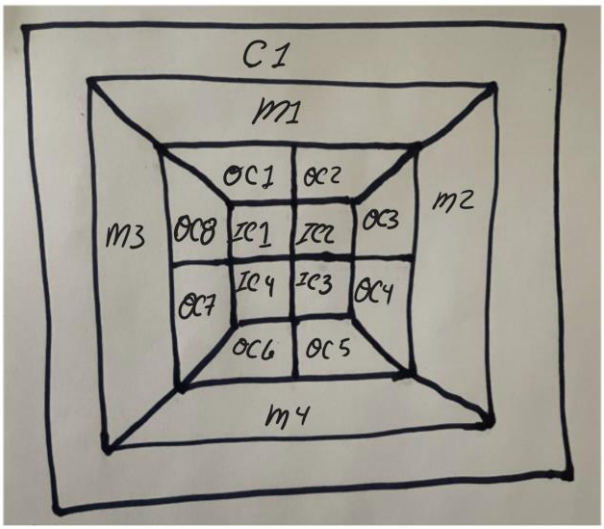
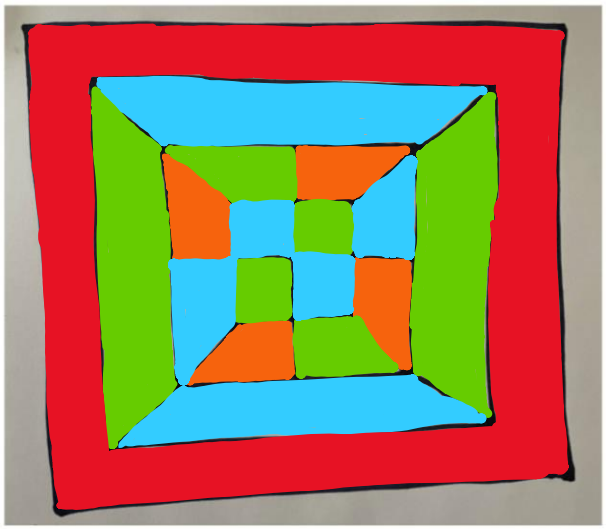
Prolog Programming Assignment #1: Various Computations

Learning Abstract

This assignment is our introduction to Prolog. This assignment encompasses querying the KB, introduction to defining facts and rules, list processing and recursion.

