venusaur
caterpie
metapod
butterfree
charmander
charmeleon
charizard
vulpix
ninetails
poliwag
poliwhirl
poliwrath
squirtle
wartortle
blastoise
staryu
starmie
true.
```

```
?- display_attacks.
gnaw
thunder-shock
leech-seed
vine-whip
poison-powder
gnaw
stun-spore
whirlwind
scratch
slash
royal-blaze
confuse-ray
fire-blast
water-gun
amnesia
dashing-punch
bubble
waterfall
hydro-pump
slap
star-freeze
true.

?- powerful(pikachu).
false.

?- powerful(blastoise).
true.

?- powerful(X), write(X), nl, fail.
raichu
venusaur
butterfree
charizard
ninetails
wartortle
blastoise
false.

?- tough(raichu).
false.
```

```
?- tough(venusaur).
true.

?- tough(Name), write(Name), nl, fail.
venusaur
butterfree
charizard
ninetails
poliwrath
blastoise
false.

?- type(caterpie,grass).
true .

?- type(pikachu,water).
false.

?- type(N,electric).
N = pikachu ;
N = raichu.

?- type(N,water), write(N), nl, fail.
poliwag
poliwhirl
poliwrath
squirtle
wartortle
blastoise
staryu
starmie
false.
```

```
?- dump_kind(water).
pokemon(name(poliwag), water, hp(60), attack(water-gun, 30)).
pokemon(name(poliwhirl), water, hp(80), attack(amnesia, 30)).
pokemon(name(poliwrath), water, hp(140), attack(dashing-punch, 5(
pokemon(name(squirtle), water, hp(40), attack(bubble, 10)).
pokemon(name(wartortle), water, hp(80), attack(waterfall, 60)).
pokemon(name(blastoise), water, hp(140), attack(hydro-pump, 60)).
pokemon(name(staryu), water, hp(40), attack(slap, 20)).
pokemon(name(starmie), water, hp(60), attack(star-freeze, 20)).


false.

?- dump_kind(fire).
pokemon(name(charmander), fire, hp(50), attack(scratch, 10)).
pokemon(name(charmeleon), fire, hp(80), attack(slash, 50)).
pokemon(name(charizard), fire, hp(170), attack(royal-blaze, 100)).
pokemon(name(vulpix), fire, hp(60), attack(confuse-ray, 20)).
pokemon(name(ninetails), fire, hp(100), attack(fire-blast, 120)).


false.

?- display_cen.
pikachu
bulbasaur
caterpie
charmander
vulpix
poliwag
squirtle
staryu
true.
```

```
?-
```

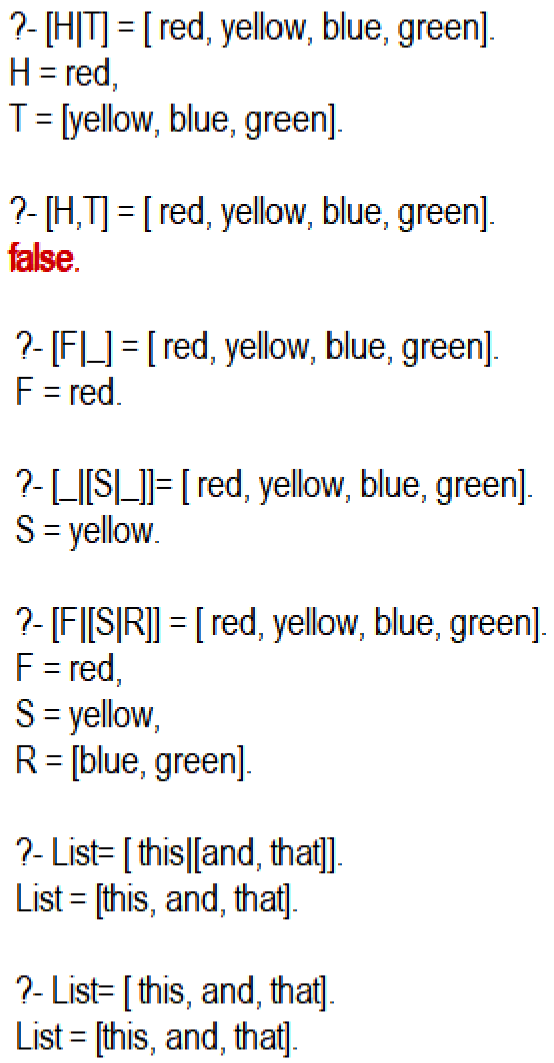
```
?- family(pikachu).
pikachu raichu
false.

