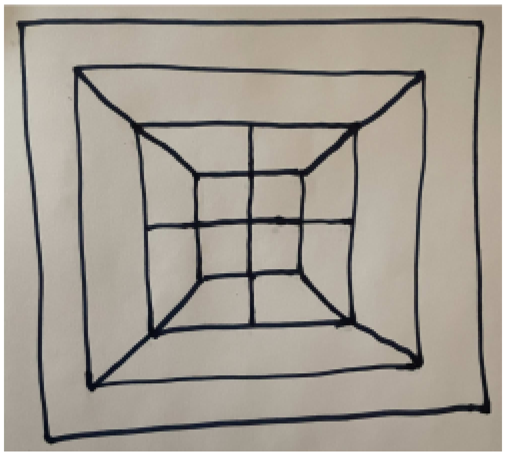
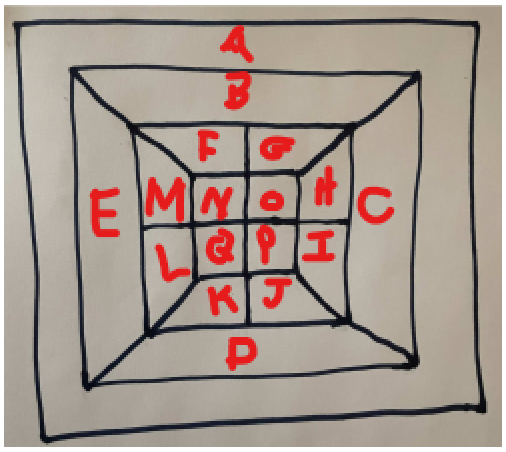
Prolog Programming Assignment #1: Various Computations

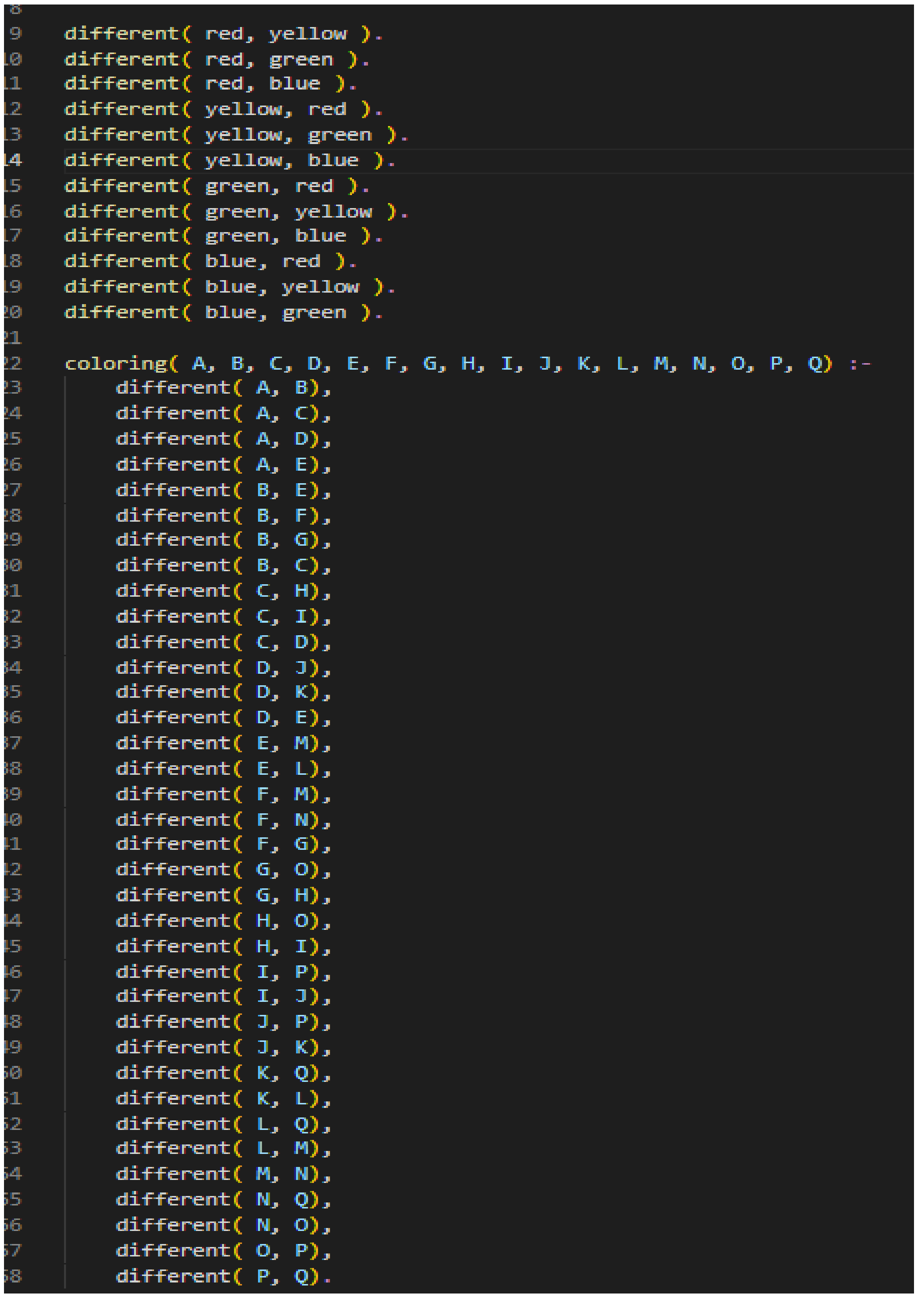
Learning Abstract:

There are several interactions with prolog to help with understanding how the language performs. Tasks include coloring maps, then some computations within Floating Worlds and Pokemon KBs, some programs written for the Pokemon KB, closing with some list processing.

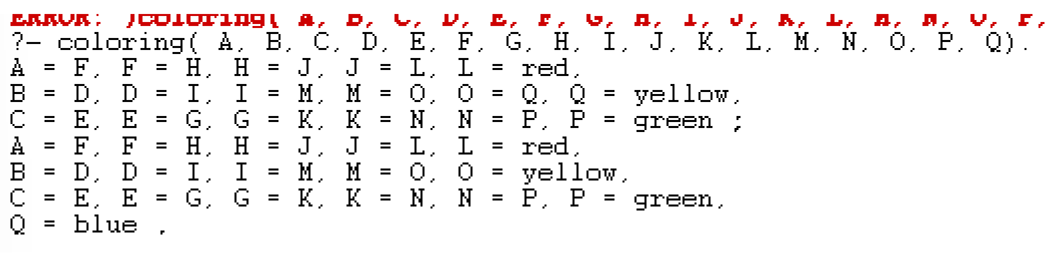
Task #1: Map Coloring



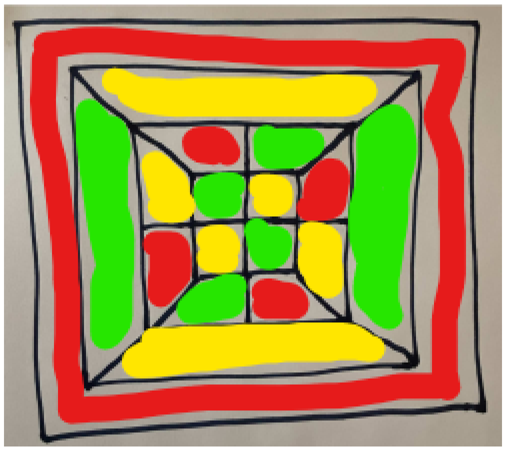




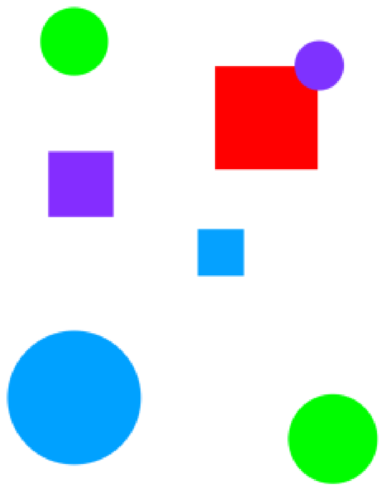
```prolog
different( red, yellow ).
different( red, green ).
different( red, blue ).
different( yellow, red ).
different( yellow, green ).
different( yellow, blue ).
different( green, red ).
different( green, yellow ).
different( green, blue ).
different( blue, red ).
different( blue, yellow ).
different( blue, green ).

coloring( A, B, C, D, E, F, G, H, I, J, K, L, M, N, O, P, Q) :-
    different( A, B),
    different( A, C),
    different( A, D),
    different( A, E),
    different( B, E),
    different( B, F),
    different( B, G),
    different( B, C),
    different( C, H),
    different( C, I),
    different( C, D),
    different( D, J),
    different( D, K),
    different( D, E),
    different( E, M),
    different( E, L),
    different( F, M),
    different( F, N),
    different( F, G),
    different( G, O),
    different( G, H),
    different( H, O),
    different( H, I),
    different( I, P),
    different( I, J),
    different( J, P),
    different( J, K),
    different( K, Q),
    different( K, L),
    different( L, Q),
    different( L, M),
    different( M, N),
    different( N, Q),
    different( N, O),
    different( O, P),
    different( P, Q).
```

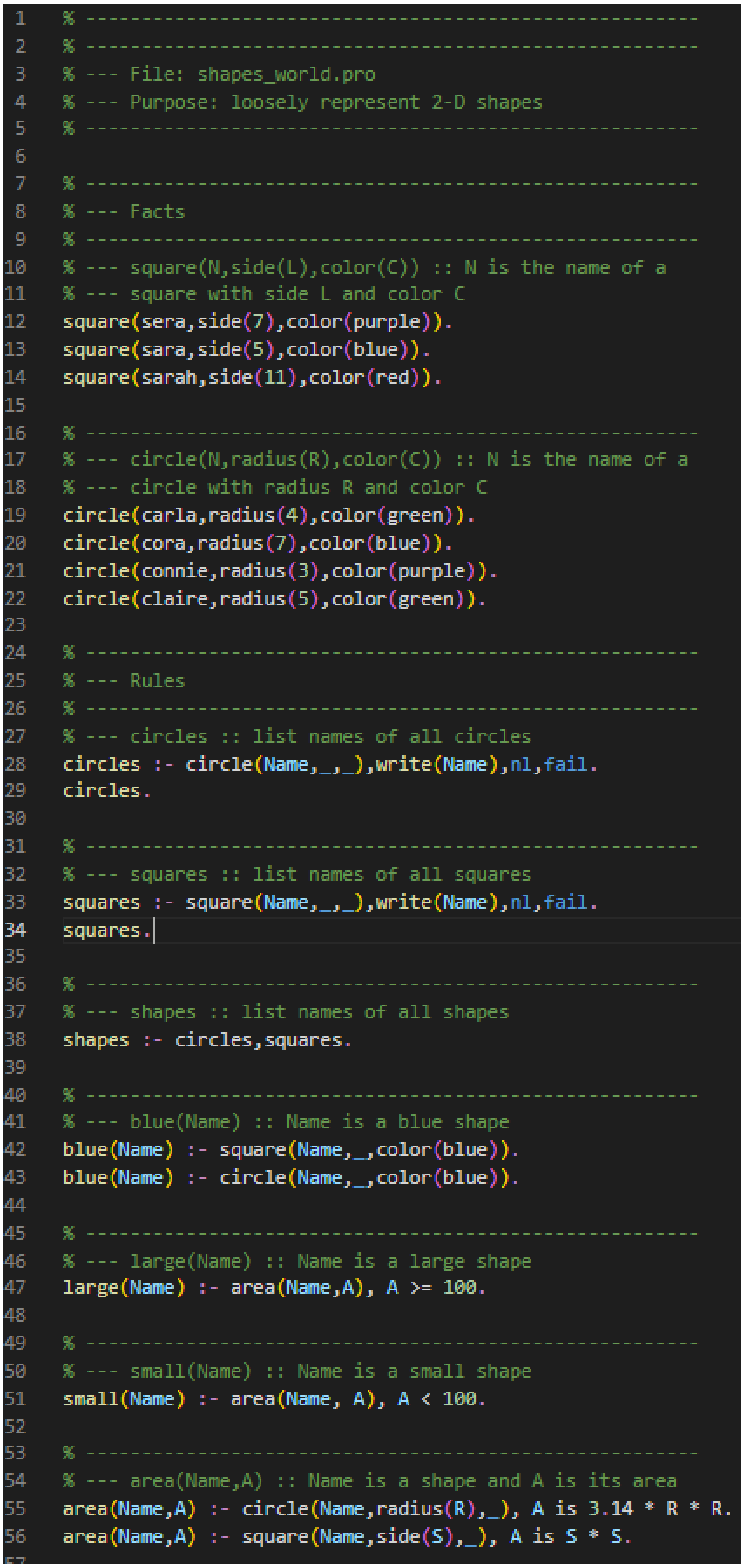


```
ERROR: )coloring( A, B, C, D, E, F, G, H, I, J, K, L, M, N, O, P,
?- coloring( A, B, C, D, E, F, G, H, I, J, K, L, M, N, O, P, Q).
A = F, F = H, H = J, J = L, L = red,
B = D, D = I, I = M, M = O, O = Q, Q = yellow,
C = E, E = G, G = K, K = N, N = P, P = green ;
A = F, F = H, H = J, J = L, L = red,
B = D, D = I, I = M, M = O, O = yellow,
C = E, E = G, G = K, K = N, N = P, P = green,
Q = blue .
```

