

---

## CSC 344 First Prolog Programming Assignment Solution

---

**Learning Abstract:** This programming assignment features some basic Prolog capabilities such as knowledge bases and list processing. There is also an interesting visual representation of the four color theorem which can be solved using Prolog.

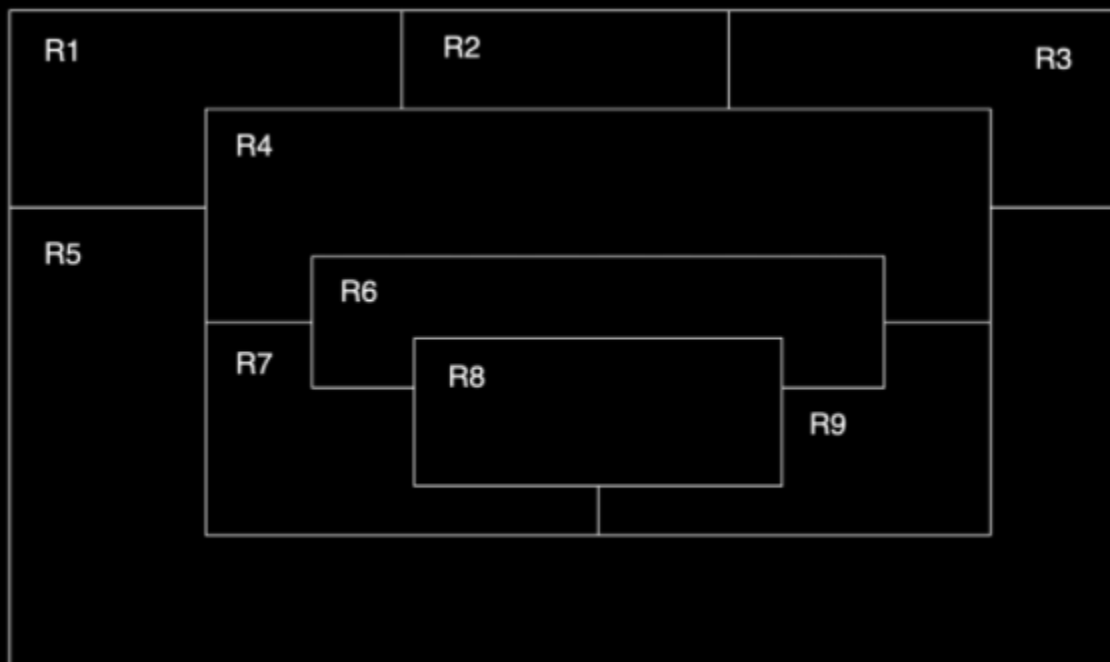
---

### First Task: Map Coloring

---

> Given Map

---



---

## > Code

---

```
% -----
% File: map_coloring.pro
% Line: Program to find a 4 color map rendering for the given image.
% More: The colors used will be red, blue, green, and orange.
% More: The standard abbreviations are used to stand for the regions.

% -----
% different(X,Y) :: X is not equal to Y

different(red,blue) .
different(red,green) .
different(red,orange) .
different(green,blue) .
different(green,orange) .
different(green,red) .
different(blue,green) .
different(blue,orange) .
different(blue,red) .
different(orange,blue) .
different(orange,green) .
different(orange,red) .

% -----
% coloring(R1,R2,R3,R4,R5,R6,R7,R8,R9) :: The regions represented by their
% standard abbreviations are colored so that none of the regions 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) .
```

