Temitope Emokpae 11/4/2022

CSC 344

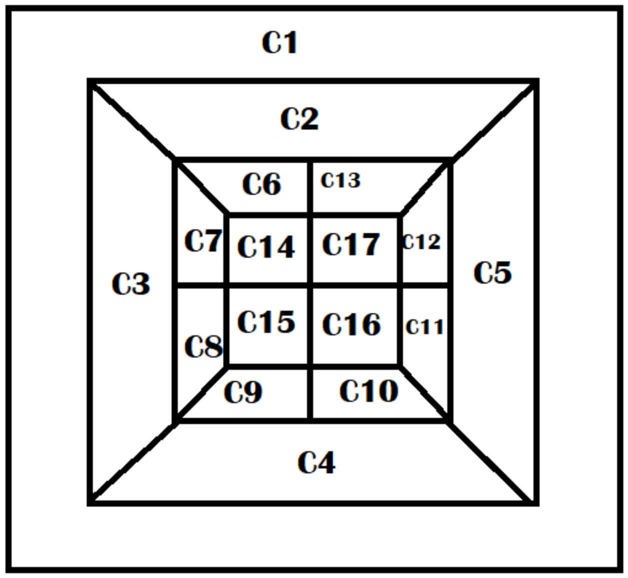
# Prolog Programming Assignment #1: Various Computations

## Learning Abstract

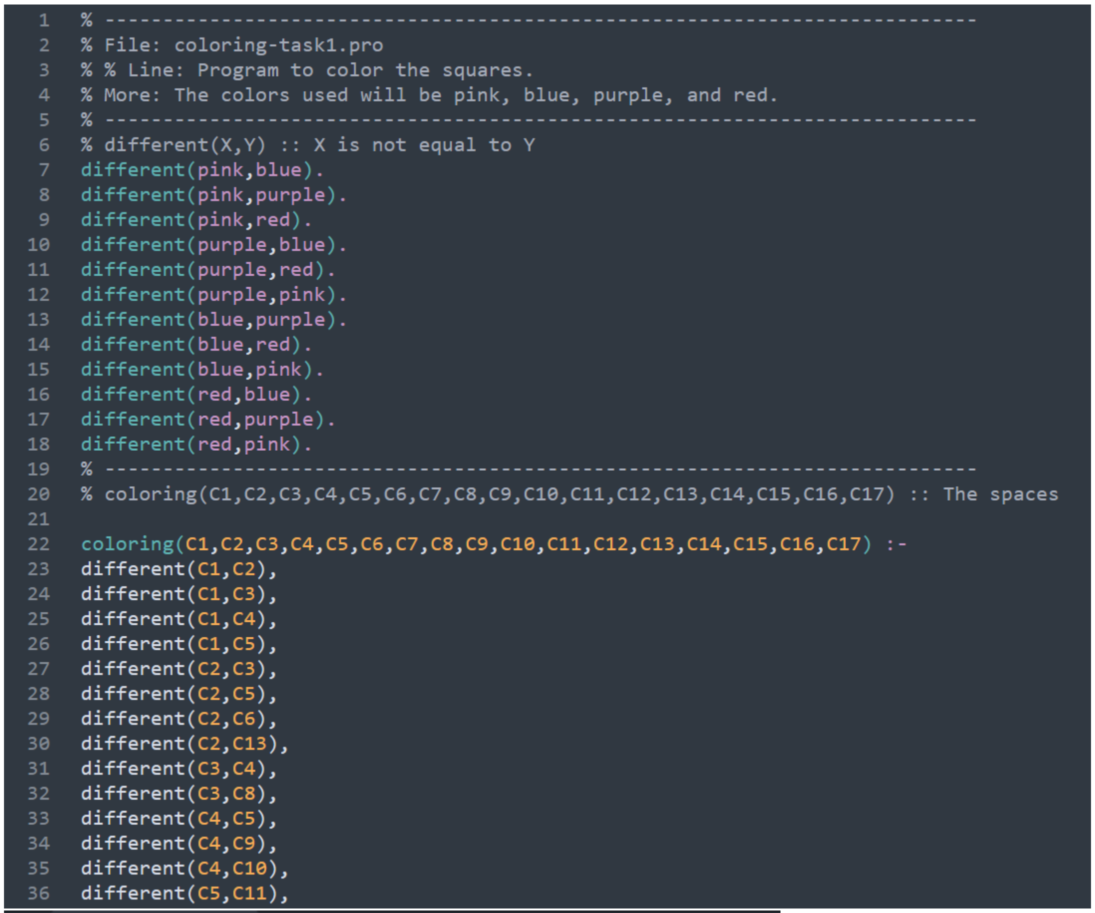
This programming assignment is about computational functions in Prolog. It involves the use of a knowledge base for coloring and the floating shape world. Then, the Pokémon KB Interaction and Programming & Lisp Processing in Prolog.

## Problem 1- Map Coloring

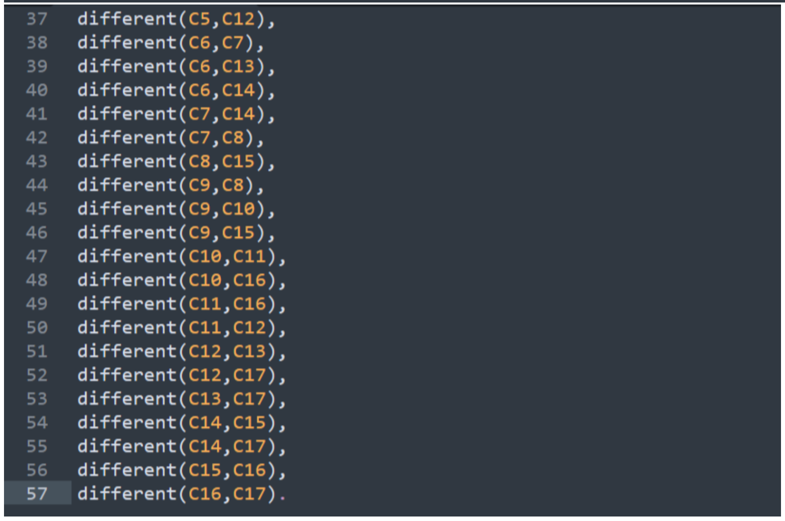
### Image:



