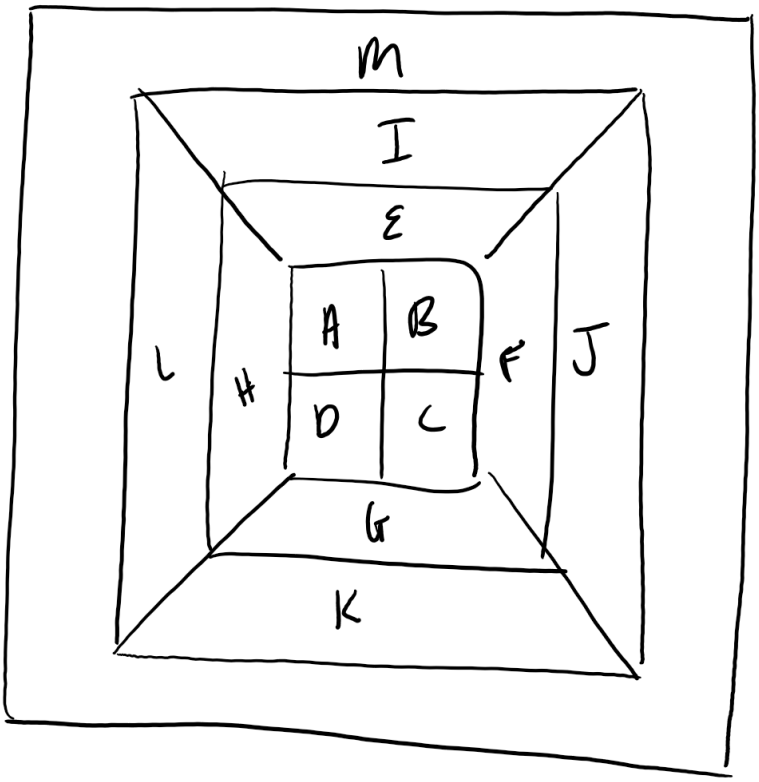
Prolog Programming Assignment #1: Various Computations

Learning Abstract

This assignment is a first introduction to Prolog and its logical ways. Using the powerful logic operators in Prolog in this assignment we color a map, consult and expand some knowledge bases, and begin list processing.

Task 1: Map Coloring

Given Map - I ended up labeling the map with just capital letters. I originally named the variables a pair of numbers with the first being the ring starting in the middle and the second being the increasing order clockwise from North. Prolog is not a fan of using numbers as variables, so this was my next best option.



Source Code -

```
% ---------------------------------
% File: coloring2.pl
% Line: Program to find a 4 color map rendering.
% More: The colors used will be red, yellow, blue, and green.
% More: The sections are labeled as two digits.
%       The first digit is the ring,
```

```
% The second is labeled from the top going counter clockwise

%---------------------------------
% 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(11,12,13,14,21,22,23,24,31,32,33,34,41) | each region
% is not colored the same as a region it shares an edge with
coloring(A,B,C,D,E,F,G,H,I,J,K,L,M) :-
    different(A, B),
    different(A, D),
    different(A, E),
    different(A, H),
    different(B, C),
    different(B, E),
    different(B, F),
    different(C, D),
    different(C, F),
    different(C, G),
    different(D, G),
    different(D, H),
    different(E, F),
    different(E, H),
    different(E, I),
    different(F, G),
    different(F, I),
    different(G, H),
    different(G, K),
    different(H, L),
    different(I, J),
    different(I, L),
    different(I, M),
    different(J, K),
    different(J, M),
    different(K, L),
    different(K, M),
```