---

> Demo

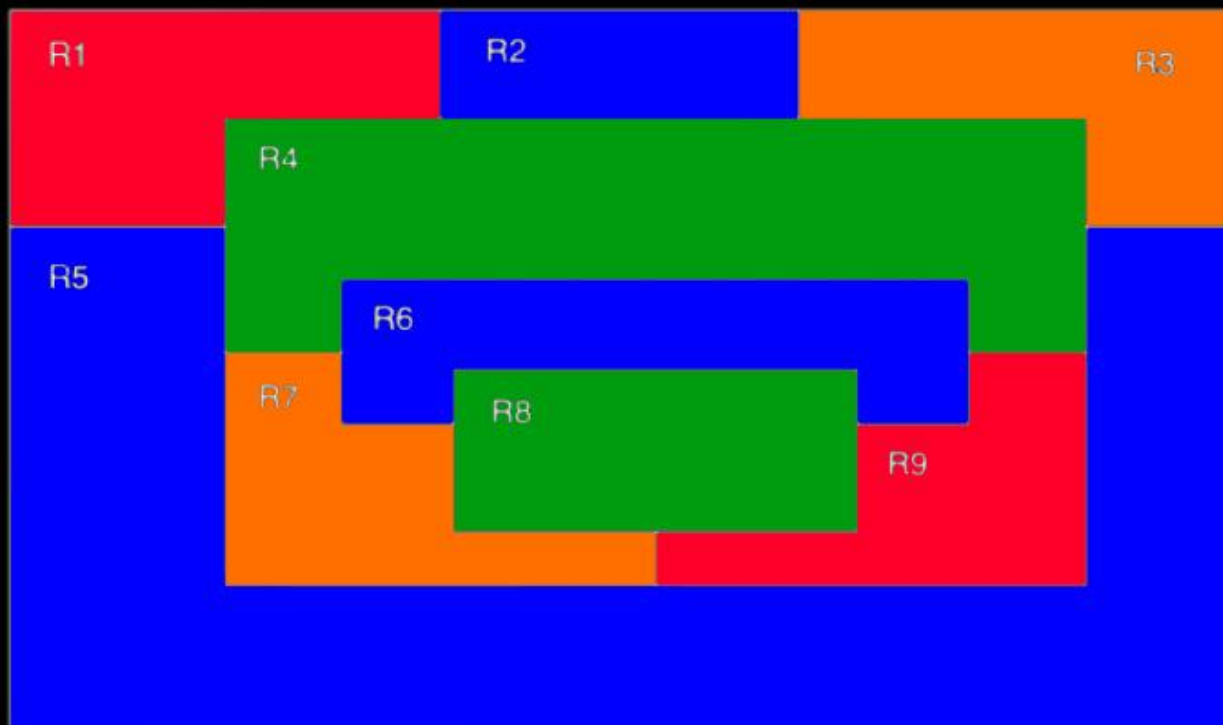
---

```
?- consult('C:/Users/godde/Downloads/CSC344/map_coloring.pro').  
true.  
  
?- coloring(R1,R2,R3,R4,R5,R6,R7,R8,R9).  
R1 = R9, R9 = red,  
R2 = R5, R5 = R6, R6 = blue,  
R3 = R7, R7 = orange,  
R4 = R8, R8 = green ,
```

---

> Final Image

---

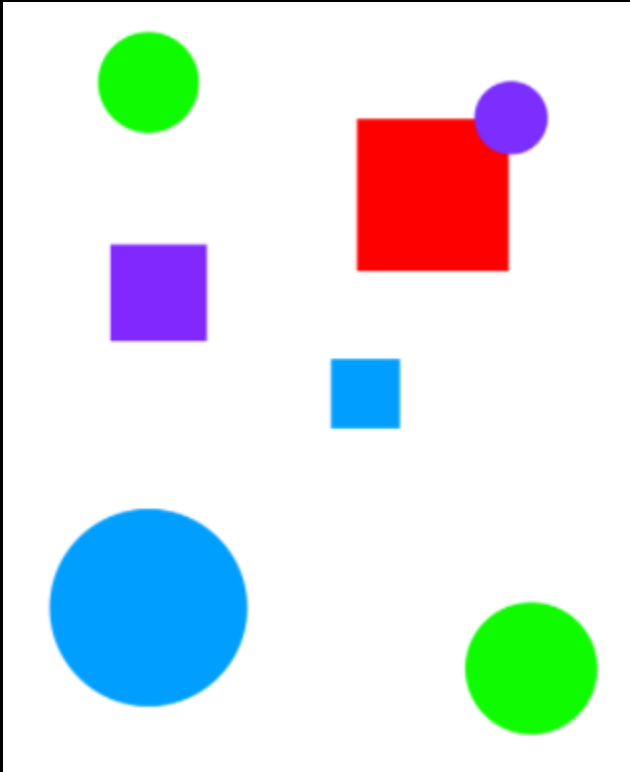


---

## Second Task: The Floating Shapes World

---

> Given Image



---

> Prolog KB

```
% -----  
% --- 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 :- 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

---

```
?- consult('C:/Users/godde/Downloads/CSC344/floating_shapes.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 ,
```

---

### Third Task: Pokemon KB Interaction and Programming

---

#### > Demo 1

---

```
?- consult('C:/Users/godde/Downloads/CSC344/pokemon.pro').  
true.
```

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

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

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

```
?- cen(Name),write(Name),nl,false.  
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,false.
bulbasaur-->venusaur
caterpie-->butterfree
charmander-->charizard
poliwag-->poliwrath
squirtle-->blastoise
false.

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

```



```

?- pokemon(name(X), fire, _, _), print(X), nl, false.
charmander
charmeleon
charizard
vulpix
ninetails
false.

?- pokemon(X, Y, _, _), print(nks(X, kind(Y))), nl, false.
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 .

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

?- pokemon(_, water, _, attack(X, _)), print(X), nl, false
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(X),_,hp(Y),_),Y>85,print(X),nl,false.
raichu
venusaur
butterfree
charizard
ninetails
poliwrath
blastoise
false.

?- pokemon(?,?,_,attack(Y,Z)),Z>60,print(Y),nl,false.
thunder-shock
poison-powder
whirlwind
royal-blaze
fire-blast
false.

