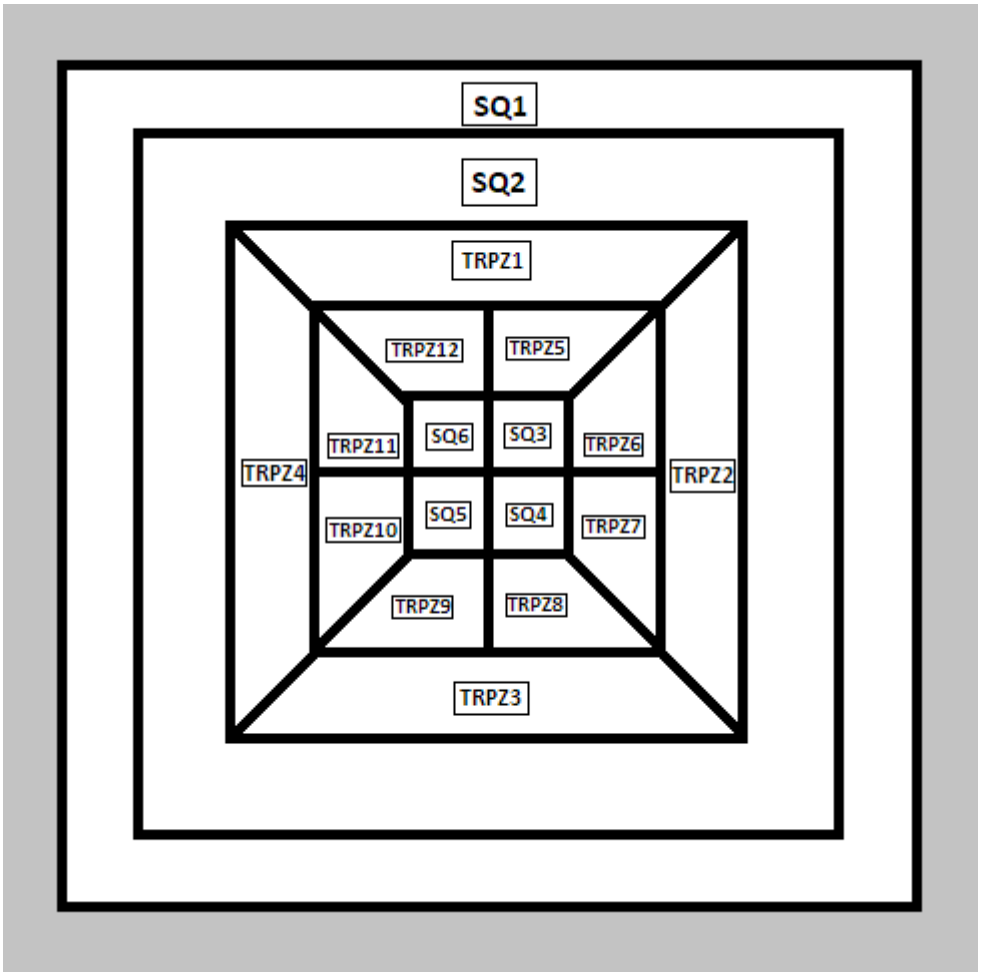
Prolog Programming Assignment #1: Various Computations

Brandon LaPointe

Learning Abstract

This assignment, being the second Racket / first Prolog assignment, includes programming exercises done within the Prolog Language environment of SWI-Prolog that focus on knowledge representation, search, and list processing done within Prolog. Task 1 pertains to a map coloring program which is used to color portions of a given map image in 4 different colors with none of the colors sharing a border with the same color. Task 2 pertains to the Floating Shapes World lesson and accompanying demo. Task 3 involves a knowledge base on pokemon trading cards in which 20 queries were needed in order to recreate the first demo followed by the creation of several programs needed to recreate the second demo provided within the lesson. The final task, task 4, involved list processing functions corresponding to the "Head/Tail Referencing Exercises", "Example List Processors", and "List Processing Exercises" demonstrations.