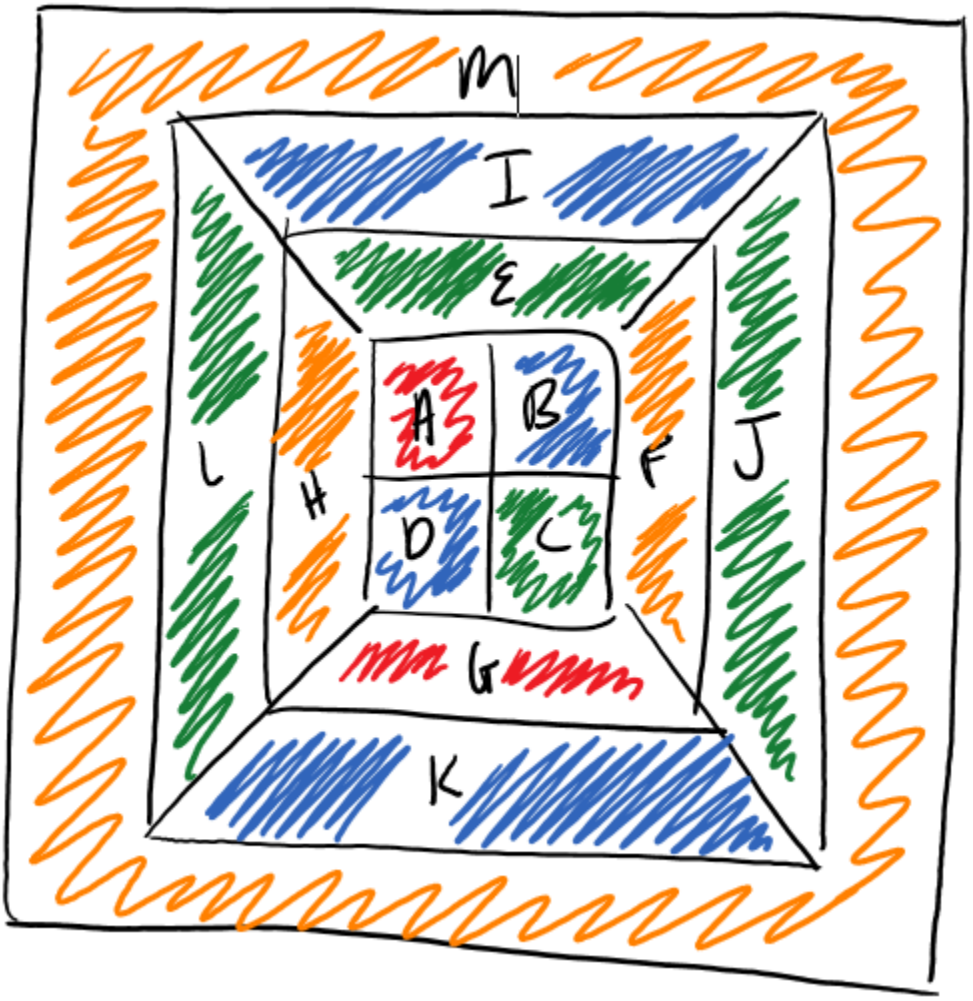
```
56      different(L, M).
```

Demo -

```
% c:/Users/allen/Programming/Public_html/CSC344Files/Prolog/PA_1/coloring2.pl
compiled 0.00 sec, 0 clauses

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

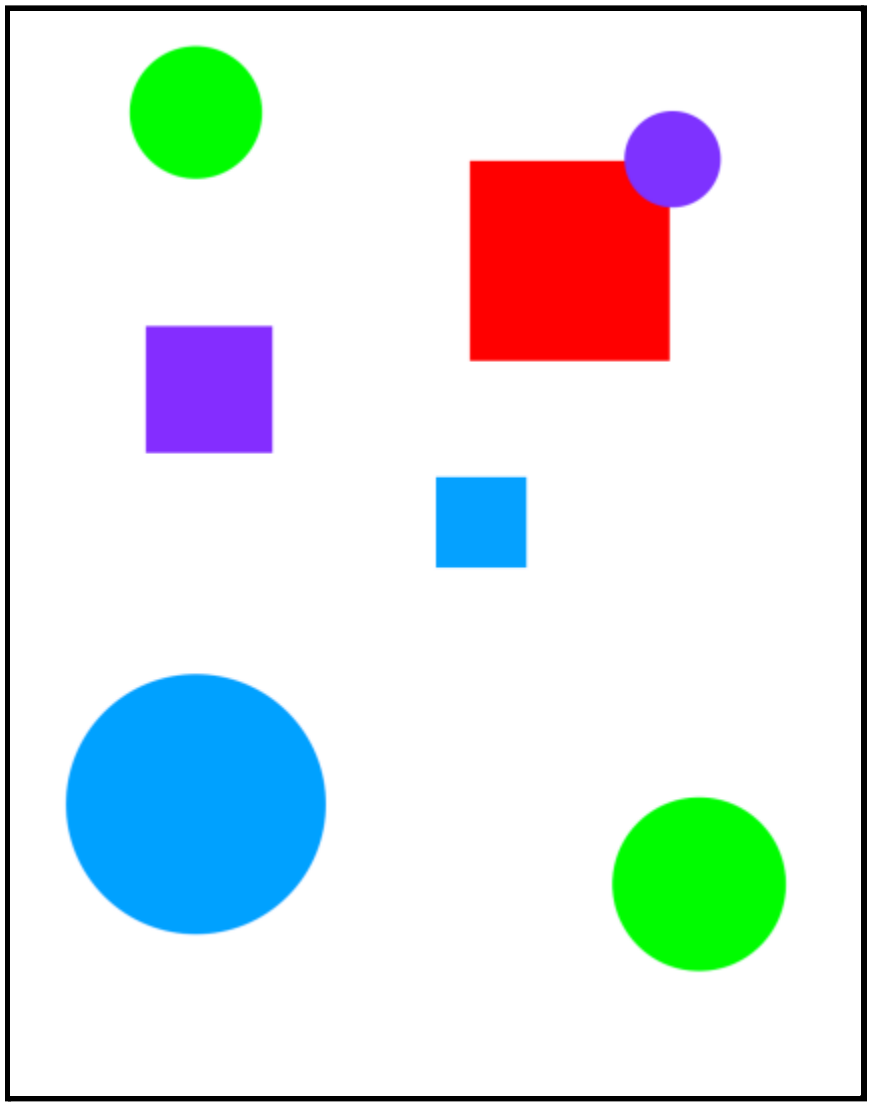
Colored Map -



Task 2: Floating Shapes World

In this task I mindfully retyped the example from class and recreated the demo. This exercise is a great first introduction to predicates in Prolog.

Shape World Image -



Demo -

```
1 ?-
2 %
c:/Users/allen/Programming/Public_html/CSC344Files/Prolog/PA_1/shapes_world.pl
compiled 0.00 sec, 18 clauses
3 ?- listing(squares).
4 squares :-
5     square(Name, _, _),
6     write(Name),
7     nl,
8     fail.
9 squares.
```

```
10
11 true.
12
13 ?- squares.
14 sera
15 sara
16 sarah
17 true.
18
19 ?- listing(circles).
20 circles :-
21     circle(Name, _, _),
22     write(Name),
23     nl,
24     fail.
25 circles.
26
27 true.
28
29 ?- circles.
30 carla
31 cora
32 connie
33 claire
34 true.
35
36 ?- listing(shapes).
37 shapes :-
38     circles,
39     squares.
40
41 true.
42
43 ?- shapes.
44 carla
45 cora
46 connie
47 claire
48 sera
49 sara
50 sarah
51 true.
52
53 ?- blue(Shape).
54 Shape = sara ;
55 Shape = cora.
56
57 ?- large(Name),write(Name),nl,fail.
58 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 .
```

Source Code -

```
% ------------------------------------
% File: shapes_world.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
```