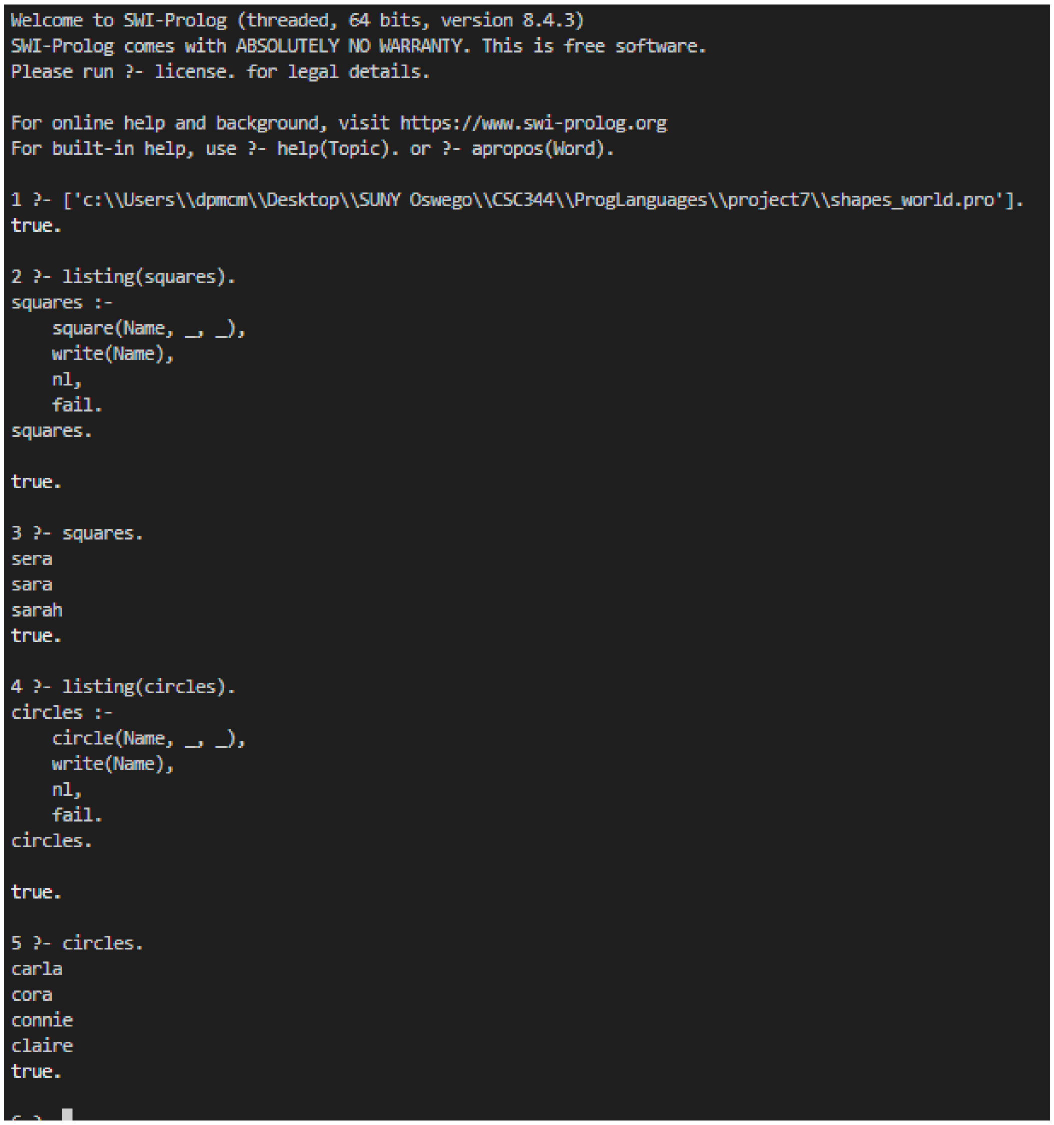


Task #2: Floating Worlds KB

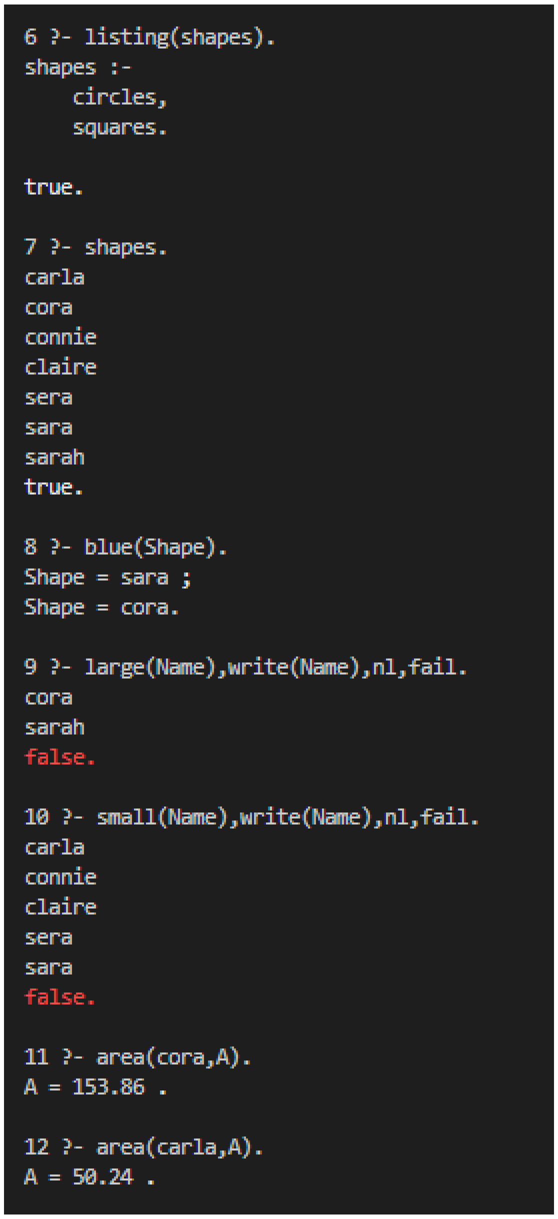




```prolog
% ---------------------------------------------------------
% ---------------------------------------------------------
% --- File: shapes_world.pro
% --- Purpose: loosely represent 2-D shapes
% ---------------------------------------------------------

% ---------------------------------------------------------
% --- 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 names of all circles
circles :- circle(Name,_,_),write(Name),nl,fail.
circles.

% ---------------------------------------------------------
% --- squares :: list names of all squares
squares :- square(Name,_,_),write(Name),nl,fail.
squares.

% ---------------------------------------------------------
% --- shapes :: list names of all 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) :: Name is a shape and A is its area
area(Name,A) :- circle(Name,radius(R),_), A is 3.14 * R * R.
area(Name,A) :- square(Name,side(S),_), A is S * S.
```

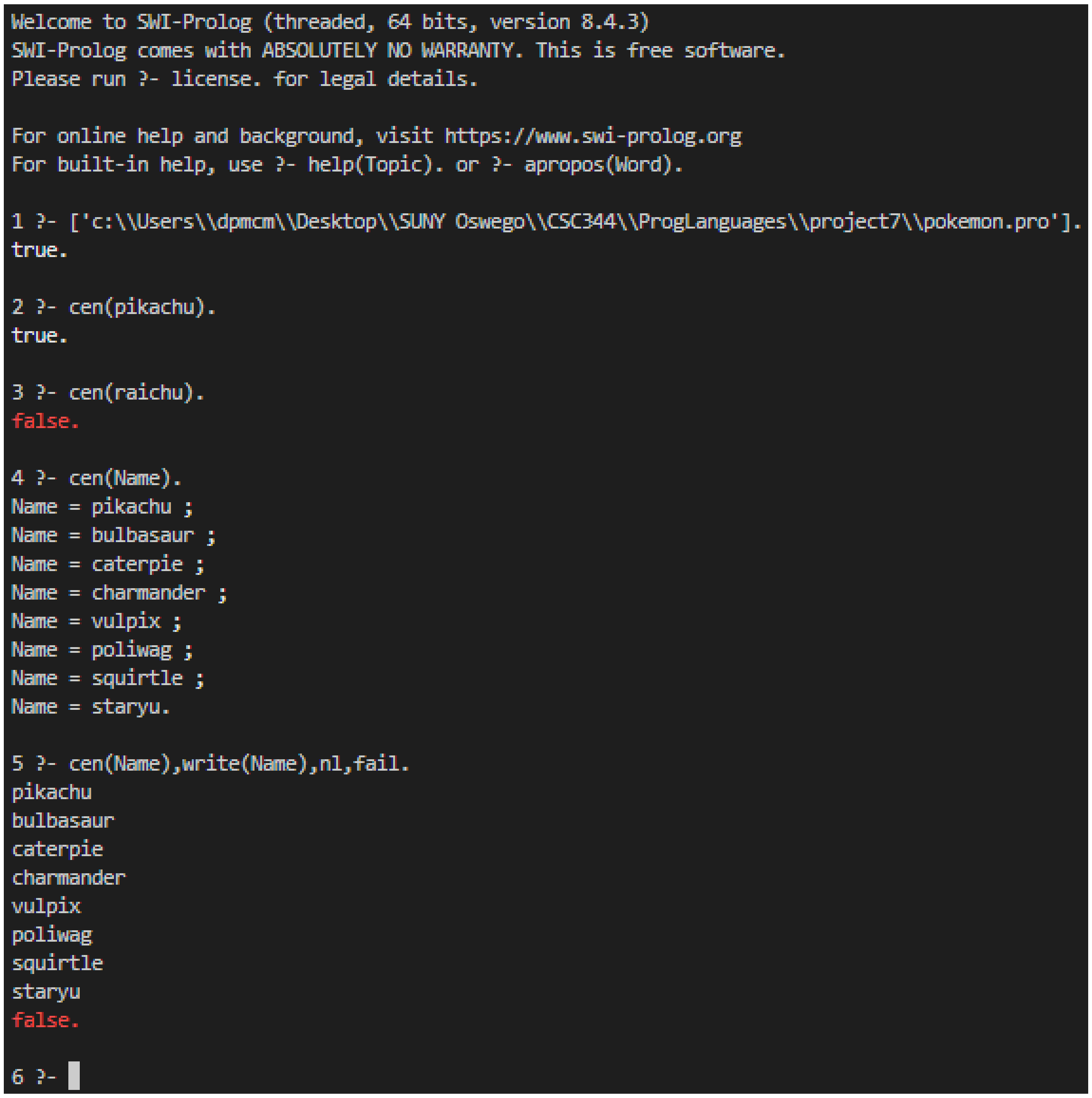


```
Welcome to SWI-Prolog (threaded, 64 bits, version 8.4.3)
SWI-Prolog comes with ABSOLUTELY NO WARRANTY. This is free software.
Please run ?- license. for legal details.

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

1 ?- ['c:\\Users\\dpmcm\\Desktop\\SUNY Oswego\\CSC344\\ProgLanguages\\project7\\shapes_world.pro'].
true.

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

true.

3 ?- squares.
sera
sara
sarah
true.

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

true.

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

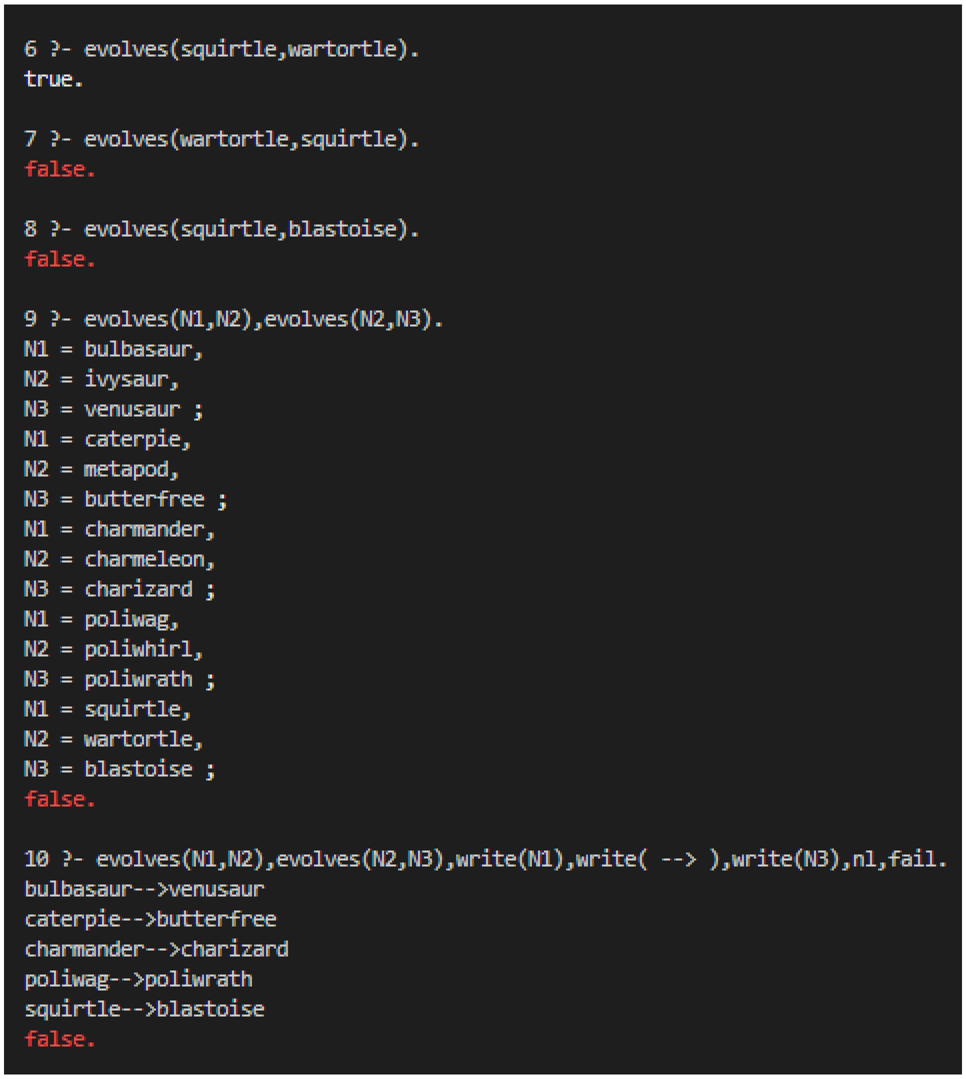

