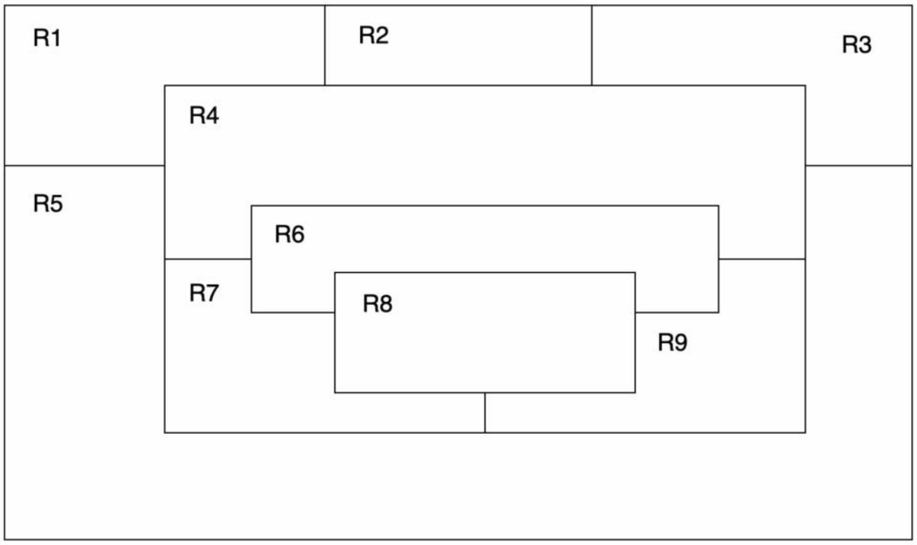
First Prolog Programming Assignment Specification

Learning Abstract

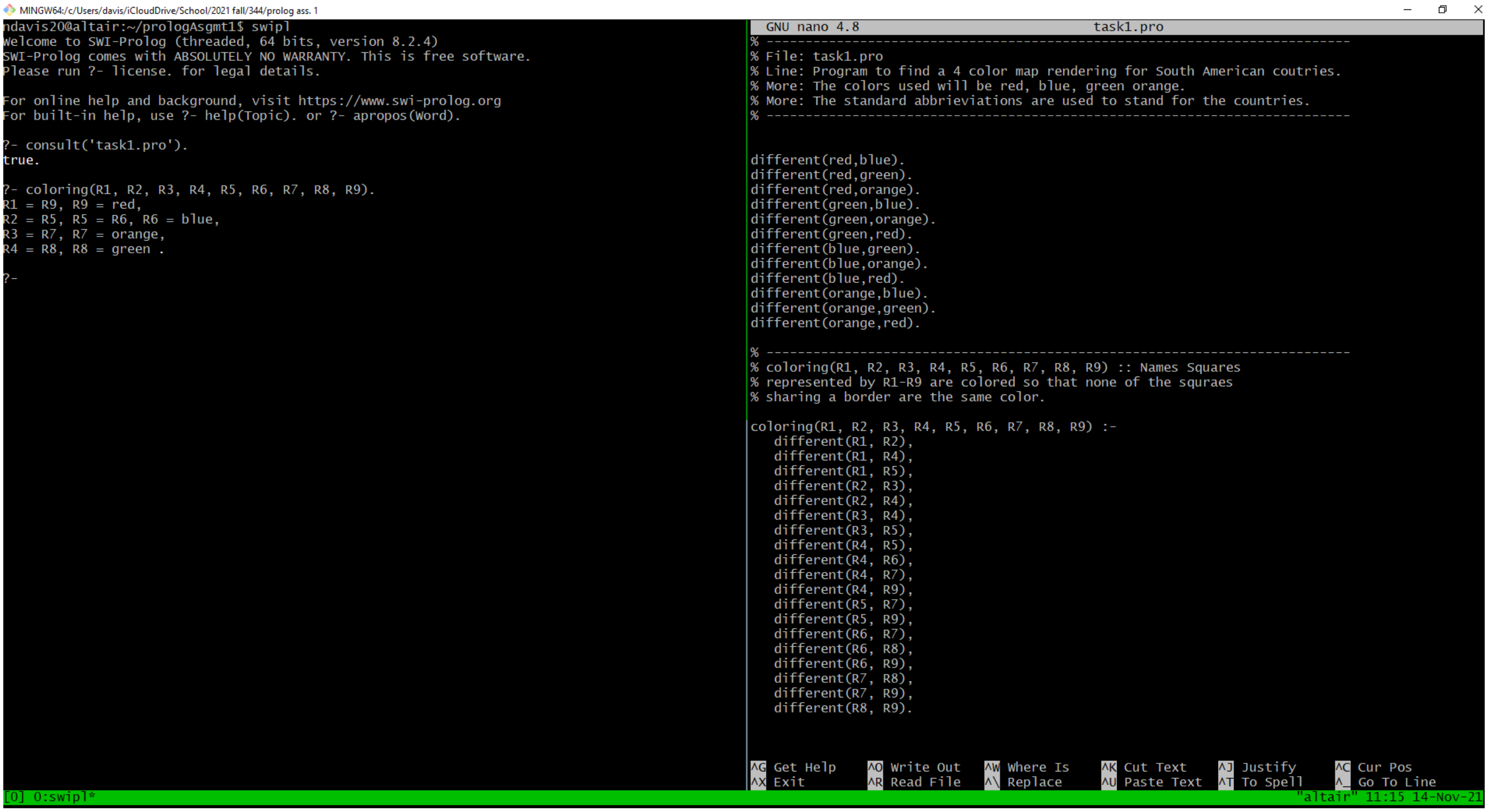
This Prolog programming assignment I got familiar with relations and facts, prolog terms(constants, variables, and compound terms), unification, how prolog can be used to perform search, and more. I got familiar with using recursion in lisp. This assignment is a good way to gain muscle memory for lisp.

Task 1: Map Coloring

Working by analogy with the map coloring program provided in class, which you can find in Lesson 3, write a map coloring program to solve the problem of coloring the following map in four colors.



Source & Demo

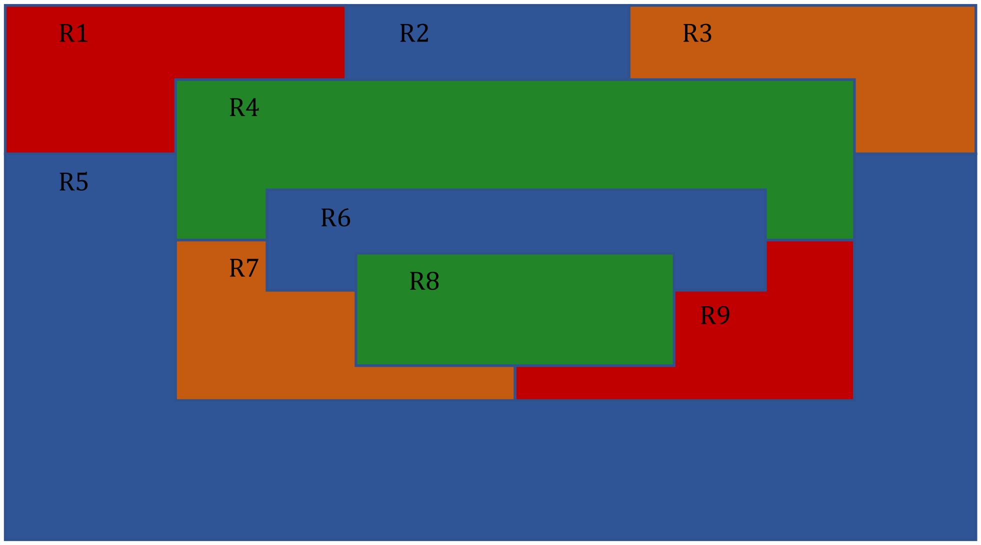


MINGW64:/c/Users/davis/iCloudDrive/School/2021 fall/344/prolog ass. 1

```
ndavis20@altair:~/prologAsgmt1$ swipl
Welcome to SWI-Prolog (threaded, 64 bits, version 8.2.4)
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('task1.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 .

?-
```