```
33
34 squares :- square(Name,_,_), write(Name),nl,fail.
35 squares.
36
37 % ---------------------------------
38 % --- squares :: list the names of all of the shapes
39
40 shapes :-
41     circles,
42     squares.
43
44 % ---------------------------------
45 % blue(Name) :: Name is a blue shape
46
47 blue(Name) :-
48     square(Name,_,color(blue)).
49 blue(Name) :-
50     circle(Name,_,color(blue)).
51
52 % ---------------------------------
53 % large(Name) :: Name is a large shape
54
55 large(Name) :-
56     area(Name,A),
57     A >= 100.
58
59 % ---------------------------------
60 % small(Name) :: Name is a small shape
61
62 small(Name) :-
63     area(Name,A),
64     A < 100.
65
66 % ---------------------------------
67 % area(Name,A) :: A is the area of the shape with name Name
68
69 area(Name,A) :-
70     circle(Name,radius(R),_),
71     A is 3.14 * R * R.
72 area(Name,A) :-
73     square(Name,side(S),_),
74     A is S * S.
```

Task 3: Pokemon KB Interaction and Programming

In this task I consult a given database and add to said knowledge base. The first demo is done in the interactive mode and just asks queries of the KB. The second demo is created using code written and added to the KB. The entire source code is at the end of the demos as having separate source code for both would be pointless.

Demo 1 -

```
1 % c:/Users/allen/Programming/Public_html/CSC344Files/Prolog/PA_1/pokemon.pl
compiled 0.00 sec, 42 clauses
2 ?- cen(pikachu).
3 true.
4
5 ?- cen(riachu).
6 false.
7
8 ?- cen(Name).
9 Name = pikachu ;
10 Name = bulbasaur ;
11 Name = caterpie ;
12 Name = charmander ;
13 Name = vulpix ;
14 Name = poliwag ;
15 Name = squirtle ;
16 Name = staryu.
17
18 ?- cen(Name), write(Name), nl, fail.
19 pikachu
20 bulbasaur
21 caterpie
22 charmander
23 vulpix
24 poliwag
25 squirtle
26 staryu
27 false.
28
29 ?- evolves(squirtle,wartortle).
30 true.
31
32 ?- evolves(wartortle,squirtle).
33 false.
34
35 ?- evolves(squirtle,blastoise).
36 false.
37
38 ?- evolves(X,Y), evolves(Y,Z).
39 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),Kind,_,_), write(wks(name(Name))), write(kind(Kind)),
nl, fail.
wks(name(pikachu))kind(electric)
wks(name(raichu))kind(electric)
wks(name(bulbasaur))kind(grass)
wks(name(ivysaur))kind(grass)
wks(name(venusaur))kind(grass)
wks(name(caterpie))kind(grass)
wks(name(metapod))kind(grass)
wks(name(butterfree))kind(grass)
wks(name(charmander))kind(fire)
wks(name(charmeleon))kind(fire)
wks(name(charizard))kind(fire)
wks(name(vulpix))kind(fire)
wks(name(ninetails))kind(fire)
wks(name(poliwag))kind(water)
wks(name(poliwhirl))kind(water)
wks(name(poliwrath))kind(water)
wks(name(squirtle))kind(water)
wks(name(wartortle))kind(water)
wks(name(blastoise))kind(water)
wks(name(staryu))kind(water)
wks(name(starmie))kind(water)
false.

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

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

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

```
137
138 ?- pokemon(name(poliwhirl),_,hp(HP),_).
139 HP = 80.
140
141 ?- pokemon(name(butterfree),_,hp(HP),_).
142 HP = 130.
143
144 ?- pokemon(name(Name),_,hp(HP),_), HP >= 85, write(Name), nl, fail.
145 raichu
146 venusaur
147 butterfree
148 charizard
149 ninetails
150 poliwrath
151 blastoise
152 false.
153
154 ?- pokemon(_,_,_,attack(Name,Dmg)), Dmg > 60, write(Name), nl, fail.
155 thunder-shock
156 poison-powder
157 whirlwind
158 royal-blaze
159 fire-blast
160 false.
161
162 ?- cen(Name), pokemon(name(Name),_,hp(HP),_), write(Name), write(:),
write(HP), nl, fail.
163 pikachu:60
164 bulbasaur:40
165 caterpie:50
166 charmander:50
167 vulpix:60
168 poliwag:60
169 squirtle:40
170 staryu:40
171 false.
172
173 ?-
```