### Code:

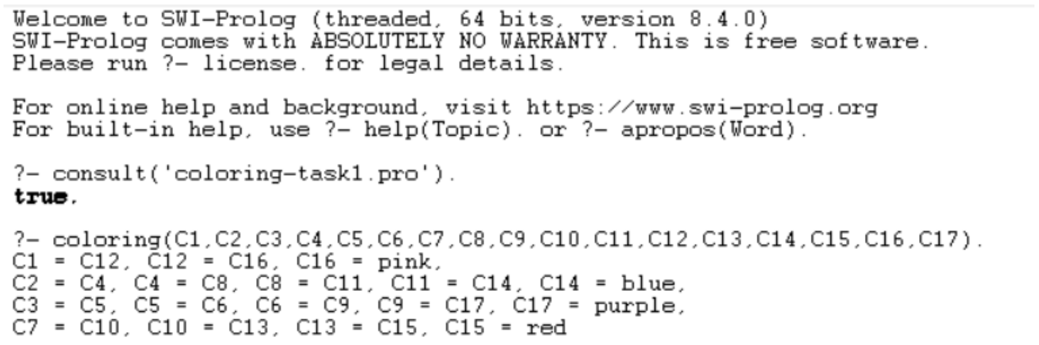


```prolog
% -------------------------------------------------------------------------
% File: coloring-task1.pro
% % Line: Program to color the squares.
% More: The colors used will be pink, blue, purple, and red.
% -------------------------------------------------------------------------
% different(X,Y) :: X is not equal to Y
different(pink,blue).
different(pink,purple).
different(pink,red).
different(purple,blue).
different(purple,red).
different(purple,pink).
different(blue,purple).
different(blue,red).
different(blue,pink).
different(red,blue).
different(red,purple).
different(red,pink).
% -------------------------------------------------------------------------
% coloring(C1,C2,C3,C4,C5,C6,C7,C8,C9,C10,C11,C12,C13,C14,C15,C16,C17) :: The spaces

coloring(C1,C2,C3,C4,C5,C6,C7,C8,C9,C10,C11,C12,C13,C14,C15,C16,C17) :-
different(C1,C2),
different(C1,C3),
different(C1,C4),
different(C1,C5),
different(C2,C3),
different(C2,C5),
different(C2,C6),
different(C2,C13),
different(C3,C4),
different(C3,C8),
different(C4,C5),
different(C4,C9),
different(C4,C10),
different(C5,C11),
```



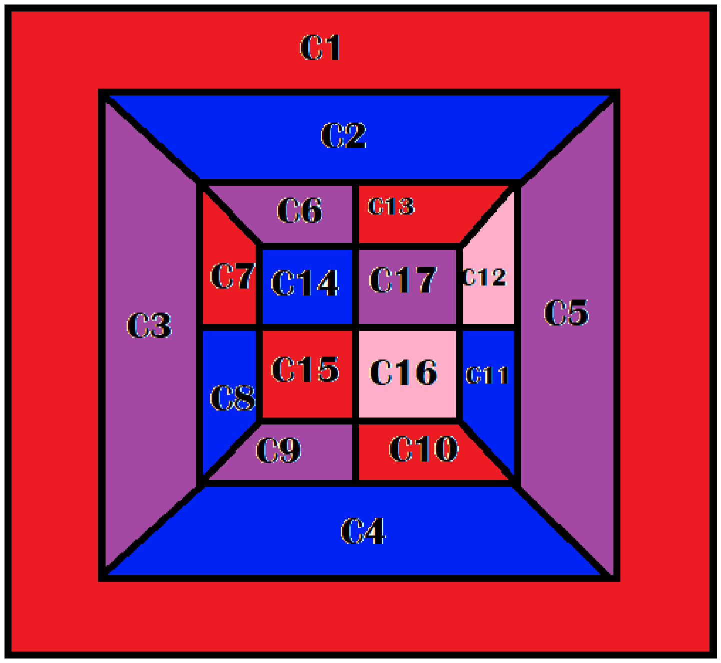
```prolog
different(C5,C12),
different(C6,C7),
different(C6,C13),
different(C6,C14),
different(C7,C14),
different(C7,C8),
different(C8,C15),
different(C9,C8),
different(C9,C10),
different(C9,C15),
different(C10,C11),
different(C10,C16),
different(C11,C16),
different(C11,C12),
different(C12,C13),
different(C12,C17),
different(C13,C17),
different(C14,C15),
different(C14,C17),
different(C15,C16),
different(C16,C17).
```

## Demo:



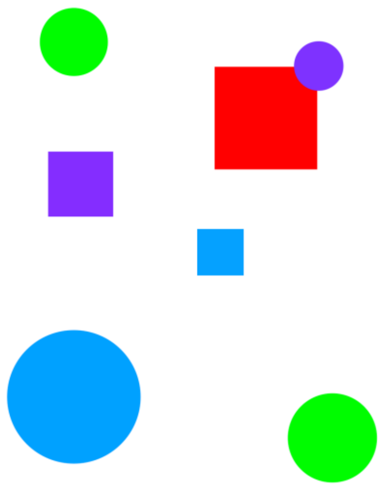
```
Welcome to SWI-Prolog (threaded, 64 bits, version 8.4.0)
SWI-Prolog comes with ABSOLUTELY NO WARRANTY. This is free software.
Please run ?- license. for legal details.

For online help and background, visit https://www.swi-prolog.org
For built-in help, use ?- help(Topic). or ?- apropos(Word).

?- consult('coloring-task1.pro').
true.

?- coloring(C1,C2,C3,C4,C5,C6,C7,C8,C9,C10,C11,C12,C13,C14,C15,C16,C17).
C1 = C12, C12 = C16, C16 = pink,
C2 = C4, C4 = C8, C8 = C11, C11 = C14, C14 = blue,
C3 = C5, C5 = C6, C6 = C9, C9 = C17, C17 = purple,
C7 = C10, C10 = C13, C13 = C15, C15 = red
```

