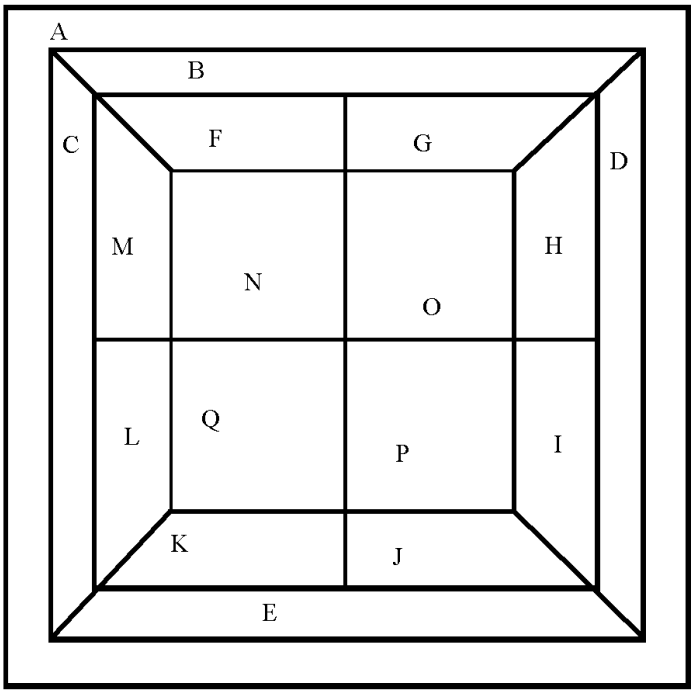
Assignment #6

Trevor Strickland, CSC344

November 2, 2022

Learning Abstract: This assignment serves as an introduction to the Prolog language, and goes through the definitions of rules and predicates, solving problems like coloring maps with each neighbor having a different color than one another, and working with pre-written databases. This assignment also touches on list processing in Prolog.

Task 1: Map Coloring



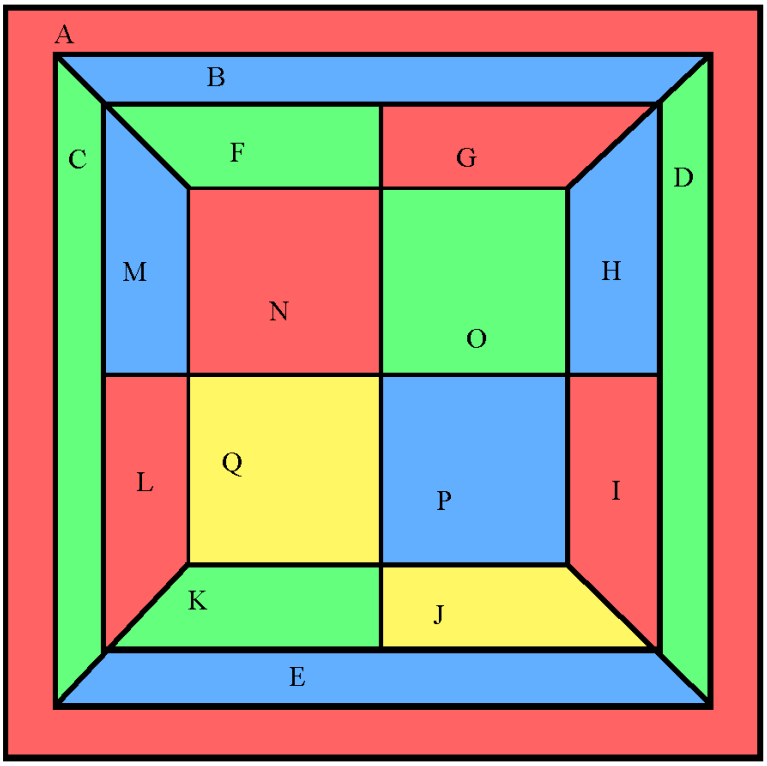
map_coloring.pro

```
diff(red, blue).
diff(red, green).
diff(red, yellow).

diff(green, blue).
diff(green, red).
diff(green, yellow).

diff(blue, green).
diff(blue, red).
diff(blue, yellow).

diff(yellow, blue).
diff(yellow, green).
diff(yellow, red).

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

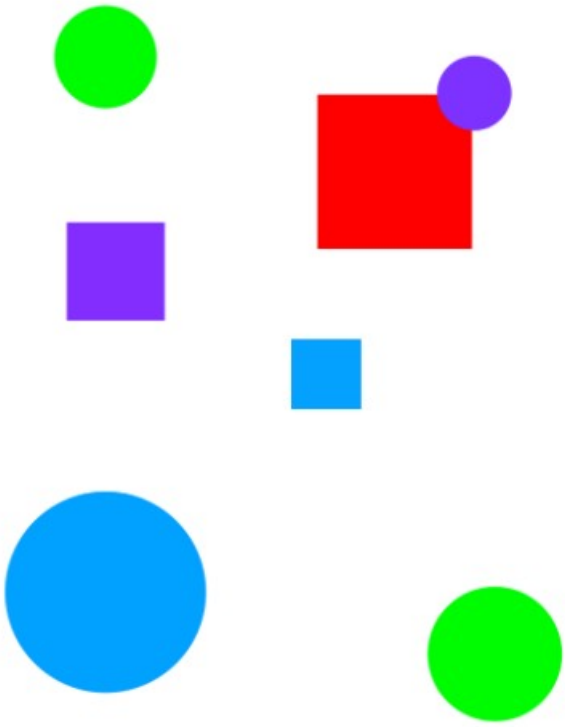
```
    diff(O, P),
    diff(P, Q).
```

Demo:

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



Task 2: Floating Shapes



floating_shapes.pro