Demo 2 -

```
 1 % c:/Users/allen/Programming/Public_html/CSC344Files/Prolog/PA_1/pokemon.pl
compiled 0.00 sec, 57 clauses
2 ?-
3 |    display_names.
4 pikachu
5 raichu
6 bulbasaur
7 ivysaur
8 venusaur
```

```
caterpie
metapod
butterfree
charmander
charmeleon
charizard
vulpix
ninetails
poliwag
poliwhirl
poliwrath
squirtle
wartortle
blastoise
staryu
starmie
true.

?-
|    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
true.

?- 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
true.

?- family(pikachu).
false.

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

?- families.
bulbasaur ivysaur venusaur
caterpie metapod butterfree
charmander charmeleon charizard
poliwag poliwhirl poliwrath
squirtle wartortle blastoise
true.

?-  lineage(caterpie).
ERROR: Unknown procedure: lineage/1 (DWIM could not correct goal)
?-
```

Source Code -

```
% ----------------
% ----------------
% --- File: pokemon.pl
```

```
4 % --- Line: Just a few facts about pokemon
5 % ----------------
6
7 % ----------------
8 % --- cen(P) :: Pokemon P was "creatio ex nihilo"
9
10 cen(pikachu).
11 cen(bulbasaur).
12 cen(caterpie).
13 cen(charmander).
14 cen(vulpix).
15 cen(poliwag).
16 cen(squirtle).
17 cen(staryu).
18
19 % ----------------
20 % --- evolves(P,Q) :: Pokemon P directly evolves to pokemon Q
21
22 evolves(pikachu,raichu).
23 evolves(bulbasaur,ivysaur).
24 evolves(ivysaur,venusaur).
25 evolves(caterpie,metapod).
26 evolves(metapod,butterfree).
27 evolves(charmander,charmeleon).
28 evolves(charmeleon,charizard).
29 evolves(vulpix,ninetails).
30 evolves(poliwag,poliwhirl).
31 evolves(poliwhirl,poliwrath).
32 evolves(squirtle,wartortle).
33 evolves(wartortle,blastoise).
34 evolves(staryu,starmie).
35
36 % ----------------
37 % --- pokemon(name(N),T,hp(H),attach(A,D)) :: There is a pokemon with
38 % --- name N, type T, hit point value H, and attach named A that does
39 % --- damage D.
40
41 pokemon(name(pikachu), electric, hp(60), attack(gnaw, 10)).
42 pokemon(name(raichu), electric, hp(90), attack(thunder-shock, 90)).
43
44 pokemon(name(bulbasaur), grass, hp(40), attack(leech-seed, 20)).
45 pokemon(name(ivysaur), grass, hp(60), attack(vine-whip, 30)).
46 pokemon(name(venusaur), grass, hp(140), attack(poison-powder, 70)).
47
48 pokemon(name(caterpie), grass, hp(50), attack(gnaw, 20)).
49 pokemon(name(metapod), grass, hp(70), attack(stun-spore, 20)).
50 pokemon(name(butterfree), grass, hp(130), attack(whirlwind, 80)).
51
52 pokemon(name(charmander), fire, hp(50), attack(scratch, 10)).
```

```
53 pokemon(name(charmeleon), fire, hp(80), attack(slash, 50)).
54 pokemon(name(charizard), fire, hp(170), attack(royal-blaze, 100)).
55
56 pokemon(name(vulpix), fire, hp(60), attack(confuse-ray, 20)).
57 pokemon(name(ninetails), fire, hp(100), attack(fire-blast, 120)).
58
59 pokemon(name(poliwag), water, hp(60), attack(water-gun, 30)).
60 pokemon(name(poliwhirl), water, hp(80), attack(amnesia, 30)).
61 pokemon(name(poliwrath), water, hp(140), attack(dashing-punch, 50)).
62
63 pokemon(name(squirtle), water, hp(40), attack(bubble, 10)).
64 pokemon(name(wartortle), water, hp(80), attack(waterfall, 60)).
65 pokemon(name(blastoise), water, hp(140), attack(hydro-pump, 60)).
66