## Image Colored:

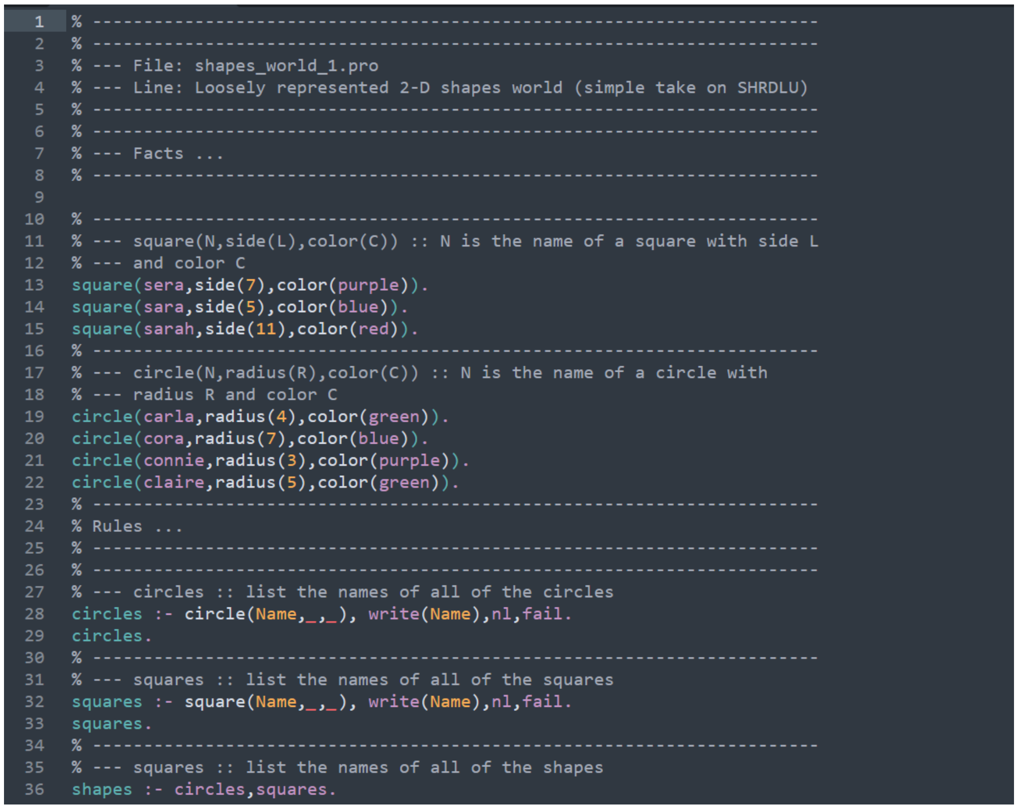


## Problem 2 - The Floating Shapes World

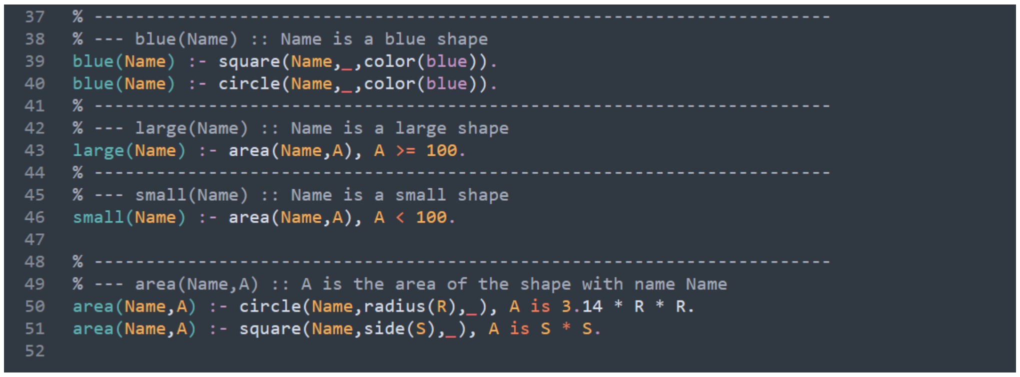
## Image:



## Demo:

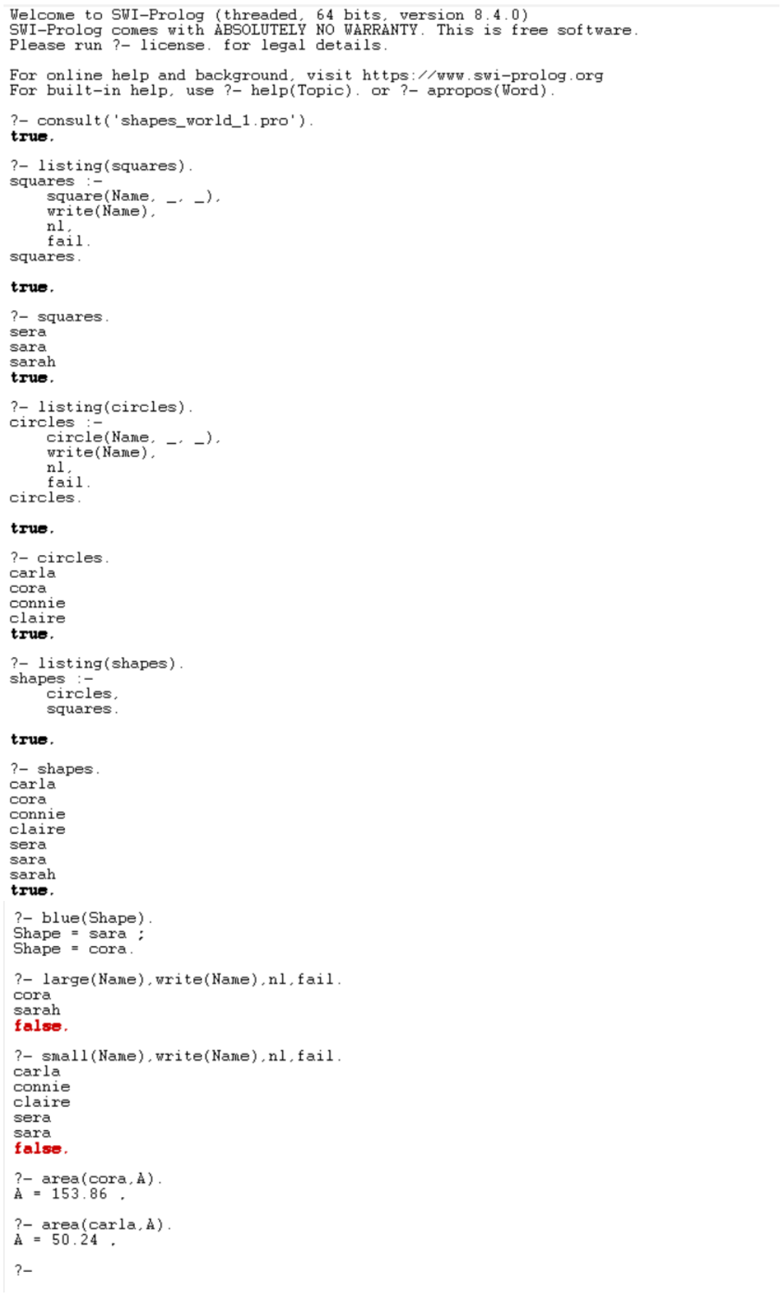


```
% ----------------------------------------------------------------------
% ----------------------------------------------------------------------
% --- File: shapes_world_1.pro
% --- Line: Loosely represented 2-D shapes world (simple take on SHRDLU)
% ----------------------------------------------------------------------
% ----------------------------------------------------------------------
% --- 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.