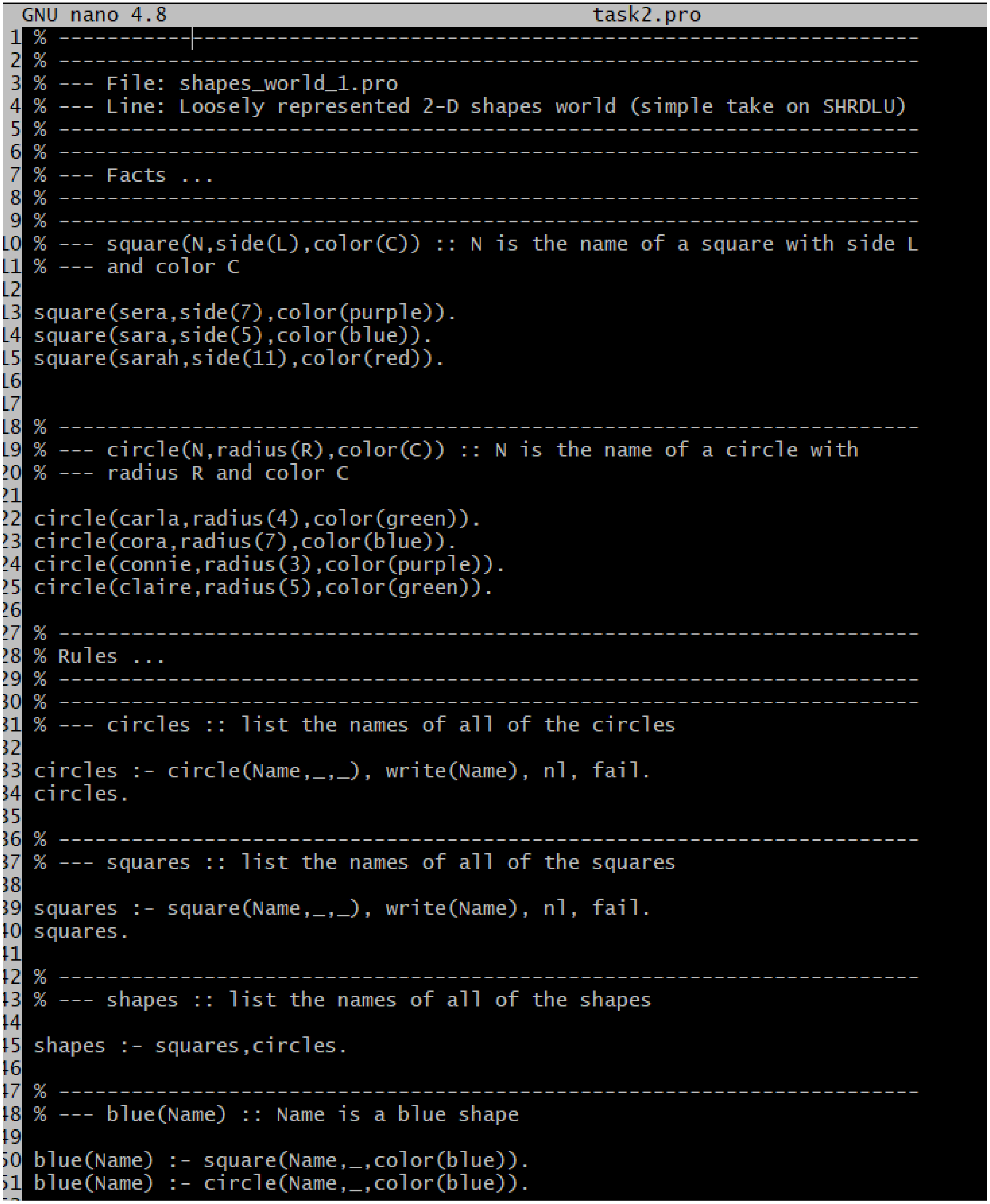
```
  GNU nano 4.8                          task1.pro
% ------------------------------------------------------------------------------
% File: task1.pro
% Line: Program to find a 4 color map rendering for South American coutries.
% More: The colors used will be red, blue, green orange.
% More: The standard abbrieviations are used to stand for the countries.
% ------------------------------------------------------------------------------


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) :: Names Squares
% represented by R1-R9 are colored so that none of the squraes
% 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).



^G Get Help    ^O Write Out   ^W Where Is    ^K Cut Text    ^J Justify     ^C Cur Pos
^X Exit        ^R Read File   ^\ Replace     ^U Paste Text  ^T To Spell    ^_ Go To Line
[0] 0:swipl*                                                        "altair" 11:15 14-Nov-21
```

Image

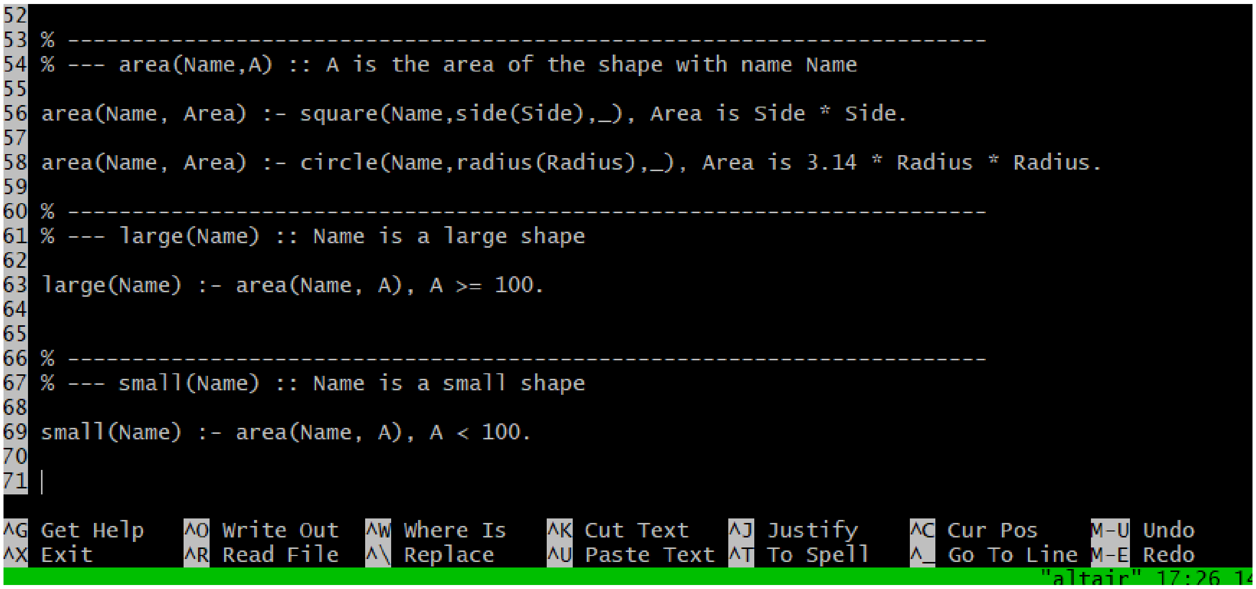


Task 2: The Floating Shapes World

Source

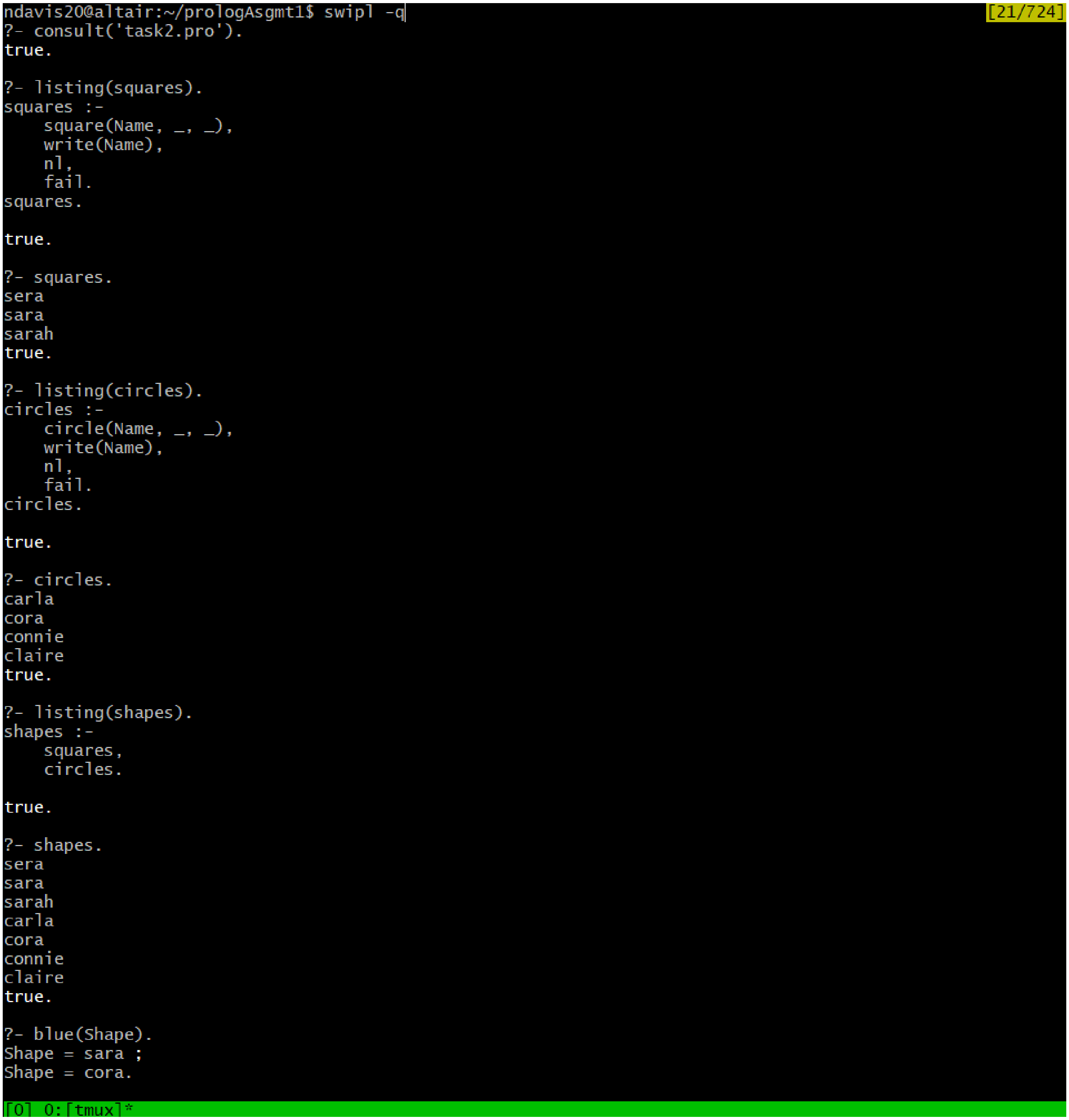


```
% ---------------------------------------------------------------------
% ---------------------------------------------------------------------
% --- File: shapes_world_1.pro
% --- Line: Loosely represented 2-D shapes world (simple take on SHRDLU)
% ---------------------------------------------------------------------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

% ---------------------------------------------------------------------
% --- shapes :: list the names of all of the shapes

shapes :- squares,circles.

% ---------------------------------------------------------------------
% --- blue(Name) :: Name is a blue shape

blue(Name) :- square(Name,_,color(blue)).
blue(Name) :- circle(Name,_,color(blue)).
```



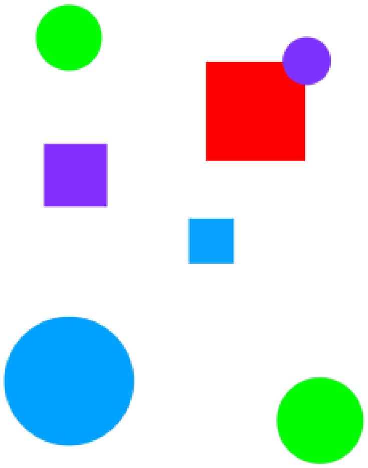
```

% -----------------------------------------------------------------------
% --- area(Name,A) :: A is the area of the shape with name Name

area(Name, Area) :- square(Name,side(Side),_), Area is Side * Side.

area(Name, Area) :- circle(Name,radius(Radius),_), Area is 3.14 * Radius * Radius.

% -----------------------------------------------------------------------
% --- 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.