```
% ------------------------------------------------------------------------
% ------------------------------------------------------------------------
% --- File: shapes_world_1.pro
% --- Line: Loosely represented 2-D shapes world (simple take on SHRDLU)
% ------------------------------------------------------------------------
% ------------------------------------------------------------------------
% --- Facts ...
% ------------------------------------------------------------------------
% ------------------------------------------------------------------------
% --- square(N,side(L),color(C)) :: N is the name of a square with side L
% --- and color C

square(sera,side(7),color(purple)).
square(sara,side(5),color(blue)).
square(sarah,side(11),color(red)).

% ------------------------------------------------------------------------
% --- circle(N,radius(R),color(C)) :: N is the name of a circle with
% --- radius R and color C

circle(carla,radius(4),color(green)).
circle(cora,radius(7),color(blue)).
circle(connie,radius(3),color(purple)).
circle(claire,radius(5),color(green)).

% ------------------------------------------------------------------------
% Rules ...
% ------------------------------------------------------------------------
% ------------------------------------------------------------------------
% --- circles :: list the names of all of the circles

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

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

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

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

shapes :- circles,squares.

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

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

```
% ------------------------------------------------------------------------
% --- large(Name) :: Name is a large shape

large(Name) :- area(Name,A), A ≥ 100.

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

small(Name) :- area(Name,A), A < 100.

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

area(Name,A) :- circle(Name,radius(R),_), A is 3.14 * R * R.
area(Name,A) :- square(Name,side(S),_), A is S * S.
```

Demo:

```
?- working_directory(_, 'C:/Users/trevo/Desktop').
true.

?- consult('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.
```

```
?- 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: The Pokemon KB

First Demo:

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

?- cen(raichu).
false.

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

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

?- evolves(squirtle, wartortle).
true.

?- evolves(wartortle, squirtle).
false.

?- evolves(squirtle, blastoise).
false.

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

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

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

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

```
?- pokemon(name(N),T,_,_),write('nks(name('),write(N),write('),kind('),write(T),write('))'),nl,fail.
nks(name(pikachu),kind(electric))
nks(name(raichu),kind(electric))
nks(name(bulbasaur),kind(grass))
nks(name(ivysaur),kind(grass))
nks(name(venusaur),kind(grass))
nks(name(caterpie),kind(grass))
nks(name(metapod),kind(grass))
nks(name(butterfree),kind(grass))
nks(name(charmander),kind(fire))
nks(name(charmeleon),kind(fire))
nks(name(charizard),kind(fire))
nks(name(vulpix),kind(fire))
nks(name(ninetails),kind(fire))
nks(name(poliwag),kind(water))
nks(name(poliwhirl),kind(water))
nks(name(poliwrath),kind(water))
nks(name(squirtle),kind(water))
nks(name(wartortle),kind(water))
nks(name(blastoise),kind(water))
nks(name(staryu),kind(water))
nks(name(starmie),kind(water))
false.

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

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

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

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

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

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

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

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

```
pokemon.pro (extended functionality):

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

powerful(N) :- pokemon(name(N),_,_,attack(_,DP)), DP > 55.
tough(N) :- pokemon(name(N),_,hp(HP),_), HP > 100.

type(N,T) :- pokemon(name(N),T,_,_).
dump_kind(T) :- pokemon(N,T,H,A), write('pokemon('), write(N), write(','),
write(T), write(','), write(H), write(','), write(A), write(').'), nl, fail.

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

family(N) :- write(N), write(' '), evolves(N, X), family(X).
families :- cen(N), family(N), nl, fail.

lineage(N) :-
     pokemon(name(N), T, H, A),write('pokemon(name('), write(N),
     write('),'), write(T), write(','), write(H), write(','), write(A),
     write(')'),
     evolves(N, X), nl, lineage(X).
```

Second Demo:

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

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

?- display_attacks.
gnaw
thunder-shock
leech-seed
vine-whip
poison-powder
gnaw
stun-spore
whirlwind
scratch
slash
royal-blaze
confuse-ray
fire-blast
water-gun
amnesia
dashing-punch
bubble
waterfall
hydro-pump
slap
star-freeze
false.
```

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

?- powerful(blastoise).
true .

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

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

?- tough(venusaur).
true.

?- tough(X), write(X), 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.
```

```
?- 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 star
mie
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

First Demo:

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

list_processing.pro

```
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).
```

Second Demo:

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

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

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

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

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

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

?- last([c,d,e,f,g,a,b], Last).
Last = 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],Result).
Result = [racket, prolog, haskell, rust].

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

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

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

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

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

?- pick([cherry,peach,apple,blueberry],Pie).
Pie = apple
Unknown action: 0 (h for help)
Action? ,

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

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

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

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

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

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

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

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