?- family(squirtle).
squirtle wartortle blastoise
true.

?- families.

pikachu raichu
bulbasaur ivysaur venusaur
caterpie metapod butterfree
charmander charmeleon charizard
vulpix ninetails
poliwag poliwhirl poliwrath
squirtle wartortle blastoise
staryu starmie
false.


?- lineage(caterpie).
pokemon(name(caterpie),grass,hp(50),attack(gnaw,20))
pokemon(name(metapod),grass,hp(70),attack(stun-spore,20))
pokemon(name(butterfree),grass,hp(130),attack(whirlwind,80))
true .

?- lineage(metapod).
pokemon(name(metapod),grass,hp(70),attack(stun-spore,20))
pokemon(name(butterfree),grass,hp(130),attack(whirlwind,80))
false.

?- lineage(butterfree).
pokemon(name(butterfree),grass,hp(130),attack(whirlwind,80))
false.
```

Task 4: Lisp Processing in Prolog

Presentational Notes for Task 4

Place the following items within the “Task 4: List Processing in Prolog” section of your presentation document:

1. Your demo corresponding to the “Head/Tale Referencing Exercises”.



```
?- [H|T] = [ red, yellow, blue, green].
H = red,
T = [yellow, blue, green].

?- [H,T] = [ red, yellow, blue, green].
false.

?- [F|_] = [ red, yellow, blue, green].
F = red.

?- [_|[S|_]]= [ red, yellow, blue, green].
S = yellow.

?- [F|[S|R]] = [ red, yellow, blue, green].
F = red,
S = yellow,
R = [blue, green].

?- List= [ this|[and, that]].
List = [this, and, that].

?- List= [ this, and, that].
List = [this, and, that].
```

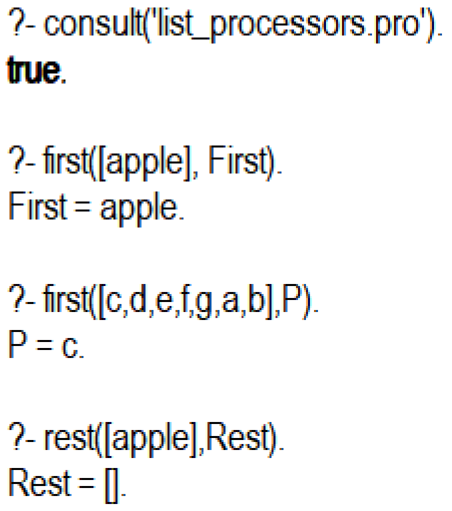
```
?- [a,[b,c]] =[a,b,c].
false.

?- [a|[b,c]] =[a,b,c].
true.

?- [cell(Row, Column)|Rest] = [cell(1,1), cell(3,2), cell(1,3)].
Row = Column, Column = 1,
Rest = [cell(3, 2), cell(1, 3)].

?- [X|Y] = [one(un, uno), two(dos, deux), three(trois, tres)].
X = one(un, uno),
Y = [two(dos, deux), three(trois, tres)].
```

2. KB:list_processors.pro.

```
% ---------------------------------------------------------------------------------
% ---------------------------------------------------------------------------------
% --- File: list_processor.pro
% ---------------------------------------------------------------------------------
first([H|_], H).

rest([_|T], T).

last([H|[]], H).
last([_|T], Result) :- last(T, Result).

nth(0,[H|_],H).
nth(N,[_|T],E) :- K is N - 1, nth(K,T,E).

writelist([]).
writelist([H|T]) :- write(H), nl, writelist(T).

sum([],0).
sum([Head|Tail],Sum) :-
sum(Tail,SumOfTail),
Sum is Head + SumOfTail.

add_first(X,L,[X|L]).

add_last(X,[],[X]).
add_last(X,[H|T],[H|TX]) :- add_last(X,T,TX).