```
6 ?- listing(shapes).
shapes :-
    circles,
    squares.

true.

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

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

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

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

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

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

Task #3: Pokemon KB

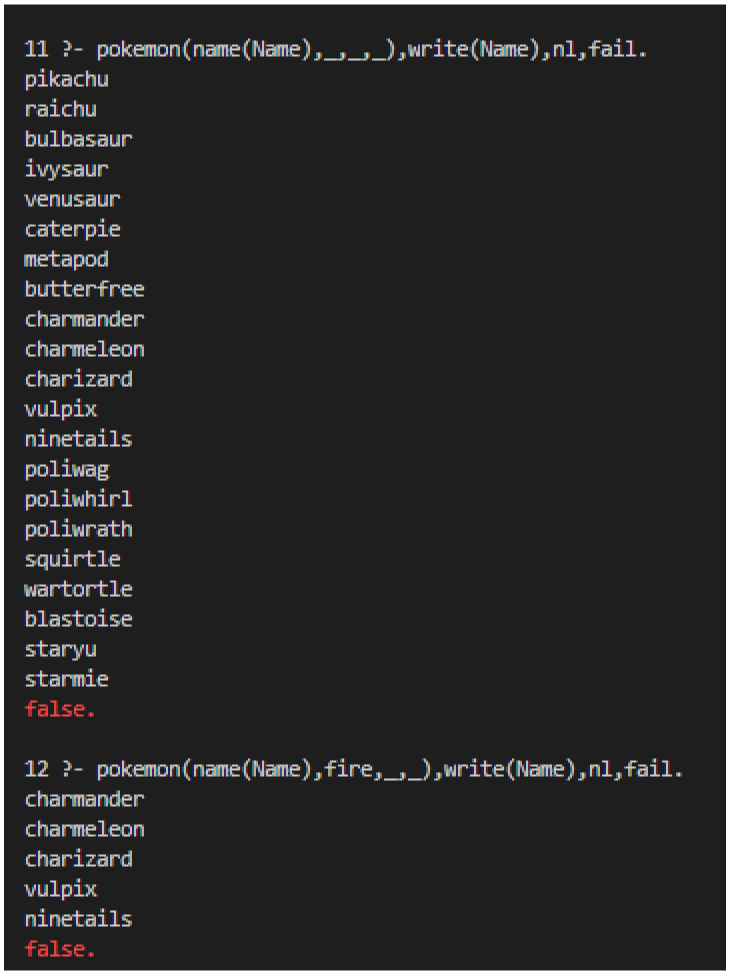
Part 1:



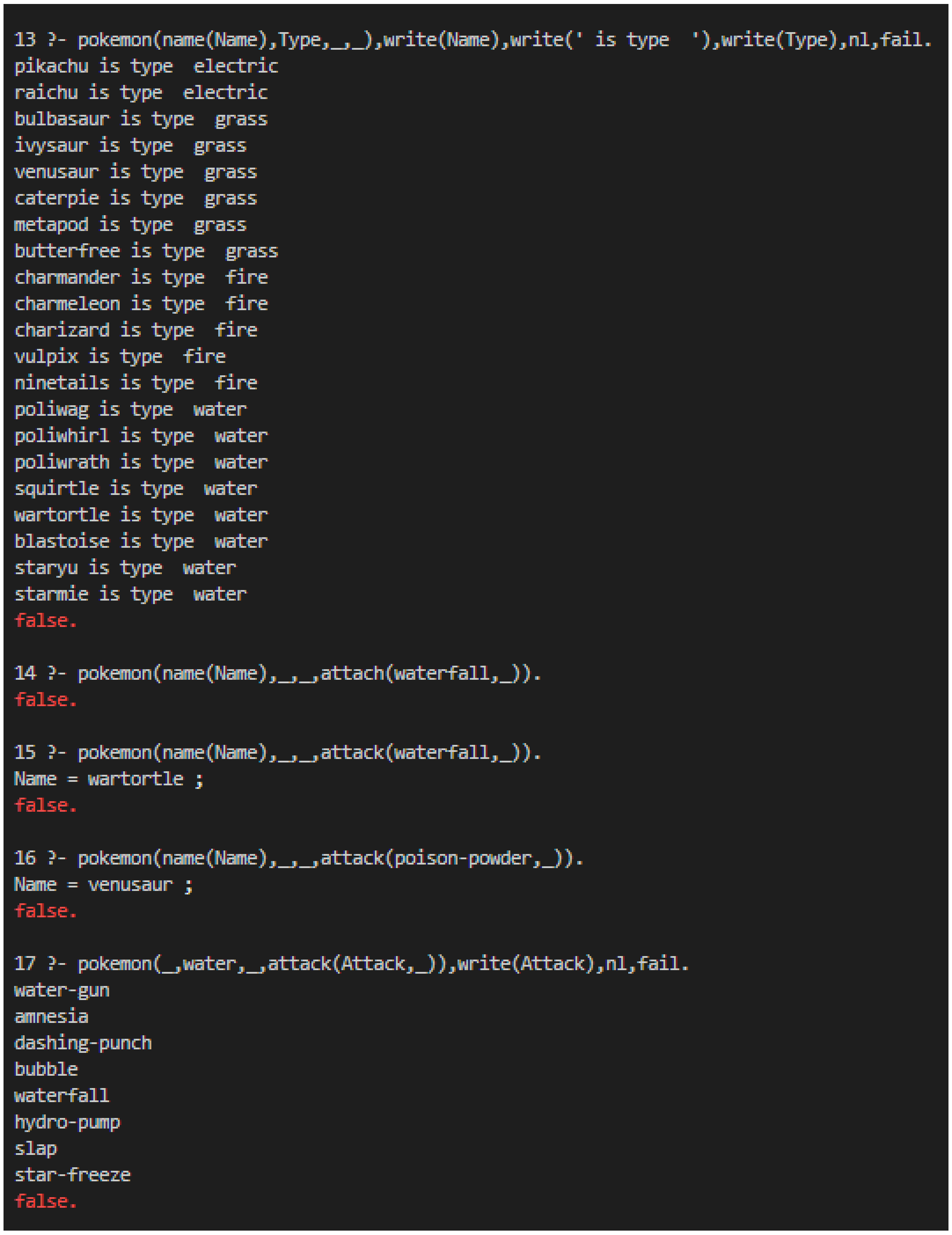
```
Welcome to SWI-Prolog (threaded, 64 bits, version 8.4.3)
SWI-Prolog comes with ABSOLUTELY NO WARRANTY. This is free software.
Please run ?- license. for legal details.

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

1 ?- ['c:\\Users\\dpmcm\\Desktop\\SUNY Oswego\\CSC344\\ProgLanguages\\project7\\pokemon.pro'].
true.

2 ?- cen(pikachu).
true.

3 ?- cen(raichu).
false.

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

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

6 ?-
```