67 pokemon(name(staryu), water, hp(40), attack(slap, 20)).
68 pokemon(name(starmie), water, hp(60), attack(star-freeze, 20)).
69
70 %---------------------
71 % prefixes
72
73 %display_names; parameterless; prints list of all names in KB
74 display_names :-
75     pokemon(name(Name),_,_,_),
76     write(Name),
77     nl,
78     fail.
79     display_names.
80
81 %display_attacks; marameterless; prints list of all attacks
82 display_attacks :-
83     pokemon(_,_,_,attack(Atk,_)),
84     write(Atk),
85     nl,
86     fail.
87     display_attacks.
88
89 %powerful(Name); 1 pokemon name; returns true if pokemons attack deals more
than 55 damage
90 powerful(Name) :-
91     pokemon(name(Name),_,_,attack(_,Dmg)),
92     Dmg > 55.
93
94 %tough(Name); 1 pokemon name; returns true if pokemons HP is greater than
100
95 tough(Name) :-
96     pokemon(name(Name),_,hp(HP),_),
97     HP > 100.
98
99 %type(Name,Type); 1 name and 1 type;
```

```
100 %suceeds if the pokemon name is of the type given
101 type(Name,Type) :-
102     pokemon(name(Name),Type,_,_).
103
104 %dump_kind(Type); 1 Type parameter
105 %prints list of all pokemon of that type
106 dump_kind(Type) :-
107     pokemon(name(Name),Type,hp(HP),attack(Atk,DMG)),
108     A = pokemon(name(Name),Type,hp(HP),attack(Atk,DMG)),
109     write(A),
110     nl,
111     fail.
112     dump_kind.
113
114 %display_cen; no parameters; displays all cen pokemon
115 display_cen :-
116     cen(A),
117     write(A),
118     nl,
119     fail.
120     display_cen.
121
122 %family(CenPokemon); one parameter a Cen pokemon
123 %prints out on one line all pokemon that evolve from given
124 family(X) :-
125     evolves(X,Y),
126     evolves(Y,Z),
127     write(X), write(" "),
128     write(Y), write(" "),
129     write(Z).
130     family.
131      :-
132     evolves(X,Y),
133     write(X), write(" "),
134     write(Y).
135 %families; no parameters; displays all families of pokemon each on one line
136 families :-
137     cen(A),
138     family(A),
139     nl,
140     fail.
141     families.
```

Task 4 - List Processing

This is a series of programs that example how list processing functions in Prolog. The first is just using the interactive mode and is fairly simple. The second is a recreated demo from a lesson that introduces some basic list processing from a KB. The third is some more simple list processing.

Demo 1 -

```
1 ?- [H|T] = [ red, yellow, blue, green].
2 H = red,
3 T = [yellow, blue, green].
4
5 ?- [ H, T ] = [ red, yellow, blue, green ].
6 false.
7
8 ?- [ F | _ ] = [ red, yellow, green, blue ].
9 F = red.
10
11 ?- [_|[S|_]] = [red, yellow, blue, green].
12 S = yellow.
13
14 ?- [F|[S|R]] = [red, yellow, blue, green].
15 F = red,
16 S = yellow,
17 R = [blue, green].
18
19 ?- List = [this|[and, that]].
20 List = [this, and, that].
21
22 ?-  List = [this, and, that].
23 List = [this, and, that].
24
25 ?- [a,[b, c]] = [a, b, c].
26 false.
27
28 ?-  [a|[b, c]] = [a, b, c].
29 true.
30
31 ?-  [cell(Row,Column)|Rest] = [cell(1,1), cell(3,2), cell(1,3)].
32 Row = Column, Column = 1,
33 Rest = [cell(3, 2), cell(1, 3)].
34
35 ?-  [X|Y] = [one(un, uno), two(dos, deux), three(trois, tres)].
36 X = one(un, uno),
37 Y = [two(dos, deux), three(trois, tres)].
38
39 ?-
```