iota(0,[]).
iota(N,IotaN) :-
K is N - 1,
iota(K,IotaK),
add_last(N,IotaK,IotaN).
```

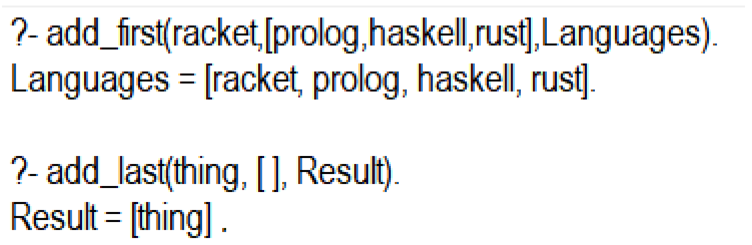
```
pick(L,Item) :-
length(L,Length),
random(0,Length,RN),
nth(RN,L,Item).

make_set([],[]).
make_set([H|T],TS) :-
member(H,T),
make_set(T,TS).
make_set([H|T],[H|TS]) :-
make_set(T,TS).
```

3. Demo

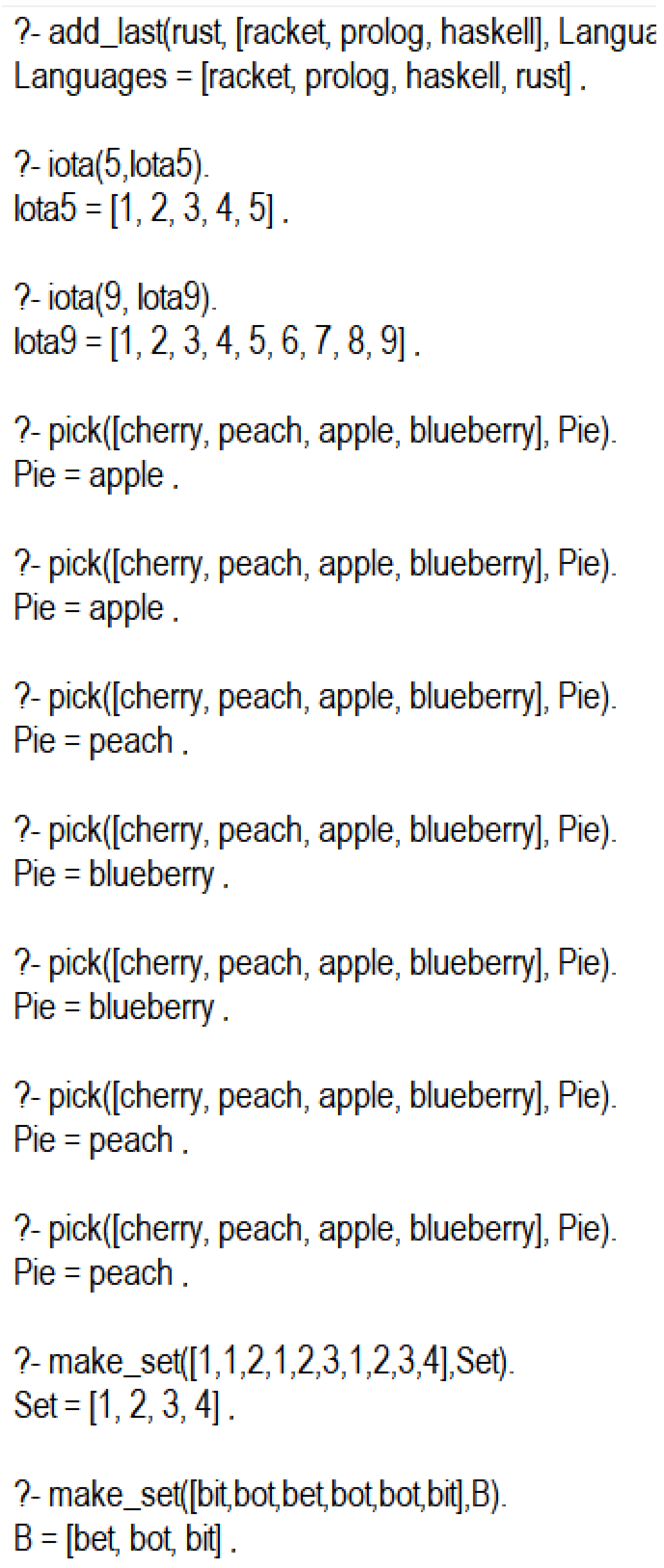


```
?- consult('list_processors.pro').
true.

?- first([apple], First).
First = apple.

?- first([c,d,e,f,g,a,b],P).
P = c.