?- cen(X),pokemon(name(X),_,hp(Y),_),print(X:Y),nl,false.
pikachu:60
bulbasaur:40
caterpie:50
charmander:50
vulpix:60
poliwag:60
squirtle:40
staryu:40
false.

```

---

## > Extended Knowledge Base

---

```

% -----
% --- display_names :: Displays all of the names of the pokemons in
% --- this knowledge base.

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

% -----
% --- display_attacks :: Displays all of the pokemon attacks in
% --- this knowledge base.

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

% -----
% --- powerful(N) :: There is a pokemon with name N whose attack
% --- is more than 55

powerful(N) :- pokemon(name(N),?,?,attack(_,D)),D > 55.

```

```

% -----
% --- tough(N) :: There is a pokemon with name N whose hp
% --- is more than 100

tough(N) :- pokemon(name(N),_,hp(H),_),H > 100.

% -----
% --- type(N,T) :: There is a pokemon with name N that is type T.

type(N,T) :- pokemon(name(N),T,_,_).

% -----
% --- dump_kind(T) :: Displays all information for pokemons with type T.

dump_kind(T) :- pokemon(name(N),T,X,Y),write(pokemon(name(N),T,X,Y)),nl,fail.

% -----
% --- display_cen :: Displays all "creatio ex nihilo" pokemon.

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

% -----
% --- family(CN) :: Displays the evolutionary family of "creatio ex nihilo"
% --- pokemon with name CN.

family(CN) :- evolves(CN,X),evolves(X,Y),write(CN),write(' '),write(X),
             write(' '),write(Y).
family(CN) :- evolves(CN,X),write(CN),write(' '),write(X).

% -----
% --- families :: Displays all of the evolutionary pokemon families.

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

% -----
% --- lineage(N) :: Displays all information of the pokemon with name N
% --- as well as all of the information on its evolutionary lineage.

lineage(N) :- pokemon(name(N),X,Y,Z),write(pokemon(name(N),X,Y,Z)),nl,
           evolves(N,A),pokemon(name(A),B,C,D),write(pokemon(name(A),B,C,D)),
           nl,evolves(A,L),pokemon(name(L),M,N,O),write(pokemon(name(L),M,N,O)).

```

---

## > Demo 2

---

```
?- consult('C:/Users/godde/Downloads/CSC344/pokemon.pro').  
true.
```

```
?- display_names.
```

```
pikachu  
raichu  
bulbasaur  
ivysaur  
venusaur  
caterpie  
metapod  
butterfree  
charmander  
charmeleon  
charizard  
vulpix  
ninetails  
poliwag  
poliwhirl  
poliwraith  
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
true.
```

```
?- 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
true.
```

```
?- lineage(caterpie).
pokemon(name(caterpie), grass, hp(50), attack(gnaw, 20))
pokemon(name(metapod), grass, hp(70), attack(stun-spore, 20))
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.
```

---

## Fourth Task: List Processing in Prolog

---

### > Demo 1

---

```
?- [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)].
```

---

## > Code

---

```
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).

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

make_list(0,_,[]).
make_list(Length,Number,[Number|Rest]) :-
    X is Length - 1,
    make_list(X,Number,Rest).

but_first([_|T],T).

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



```

is_palindrome([]).
is_palindrome([_]).
is_palindrome(List) :-
    first(List,First),
    last(List,Last),
    First == Last ->
    but_first(List,X),but_last(X,Y),
    is_palindrome(Y).

noun_phrase(NP) :-
    pick([shiny,dull,dirty,pretty,green,huge],A),
    pick([cat,computer,plane,book,carpet,spider,sweater,table],N),
    add_last(A,[the],Adjectives),
    add_last(N,Adjectives,NP).

sentence(Sentence) :-
    pick([ran,flew,slept,ate,drank,sat,talked],V),
    noun_phrase(First),
    noun_phrase(Second),
    add_last(V,First,S),
    append(S,Second,Sentence).

```

---

> Demo 2

---

```

?- consult('C:/Users/godde/Downloads/CSC344/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 = apple .

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

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

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

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

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

?- 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] .
```

---

## > Demo 3

---

```
?- consult('C:/Users/godde/Downloads/CSC344/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,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, pretty, spider] ;
false.

?- noun_phrase(NP).
NP = [the, dirty, cat] ;
false.

?- noun_phrase(NP).
NP = [the, green, computer] ;
false.

?- noun_phrase(NP).
NP = [the, pretty, spider] ;
false.

?- noun_phrase(NP).
NP = [the, pretty, plane] ;
false.

?- sentence(S).
S = [the, green, table, slept, the, dirty, book] .

?- sentence(S).
S = [the, dull, spider, ran, the, dirty, cat] .

?- sentence(S).
S = [the, huge, carpet, slept, the, dull, cat] .

?- sentence(S).
S = [the, huge, plane, flew, the, dirty, sweater] .

?- sentence(S).
S = [the, pretty, table, slept, the, huge, spider] .
```