```

Demo



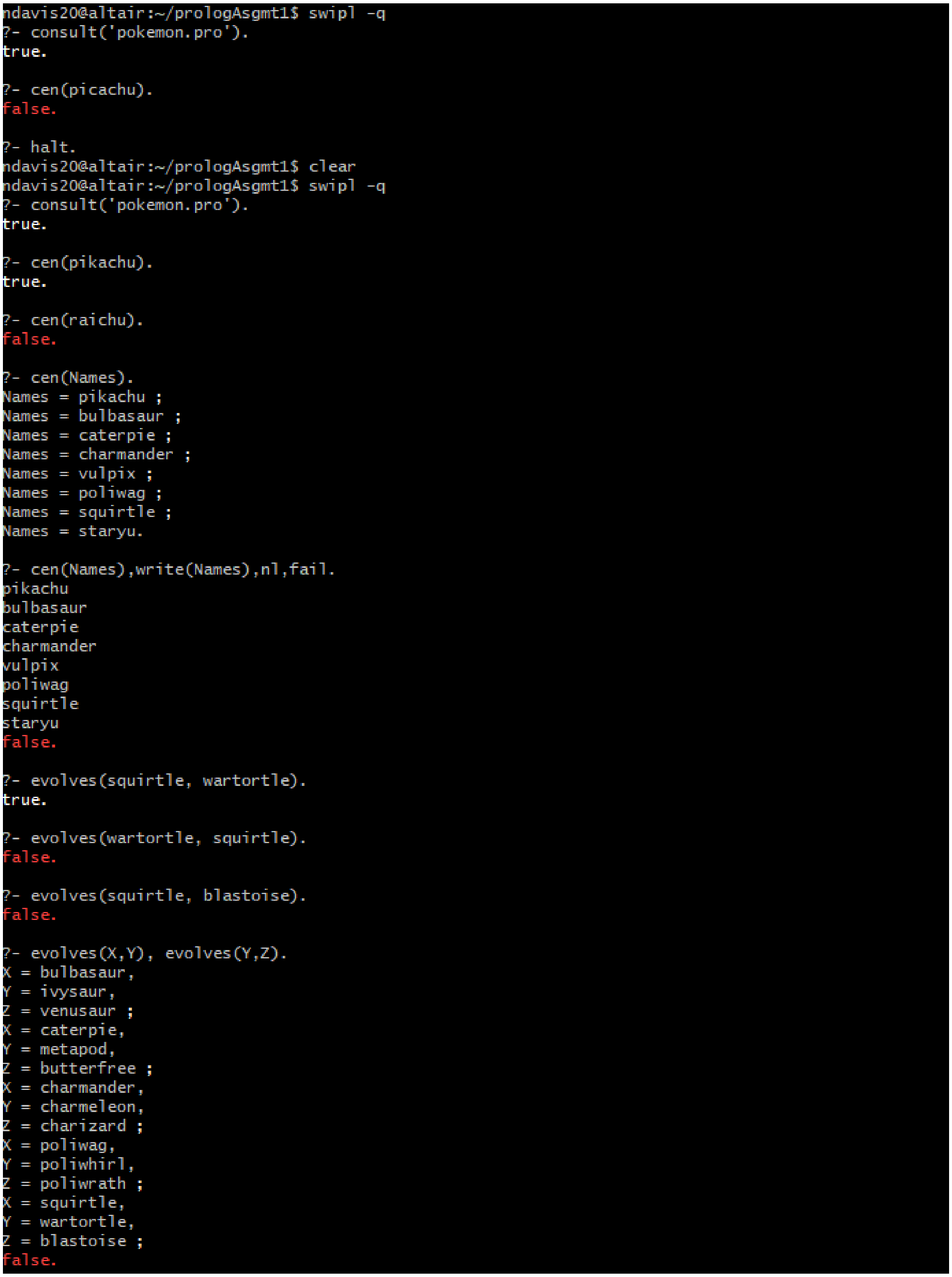
```
ndavis20@altair:~/prologAsgmt1$ swipl -q                                [21/724]
?- consult('task2.pro').
true.

?- 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 :-
    squares,
    circles.

true.

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

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

[0] 0:[tmux]*
```

