Prolog Programming Assignment #1: Various Computations

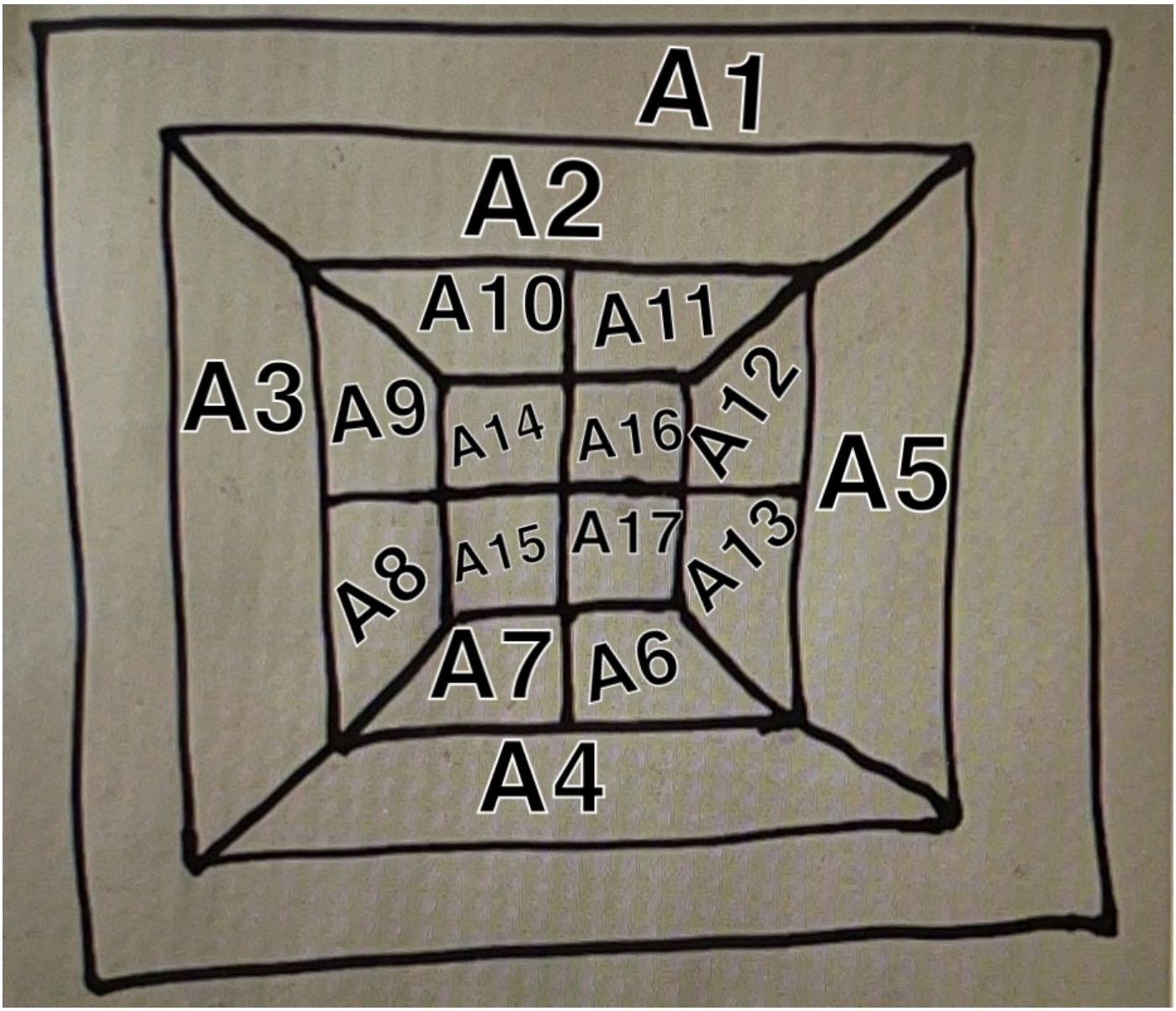
◇◇◇◇◇◇◇◇◇◇◇◇◇◇◇◇◇◇◇◇◇◇◇◇◇◇◇◇◇◇◇◇..

Learning Abstract:

I learned how to do programming exercises in Prolog based on knowledge representation, search, and list processing.

◇◇◇◇◇◇◇◇◇◇◇◇◇◇◇◇◇◇◇◇◇◇◇◇◇◇◇◇◇◇◇◇..