```

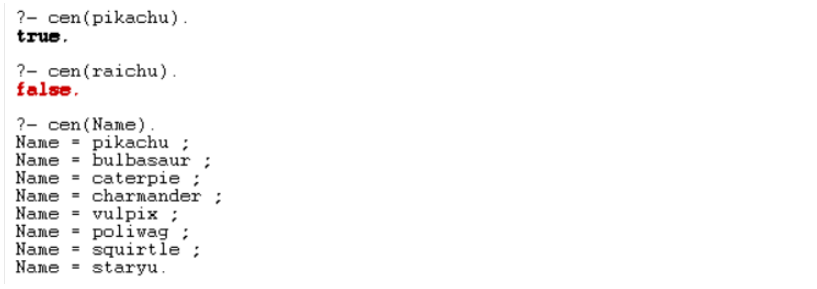
## Demo:



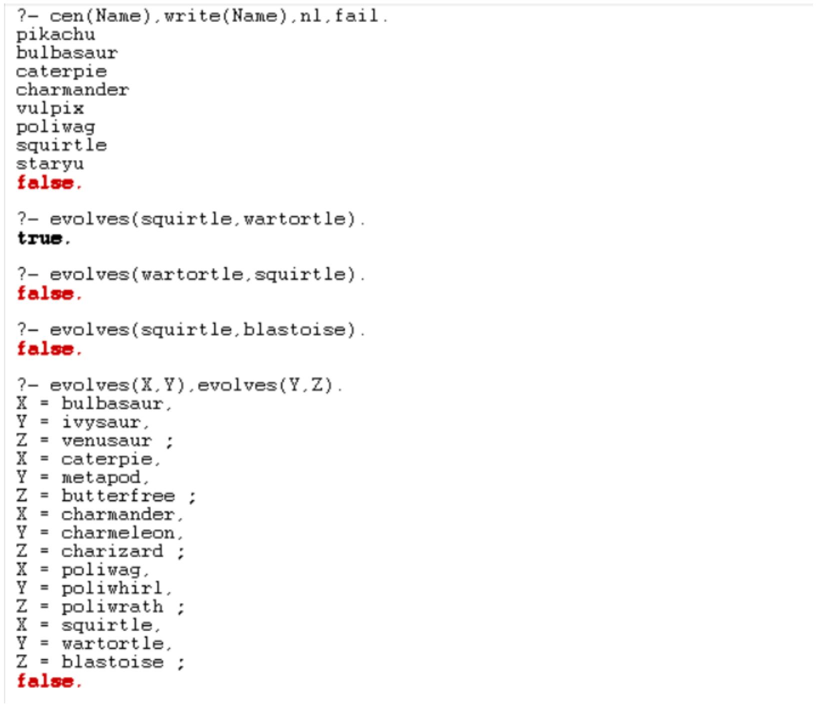
```
Welcome to SWI-Prolog (threaded, 64 bits, version 8.4.0)
SWI-Prolog comes with ABSOLUTELY NO WARRANTY. This is free software.
Please run ?- license. for legal details.

For online help and background, visit https://www.swi-prolog.org
For built-in help, use ?- help(Topic). or ?- apropos(Word).

?- 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 .

?-
```

## Problem 3- Pokemon KB Interaction and Programming

### Part 1: Queries

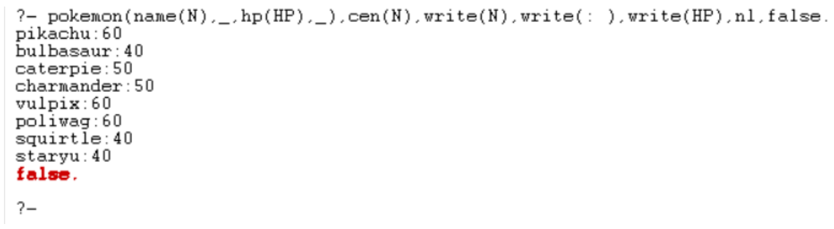


```
?- cen(pikachu).
true.

?- cen(raichu).
false.

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



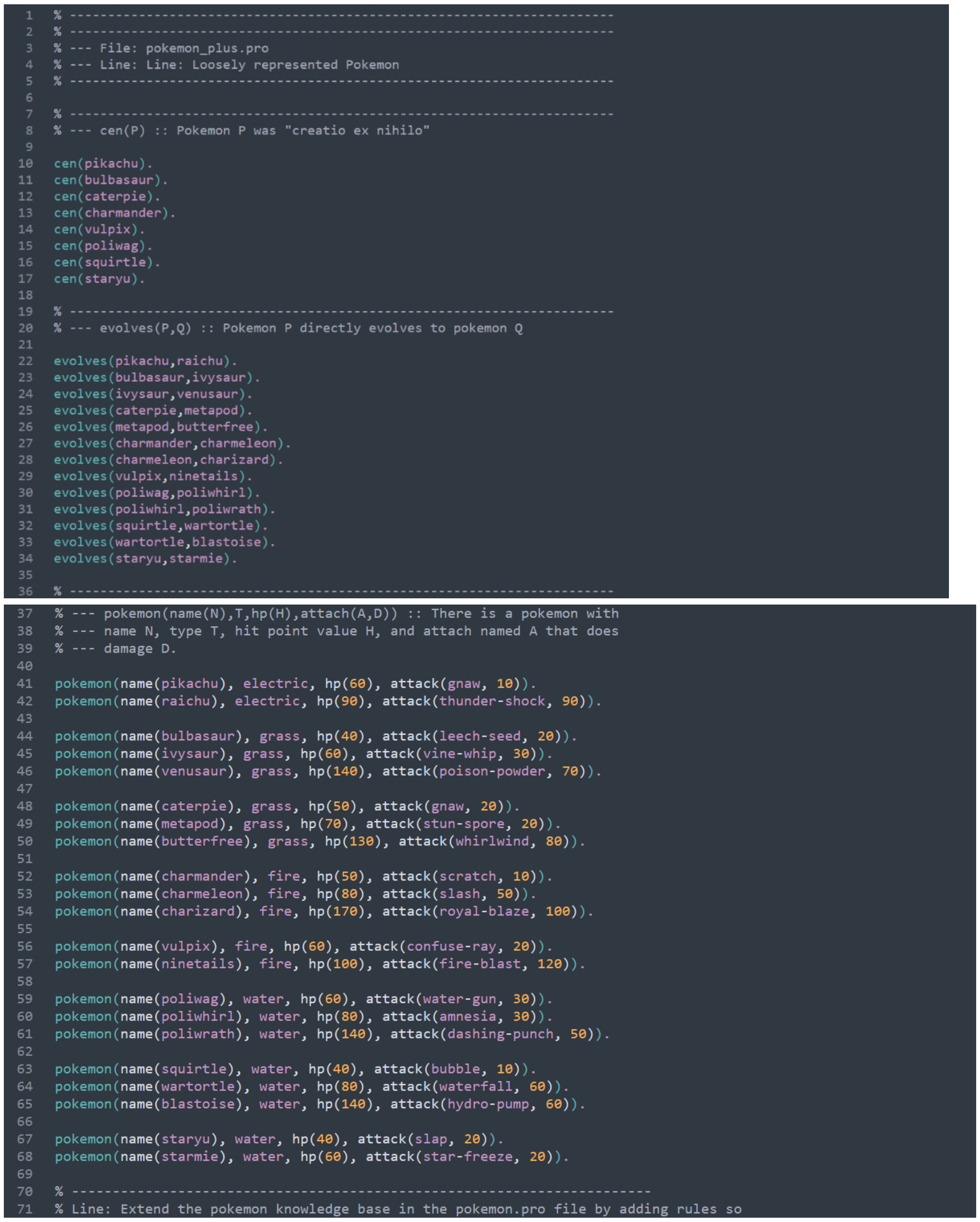
```
?- cen(Name),write(Name),nl,fail.
pikachu
bulbasaur
caterpie
charmander
vulpix
poliwag
squirtle
staryu
false.