Image

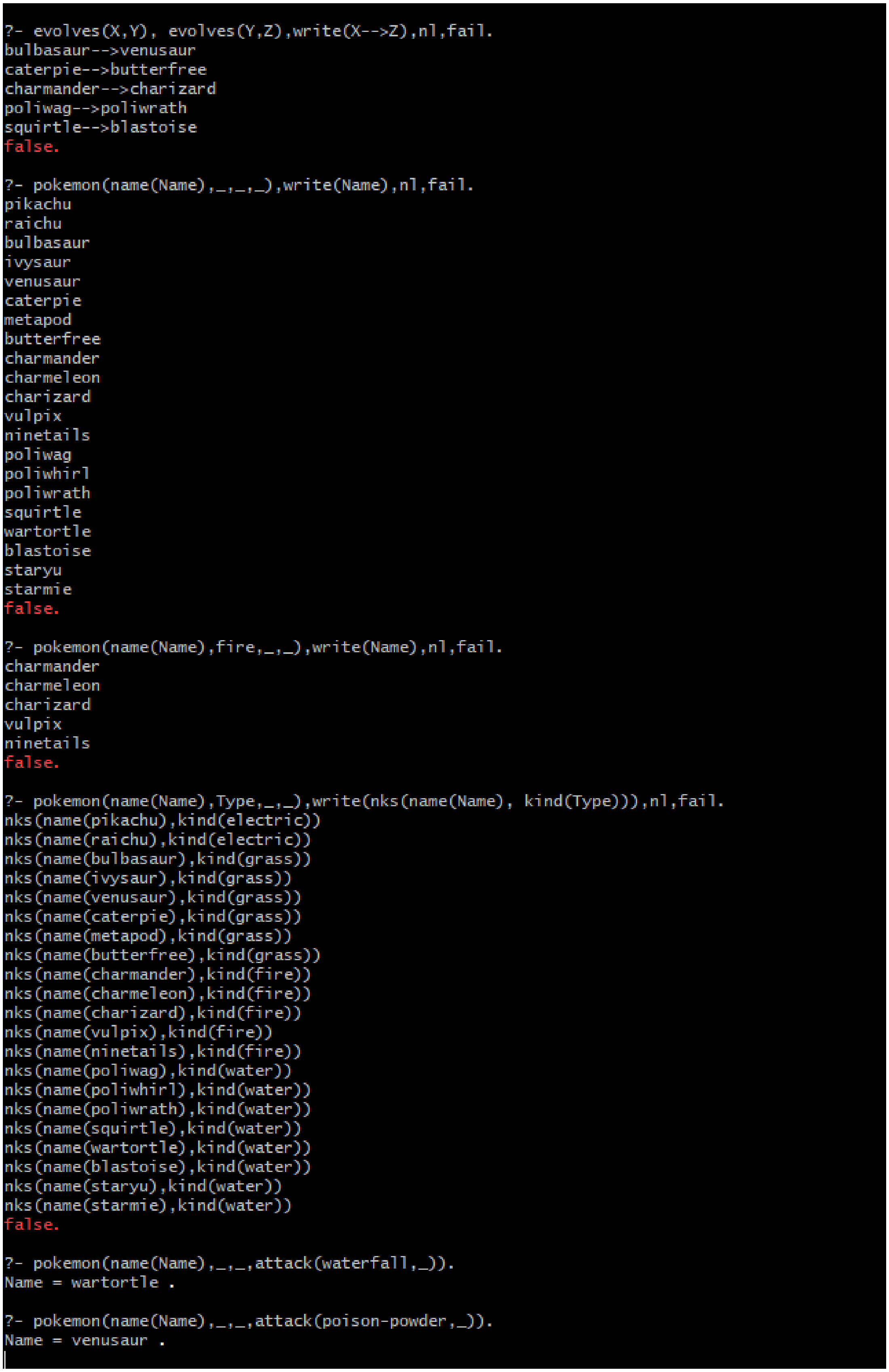


Task 3: Pokemon KB Interaction and Programming

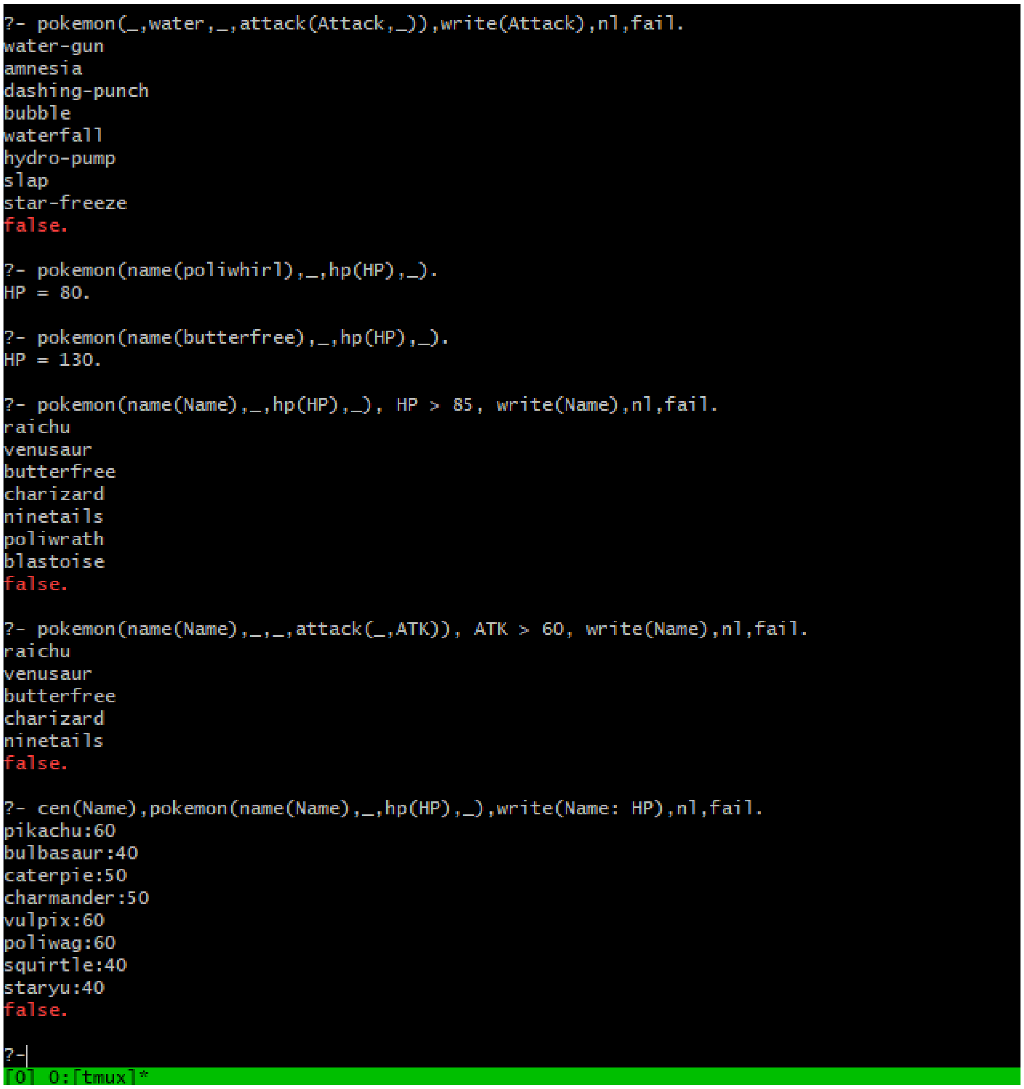
Demo pt 1



```
ndavis20@altair:~/prologAsgmt1$ swipl -q
?- consult('pokemon.pro').
true.

?- cen(picachu).
false.

?- halt.
ndavis20@altair:~/prologAsgmt1$ clear
ndavis20@altair:~/prologAsgmt1$ swipl -q
?- consult('pokemon.pro').
true.

?- cen(pikachu).
true.

?- cen(raichu).
false.

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

?- cen(Names),write(Names),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.
```

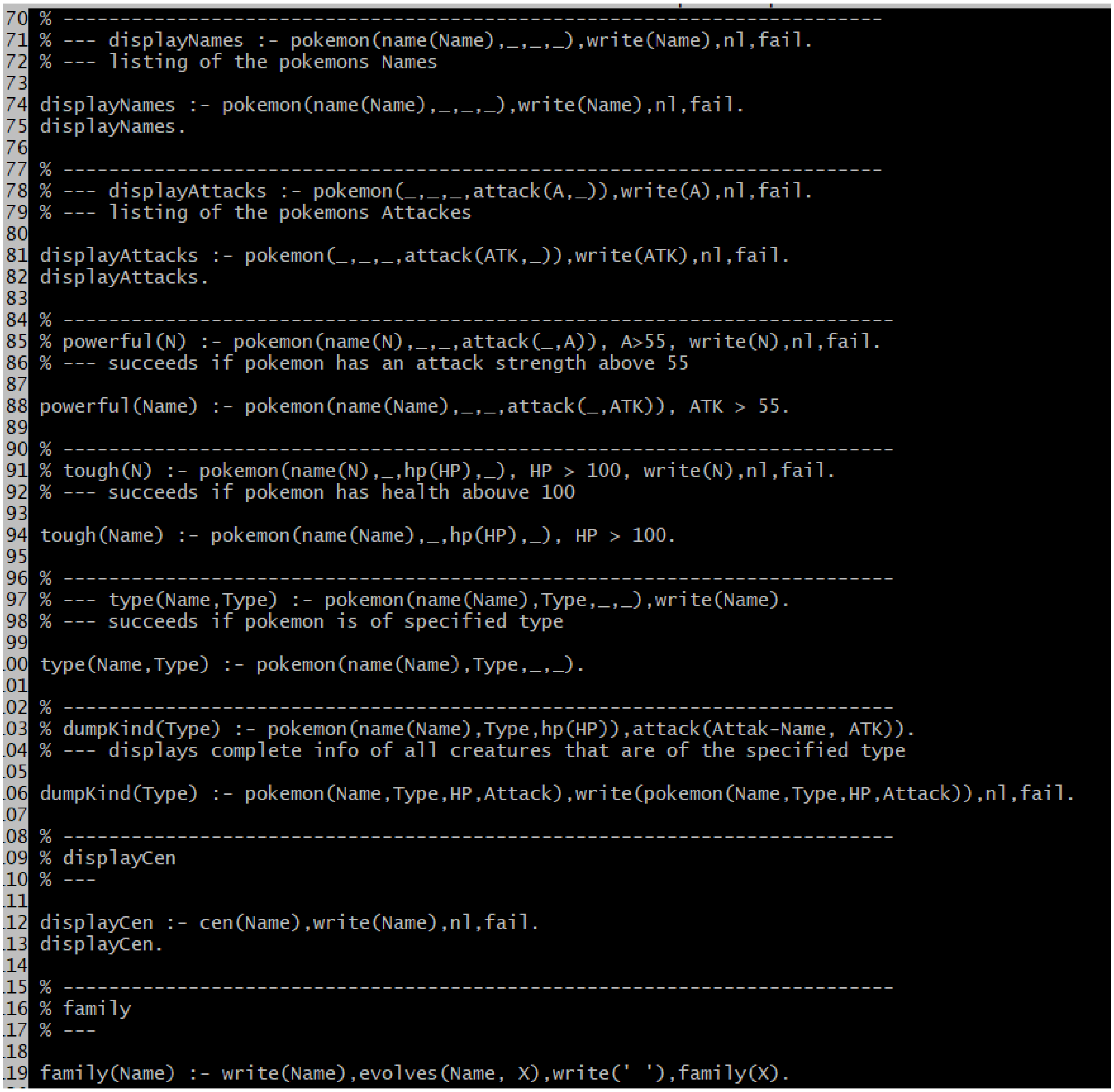


```
?- evolves(X,Y), evolves(Y,Z),write(X-->Z),nl,fail.
bulbasaur-->venusaur
caterpie-->butterfree
charmander-->charizard
poliwag-->poliwrath
squirtle-->blastoise
false.

?- 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),Type,_,_),write(nks(name(Name), kind(Type))),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(Name),_,_,attack(waterfall,_)).
Name = wartortle .

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

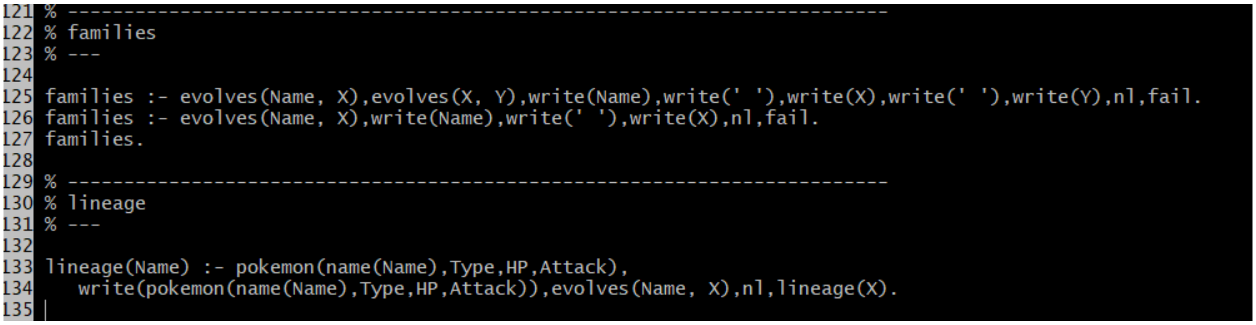


```
?- pokemon(_,water,_,attack(Attack,_)),write(Attack),nl,fail.
water-gun
amnesia
dashing-punch
bubble
waterfall
hydro-pump
slap
star-freeze
false.

?- pokemon(name(poliwhirl),_,hp(HP),_).
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(name(Name),_,_,attack(_,ATK)), ATK > 60, write(Name),nl,fail.
raichu
venusaur
butterfree
charizard
ninetails
false.