Task 1: Map Coloring – Labelled Map



Task 1: Map Coloring – Source Code

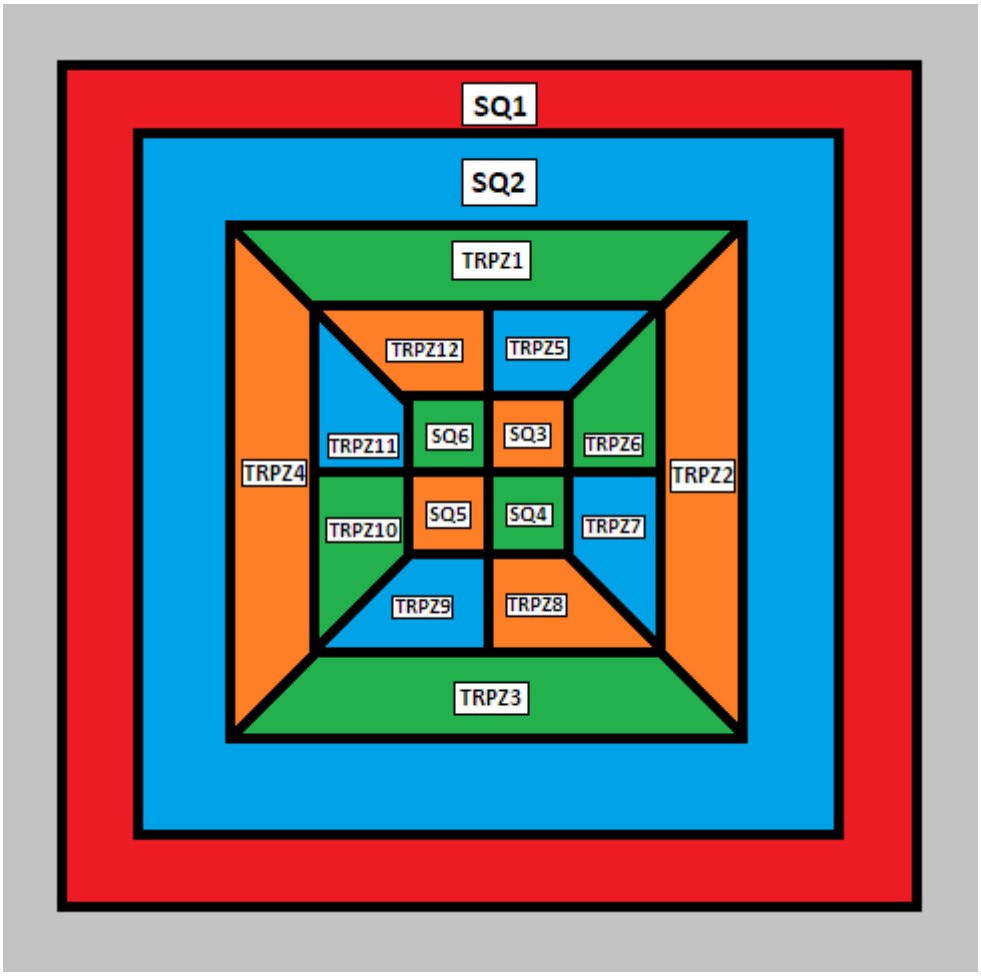
```
% ----------------------------------------------------------------------
% File: map_coloring.pro
% Program to find a 4 color map rendering for the given task 1 map.
% The colors used will be red, blue, green, and orange.
% The standard for abbrieviations used to stand for the shapes are given
% in the format of an abbreviation of the shape in the form of SQ for square
% or TRPZ for trapezoid, ending with a number signifying the position of
% shape within the ring starting at the 12 o'clock position within the outer
% layer ring of the given map, increasing numbers making our way towards the center.

% ----------------------------------------------------------------------
% 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(SQ1,SQ2,TRPZ1,TRPZ2,TRPZ3,TRPZ4,TRPZ5,TRPZ6,TRPZ7,TRPZ8,TRPZ9,TRPZ10,TRPZ11,TRPZ12,SQ3,SQ4,SQ5,SQ6). ::
% The shapes within the given map represented by a shape abbrieviation and a number signifying the shape within
% the ring starting at the 12 o'clock position of the outer most layer are colored so that none of the shapes
% sharing a border are the same color.

coloring(SQ1,SQ2,TRPZ1,TRPZ2,TRPZ3,TRPZ4,TRPZ5,TRPZ6,TRPZ7,TRPZ8,TRPZ9,TRPZ10,TRPZ11,TRPZ12,SQ3,SQ4,SQ5,SQ6) :-
        different(SQ1, SQ2),
        different(SQ2, TRPZ1),
        different(SQ2, TRPZ2),
        different(SQ2, TRPZ3),
        different(SQ2, TRPZ4),
        different(TRPZ1, TRPZ2),
```