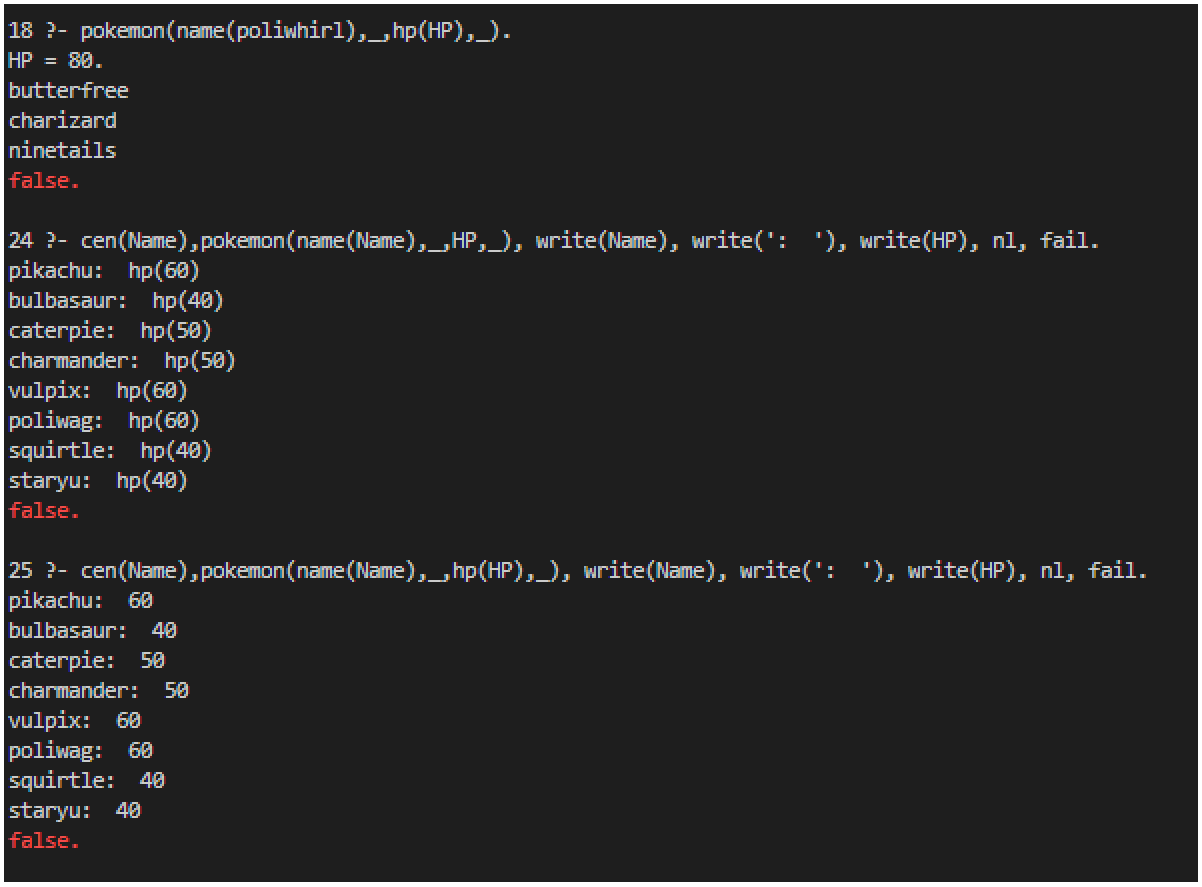
```
6 ?- evolves(squirtle,wartortle).
true.

7 ?- evolves(wartortle,squirtle).
false.

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

9 ?- evolves(N1,N2),evolves(N2,N3).
N1 = bulbasaur,
N2 = ivysaur,
N3 = venusaur ;
N1 = caterpie,
N2 = metapod,
N3 = butterfree ;
N1 = charmander,
N2 = charmeleon,
N3 = charizard ;
N1 = poliwag,
N2 = poliwhirl,
N3 = poliwrath ;
N1 = squirtle,
N2 = wartortle,
N3 = blastoise ;
false.

10 ?- evolves(N1,N2),evolves(N2,N3),write(N1),write( --> ),write(N3),nl,fail.
bulbasaur-->venusaur
caterpie-->butterfree
charmander-->charizard
poliwag-->poliwrath
squirtle-->blastoise
false.
```



```
11 ?- 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.

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

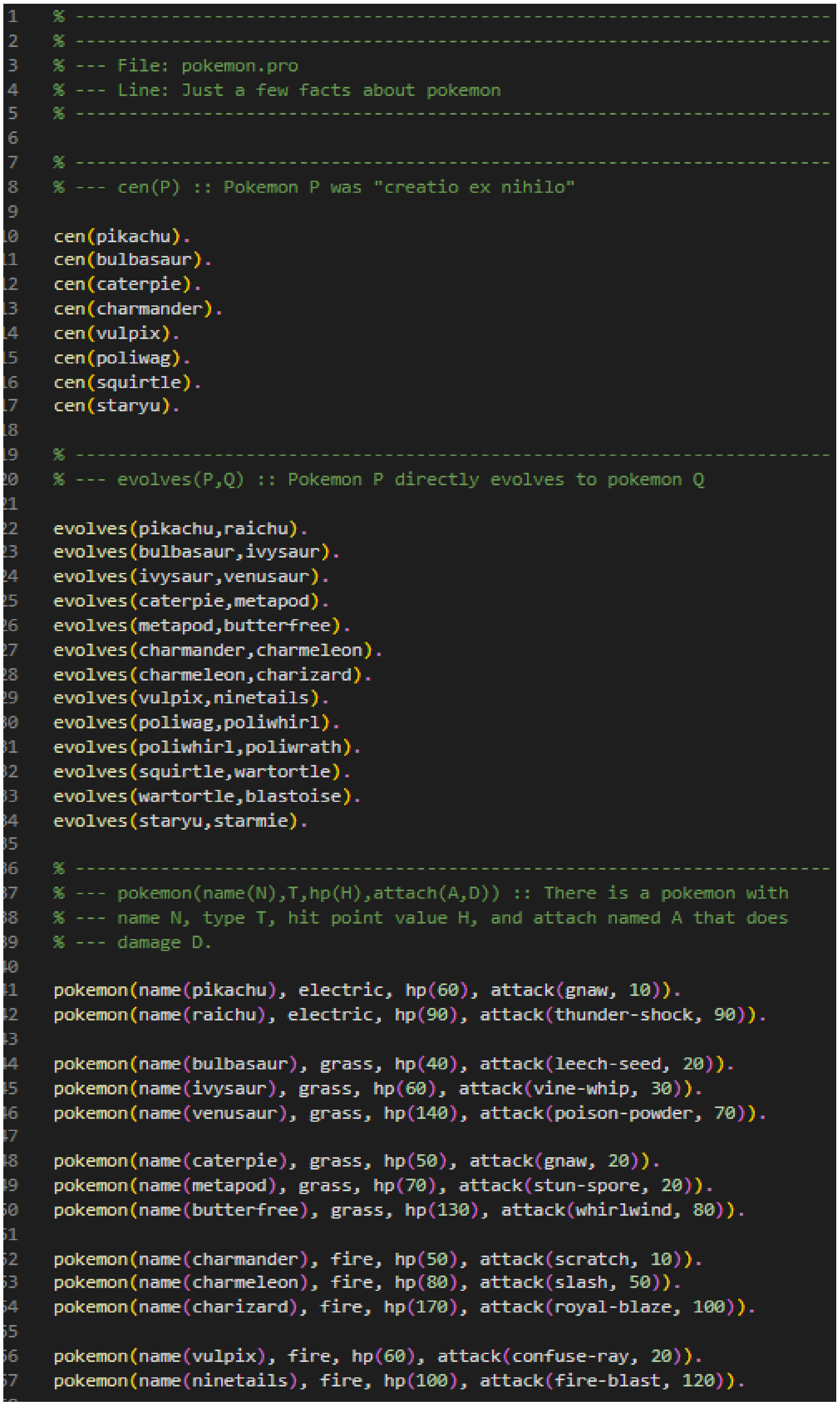


```
13 ?- pokemon(name(Name),Type,_,_),write(Name),write(' is type  '),write(Type),nl,fail.
pikachu is type  electric
raichu is type  electric
bulbasaur is type  grass
ivysaur is type  grass
venusaur is type  grass
caterpie is type  grass
metapod is type  grass
butterfree is type  grass
charmander is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

14 ?- pokemon(name(Name),_,_,attach(waterfall,_)).
false.

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

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

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

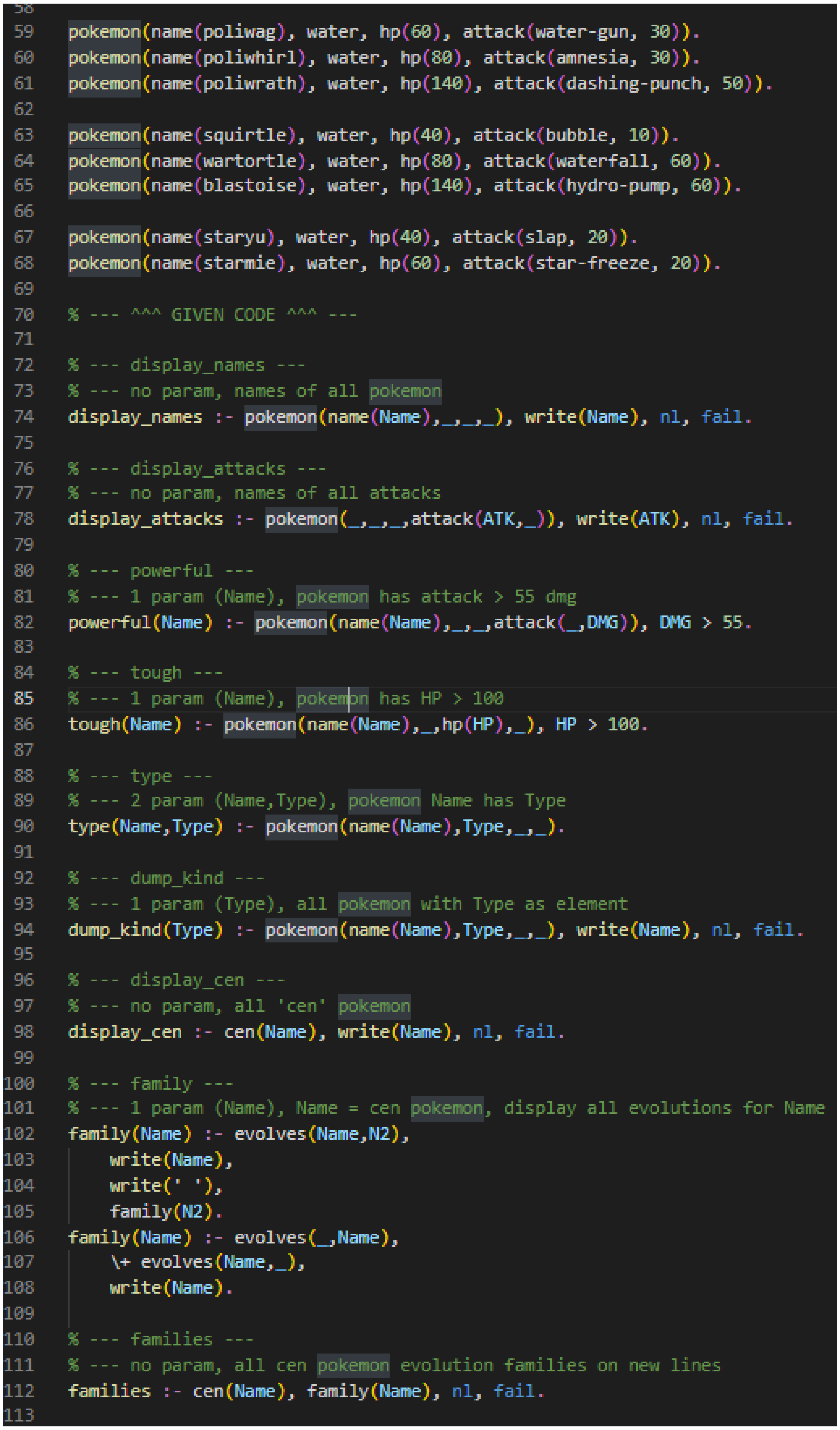


```
18 ?- pokemon(name(poliwhirl),_,hp(HP),_).
HP = 80.
butterfree
charizard
ninetails
false.

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

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

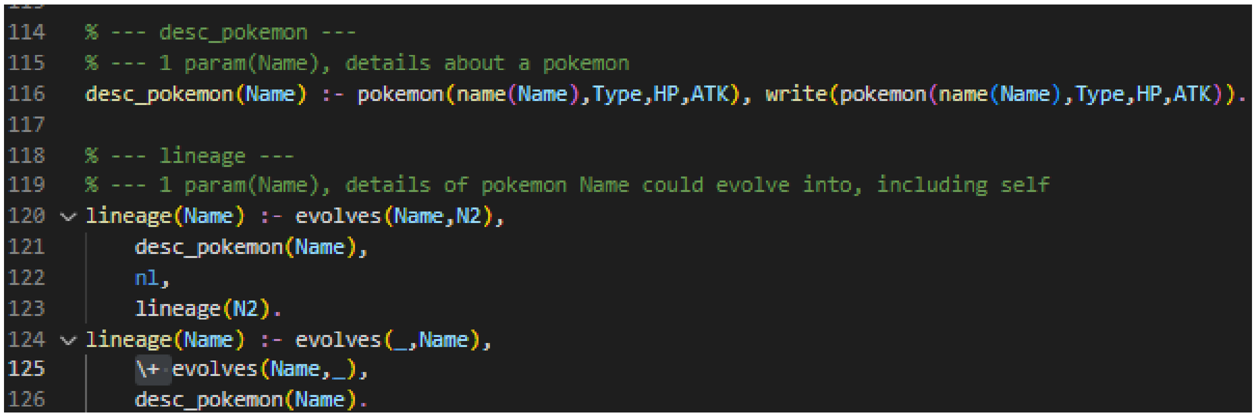
Part 2:



```
% ----------------------------------------------------------------------
% ----------------------------------------------------------------------
% --- File: pokemon.pro
% --- Line: Just a few facts about 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)).
```

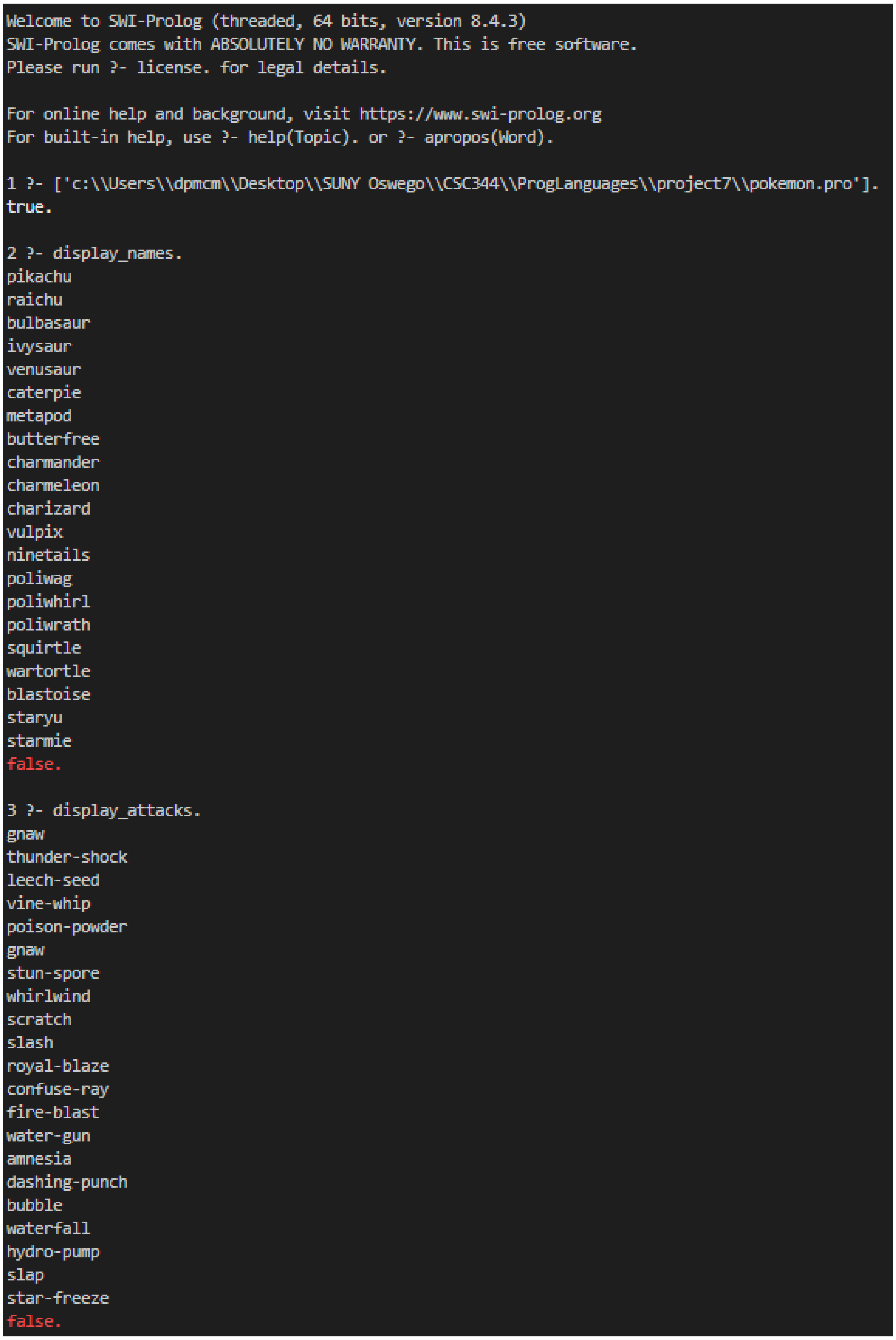


```
pokemon(name(poliwag), water, hp(60), attack(water-gun, 30)).
pokemon(name(poliwhirl), water, hp(80), attack(amnesia, 30)).
pokemon(name(poliwrath), water, hp(140), attack(dashing-punch, 50)).