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

?-
[0] 0:[tmux]*
```

Extended Knowledge base

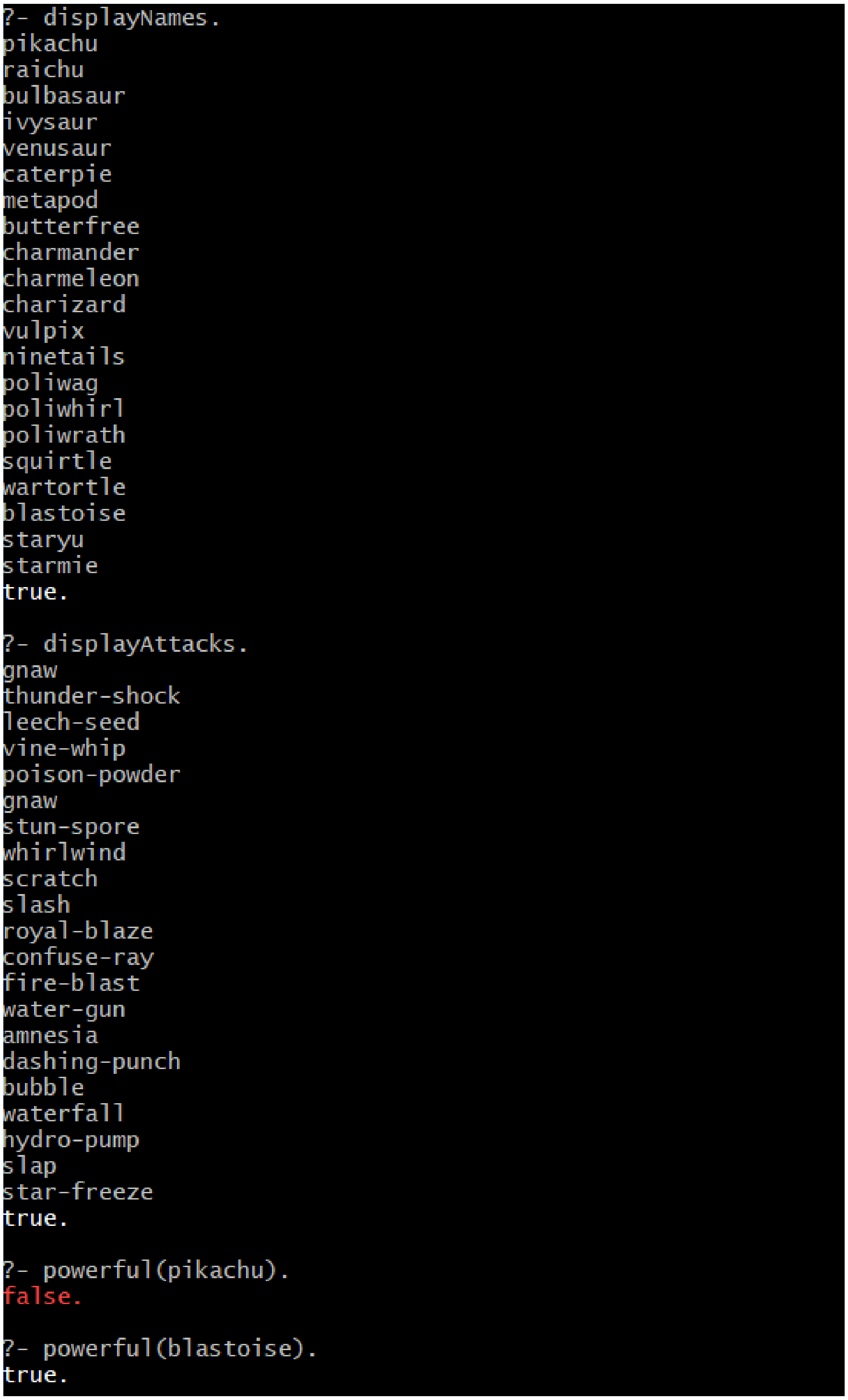


```
% ------------------------------------------------------------------------
% --- displayNames :- pokemon(name(Name),_,_,_),write(Name),nl,fail.
% --- listing of the pokemons Names

displayNames :- pokemon(name(Name),_,_,_),write(Name),nl,fail.
displayNames.

% ------------------------------------------------------------------------
% --- displayAttacks :- pokemon(_,_,_,attack(A,_)),write(A),nl,fail.
% --- listing of the pokemons Attackes

displayAttacks :- pokemon(_,_,_,attack(ATK,_)),write(ATK),nl,fail.
displayAttacks.

% -------------------------------------------------------------------------
% powerful(N) :- pokemon(name(N),_,_,attack(_,A)), A>55, write(N),nl,fail.
% --- succeeds if pokemon has an attack strength above 55

powerful(Name) :- pokemon(name(Name),_,_,attack(_,ATK)), ATK > 55.

% -------------------------------------------------------------------------
% tough(N) :- pokemon(name(N),_,hp(HP),_), HP > 100, write(N),nl,fail.
% --- succeeds if pokemon has health abouve 100

tough(Name) :- pokemon(name(Name),_,hp(HP),_), HP > 100.

% -------------------------------------------------------------------------
% --- type(Name,Type) :- pokemon(name(Name),Type,_,_),write(Name).
% --- succeeds if pokemon is of specified type

type(Name,Type) :- pokemon(name(Name),Type,_,_).

% -------------------------------------------------------------------------
% dumpKind(Type) :- pokemon(name(Name),Type,hp(HP)),attack(Attak-Name, ATK)).
% --- displays complete info of all creatures that are of the specified type

dumpKind(Type) :- pokemon(Name,Type,HP,Attack),write(pokemon(Name,Type,HP,Attack)),nl,fail.

% -------------------------------------------------------------------------
% displayCen
% ---

displayCen :- cen(Name),write(Name),nl,fail.
displayCen.

% -------------------------------------------------------------------------
% family
% ---

family(Name) :- write(Name),evolves(Name, X),write(' '),family(X).
```

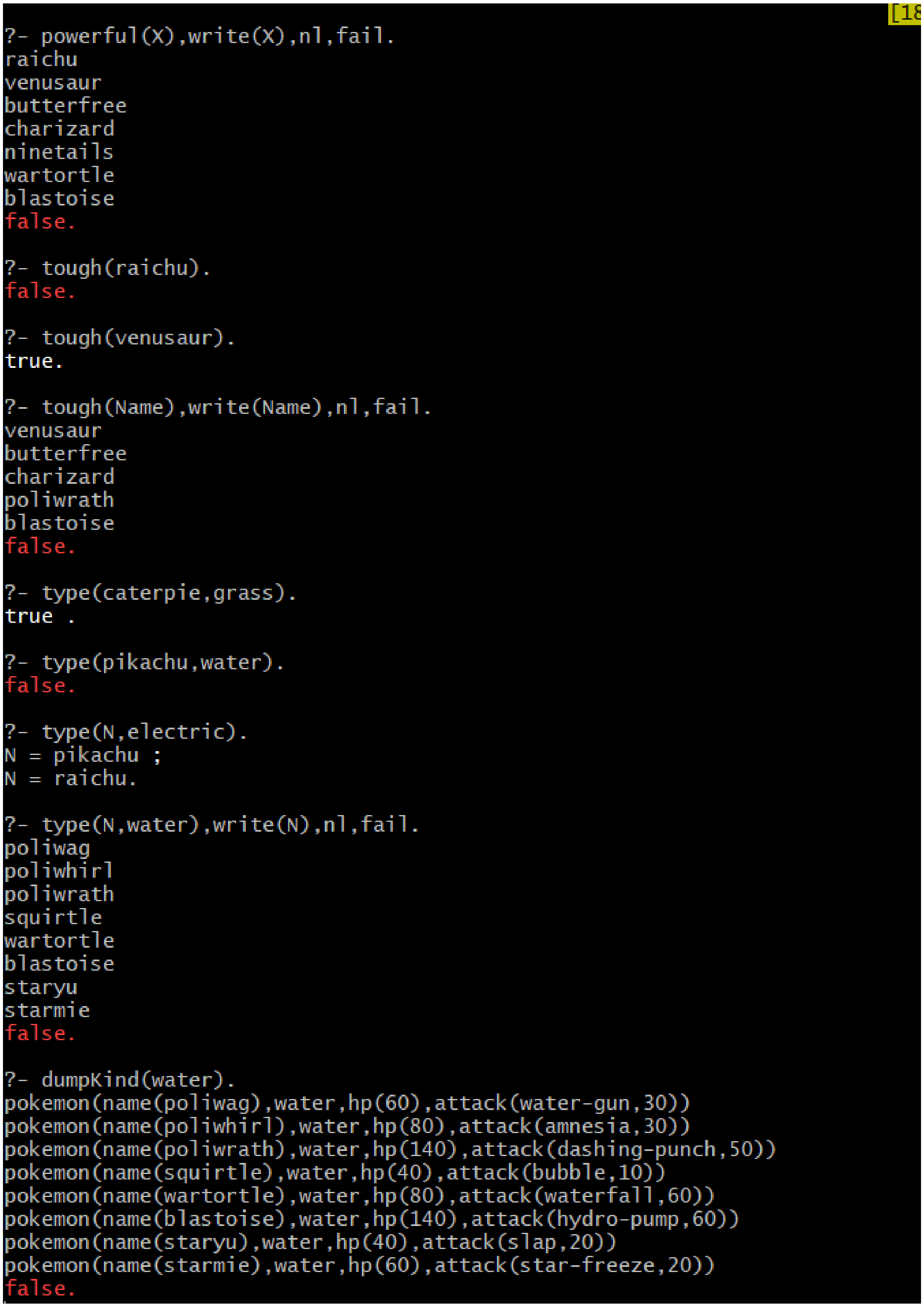


```prolog
% -------------------------------------------------------------------------
% families
% ---

families :- evolves(Name, X),evolves(X, Y),write(Name),write(' '),write(X),write(' '),write(Y),nl,fail.
families :- evolves(Name, X),write(Name),write(' '),write(X),nl,fail.
families.