different(TRPZ2, TRPZ3),
different(TRPZ3, TRPZ4),
different(TRPZ4, TRPZ1),
different(TRPZ1, TRPZ5),
different(TRPZ1, TRPZ12),
different(TRPZ2, TRPZ6),
different(TRPZ2, TRPZ7),
different(TRPZ3, TRPZ8),
different(TRPZ3, TRPZ9),
different(TRPZ4, TRPZ10),
different(TRPZ4, TRPZ11),
different(TRPZ5, TRPZ6),
different(TRPZ6, TRPZ7),
different(TRPZ7, TRPZ8),
different(TRPZ8, TRPZ9),
different(TRPZ9, TRPZ10),
different(TRPZ10, TRPZ11),
different(TRPZ11, TRPZ12),
different(TRPZ12, TRPZ5),
different(TRPZ5, SQ3),
different(TRPZ6, SQ3),
different(TRPZ7, SQ4),
different(TRPZ8, SQ4),
different(TRPZ9, SQ5),
different(TRPZ10, SQ5),
different(TRPZ11, SQ6),
different(TRPZ12, SQ6),
different(SQ3, SQ4),
different(SQ4, SQ5),
different(SQ5, SQ6),
different(SQ6, SQ1).

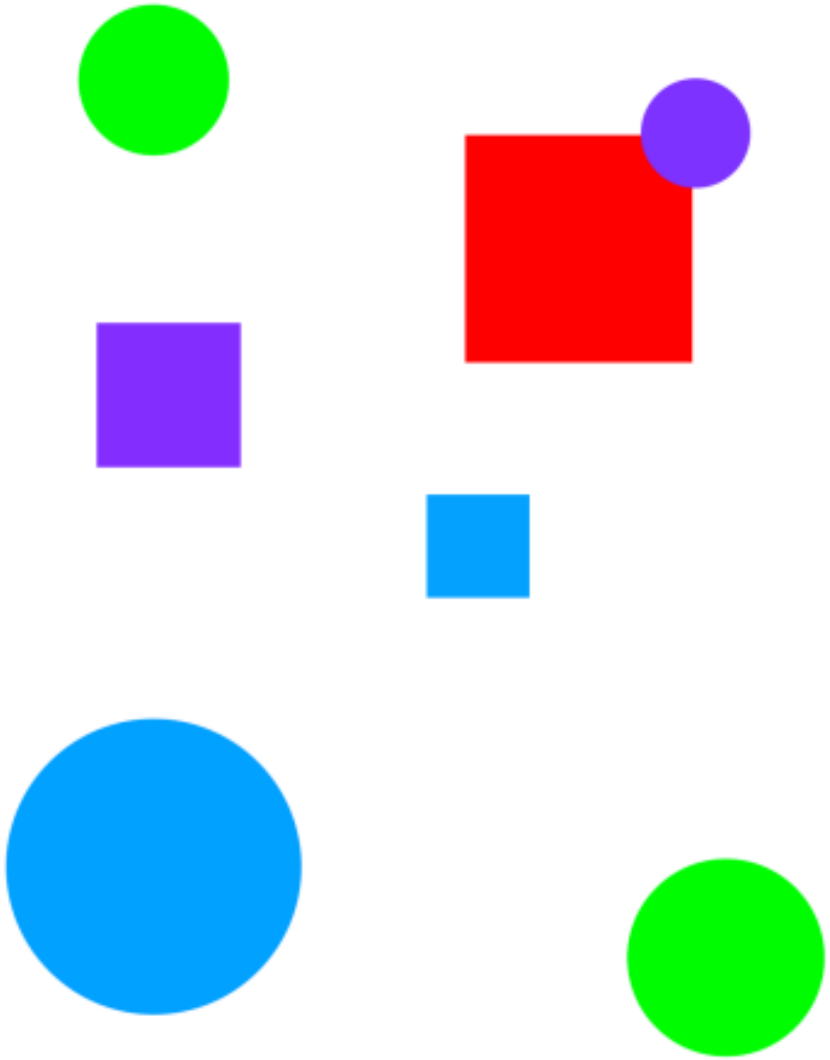
Task 1: Map Coloring Demo

```
Welcome to SWI-Prolog (threaded, 64 bits, version 8.4.3)
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('map_coloring.pro').
true.

?- coloring(SQ1,SQ2,TRPZ1,TRPZ2,TRPZ3,TRPZ4,TRPZ5,TRPZ6,TRPZ7,TRPZ8,TRPZ9,TRPZ10,TRPZ11,TRPZ12,SQ3,SQ4,SQ5,SQ6).
SQ1 = red,
SQ2 = TRPZ5, TRPZ5 = TRPZ7, TRPZ7 = TRPZ9, TRPZ9 = TRPZ11, TRPZ11 = blue,
TRPZ1 = TRPZ3, TRPZ3 = TRPZ6, TRPZ6 = TRPZ10, TRPZ10 = SQ4, SQ4 = SQ6, SQ6 = green,
TRPZ2 = TRPZ4, TRPZ4 = TRPZ8, TRPZ8 = TRPZ12, TRPZ12 = SQ3, SQ3 = SQ5, SQ5 = orange .

?- halt.
```