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)).

% --- ^^^ GIVEN CODE ^^^ ---

% --- display_names ---
% --- no param, names of all pokemon
display_names :- pokemon(name(Name),_,_,_), write(Name), nl, fail.

% --- display_attacks ---
% --- no param, names of all attacks
display_attacks :- pokemon(_,_,_,attack(ATK,_)), write(ATK), nl, fail.

% --- powerful ---
% --- 1 param (Name), pokemon has attack > 55 dmg
powerful(Name) :- pokemon(name(Name),_,_,attack(_,DMG)), DMG > 55.

% --- tough ---
% --- 1 param (Name), pokemon has HP > 100
tough(Name) :- pokemon(name(Name),_,hp(HP),_), HP > 100.

% --- type ---
% --- 2 param (Name,Type), pokemon Name has Type
type(Name,Type) :- pokemon(name(Name),Type,_,_).

% --- dump_kind ---
% --- 1 param (Type), all pokemon with Type as element
dump_kind(Type) :- pokemon(name(Name),Type,_,_), write(Name), nl, fail.

% --- display_cen ---
% --- no param, all 'cen' pokemon
display_cen :- cen(Name), write(Name), nl, fail.

% --- family ---
% --- 1 param (Name), Name = cen pokemon, display all evolutions for Name
family(Name) :- evolves(Name,N2),
    write(Name),
    write(' '),
    family(N2).
family(Name) :- evolves(_,Name),
    \+ evolves(Name,_),
    write(Name).

% --- families ---
% --- no param, all cen pokemon evolution families on new lines
families :- cen(Name), family(Name), nl, fail.
```

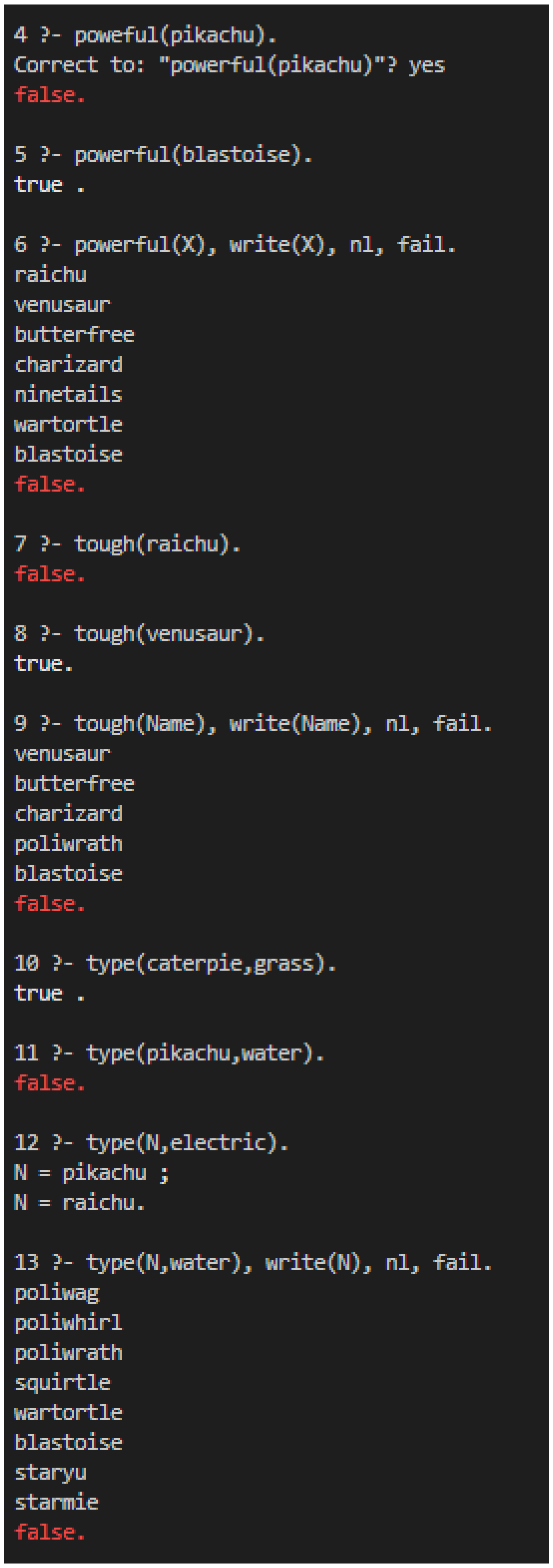


```prolog
% --- desc_pokemon ---
% --- 1 param(Name), details about a pokemon
desc_pokemon(Name) :- pokemon(name(Name),Type,HP,ATK), write(pokemon(name(Name),Type,HP,ATK)).

% --- lineage ---
% --- 1 param(Name), details of pokemon Name could evolve into, including self
lineage(Name) :- evolves(Name,N2),
    desc_pokemon(Name),
    nl,
    lineage(N2).
lineage(Name) :- evolves(_,Name),
    \+ evolves(Name,_),
    desc_pokemon(Name).
```

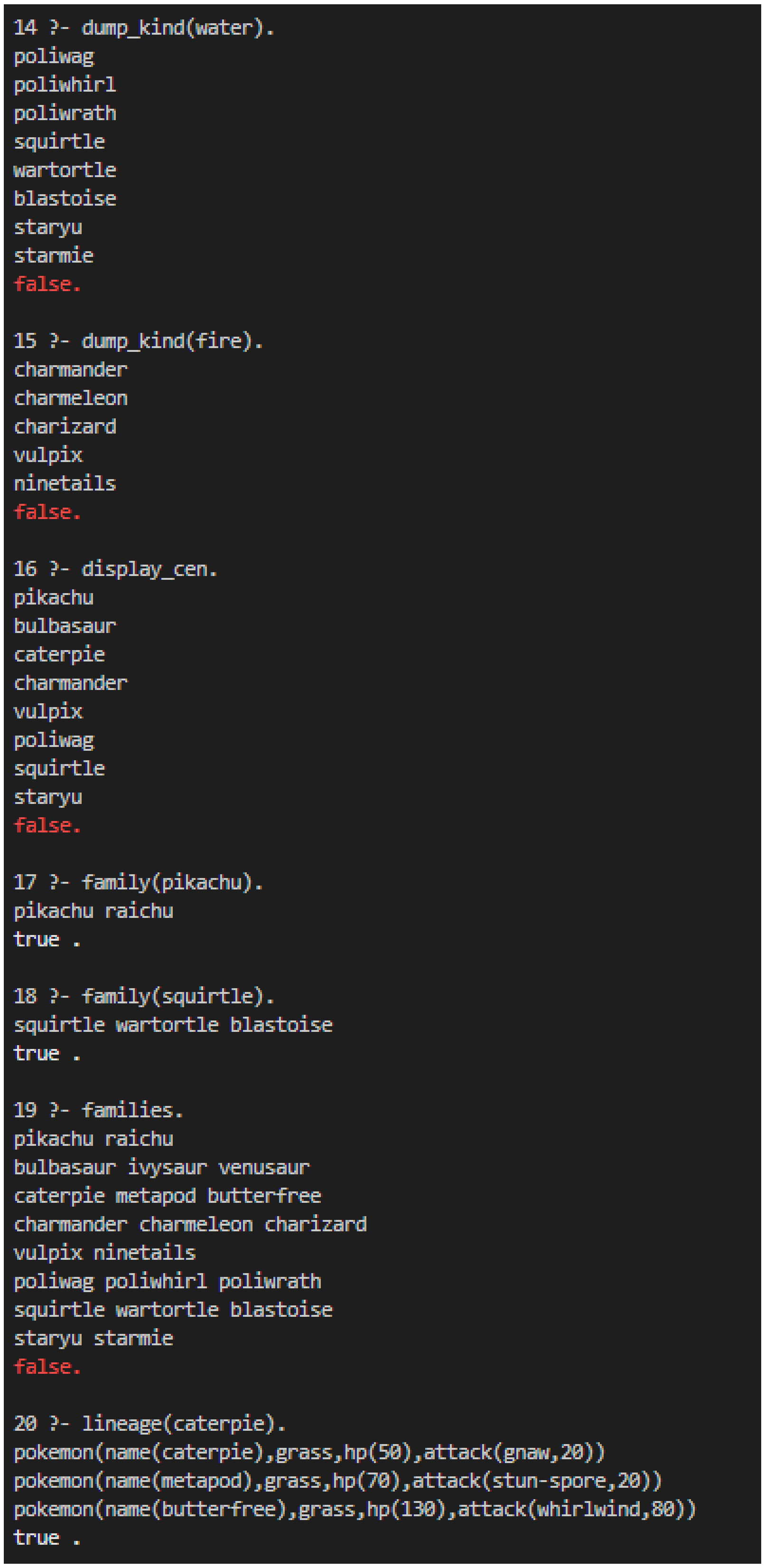
Part 3:



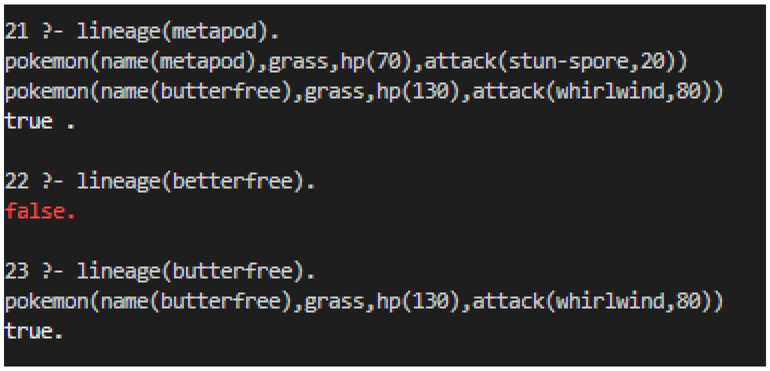
```
Welcome to SWI-Prolog (threaded, 64 bits, version 8.4.3)
SWI-Prolog comes with ABSOLUTELY NO WARRANTY. This is free software.
Please run ?- license. for legal details.

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

1 ?- ['c:\\Users\\dpmcm\\Desktop\\SUNY Oswego\\CSC344\\ProgLanguages\\project7\\pokemon.pro'].
true.

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

3 ?- 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.
```



```
4 ?- poweful(pikachu).
Correct to: "powerful(pikachu)"? yes
false.

5 ?- powerful(blastoise).
true .

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

7 ?- tough(raichu).
false.

8 ?- tough(venusaur).
true.

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

10 ?- type(caterpie,grass).
true .

11 ?- type(pikachu,water).
false.