% -------------------------------------------------------------------------
% lineage
% ---

lineage(Name) :- pokemon(name(Name),Type,HP,Attack),
   write(pokemon(name(Name),Type,HP,Attack)),evolves(Name, X),nl,lineage(X).

```

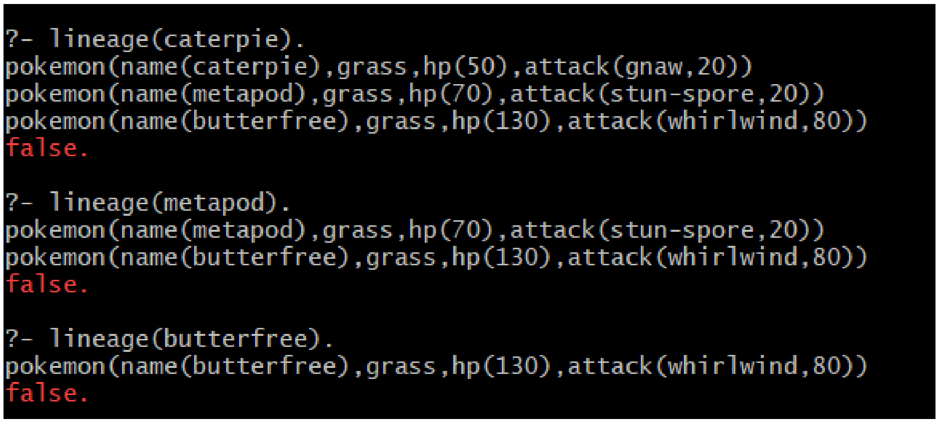
Pt 2 Demo



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

?- displayAttacks.
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.
```



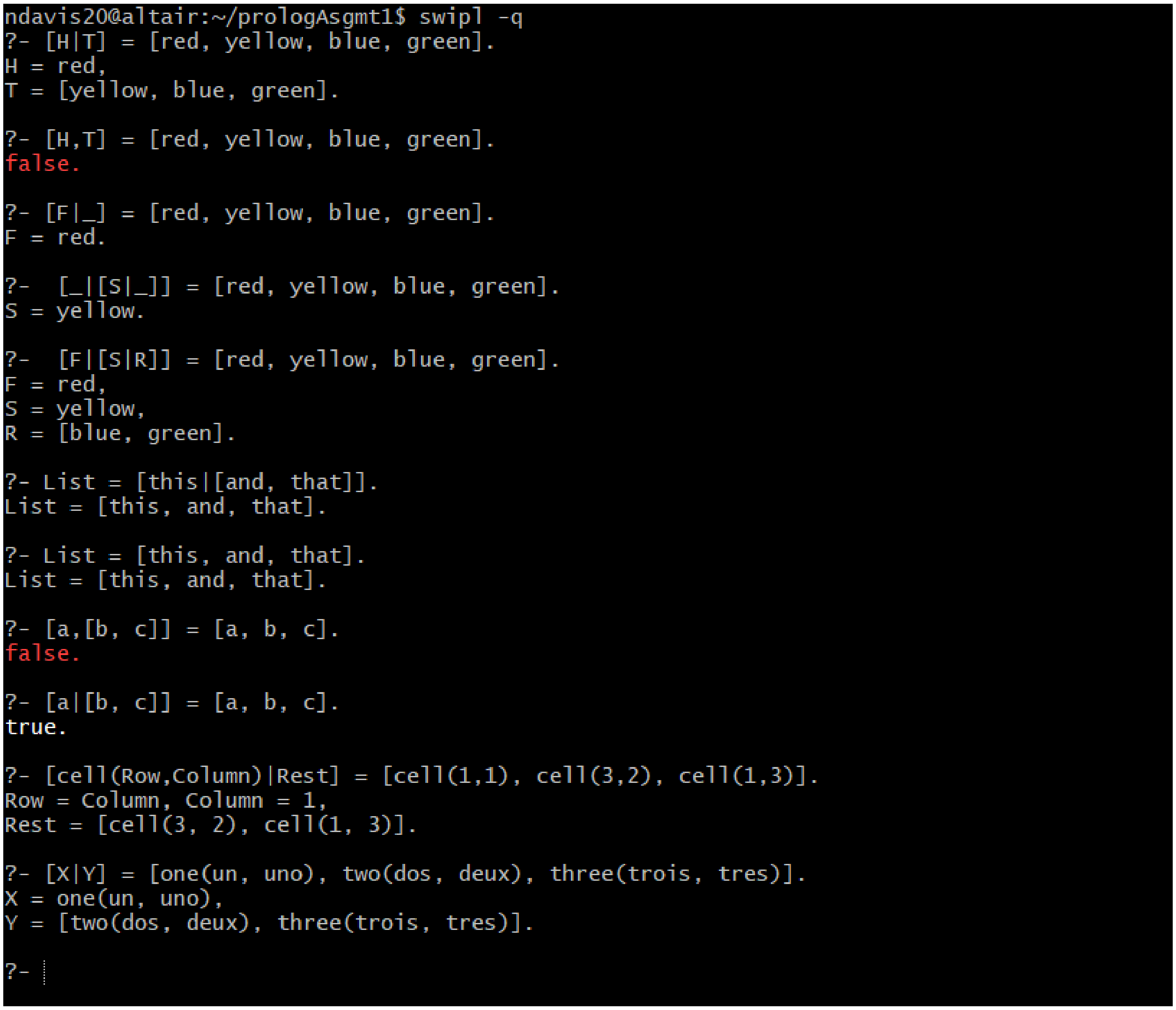
```
                                                                                [18
?- powerful(X),write(X),nl,fail.
raichu
venusaur
butterfree
charizard
ninetails
wartortle
blastoise
false.

?- 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.

?- dumpKind(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.
```



```
?- 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))
false.

?- 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.
```