?- rest([apple],Rest).
Rest = [].
```

```
?- rest( [c, d, e, f ,g, a, b], P).
P = [d, e, f, g, a, b].

?- rest( [c, d, e, f ,g, a, b], Rest).
Rest = [d, e, f, g, a, b].

?- last([peach], Last).
Last = peach .

?- last([c,d,e,f,g,a,b],P).
P = b .

?- nth(0,[zero,one,two,three,four],Element).
Element = zero .

?- nth(0,[four,three,two,one,zero],Element).
Element = four .

?- nth(3,[four,three,two,one,zero],Element).
Element = one .

?- writelist([red,yellow,blue,green,purple,orange]).
red
yellow
blue
green
purple
orange
true.

?- sum([ ],Sum).
Sum = 0.

?- sum([2, 3, 5, 7, 11], SumOfPrimes).
SumOfPrimes = 28.

?- add_first(thing, [ ], Result).
Result = [thing].
```



```
?- add_first(racket,[prolog,haskell,rust],Languages).
Languages = [racket, prolog, haskell, rust].

?- add_last(thing, [ ], Result).
Result = [thing] .
```



```
?- add_last(rust, [racket, prolog, haskell], Langua
Languages = [racket, prolog, haskell, rust] .

?- iota(5,Iota5).
Iota5 = [1, 2, 3, 4, 5] .

?- iota(9, Iota9).
Iota9 = [1, 2, 3, 4, 5, 6, 7, 8, 9] .

?- pick([cherry, peach, apple, blueberry], Pie).
Pie = apple .

?- pick([cherry, peach, apple, blueberry], Pie).
Pie = apple .

?- pick([cherry, peach, apple, blueberry], Pie).
Pie = peach .

?- pick([cherry, peach, apple, blueberry], Pie).
Pie = blueberry .

?- pick([cherry, peach, apple, blueberry], Pie).
Pie = blueberry .

?- pick([cherry, peach, apple, blueberry], Pie).
Pie = peach .

?- pick([cherry, peach, apple, blueberry], Pie).
Pie = peach .

?- make_set([1,1,2,1,2,3,1,2,3,4],Set).
Set = [1, 2, 3, 4] .

?- make_set([bit,bot,bet,bot,bot,bit],B).
B = [bet, bot, bit] .
```

4. KB associated with the “Example List Processors” that is provided in Lesson 5.

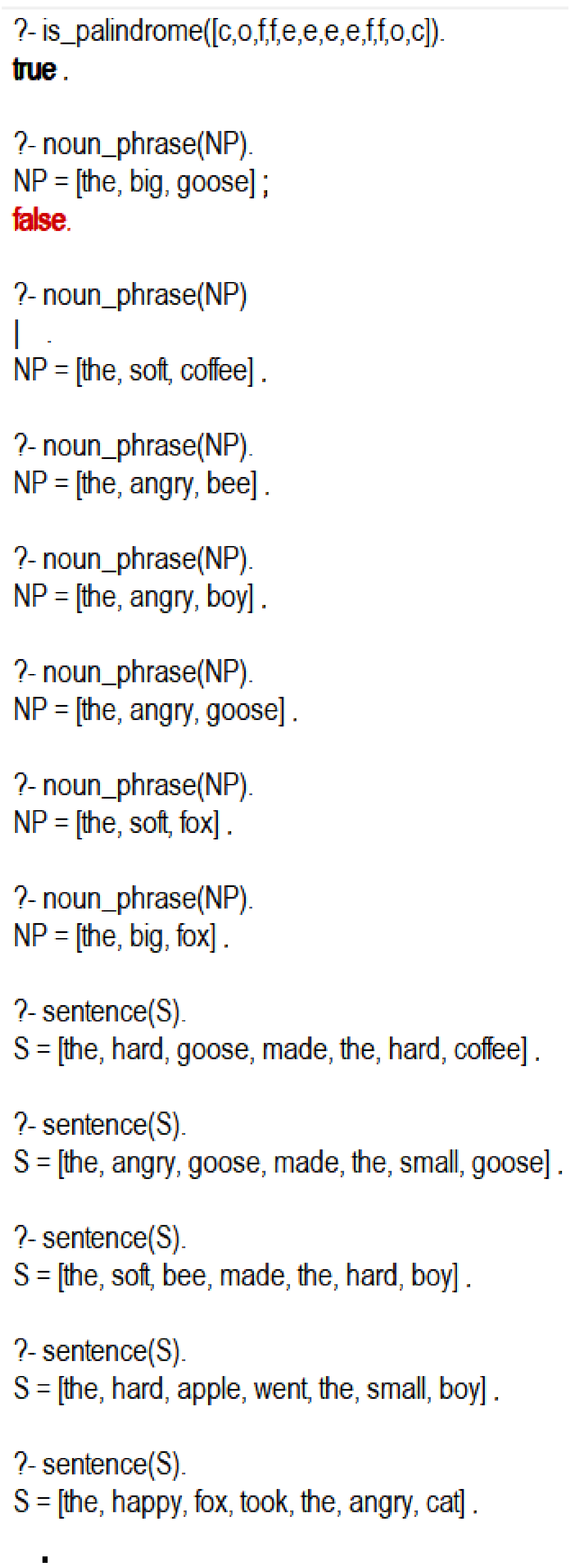
```
product([],1).
product([H|T],Product):-
product(T,P),
Product is H * P.