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

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



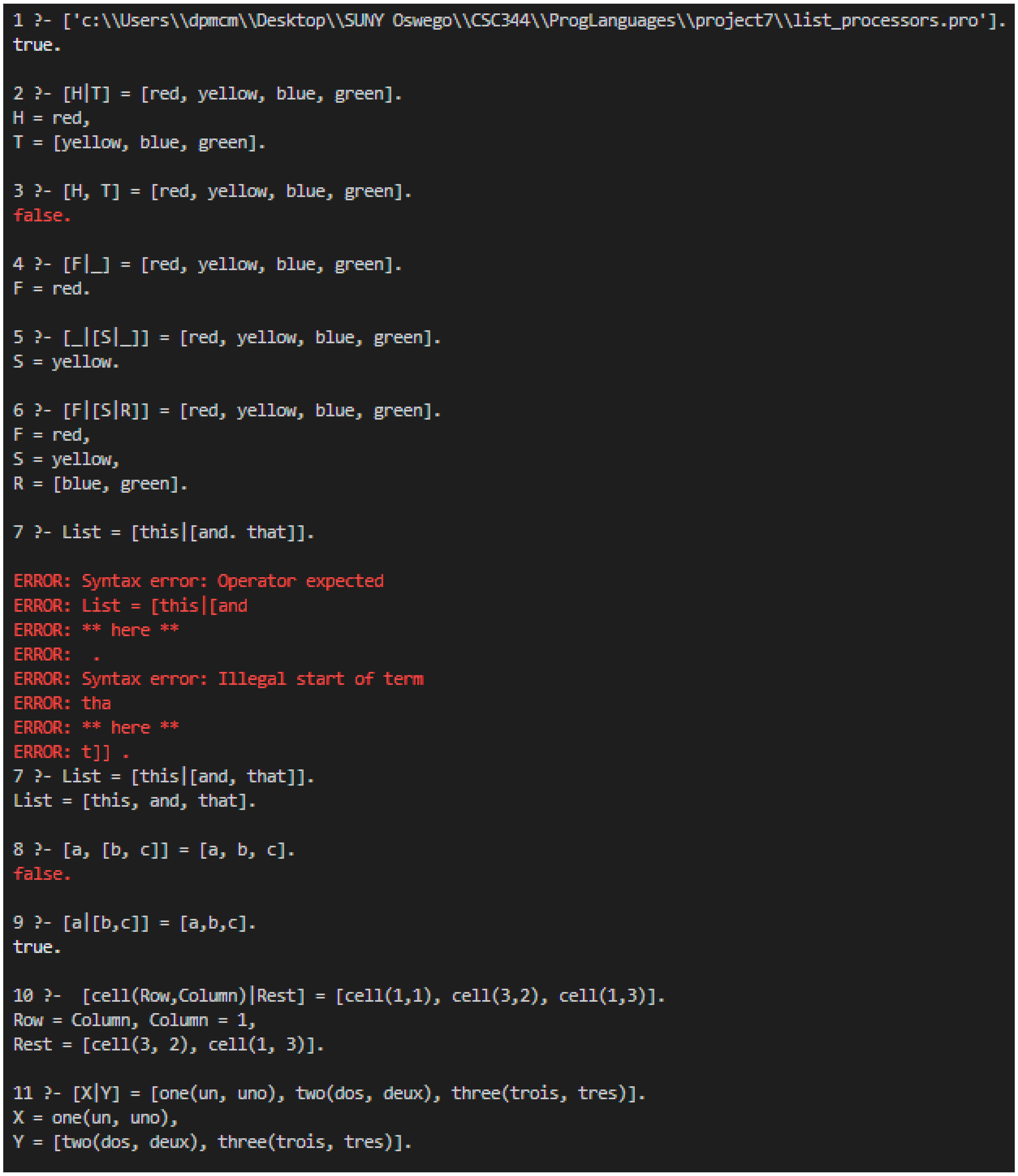
```
14 ?- dump_kind(water).
poliwag
poliwhirl
poliwrath
squirtle
wartortle
blastoise
staryu
starmie
false.

15 ?- dump_kind(fire).
charmander
charmeleon
charizard
vulpix
ninetails
false.

16 ?- display_cen.
pikachu
bulbasaur
caterpie
charmander
vulpix
poliwag
squirtle
staryu
false.

17 ?- family(pikachu).
pikachu raichu
true .

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

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

20 ?- 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 .
```



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

22 ?- lineage(betterfree).
false.

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

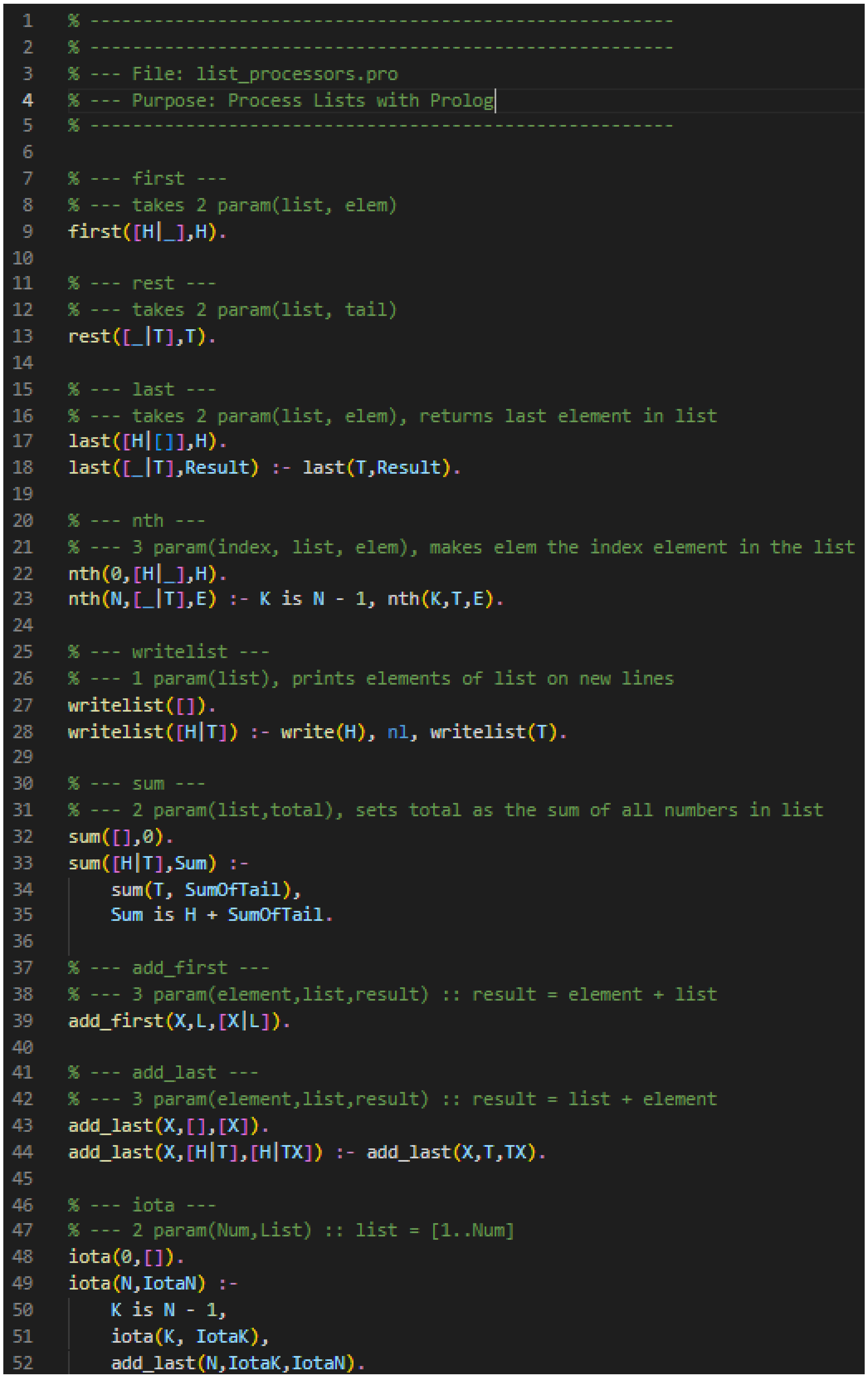
Task #4: List Processing in Prolog

Part 1:

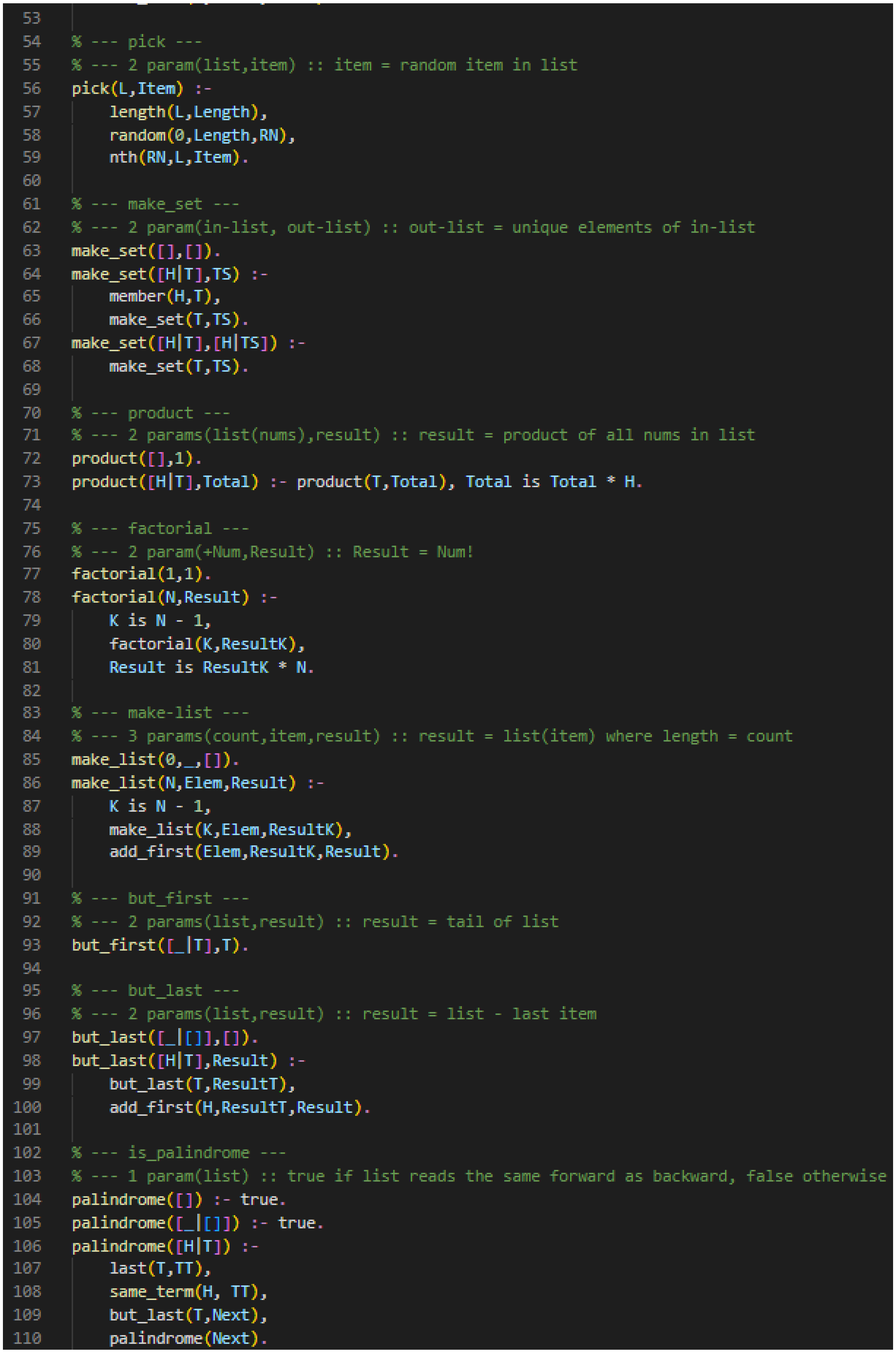


```
1 ?- ['c:\\Users\\dpmcm\\Desktop\\SUNY Oswego\\CSC344\\ProgLanguages\\project7\\list_processors.pro'].
true.

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

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

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

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

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

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

ERROR: Syntax error: Operator expected
ERROR: List = [this|[and
ERROR: ** here **
ERROR:  .
ERROR: Syntax error: Illegal start of term
ERROR: tha
ERROR: ** here **
ERROR: t]] .
7 ?- List = [this|[and, that]].
List = [this, and, that].

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

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

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

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

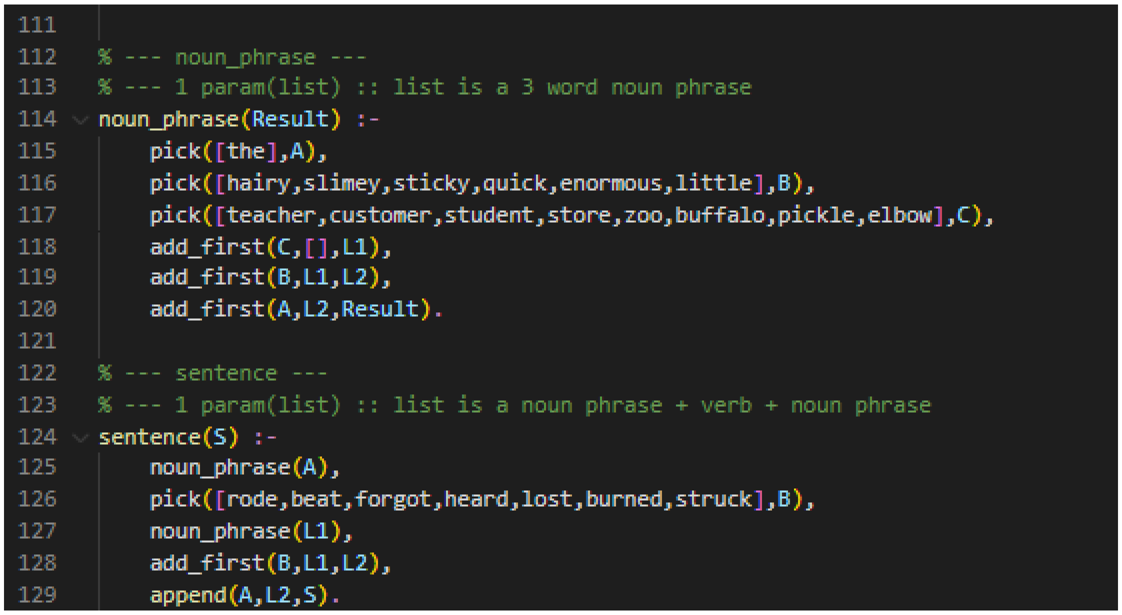
Part 2:



```
% ------------------------------------------------------
% ------------------------------------------------------
% --- File: list_processors.pro
% --- Purpose: Process Lists with Prolog
% ------------------------------------------------------

% --- first ---
% --- takes 2 param(list, elem)
first([H|_],H).