Demo 2 -

```
% c:/Users/allen/Programming/Public_html/CSC344Files/Prolog/PA_1/list_processors.pl 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 .

?- 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].
```

```
47
48 ?- add_last(thing,[],Result).
49 Result = [thing] .
50
51 ?- add_last(rust,[racket,prolog,haskell],Languages).
52 Languages = [racket, prolog, haskell, rust] .
53
54 ?- iota(5,Iota5).
55 Iota5 = [1, 2, 3, 4, 5] .
56
57 ?- iota(9,Iota9).
58 Iota9 = [1, 2, 3, 4, 5, 6, 7, 8, 9] .
59
60 ?- pick([cherry,peach,apple,blueberry],Pie).
61 Pie = apple .
62
63 ?- pick([cherry,peach,apple,blueberry],Pie).
64 Pie = blueberry .
65
66 ?- pick([cherry,peach,apple,blueberry],Pie).
67 Pie = cherry .
68
69 ?- pick([cherry,peach,apple,blueberry],Pie).
70 Pie = apple .
71
72 ?- pick([cherry,peach,apple,blueberry],Pie).
73 Pie = cherry .
74
75 ?- pick([cherry,peach,apple,blueberry],Pie).
76 Pie = peach .
77
78 ?- pick([cherry,peach,apple,blueberry],Pie).
79 Pie = apple .
80
81 ?-
82 |    pick([cherry,peach,apple,blueberry],Pie).
83 Pie = peach .
84
85 ?- make_set([1,1,2,1,2,3,1,2,3,4],Set).
86 Set = [1, 2, 3, 4] .
87
88 ?- make_set([bit,bot,bet,bot,bot,bit],B).
89 B = [bet, bot, bit] .
90
91 ?-
```

Code 2 -

```
1 %-----------------
2 % example list processors
```

```
3
4 %-----
5 % Name
6
7 %-----
8 % first
9 first([H|_], H).
10
11 %rest
12 rest([_|T], T).
13
14
15 %last
16 last([H|[]], H).
17 last([_|T], Result) :-
18     last(T, Result).
19
20 %nth
21 nth(0,[H|_],H).
22 nth(N,[_|T],E) :-
23     K is N - 1,
24     nth(K,T,E).
25
26 %writelist
27 writelist([]).
28 writelist([H|T]) :-
29     write(H),
30     nl,
31     writelist(T).
32
33 %sum
34 sum([],0).
35 sum([Head|Tail],Sum) :-
36     sum(Tail,SumOfTail),
37     Sum is Head + SumOfTail.
38
39 %add_first
40 add_first(X,L,[X|L]).
41
42 %add_last
43 add_last(X,[],[X]).
44 add_last(X,[H|T],[H|TX]) :-
45     add_last(X,T,TX).
46
47 %iota
48 iota(0,[]).
49 iota(N,IotaN) :-
50     K is N - 1,
51     iota(K,IotaK),
```

```
52      add_last(N,IotaK,IotaN).
53
54  %pick
55  pick(L,Item) :-
56      length(L,Length),
57      random(0,Length,RN),
58      nth(RN,L,Item).
59
60  %make_set
61  make_set([],[]).
62  make_set([H|T],TS) :-
63      member(H,T),
64      make_set(T,TS).
65  make_set([H|T],[H|TS]) :-
66      make_set(T,TS).
67
```