?- evolves(squirtle,wartortle).
true.

?- evolves(wartortle,squirtle).
false.

?- evolves(squirtle,blastoise).
false.

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

```
?- evolves(X,Y), evolves(Y,Z), write(X), write( '-->'), write(Z),nl,fail.
bulbasaur-->venusaur
caterpie-->butterfree
charmander-->charizard
poliwag-->poliwrath
squirtle-->blastoise
false.

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

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

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

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

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

?- pokemon(name(poliwhirl),_,hp(HP),_).
HP = 80.

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

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

?- pokemon(name(N),_,_,attack(_,A)),A>60,write(N),nl,false.
raichu
venusaur
butterfree
charizard
ninetails
false.
```

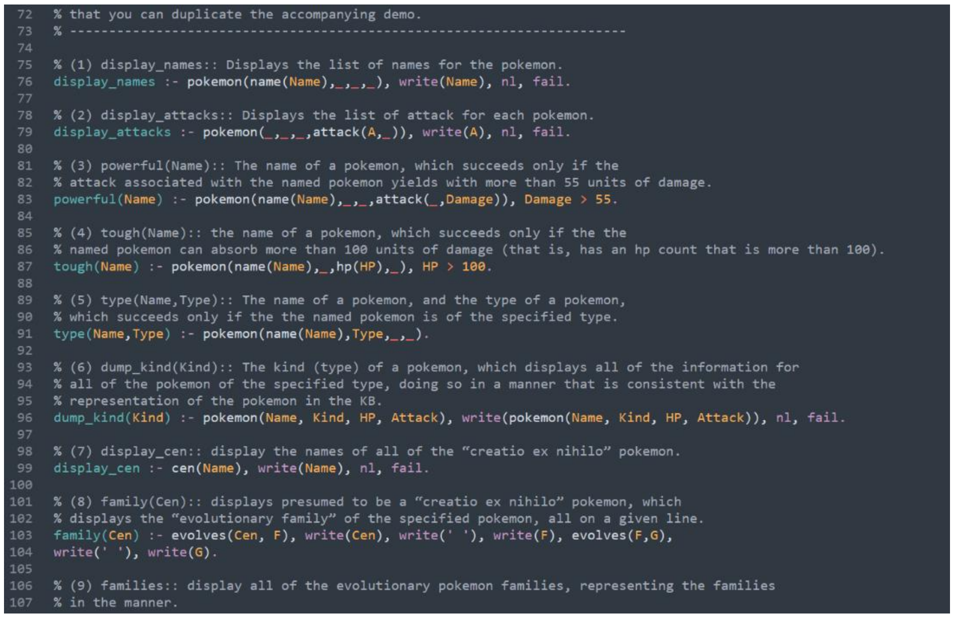


```
?- pokemon(name(N),_,hp(HP),_),cen(N),write(N),write(: ),write(HP),nl,false.
pikachu:60
bulbasaur:40
caterpie:50
charmander:50
vulpix:60
poliwag:60
squirtle:40
staryu:40
false.

?-
```

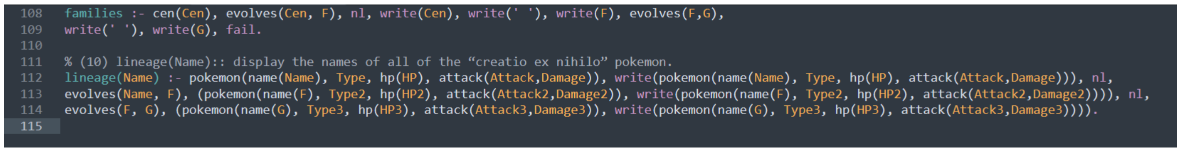
## Part 2: Programs



```
% ---------------------------------------------------------------------
% ---------------------------------------------------------------------
% --- File: pokemon_plus.pro
% --- Line: Line: Loosely represented Pokemon
% ---------------------------------------------------------------------

% ---------------------------------------------------------------------
% --- 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)).

% ---------------------------------------------------------------------
% Line: Extend the pokemon knowledge base in the pokemon.pro file by adding rules so
```