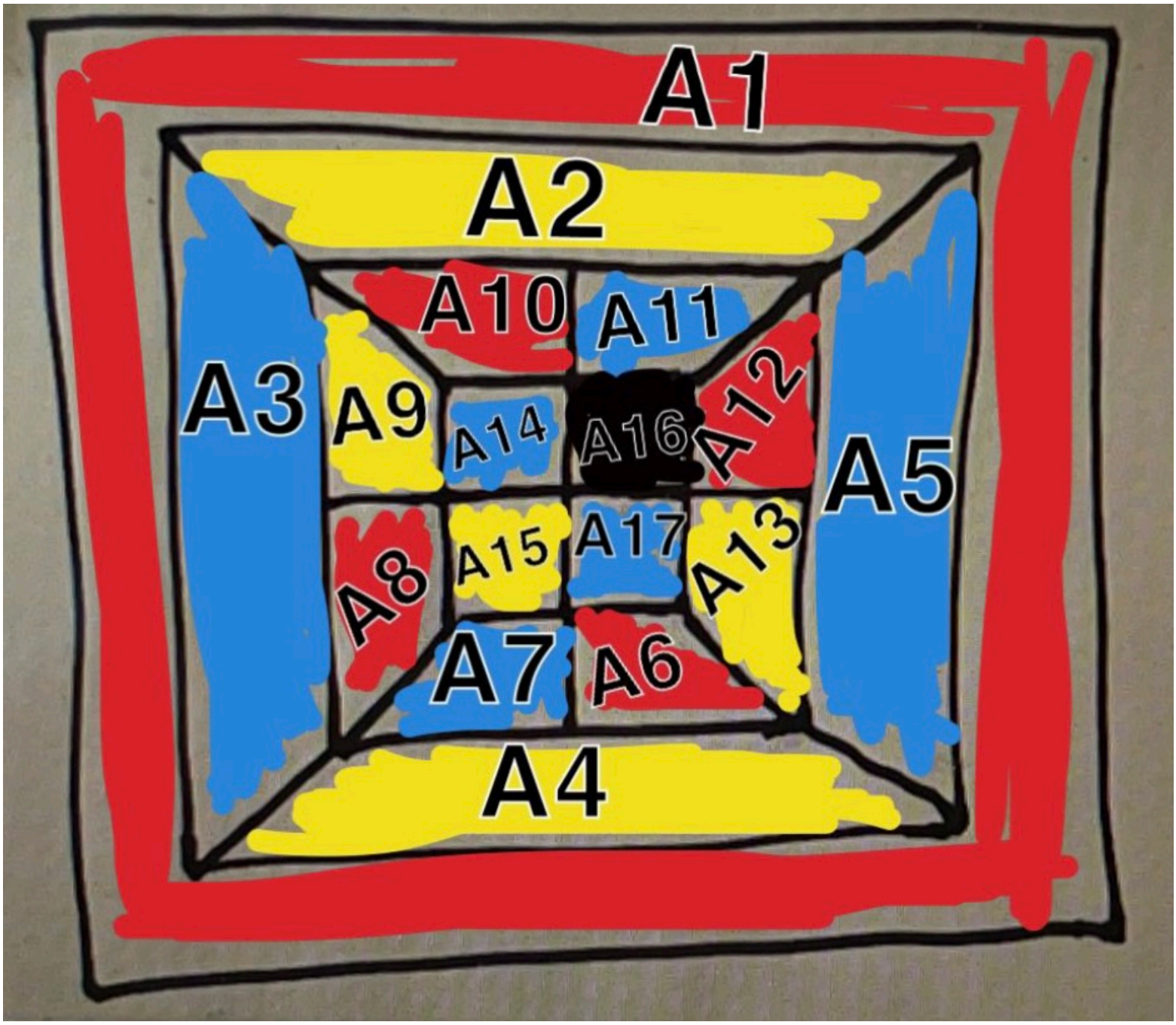
Task 1: Map Coloring



```
different(red,  yellow).
different(red, blue).
different(red, black).
different(yellow, red).
different(yellow, blue).
different(yellow, black).
different(blue, yellow).
different(blue, red).
different(blue, black).
different(black, red).
different(black, yellow).
different(black, blue).

coloring(A1, A2, A3, A4, A5, A6 ,A7, A8, A9, A10, A11, A12, A13, A14, A15, A16,A17):-
    different(A1, A2),
    different(A1, A3),
    different(A1, A4),
    different(A1, A5),
    different(A2, A3),
    different(A2, A10),
    different(A2, A11),
    different(A2, A5),
    different(A3, A9),
    different(A3, A8),
    different(A3, A4),
    different(A4, A7),
    different(A4, A6),
    different(A4, A5),
    different(A5, A13),
    different(A5, A12),
    different(A6, A7),
    different(A6, A17),
    different(A6, A13),
    different(A7, A15),
    different(A7, A8),
    different(A8, A15),
    different(A8, A9),
    different(A9, A14),
    different(A9, A10),
    different(A10, A14),
    different(A10, A11),
    different(A9, A14),
    different(A9, A10),
    different(A10, A14),
    different(A10, A11),
    different(A11, A16),
    different(A11, A12),
    different(A12, A16),
    different(A12, A13),
    different(A13, A17),
    different(A14, A15),
    different(A14, A16),
    different(A15, A17),
    different(A16, A17).
```

```
?- coloring(A1, A2, A3, A4, A5, A6 ,A7, A8, A9, A10, A11, A12, A13, A14, A15, A16,A17).
A1 = A6, A6 = A8, A8 = A10, A10 = A12, A12 = red,
A2 = A4, A4 = A9, A9 = A13, A13 = A15, A15 = A16, A16 = yellow,
A3 = A5, A5 = A7, A7 = A11, A11 = A14, A14 = A17, A17 = blue ;
A1 = A6, A6 = A8, A8 = A10, A10 = A12, A12 = red,
A2 = A4, A4 = A9, A9 = A13, A13 = A15, A15 = yellow,
A3 = A5, A5 = A7, A7 = A11, A11 = A14, A14 = A17, A17 = blue,
A16 = black .
```