Task 1: Map Coloring – Colored Map



Task 2: The Floating Shapes World – Presented Image



Task 2: The Floating Shapes World – Prolog Knowledge Base

```
% ------------------------------------------------------------------
% ------------------------------------------------------------------
% --- 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

        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.
```

Task 2: The Floating Shapes World – Demo

```
Welcome to SWI-Prolog (threaded, 64 bits, version 8.4.3)
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('shapes_world.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 .

?- halt.
```

Task 3: Pokémon KB Interactions & Programming – Part 1 Demo

```
SWI-Prolog (AMD64, Multi-threaded, version 8.4.3)
File  Edit  Settings  Run  Debug  Help
Welcome to SWI-Prolog (threaded, 64 bits, version 8.4.3)
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('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,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(Name1,Name2),evolves(Name2,Name3),write(Name1),write(" --> "),write(Name3),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,Kind, _, _),write("nks("),write(Name),write(',kind('),write(Kind),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(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( _, _, _, attack(ATK,DAM)),DAM>60,write(ATK),nl,fail.
thunder-shock
poison-powder
whirlwind
royal-blaze
fire-blast
false.

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

?-
```

Task 3: Pokémon KB Interactions & Programming – Extended KB

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

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

% ------------------------------------------------------------------------
% --- display_names :: list the names of all of the pokemon

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

% ------------------------------------------------------------------------
% --- display_attacks :: list the attacks of all of the pokemon

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

% ------------------------------------------------------------------------
% --- powerful(Name) :: succeeds if the pokemon's attack yields over 55 damage

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

% ------------------------------------------------------------------------
% --- tough(Name) :: succeeds if the pokemon's HP is greater or equal to 100

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

% ------------------------------------------------------------------------
% --- type(Name,Type) :: succeeds if the pokemon is of the specified type

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

% ------------------------------------------------------------------------
% --- dump_kind(T) :: displays all information on all pokemon of the given type

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

% ------------------------------------------------------------------------
% --- display_cen :: displays names of all "creatio ex nihilo" pokemon

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

%------------------------------------------------------------------------
% --- family(Name1) :: displays names of all related pokemon to a presumed CEN pokemon

		family(Name1) :- evolves(Name1,Name2), write(Name1), write(' '), write(Name2), evolves(Name2,Name3), write(' '), write(Name3).
% -- Returns true but gives singleton variable warning with the addition of the line below
		family(Name1).

%------------------------------------------------------------------------
% --- families :: displays all families of pokemon

		families :- cen(Name1), evolves(Name1,Name2), nl, write(Name1), write(' '), write(Name2), evolves(Name2,Name3), write(' '), write(Name3), fail.
		families.

%------------------------------------------------------------------------
% --- lineage(Name1) :: displays all information on the family of the given pokemon

		lineage(Name1) :- pokemon(name(Name1),_,_,_), listing(pokemon(name(Name1),_,_,_)), evolves(Name1,Name2),
listing(pokemon(name(Name2),_,_,_)), evolves(Name2,Name3), listing(pokemon(name(Name3),_,_,_)).