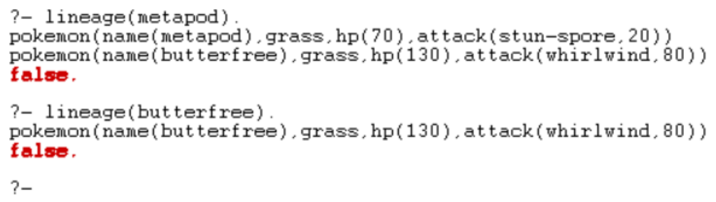
```prolog
% that you can duplicate the accompanying demo.
% ----------------------------------------------------------------------

% (1) display_names:: Displays the list of names for the pokemon.
display_names :- pokemon(name(Name),_,_,_), write(Name), nl, fail.

% (2) display_attacks:: Displays the list of attack for each pokemon.
display_attacks :- pokemon(_,_,_,attack(A,_)), write(A), nl, fail.

% (3) powerful(Name):: The name of a pokemon, which succeeds only if the
% attack associated with the named pokemon yields with more than 55 units of damage.
powerful(Name) :- pokemon(name(Name),_,_,attack(_,Damage)), Damage > 55.

% (4) tough(Name):: the name of a pokemon, which succeeds only if the the
% named pokemon can absorb more than 100 units of damage (that is, has an hp count that is more than 100).
tough(Name) :- pokemon(name(Name),_,hp(HP),_), HP > 100.

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

% (6) dump_kind(Kind):: The kind (type) of a pokemon, which displays all of the information for
% all of the pokemon of the specified type, doing so in a manner that is consistent with the
% representation of the pokemon in the KB.
dump_kind(Kind) :- pokemon(Name, Kind, HP, Attack), write(pokemon(Name, Kind, HP, Attack)), nl, fail.

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

% (8) family(Cen):: displays presumed to be a “creatio ex nihilo” pokemon, which
% displays the “evolutionary family” of the specified pokemon, all on a given line.
family(Cen) :- evolves(Cen, F), write(Cen), write(' '), write(F), evolves(F,G),
write(' '), write(G).

% (9) families:: display all of the evolutionary pokemon families, representing the families
% in the manner.
```



```prolog
families :- cen(Cen), evolves(Cen, F), nl, write(Cen), write(' '), write(F), evolves(F,G),
write(' '), write(G), fail.

% (10) lineage(Name):: display the names of all of the “creatio ex nihilo” 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, F), (pokemon(name(F), Type2, hp(HP2), attack(Attack2,Damage2)), write(pokemon(name(F), Type2, hp(HP2), attack(Attack2,Damage2)))), nl,
evolves(F, G), (pokemon(name(G), Type3, hp(HP3), attack(Attack3,Damage3)), write(pokemon(name(G), Type3, hp(HP3), attack(Attack3,Damage3)))).

```

```
?- consult('pokemon_plus.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
false.

?- 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
false.
```

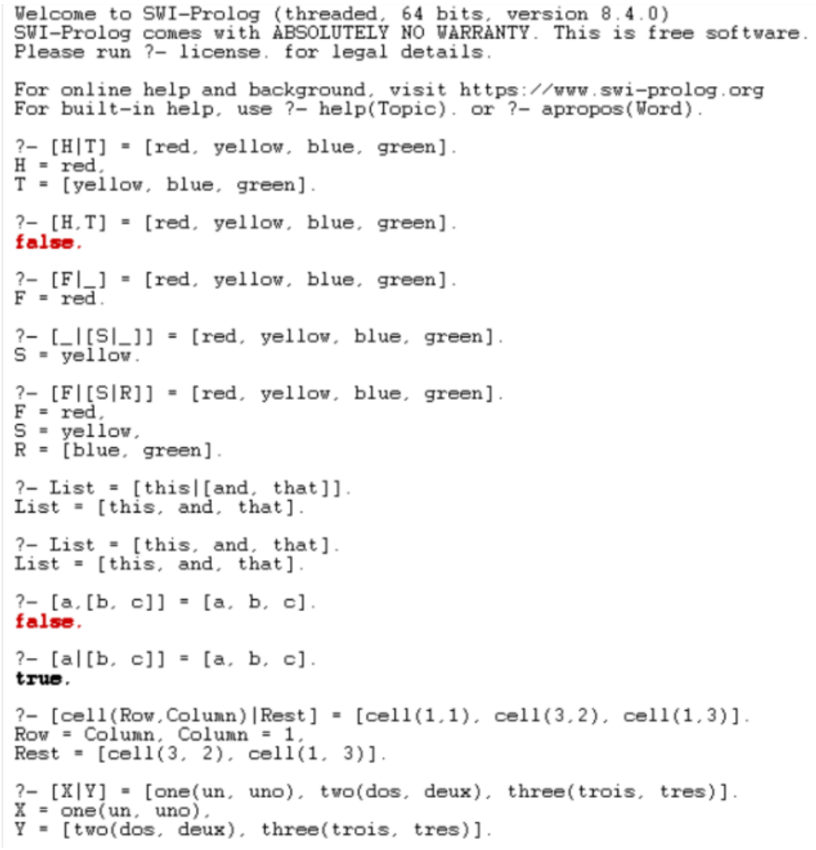
```
?- powerful(pikachu).
false.

?- 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
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,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))
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
false.

?- 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.

?-
```

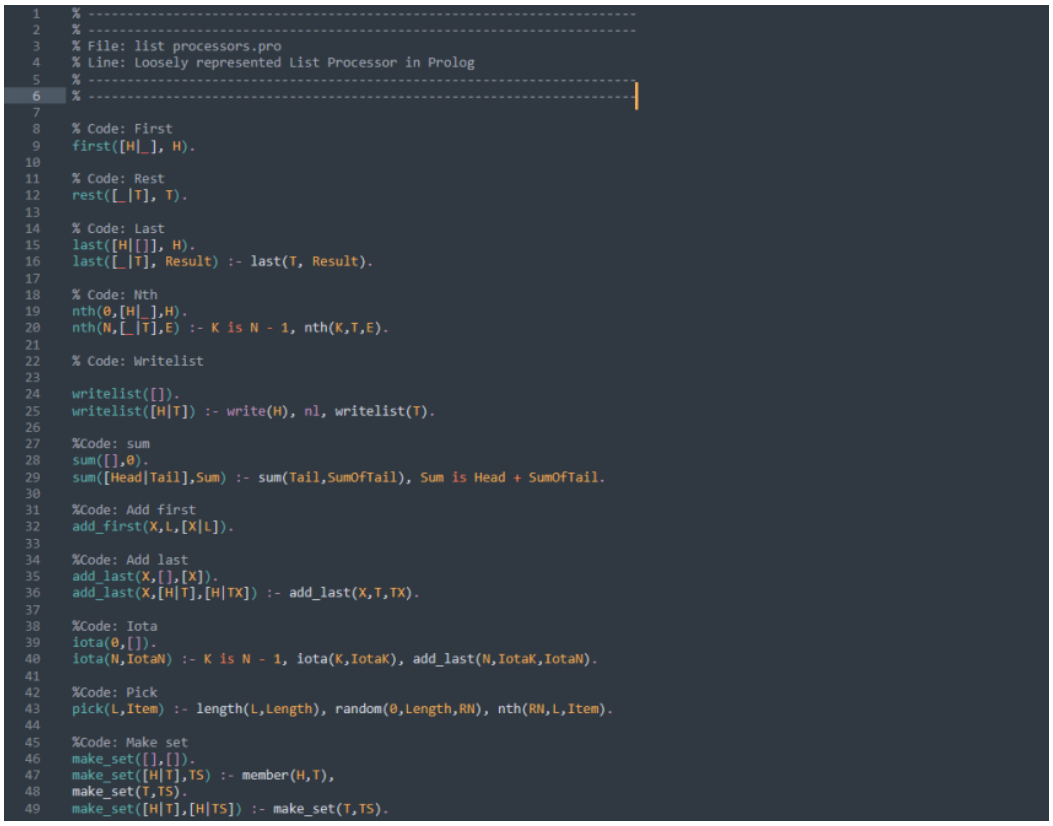
## Problem 4- Lisp Processing in Prolog

### Task 1:



```
Welcome to SWI-Prolog (threaded, 64 bits, version 8.4.0)
SWI-Prolog comes with ABSOLUTELY NO WARRANTY. This is free software.
Please run ?- license. for legal details.

For online help and background, visit https://www.swi-prolog.org
For built-in help, use ?- help(Topic). or ?- apropos(Word).

?- [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)].
```