Task 1: Map Coloring





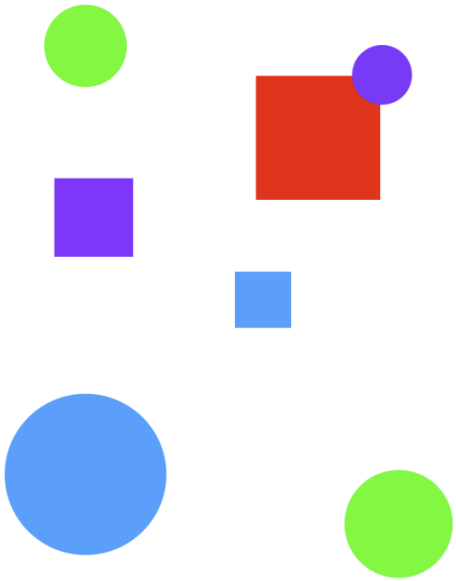
```
:-
% c:/Users/stick/Documents/Prolog/map_coloring.pro compiled 0.00 sec, 13 clauses
?-
|    coloring(C1,M1,M2,M3,M4,OC1,OC2,OC3,OC4,OC5,OC6,OC7,OC8,IC1,IC2,IC3,IC4).
C1 = red,
M1 = M4, M4 = OC3, OC3 = OC7, OC7 = IC1, IC1 = IC3, IC3 = blue,
M2 = M3, M3 = OC1, OC1 = OC5, OC5 = IC2, IC2 = IC4, IC4 = green,
OC2 = OC4, OC4 = OC6, OC6 = OC8, OC8 = orange ;
```

```
%-----------------------------------------
%Task 1: CSC344 Map Coloring
%-----------------------------------------

% ------------------------------------------------------------------------
% 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(C1,M1,M2,M3,M4,OC1,OC2,OC3,OC4,OC5,OC6,OC7,OC8,IC1,IC2,IC3,IC4) :-
different(C1, M1),
different(C1, M2),
different(C1, M3),
different(C1, M4),

different(M1, M2),
different(M1, M3),
different(M1, OC1),
different(M1, OC2),
different(M1, C1),

different(M2, M1),
different(M2, M4),
different(M2, OC3),
different(M2, OC4),

different(M3, M1),
different(M3, M4),
different(M3, OC7),
different(M3, OC8),

different(M4, M3),
different(M4, M2),
different(M4, OC5),
different(M4, OC6),

different(OC1, M1),
different(OC1, OC2),
different(OC1, IC1),
%different(OC1, IC2),
different(OC1, OC8),

different(OC2, M1),
different(OC2, OC1),
%different(OC2, IC1),
different(OC2, IC2),
different(OC2, OC3),

different(OC3, M2),
different(OC3, OC2),
different(OC3, IC2),
%different(OC3, IC3),
different(OC3, OC4),

different(OC4, M2),
different(OC4, OC3),
%different(OC4, IC2),
different(OC4, IC3),
```

```
different(OC4, OC5),

different(OC5, M4),
different(OC5, OC4),
different(OC5, IC3),
%different(OC5, IC4),
different(OC5, OC6),

different(OC6, M4),
different(OC6, OC5),
%different(OC6, IC3),
different(OC6, IC4),
different(OC6, OC7),

different(OC7, M3),
different(OC7, OC6),
different(OC7, IC4),
%different(OC7, IC1),
different(OC7, OC8),

different(OC8, M3),
different(OC8, OC7),
%different(OC8, IC4),
different(OC8, IC1),
different(OC8, OC1),

different(IC1, OC1),
%different(IC1, OC2),
different(IC1, IC2),
%different(IC1, IC3),
different(IC1, IC4),
%different(IC1, OC7),
different(IC1, OC8),

different(IC2, OC2),
different(IC2, OC3),
%different(IC2, OC4),
different(IC2, IC3),
%different(IC2, IC4),
different(IC2, IC1),
%different(IC2, OC1),

%different(IC3, OC3),
different(IC3, OC4),
different(IC3, OC5),
%different(IC3, OC6),
different(IC3, IC4),
%different(IC3, IC1),
different(IC3, IC2),

different(IC4, OC6),
different(IC4, OC7),
%different(IC4, OC8),
different(IC4, IC1),
%different(IC4, IC2),
different(IC4, IC3).
%different(IC4, OC5).
```

Task 2: The Floating Shapes World



```
% ------------------------------------------------------------------------
% ------------------------------------------------------------------------
% --- 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.
```

```
% c:/Users/stick/Documents/Prolog/shapes_world_1.pro compiled 0.00 sec, 18 clauses
?-
|    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

Part 1:

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

% ------------------------------------------------------------------------
% appened program to task 2b specifcations
% ------------------------------------------------------------------------

display_names :-
    pokemon(name(N),_,_,_),
    write(N), nl,
    fail.

display_attacks :-
    pokemon(_,_,_,attack(A,_)),
    write(A), nl,
    fail.

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

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

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

dump_kind(Type) :-
    listing(pokemon(_,Type,_,_)).

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

family(N1) :-
    write(N1), write(" "),
    evolves(N1,N2),
    write(N2), write(" "),
    evolves(N2,N3),
    write(N3),nl,fail.

families :-
    cen(N1),
    write(N1), write(" "),
    evolves(N1,N2),
    write(N2), write(" "),nl,
    evolves(N2,N3),
    write(N3),nl,
    fail,
    families.

lineage(N1) :-
    pokemon(name(N1),_,_,_),
    listing(pokemon(name(N1),_,_,_)),
    evolves(N1,N2),
    listing(pokemon(name(N2),_,_,_)),
    evolves(N2,N3),
    listing(pokemon(name(N3),_,_,_)).
```

```
% c:/Users/stick/Documents/Prolog/pokemon.pro compiled 0.00 sec, 42 clauses
?- 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(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),fire,_,_),write(nks(name(pikachu),kind(electric))),nl,fail.
nks(name(pikachu),kind(electric))
nks(name(pikachu),kind(electric))
nks(name(pikachu),kind(electric))
nks(name(pikachu),kind(electric))
nks(name(pikachu),kind(electric))
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 ;
false.

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

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

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

```
?- 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:

```
% c:/Users/stick/Documents/Prolog/pokemon.pro compiled 0.00 sec, 50 clauses
?- 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)).

true.

?- 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)).

true.
```

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

?- family(pikachu).
pikachu raichu
false.