% -- Returns true but gives singleton variable warning with the addition of the line below
		lineage(Name1).
```

Task 3: Pokémon KB Interactions & Programming – Part 2 Demo

```
Welcome to SWI-Prolog (threaded, 64 bits, version 8.4.3)
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('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
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
ninetails
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
true.

?- 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)).

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)).

true.

?- lineage(butterfree).
pokemon(name(butterfree), grass, hp(130), attack(whirlwind, 80)).

true.

?- 
```

Task 4: List Processing in Prolog – Head / Tail Referencing Exercise Guesses

?- [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].
[this, and, that]

?- List = [this|[and, that]].
[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)].
cell(1,1), 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)].

Task 4: List Processing in Prolog – Head / Tail Referencing Exercise Demo

```
Welcome to SWI-Prolog (threaded, 64 bits, version 8.4.3)
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('list_processors.pro').
true.

?- [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)].

?-
```

Task 4: List Processing in Prolog – List Processing Function Definitions

```
% ------------------------------------------------------------------------
% ------------------------------------------------------------------------
% --- File: list_processors.pro
% --- Line:
% ------------------------------------------------------------------------

% ------------------------------------------------------------------------
% Preliminary Code...
% ------------------------------------------------------------------------

% ------------------------------------------------------------------------
% --- first(List,"First") :: Outputs the "car" of the given list in
% the form "First" = ...

        first([H|_], H).

% ------------------------------------------------------------------------
% --- rest(List,"Rest") :: Outputs the "cdr" of the given list in
% the form "Rest" = ...

        rest([_|T], T).

% ------------------------------------------------------------------------
% --- last(List,"Last") :: Outputs the last item of the given list
% in the form "Last" = ...

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

% ------------------------------------------------------------------------
% --- Nth(N,List,"Element") :: Outputs the Nth item of the given list
% in the form "Element" = ...

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

% ------------------------------------------------------------------------
% --- Writelist(List) :: Outputs each element of the given list on a new line

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

% ------------------------------------------------------------------------
% --- sum(List,"Sum") :: Adds all of the elements of a list and outputs
% in the form "Sum" = ...

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

% ------------------------------------------------------------------------
% --- Add_first(Item,List,"Result") :: Adds item to first position of list
% and outputs "Result" = ...

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

% ------------------------------------------------------------------------
% --- Add_last(Item,List,"Result") :: Adds item to last position of list
% and outputs "Result" = ...