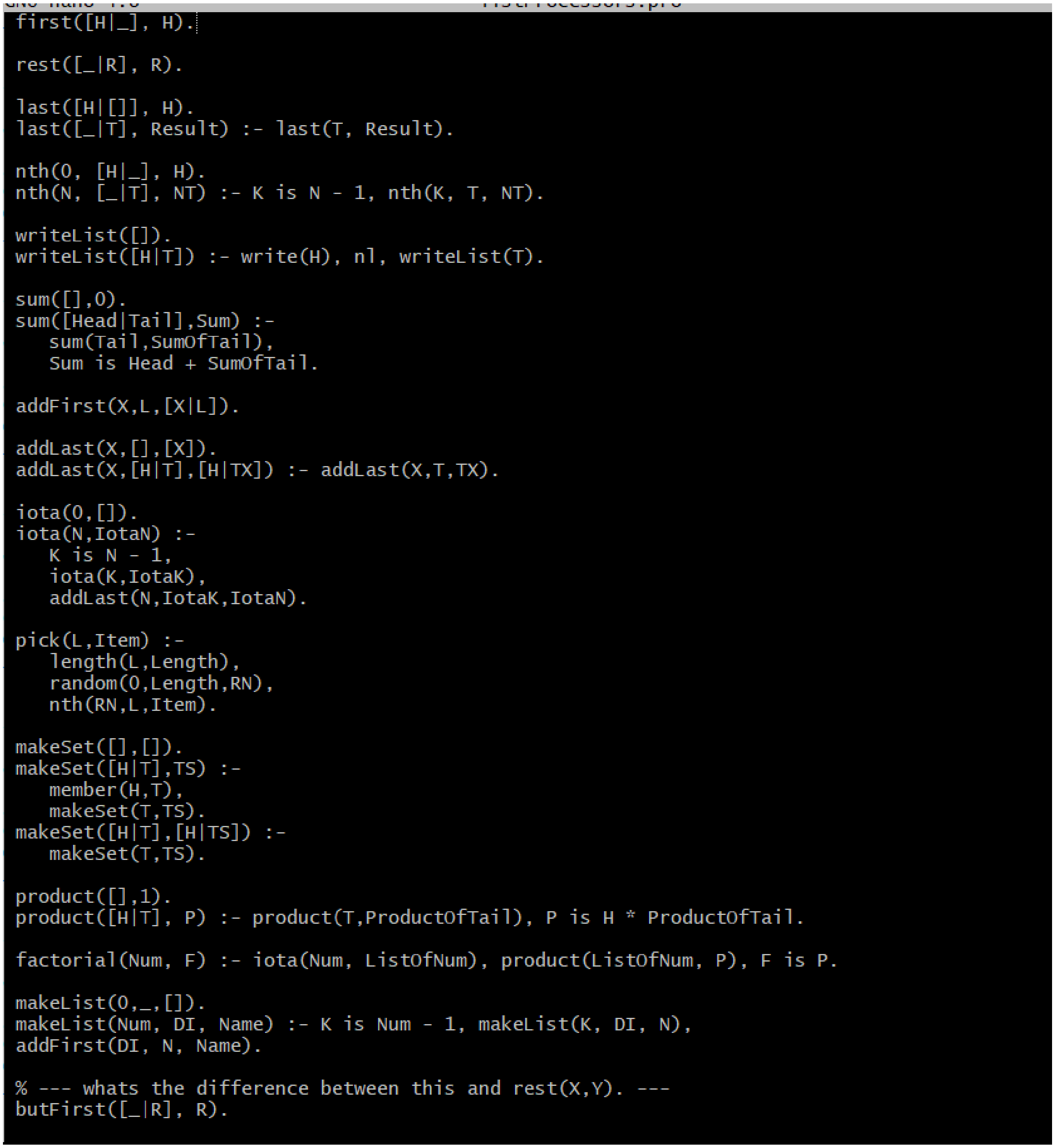
Task 4: Lisp Processing in Prolog

Head/Tail Demo

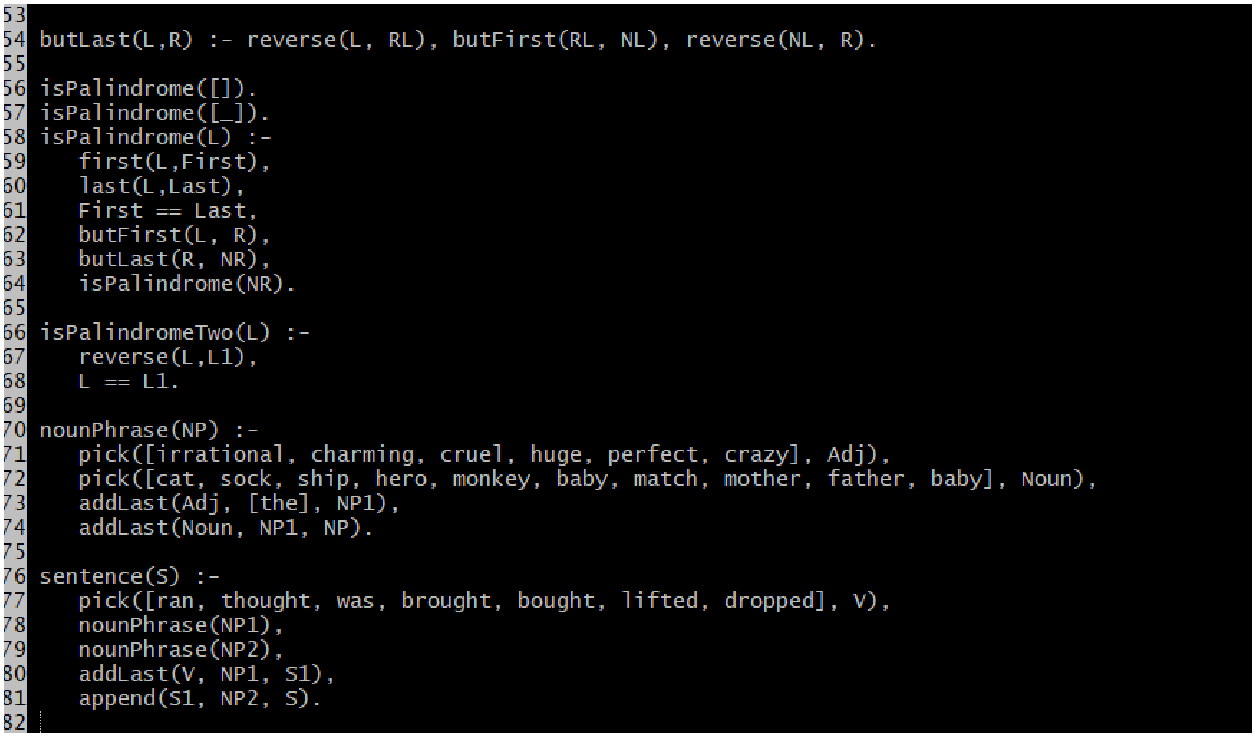


```
ndavis20@altair:~/prologAsgmt1$ swipl -q
?- [H|T] = [red, yellow, blue, green].
H = red,
T = [yellow, blue, green].

?- [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)].

?- 
```

ListProcessor.pro Source

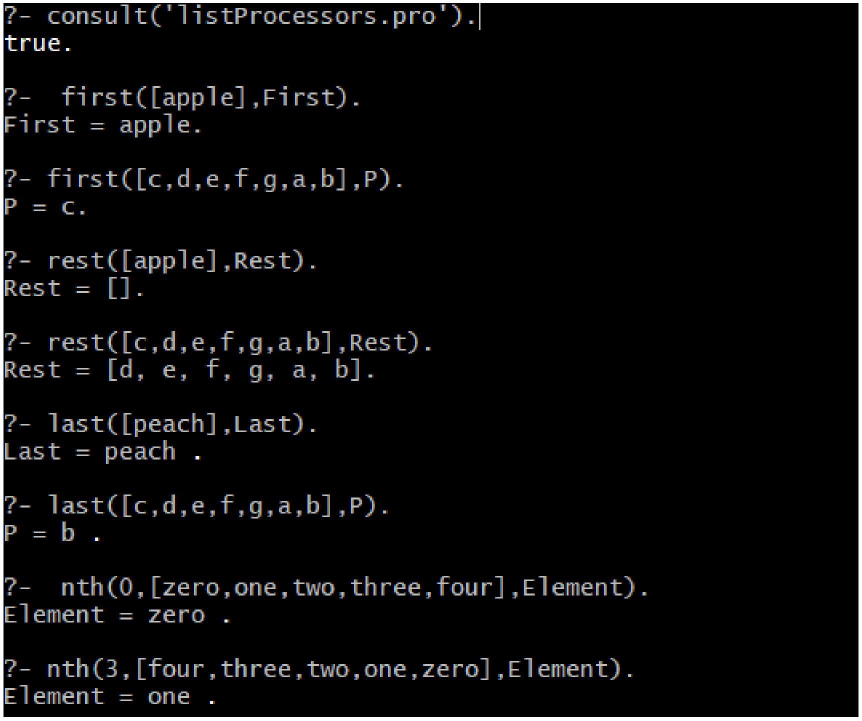


```
first([H|_], H).

rest([_|R], R).

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

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

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

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

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

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

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

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

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

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

factorial(Num, F) :- iota(Num, ListOfNum), product(ListOfNum, P), F is P.

makeList(0,_,[]).
makeList(Num, DI, Name) :- K is Num - 1, makeList(K, DI, N),
addFirst(DI, N, Name).

% --- whats the difference between this and rest(X,Y). ---
butFirst([_|R], R).
```

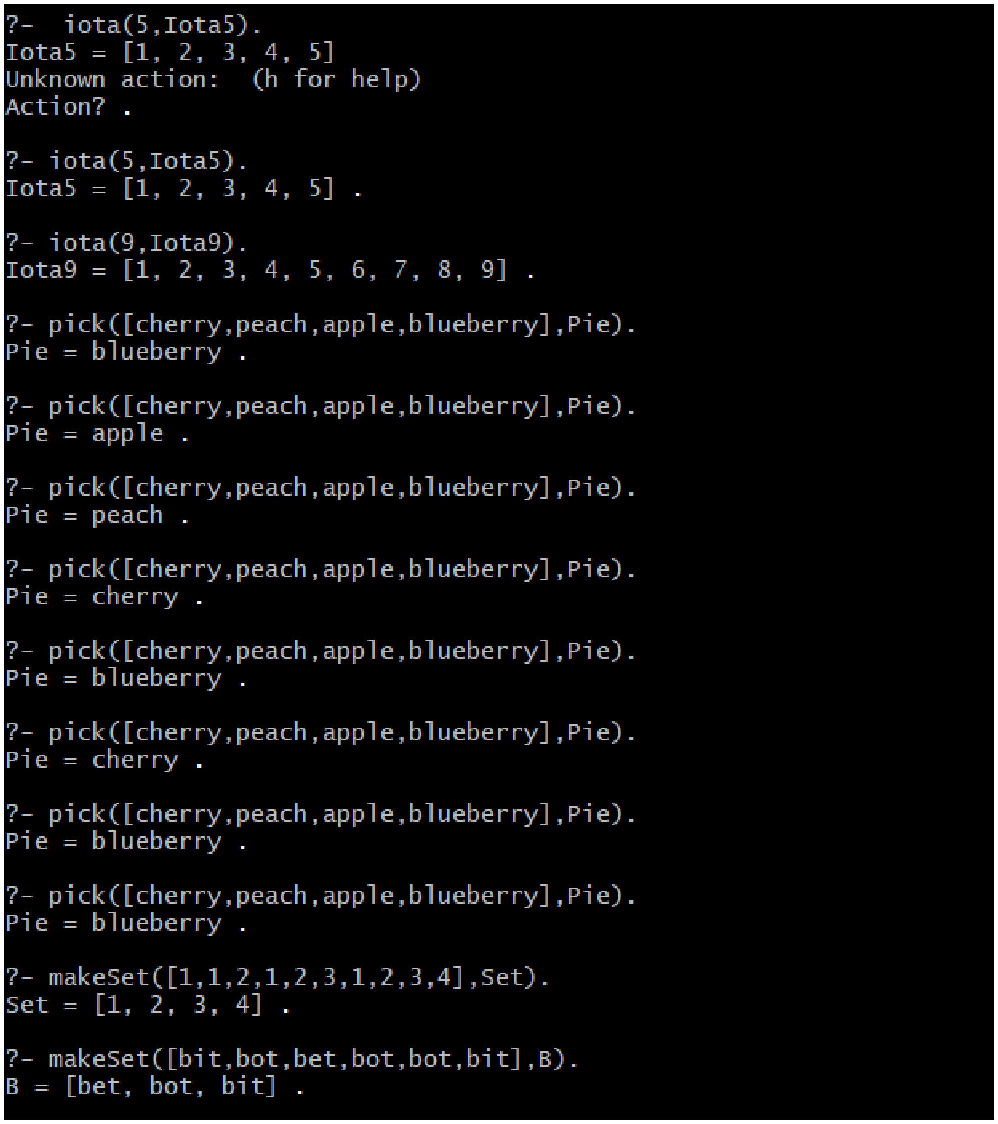


```

butLast(L,R) :- reverse(L, RL), butFirst(RL, NL), reverse(NL, R).

isPalindrome([]).
isPalindrome([_]).
isPalindrome(L) :-
   first(L,First),
   last(L,Last),
   First == Last,
   butFirst(L, R),
   butLast(R, NR),
   isPalindrome(NR).

isPalindromeTwo(L) :-
   reverse(L,L1),
   L == L1.

nounPhrase(NP) :-
   pick([irrational, charming, cruel, huge, perfect, crazy], Adj),
   pick([cat, sock, ship, hero, monkey, baby, match, mother, father, baby], Noun),
   addLast(Adj, [the], NP1),
   addLast(Noun, NP1, NP).

sentence(S) :-
   pick([ran, thought, was, brought, bought, lifted, dropped], V),
   nounPhrase(NP1),
   nounPhrase(NP2),
   addLast(V, NP1, S1),
   append(S1, NP2, S).