% --- rest ---
% --- takes 2 param(list, tail)
rest([_|T],T).

% --- last ---
% --- takes 2 param(list, elem), returns last element in list
last([H|[]],H).
last([_|T],Result) :- last(T,Result).

% --- nth ---
% --- 3 param(index, list, elem), makes elem the index element in the list
nth(0,[H|_],H).
nth(N,[_|T],E) :- K is N - 1, nth(K,T,E).

% --- writelist ---
% --- 1 param(list), prints elements of list on new lines
writelist([]).
writelist([H|T]) :- write(H), nl, writelist(T).

% --- sum ---
% --- 2 param(list,total), sets total as the sum of all numbers in list
sum([],0).
sum([H|T],Sum) :-
    sum(T, SumOfTail),
    Sum is H + SumOfTail.

% --- add_first ---
% --- 3 param(element,list,result) :: result = element + list
add_first(X,L,[X|L]).

% --- add_last ---
% --- 3 param(element,list,result) :: result = list + element
add_last(X,[],[X]).
add_last(X,[H|T],[H|TX]) :- add_last(X,T,TX).

% --- iota ---
% --- 2 param(Num,List) :: list = [1..Num]
iota(0,[]).
iota(N,IotaN) :-
    K is N - 1,
    iota(K, IotaK),
    add_last(N,IotaK,IotaN).
```

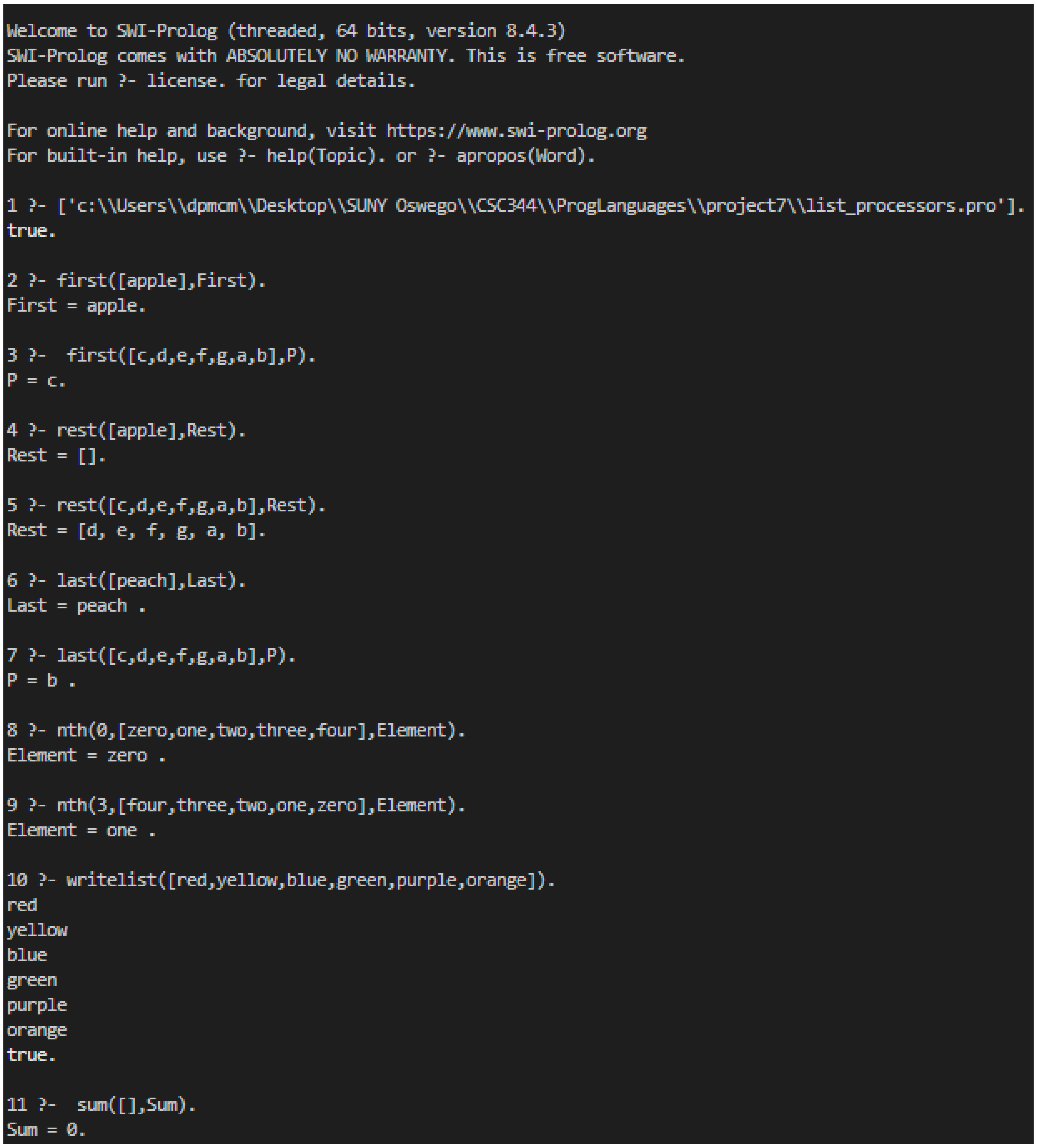


```prolog

% --- pick ---
% --- 2 param(list,item) :: item = random item in list
pick(L,Item) :-
    length(L,Length),
    random(0,Length,RN),
    nth(RN,L,Item).

% --- make_set ---
% --- 2 param(in-list, out-list) :: out-list = unique elements of in-list
make_set([],[]).
make_set([H|T],TS) :-
    member(H,T),
    make_set(T,TS).
make_set([H|T],[H|TS]) :-
    make_set(T,TS).

% --- product ---
% --- 2 params(list(nums),result) :: result = product of all nums in list
product([],1).
product([H|T],Total) :- product(T,Total), Total is Total * H.

% --- factorial ---
% --- 2 param(+Num,Result) :: Result = Num!
factorial(1,1).
factorial(N,Result) :-
    K is N - 1,
    factorial(K,ResultK),
    Result is ResultK * N.

% --- make-list ---
% --- 3 params(count,item,result) :: result = list(item) where length = count
make_list(0,_,[]).
make_list(N,Elem,Result) :-
    K is N - 1,
    make_list(K,Elem,ResultK),
    add_first(Elem,ResultK,Result).

% --- but_first ---
% --- 2 params(list,result) :: result = tail of list
but_first([_|T],T).

% --- but_last ---
% --- 2 params(list,result) :: result = list - last item
but_last([_|[]],[]).
but_last([H|T],Result) :-
    but_last(T,ResultT),
    add_first(H,ResultT,Result).

% --- is_palindrome ---
% --- 1 param(list) :: true if list reads the same forward as backward, false otherwise
palindrome([]) :- true.
palindrome([_|[]]) :- true.
palindrome([H|T]) :-
    last(T,TT),
    same_term(H, TT),
    but_last(T,Next),
    palindrome(Next).
```

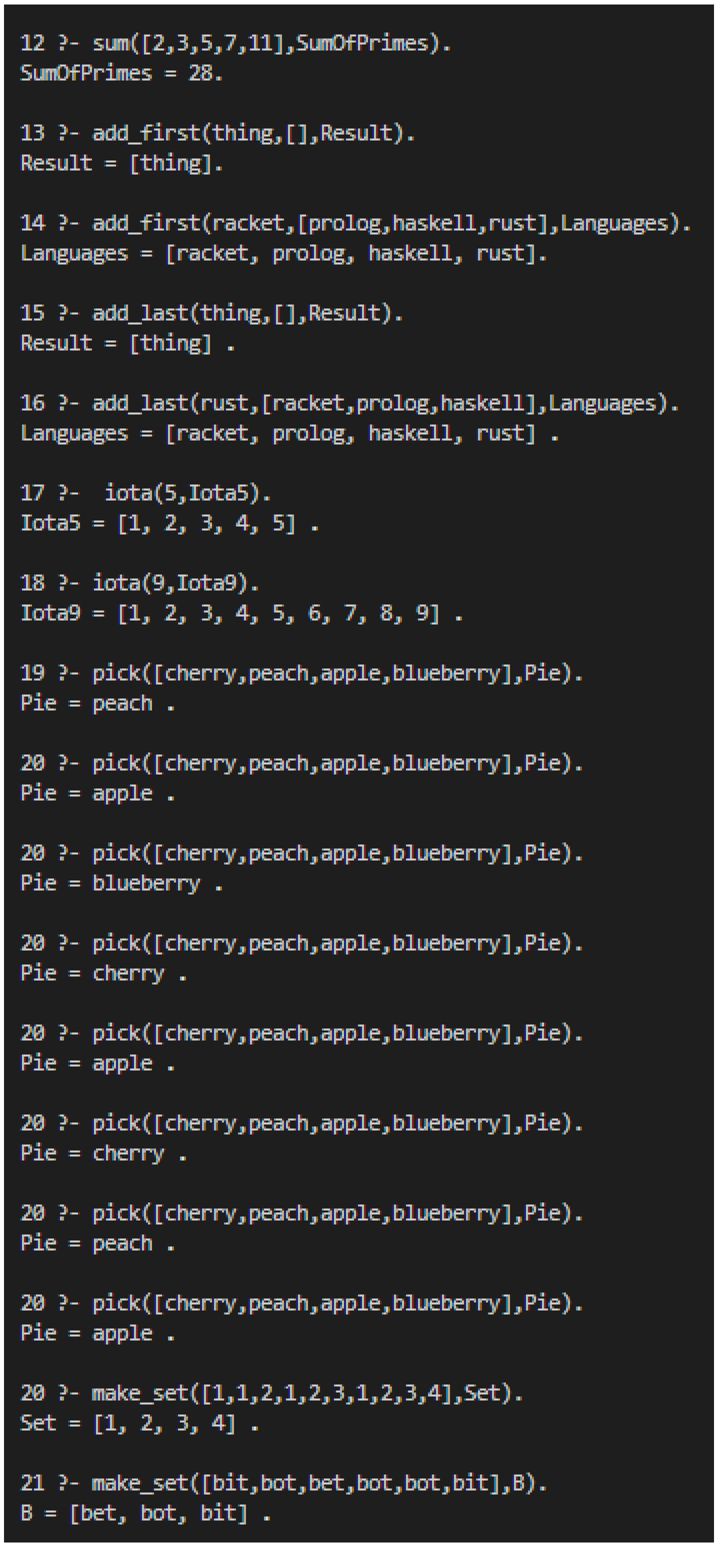


```prolog

% --- noun_phrase ---
% --- 1 param(list) :: list is a 3 word noun phrase
noun_phrase(Result) :-
    pick([the],A),
    pick([hairy,slimey,sticky,quick,enormous,little],B),
    pick([teacher,customer,student,store,zoo,buffalo,pickle,elbow],C),
    add_first(C,[],L1),
    add_first(B,L1,L2),
    add_first(A,L2,Result).