        add_last(X,[],[X]).
```

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

% ----------------------------------------------------------------------
% --- Iota(N,IotaN) :: Creates in order number list of N length starting at 1

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

% ----------------------------------------------------------------------
% --- Pick(List,"Item") :: Selects and outputs random item from list as "Item" = ...

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

% ----------------------------------------------------------------------
% --- Make_set(List,"Set") :: Outputs elements of list with more than one
% representation as "Set" = ...

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

% ----------------------------------------------------------------------
% Extended Code...
% ----------------------------------------------------------------------

% ----------------------------------------------------------------------
% --- product(Numlist,"Product") :: Takes a list of numbers as an input parameter
% and produces the product of the numbers in the list as an output parameter and
% outputs the result as "Product" = ...

		product([],1).
		product([Head|Tail],Product) :-
			product(Tail,ProductOfTail),
			Product is Head * ProductOfTail.

% ----------------------------------------------------------------------
% --- factorial(Num,"Result") ::  Takes a positive integer as an input
% parameter and produces the factorial of the given number as an output
% parameter in the form "Result" = ...

		factorial(0,0).
		factorial(Num,Result) :- iota(Num,Iota),
		product(Iota,Product),
		Result is Product.

% ----------------------------------------------------------------------
% --- make_list(Num,Dataitem,"List") :: Takes a nonnegative integer as
% its first input parameter, a data item as its second input parameter,
% and which produces a list consisting of the specified number of
% occurrences of the specified piece of data as its output parameter
% in the form "Result" = ...

		make_list(0,_,[]).
		make_list(Num,DataItem,List) :-
			K is Num - 1,
			make_list(K,DataItem,ListK),
			add_last(DataItem,ListK,List).
```

```
% ------------------------------------------------------------------------
% --- but_first(List,"Result") :: Takes a non-empty list as input parameter
% and produces the “cdr” of the list as its output parameter in the form
% "Result" = ...

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


% ------------------------------------------------------------------------
% --- but_last(List,"Result") ::  Takes a non-empty list as input parameter
% and produces the “rdc” of the list as its output parameter making use of
% the primitive predicate reverse two times in defining this predicate
% in the form "Result" = ...

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


% ------------------------------------------------------------------------
% --- is_palindrome(List) ::  Takes a list as its sole parameter and succeeds
% only if the list is a palindrome using first, last, but_first, and but_last
% while also using recursion. Outputs true or false.

        is_palindrome([]).
        is_palindrome([_]).
        is_palindrome(List) :-
                first(List,First), last(List,Last),
                First = Last,
                but_first(List,A), but_last(A,B),
                is_palindrome(B).

% ------------------------------------------------------------------------
% --- noun_phrase(List,"Phrase") ::  Produces a noun phrase of length three
% consisting of the word “the” followed by an adjective selected at random
% from a list of six adjectives followed by a noun selected at random from
% a list of eight nouns in the form "Phrase" = ...

        noun_phrase(Phrase) :-
                pick([captivating,rude,zesty,wrathful,ordinary,joyous],Adj),
                pick([investment,instance,recipe,apple,president,solution,possibility,strategy],Noun),
                add_last(Adj,[the],Start), add_last(Noun,Start,Phrase).

% ------------------------------------------------------------------------
% --- sentence(List,"Sentence") ::  Produces a sentence consisting of a random
% noun phrase followed by a past tense verb followed by a noun phrase using
% the the tokennoun_phrase predicate as well as the pick predicate in the
% form “Sentence” = …

        sentence(Sentence) :-
                noun_phrase(NP1),
                noun_phrase(NP2),
                pick([ate,assured,fixed,zapped,composed,justified,sketched],Verb),
                add_last(Verb,NP1,Temp),
                append(Temp,NP2,Sentence).
```

Task 4: List Processing in Prolog – Example List Processors Demo

```
Welcome to SWI-Prolog (threaded, 64 bits, version 8.4.3)
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('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 = blueberry ,

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

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

```
?- 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 = 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 = blueberry ,

?- 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] ,

?- 
```

Task 4: List Processing in Prolog – List Processing Exercises Demo

```
Welcome to SWI-Prolog (threaded, 64 bits, version 8.4.3)
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('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,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, zesty, strategy] ;
false.

?- noun_phrase(NP).
NP = [the, captivating, apple] .

?- noun_phrase(NP).
NP = [the, captivating, recipe] .

?- noun_phrase(NP).
NP = [the, ordinary, president] .

?- noun_phrase(NP).
NP = [the, wrathful, solution] .

?- sentence(S).
S = [the, ordinary, president, ate, the, captivating, recipe] .

?- sentence(S).
S = [the, joyous, recipe, composed, the, wrathful, investment] .

?- sentence(S).
S = [the, captivating, investment, fixed, the, captivating, instance] .

?- sentence(S).
S = [the, captivating, possibility, fixed, the, ordinary, instance] .

?- sentence(S).
S = [the, joyous, strategy, zapped, the, rude, instance] .

?- sentence(S).
S = [the, joyous, apple, zapped, the, joyous, instance] .

?- sentence(S).
S = [the, joyous, solution, fixed, the, rude, investment] .

?- sentence(S).
S = [the, rude, president, assured, the, joyous, recipe] .

?- sentence(S).
S = [the, wrathful, president, composed, the, wrathful, solution] .

?- sentence(S).
S = [the, wrathful, recipe, ate, the, joyous, investment] .

?- sentence(S).
S = [the, joyous, solution, assured, the, wrathful, president] .

?- sentence(S).
S = [the, ordinary, instance, ate, the, joyous, strategy] .

?- sentence(S).
S = [the, ordinary, apple, ate, the, zesty, instance] .

?- sentence(S).
S = [the, ordinary, investment, justified, the, rude, strategy] .

?- sentence(S).
S = [the, rude, possibility, assured, the, rude, investment] .

?- sentence(S).
S = [the, ordinary, instance, composed, the, rude, recipe] .

?-
```