```

Example list processors Demo

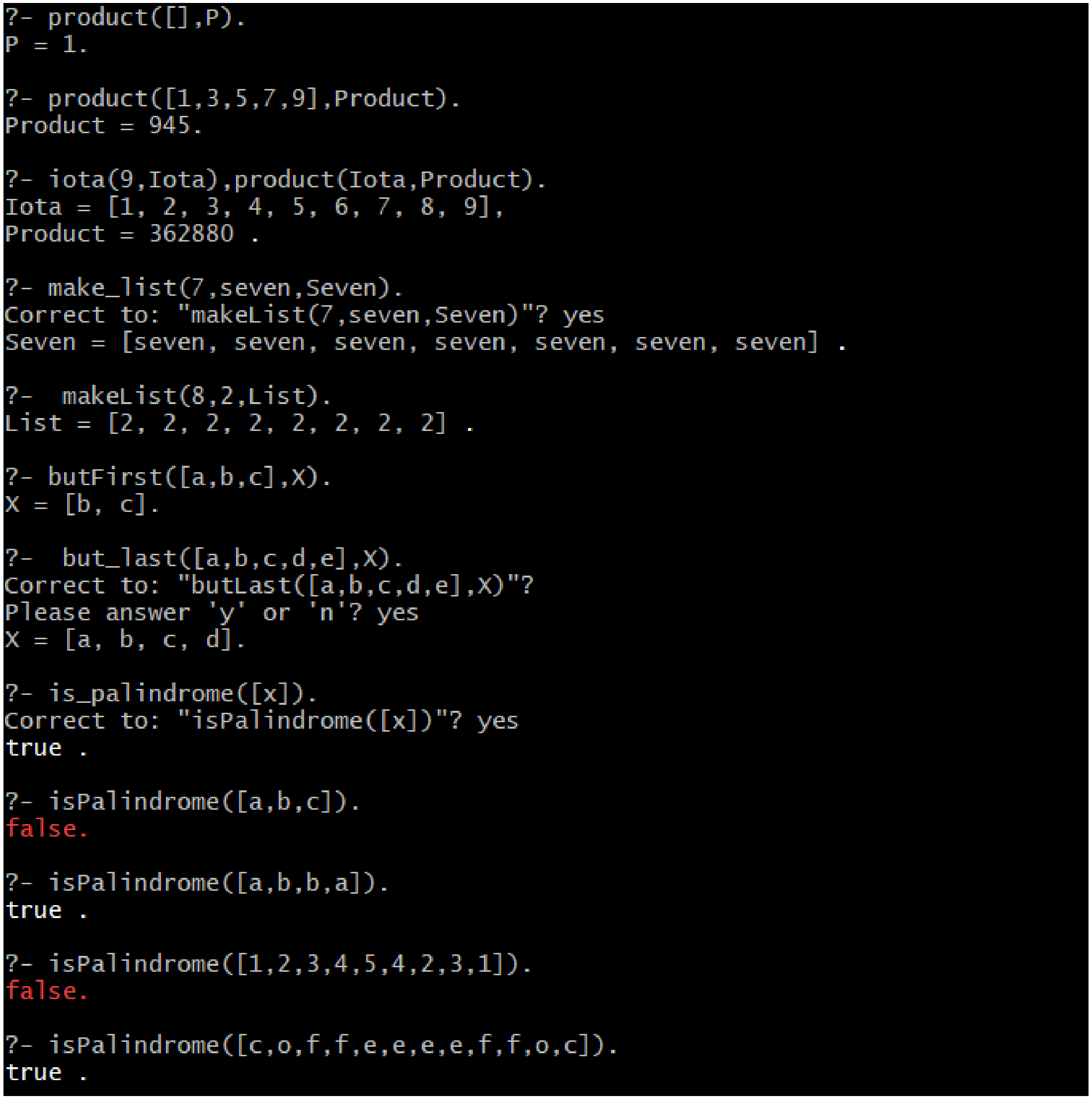


```
?- consult('listProcessors.pro').
true.

?-  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 .
```

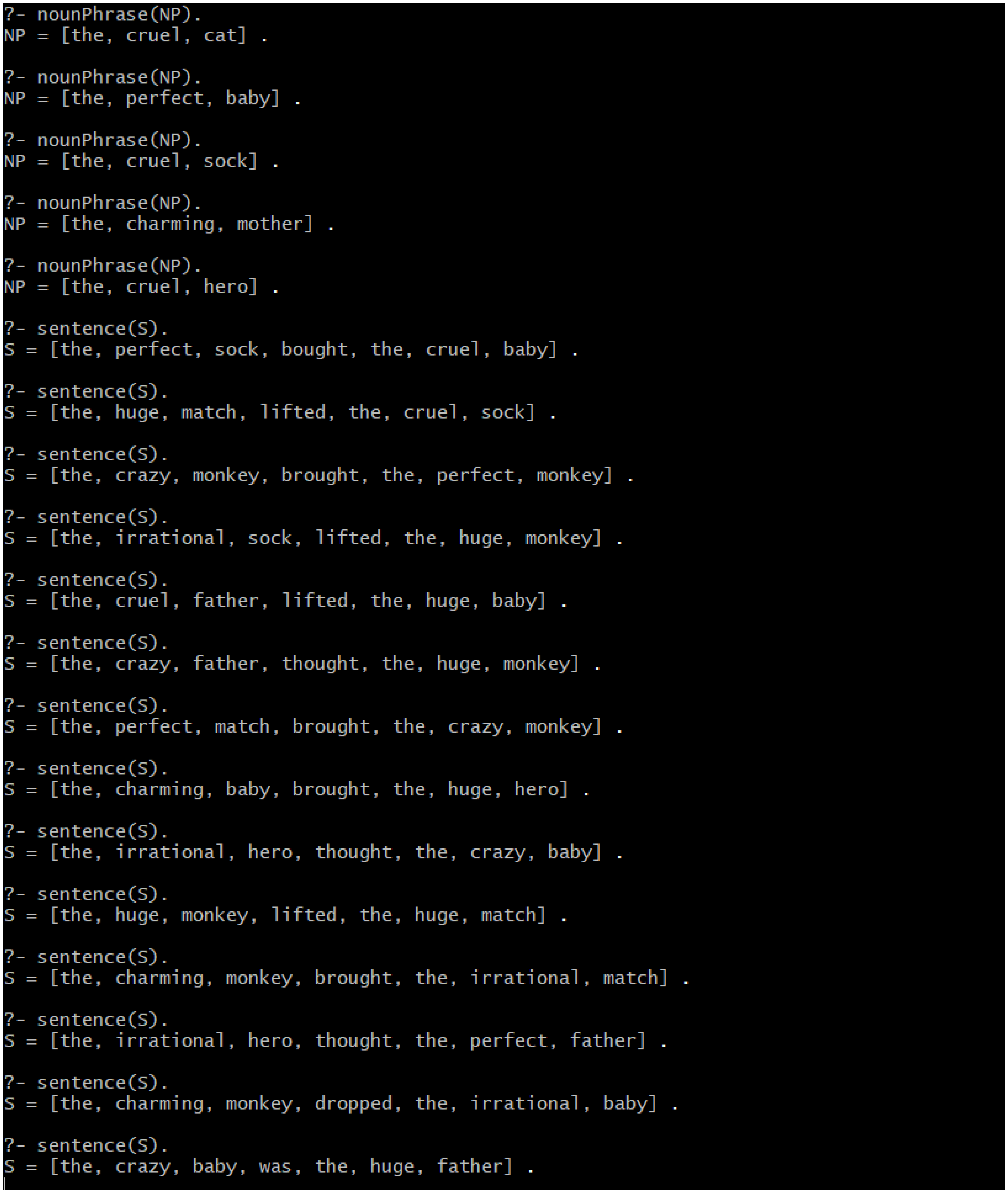


```
?-  iota(5,Iota5).
Iota5 = [1, 2, 3, 4, 5]
Unknown action:  (h for help)
Action? .

?- 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 = blueberry .

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

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

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

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

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

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

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

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

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

List processing exercises Demo



```
?- 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).
Correct to: "makeList(7,seven,Seven)"? yes
Seven = [seven, seven, seven, seven, seven, seven, seven] .

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

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

?-  but_last([a,b,c,d,e],X).
Correct to: "butLast([a,b,c,d,e],X)"?
Please answer 'y' or 'n'? yes
X = [a, b, c, d].

?- is_palindrome([x]).
Correct to: "isPalindrome([x])"? yes
true .

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

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

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

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



```
?- nounPhrase(NP).
NP = [the, cruel, cat] .

?- nounPhrase(NP).
NP = [the, perfect, baby] .

?- nounPhrase(NP).
NP = [the, cruel, sock] .

?- nounPhrase(NP).
NP = [the, charming, mother] .

?- nounPhrase(NP).
NP = [the, cruel, hero] .

?- sentence(S).
S = [the, perfect, sock, bought, the, cruel, baby] .

?- sentence(S).
S = [the, huge, match, lifted, the, cruel, sock] .

?- sentence(S).
S = [the, crazy, monkey, brought, the, perfect, monkey] .

?- sentence(S).
S = [the, irrational, sock, lifted, the, huge, monkey] .

?- sentence(S).
S = [the, cruel, father, lifted, the, huge, baby] .

?- sentence(S).
S = [the, crazy, father, thought, the, huge, monkey] .

?- sentence(S).
S = [the, perfect, match, brought, the, crazy, monkey] .

?- sentence(S).
S = [the, charming, baby, brought, the, huge, hero] .

?- sentence(S).
S = [the, irrational, hero, thought, the, crazy, baby] .

?- sentence(S).
S = [the, huge, monkey, lifted, the, huge, match] .

?- sentence(S).
S = [the, charming, monkey, brought, the, irrational, match] .

?- sentence(S).
S = [the, irrational, hero, thought, the, perfect, father] .

?- sentence(S).
S = [the, charming, monkey, dropped, the, irrational, baby] .

?- sentence(S).
S = [the, crazy, baby, was, the, huge, father] .
```