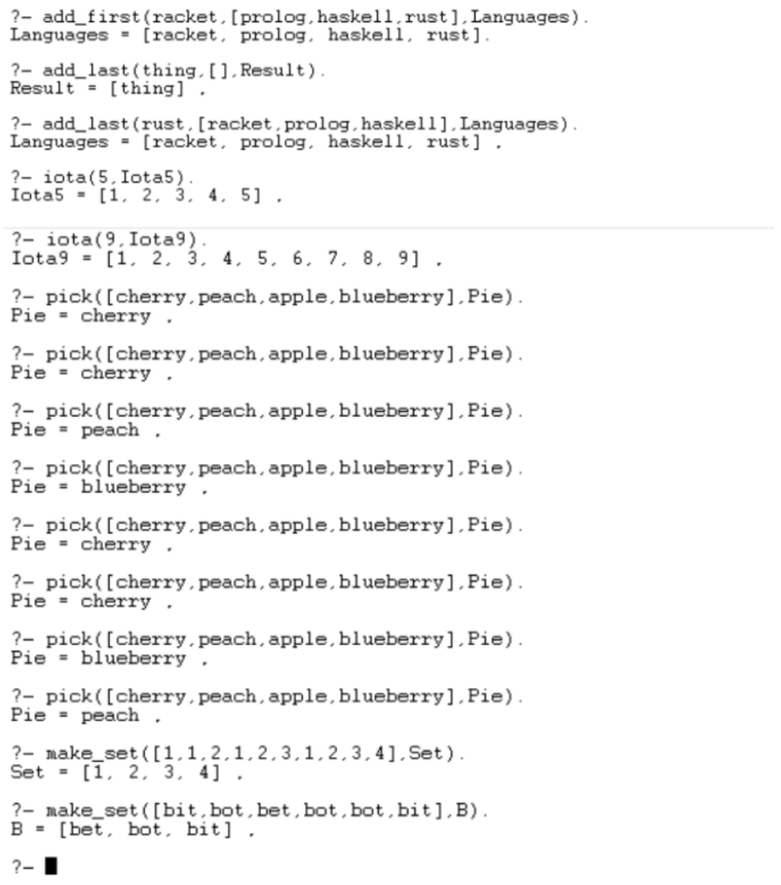
### Task 2:



```prolog
% --------------------------------------------------------------------
% --------------------------------------------------------------------
% File: list processors.pro
% Line: Loosely represented List Processor in Prolog
% --------------------------------------------------------------------
% --------------------------------------------------------------------

% Code: First
first([H|_], H).

% Code: Rest
rest([_|T], T).

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

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

% Code: Writelist

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

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

%Code: Add first
add_first(X,L,[X|L]).

%Code: Add last
add_last(X,[],[X]).
add_last(X,[H|T],[H|TX]) :- add_last(X,T,TX).

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

%Code: Pick
pick(L,Item) :- length(L,Length), random(0,Length,RN), nth(RN,L,Item).

%Code: Make set
make_set([],[]).
make_set([H|T],TS) :- member(H,T),
make_set(T,TS).
make_set([H|T],[H|TS]) :- make_set(T,TS).
```

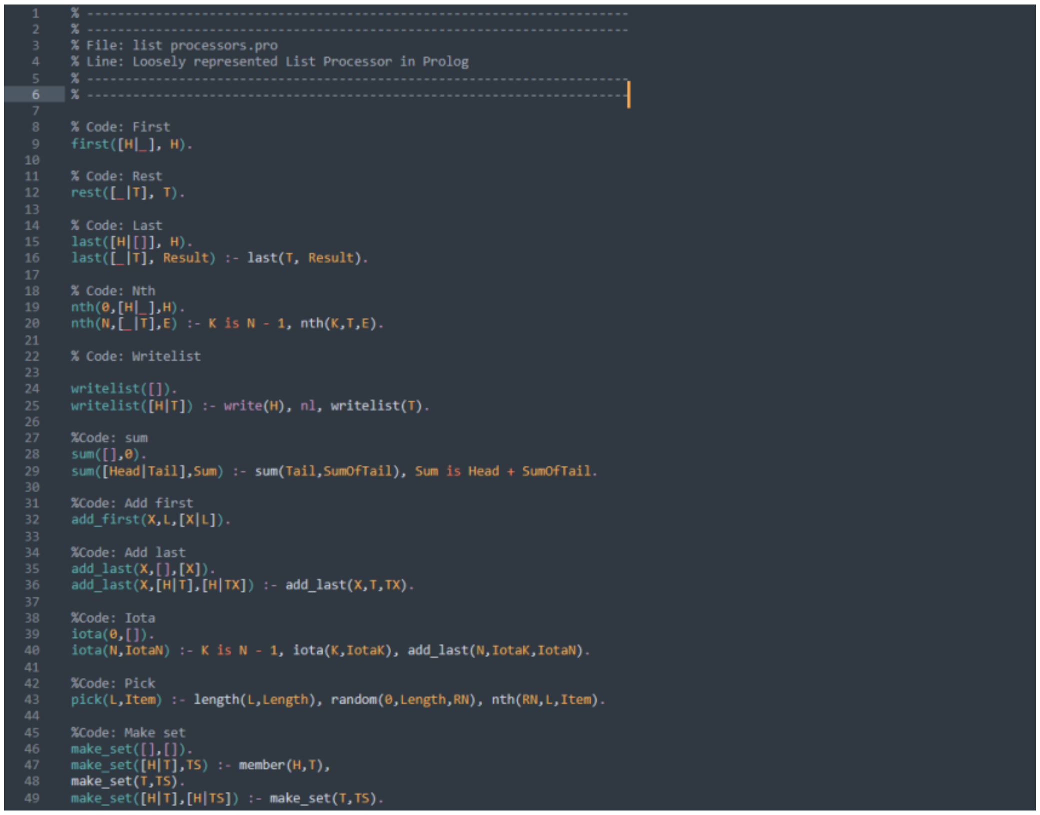
## Task 3:

```
?- 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],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(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].
```