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


?- 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: List Processing in Prolog

```
?-  [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)].

?-
```

```
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)
%-----------------------------------------------------------
```

```
% c:/Users/stick/Documents/Prolog/list_processors.pro compiled 0.00 sec, 19 clauses
?- 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
Unknown action: 0 (h for help)
Action?
Unknown action: 0 (h for help)
Action? ,

?- 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 = apple ,

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

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

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

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

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

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

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

?- 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] ,

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

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

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

?-
```

```
%-----------------------------------------------
product([],0).
product([H|T],P) :-
    product(T,PT),
    P is H * PT.

factorial(0,1).
factorial(N,F) :-
    iota(N,IotaN),
    product(IotaN,F),
    N is F.

make_list(0,[]).
make_list(N,[N|T]) :-
    K is N - 1,
    make_list(K,T).

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

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

isPalindrome([]).
isPalindrome([_]).
isPalindrome(L) :-
    first(L,First),
    last(L,Last),
    First = Last,
    but_first(L,BF),
    but_last(BF,BL),
    isPalindrome(BL).
```

```
noun_phrase(P) :-
    pick([graceful,
          possible,
          dashing,
          delightful,
          wanting,
          imaginary],Adjective),
    pick([menu,
          establishment,
          refrigerator,
          procedure,
          selection,
          employer,
          driver,
          community],Noun),
    add_last(Adjective,[the],P1),
    add_last(Noun,P1,P).

sentence(S) :-
    noun_phrase(NP1),
    noun_phrase(NP2),
    pick([arranged,
          attached,
          allowed,
          extended,
          remarked,
          planted,
          revised],Verb),
    add_last(Verb,NP1,S1),
    append(S1,NP2,S).
```

```
% c:/users/stick/documents/prolog/list_processors.pro compiled 0.02 sec, 0 clauses
?- product([],P).
P = 1.

?- product([1,3,5,7,9],P).
P = 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_first([a,b,c,d,e],X).
X = [b, c, d, e].

?- 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, dashing, menu] ;
false.

?- noun_phrase(NP).
NP = [the, possible, establishment] .

?- noun_phrase(NP).
NP = [the, possible, selection] .

?- noun_phrase(NP).
NP = [the, delightful, employer] .

?- noun_phrase(NP).
NP = [the, dashing, menu] .

?- noun_phrase(NP).
NP = [the, imaginary, employer] .
```

```
?- sentence(S).
S = [the, graceful, selection, extended, the, graceful, refrigerator] ;

?- sentence(S).
S = [the, possible, community, planted, the, possible, community] ;

?- sentence(S).
S = [the, imaginary, procedure, planted, the, wanting, refrigerator] ;

?- sentence(S).
S = [the, dashing, community, attached, the, graceful, menu] ;

?- sentence(S).
S = [the, possible, driver, revised, the, wanting, driver] ;

?- sentence(S).
S = [the, possible, selection, planted, the, graceful, establishment] ;

?- sentence(S).
S = [the, graceful, driver, arranged, the, graceful, employer] ;

?- sentence(S).
S = [the, imaginary, menu, allowed, the, imaginary, community] ;

?- sentence(S).
S = [the, dashing, community, revised, the, possible, establishment] ;

?- sentence(S).
S = [the, graceful, establishment, allowed, the, possible, procedure] ;

?- sentence(S).
S = [the, dashing, procedure, allowed, the, possible, refrigerator] ;

?- sentence(S).
S = [the, delightful, establishment, planted, the, imaginary, driver] ;

?- sentence(S).
S = [the, graceful, driver, revised, the, graceful, employer] ;

?-
|    sentence(S).
S = [the, dashing, menu, revised, the, dashing, employer] ;

?- sentence(S).
S = [the, graceful, procedure, attached, the, graceful, establishment] ;

?- sentence(S).
S = [the, imaginary, community, allowed, the, graceful, driver] ;

?-
```