% --- sentence ---
% --- 1 param(list) :: list is a noun phrase + verb + noun phrase
sentence(S) :-
    noun_phrase(A),
    pick([rode,beat,forgot,heard,lost,burned,struck],B),
    noun_phrase(L1),
    add_first(B,L1,L2),
    append(A,L2,S).
```

Part 3:

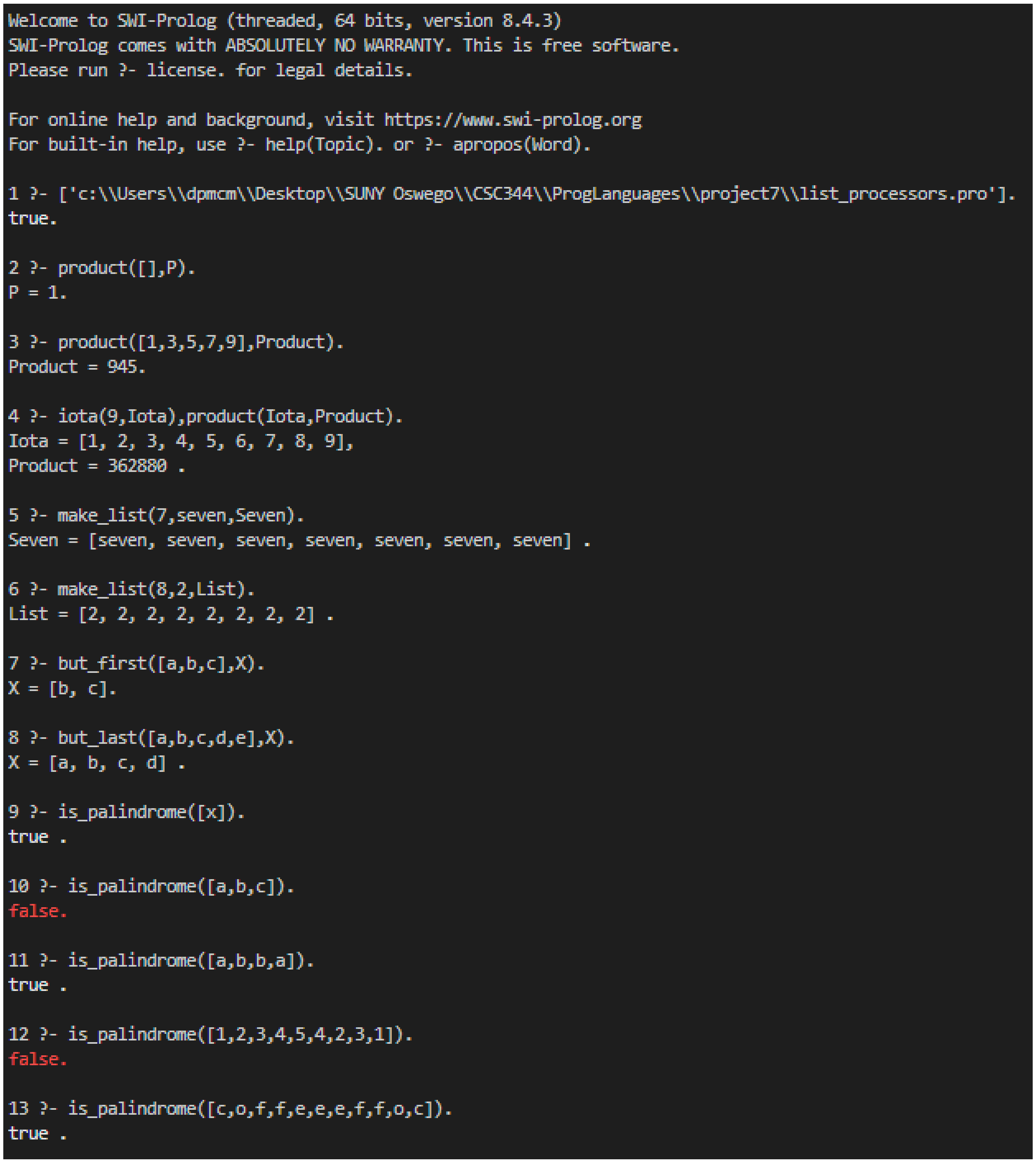


```
Welcome to SWI-Prolog (threaded, 64 bits, version 8.4.3)
SWI-Prolog comes with ABSOLUTELY NO WARRANTY. This is free software.
Please run ?- license. for legal details.

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

1 ?- ['c:\\Users\\dpmcm\\Desktop\\SUNY Oswego\\CSC344\\ProgLanguages\\project7\\list_processors.pro'].
true.

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

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

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

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

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

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

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

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

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

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

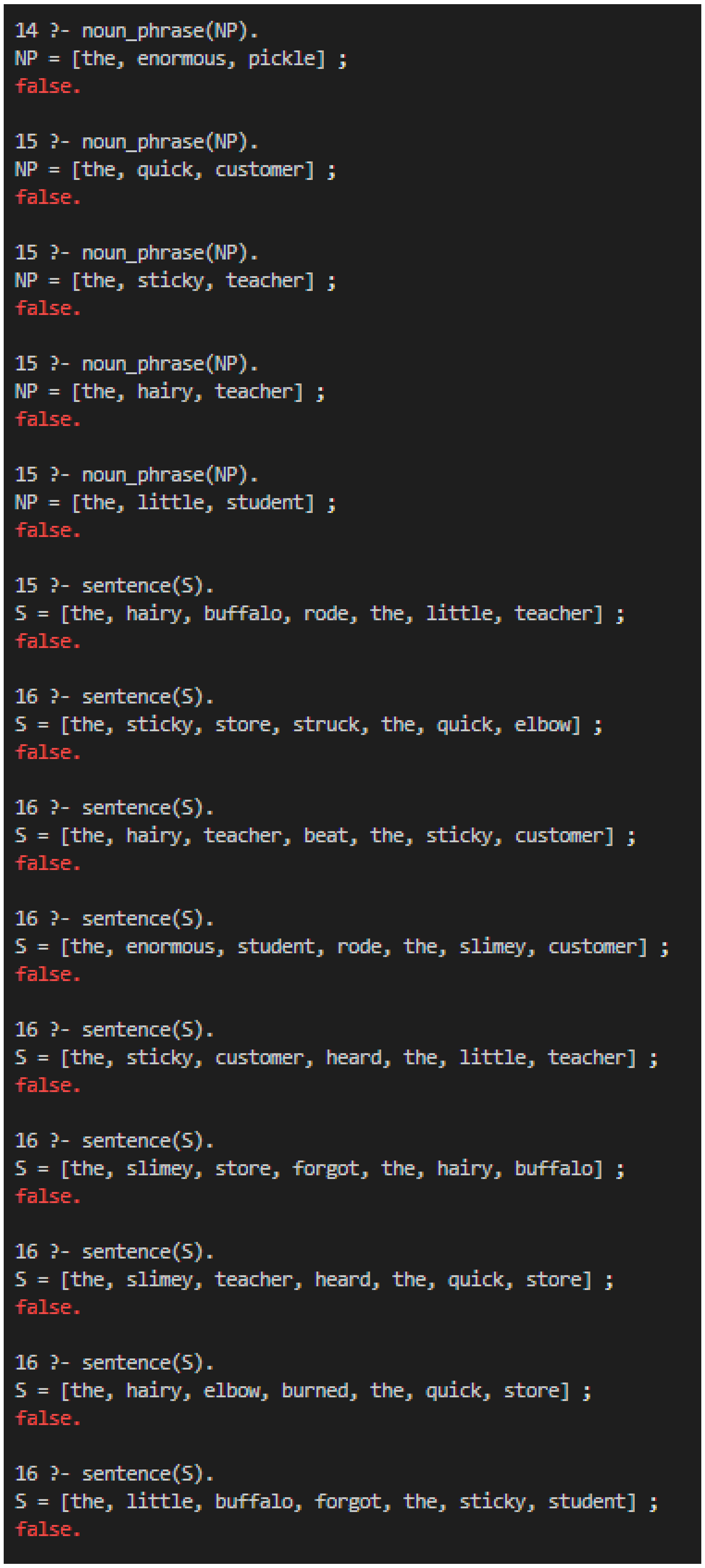


```
12 ?- sum([2,3,5,7,11],SumOfPrimes).
SumOfPrimes = 28.

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

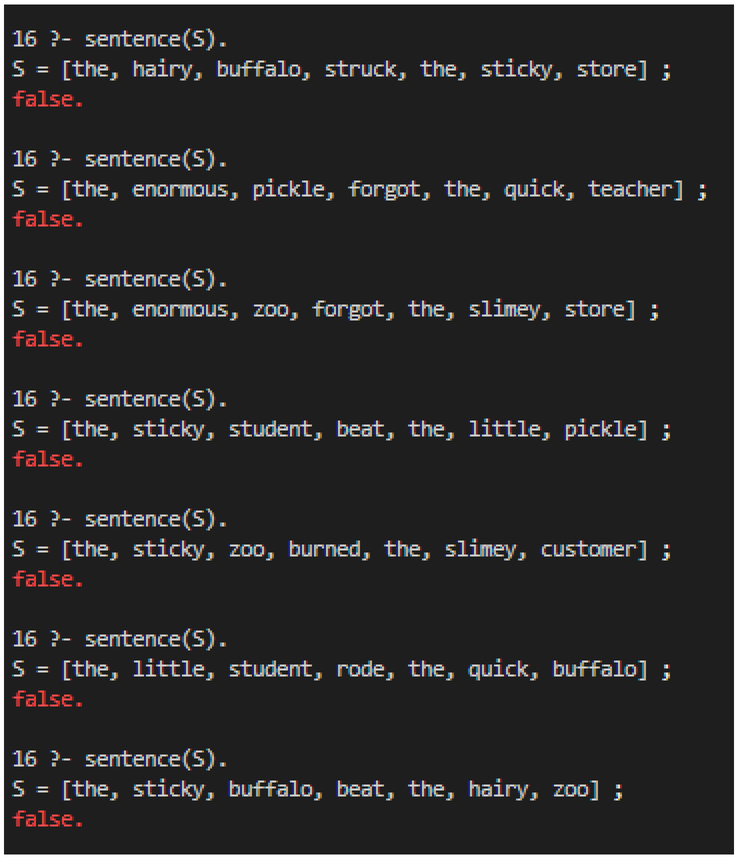
Part 4:



```
Welcome to SWI-Prolog (threaded, 64 bits, version 8.4.3)
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).

1 ?- ['c:\\Users\\dpmcm\\Desktop\\SUNY Oswego\\CSC344\\ProgLanguages\\project7\\list_processors.pro'].
true.

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

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

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

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

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

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

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

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

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

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

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

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



```
14 ?- noun_phrase(NP).
NP = [the, enormous, pickle] ;
false.

15 ?- noun_phrase(NP).
NP = [the, quick, customer] ;
false.

15 ?- noun_phrase(NP).
NP = [the, sticky, teacher] ;
false.

15 ?- noun_phrase(NP).
NP = [the, hairy, teacher] ;
false.

15 ?- noun_phrase(NP).
NP = [the, little, student] ;
false.

15 ?- sentence(S).
S = [the, hairy, buffalo, rode, the, little, teacher] ;
false.

16 ?- sentence(S).
S = [the, sticky, store, struck, the, quick, elbow] ;
false.

16 ?- sentence(S).
S = [the, hairy, teacher, beat, the, sticky, customer] ;
false.

16 ?- sentence(S).
S = [the, enormous, student, rode, the, slimey, customer] ;
false.

16 ?- sentence(S).
S = [the, sticky, customer, heard, the, little, teacher] ;
false.

16 ?- sentence(S).
S = [the, slimey, store, forgot, the, hairy, buffalo] ;
false.

16 ?- sentence(S).
S = [the, slimey, teacher, heard, the, quick, store] ;
false.

16 ?- sentence(S).
S = [the, hairy, elbow, burned, the, quick, store] ;
false.

16 ?- sentence(S).
S = [the, little, buffalo, forgot, the, sticky, student] ;
false.
```



```
16 ?- sentence(S).
S = [the, hairy, buffalo, struck, the, sticky, store] ;
false.

16 ?- sentence(S).
S = [the, enormous, pickle, forgot, the, quick, teacher] ;
false.

16 ?- sentence(S).
S = [the, enormous, zoo, forgot, the, slimey, store] ;
false.

16 ?- sentence(S).
S = [the, sticky, student, beat, the, little, pickle] ;
false.

16 ?- sentence(S).
S = [the, sticky, zoo, burned, the, slimey, customer] ;
false.

16 ?- sentence(S).
S = [the, little, student, rode, the, quick, buffalo] ;
false.

16 ?- sentence(S).
S = [the, sticky, buffalo, beat, the, hairy, zoo] ;
false.
```