factorial(0,0).
factorial(N,Name):-
iota(N,I), product(I,Product),
Name is Product.

make_list(0,_,[]).
make_list(T,E,N):-
K is T -1,
make_list(K,E,Nk),add_last(E,Nk,N).

but_first([],[]).
but_first([_],[]).
but_first([_|T],T).

but_last([],[]).
but_last([_],[]).
but_last([H|T], N):-
reverse(T, [_|B]), reverse(B,RDC), add_first(H,RDC,N).

is_palindrome([]).
is_palindrome([_]).
is_palindrome(L):-
first(L,FE), last(L,LE),
FE = LE,
but_first(L,A), but_last(A,B), is_palindrome(B).

noun_phrase(N):-
pick([big,small,soft,hard,angry,happy],Adjective),
pick([apple,bee,cat,dog,fox,goose,boy,coffee],Noun),
add_last(Adjective,[the],Theadj), add_last(Noun,Theadj,N).

sentence(N):-
pick([saw,had,did,went,said,took,made],Verb),
noun_phrase(A), noun_phrase(B),
add_last(Verb,A,X),append(X,B,N).
```

5. The demo that you are asked to create in the “List Processing Exercises” section of Lesson 5.

```
?- consult('list_processors.pro').
true.

?- product([ ],P).
P = 1.

?- product([1,3,5,7,9],Product).
Product = 945.

?- consult('list_processors.pro').
true.

?- product([ ],P).
P = 1.

?- product([1,3,5,7,9],Product).
Product = 945.

?- factorial(9,Product).
Product = 362880 .

?- iota(9,Iota),product(Iota,Product).
Iota = [1, 2, 3, 4, 5, 6, 7, 8, 9],
Product = 362880 .

?- make_list(7, seven, Seven).
Seven = [seven, seven, seven, seven, seven, seven, seven] .

?- make_list(8,2,List).
List = [2, 2, 2, 2, 2, 2, 2, 2] .

?- but_first([a,b,c],X).
X = [b, c].
```

```
?- but_last([a,b,c,d,e],X).
X = [a, b, c, d].

?- is_palindrome([x]).
true
```

```
?- is_palindrome([a,b,c]).
false.

?- is_palindrome([a,b,b,a]).
true .

?- is_palindrome([1,2,3,4,5,4,2,3,1]).
false.
```



```
?- is_palindrome([c,o,f,f,e,e,e,e,f,f,o,c]).
true .

?- noun_phrase(NP).
NP = [the, big, goose] ;
false.

?- noun_phrase(NP)
|    .
NP = [the, soft, coffee] .

?- noun_phrase(NP).
NP = [the, angry, bee] .

?- noun_phrase(NP).
NP = [the, angry, boy] .

?- noun_phrase(NP).
NP = [the, angry, goose] .

?- noun_phrase(NP).
NP = [the, soft, fox] .

?- noun_phrase(NP).
NP = [the, big, fox] .

?- sentence(S).
S = [the, hard, goose, made, the, hard, coffee] .

?- sentence(S).
S = [the, angry, goose, made, the, small, goose] .

?- sentence(S).
S = [the, soft, bee, made, the, hard, boy] .

?- sentence(S).
S = [the, hard, apple, went, the, small, boy] .

?- sentence(S).
S = [the, happy, fox, took, the, angry, cat] .
```