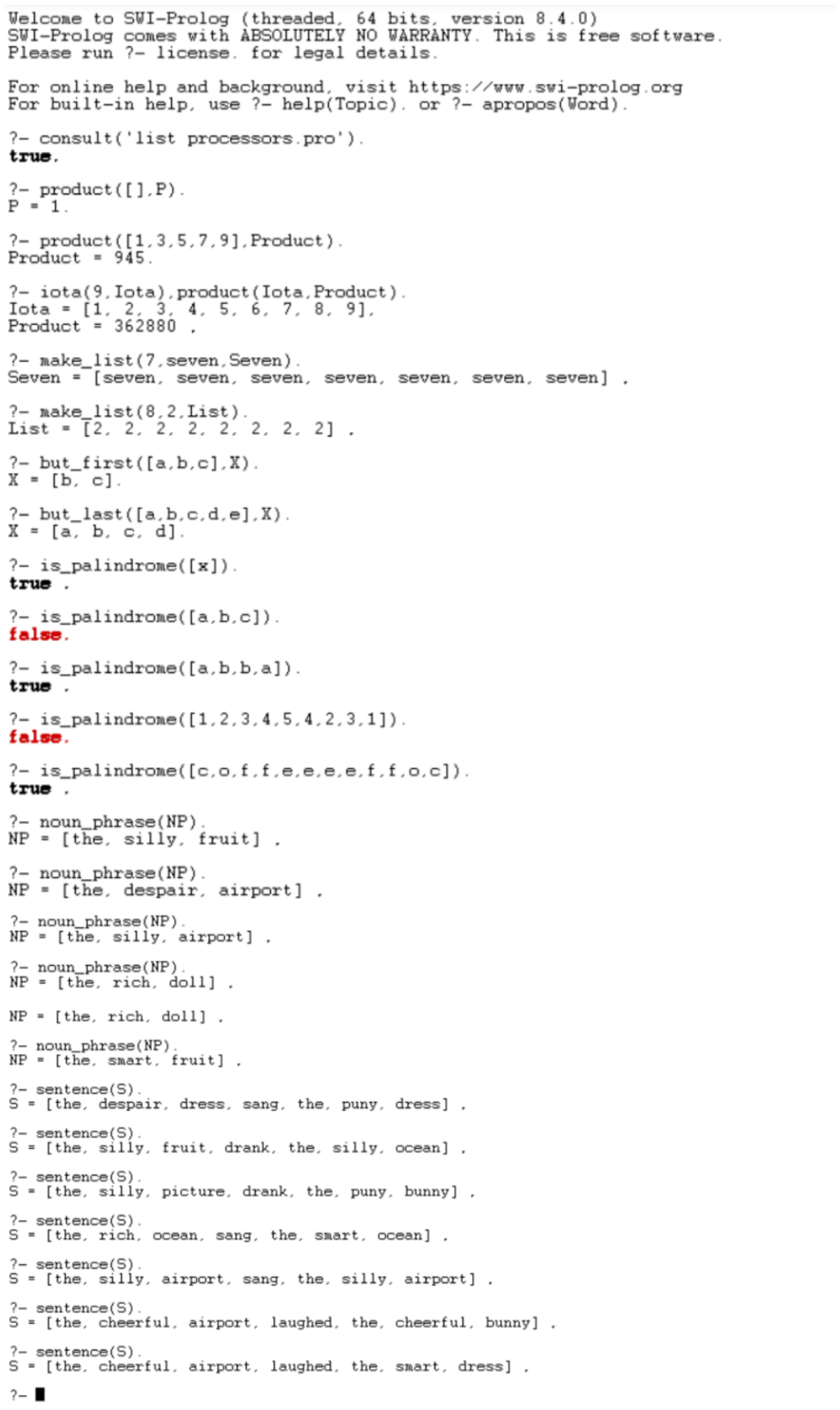
```
?- add_first(racket,[prolog,haskell,rust],Languages).
Languages = [racket, prolog, haskell, rust].

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

?- add_last(rust,[racket,prolog,haskell],Languages).
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 = cherry .

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

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

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

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

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

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

?- 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] .

?- █
```

## Task 4:



```
% ----------------------------------------------------------------------
% ----------------------------------------------------------------------
% File: list processors.pro
% Line: Loosely represented List Processor in Prolog
% ----------------------------------------------------------------------
% ----------------------------------------------------------------------

% Code: First
first([H|_], H).

% Code: Rest
rest([_|T], T).

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

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

% Code: Writelist

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

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

%Code: Add first
add_first(X,L,[X|L]).

%Code: Add last
add_last(X,[],[X]).
add_last(X,[H|T],[H|TX]) :- add_last(X,T,TX).

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

%Code: Pick
pick(L,Item) :- length(L,Length), random(0,Length,RN), nth(RN,L,Item).

%Code: Make set
make_set([],[]).
make_set([H|T],TS) :- member(H,T),
make_set(T,TS).
make_set([H|T],[H|TS]) :- make_set(T,TS).
```

## Task 5:



```
Welcome to SWI-Prolog (threaded, 64 bits, version 8.4.0)
SWI-Prolog comes with ABSOLUTELY NO WARRANTY. This is free software.
Please run ?- license. for legal details.

For online help and background, visit https://www.swi-prolog.org
For built-in help, use ?- help(Topic). or ?- apropos(Word).

?- consult('list processors.pro').
true.

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

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

?- 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, silly, fruit] .

?- noun_phrase(NP).
NP = [the, despair, airport] .

?- noun_phrase(NP).
NP = [the, silly, airport] .

?- noun_phrase(NP).
NP = [the, rich, doll] .

NP = [the, rich, doll] .

?- noun_phrase(NP).
NP = [the, smart, fruit] .

?- sentence(S).
S = [the, despair, dress, sang, the, puny, dress] .

?- sentence(S).
S = [the, silly, fruit, drank, the, silly, ocean] .

?- sentence(S).
S = [the, silly, picture, drank, the, puny, bunny] .

?- sentence(S).
S = [the, rich, ocean, sang, the, smart, ocean] .

?- sentence(S).
S = [the, silly, airport, sang, the, silly, airport] .

?- sentence(S).
S = [the, cheerful, airport, laughed, the, cheerful, bunny] .

?- sentence(S).
S = [the, cheerful, airport, laughed, the, smart, dress] .

?- █
```