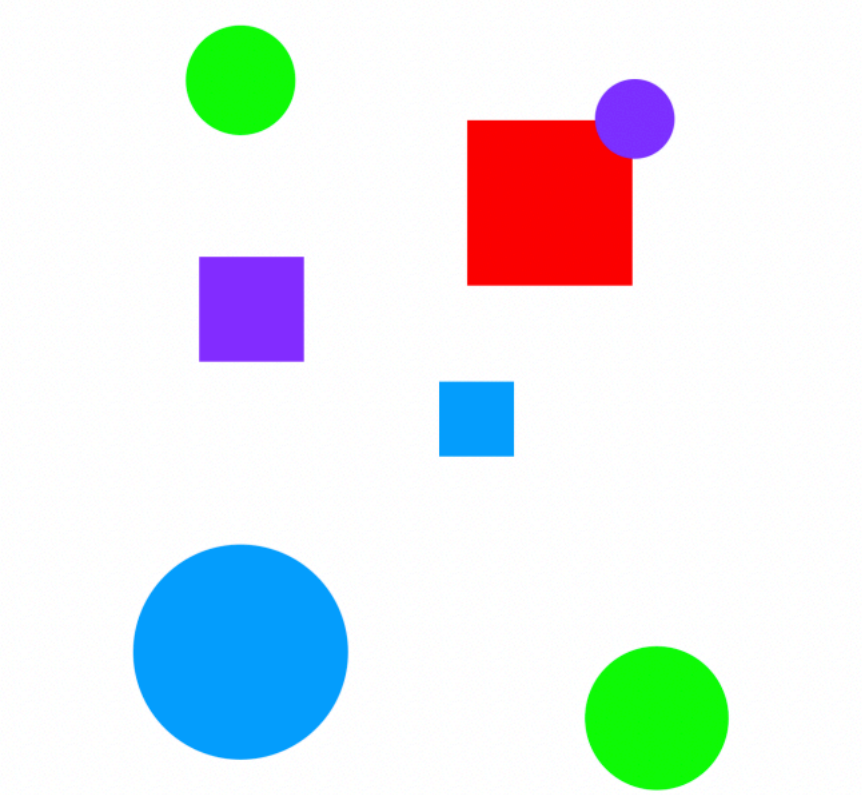


Task 2: The Floating Shapes World

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

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

circles :- circle(Name,_,_), write(Name),nl,fail.
circles.
squares :- square(Name,_,_), write(Name),nl,fail.
squares.
shapes :- circles,squares.
blue(Name) :- square(Name,_,color(blue)).
blue(Name) :- circle(Name,_,color(blue)).
large(Name) :- area(Name,A), A >= 100.
small(Name) :- area(Name,A), A < 100.
area(Name,A) :- circle(Name,radius(R),_), A is 3.14 * R * R.
area(Name,A) :- square(Name,side(S),_), A is S * S.
```



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

```
?- 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 .

?- 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 - Code:

```
% --- 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(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)).
display_names :- pokemon(name(N),_,_,_),write(N),nl,fail.
display_attacks :-
pokemon(_,_,_,attack(Ok,_)),write(Ok),nl,fail.
powerful(Name) :- pokemon(name(Name),_,_,attack(_,A)),A>55.
tough(Name) :- pokemon(name(Name),_,hp(H),_),H>100.
type(Name,Power) :- pokemon(name(Name),Power,_,_).
dump_kind(Element) :- pokemon(Name,Element,Hp,Attack),
write(pokemon(Name,Element,Hp,Attack)),nl,false.
display_cen :- cen(Name),write(Name),nl,fail.
family(Name) :- evolves(Name,X), write(Name), write(" "),
write(X), evolves(X,Y),write(" "), write(Y).
families :- cen(Name), evolves(Name,X), nl, write(Name), write("
"), write(X), evolves(X,Y),write(" "), write(Y),fail.
lineage(Name) :- pokemon(name(Name),Type,hp(H),attack(Attack,Damage)),
write(pokemon(name(Name),Type,hp(H),attack(Attack,Damage))),nl,
evolves(Name,X), pokemon(name(X),Type2,hp(I),attack(Atk2,Dmg2)),
write(pokemon(name(Y),Type2,hp(I),attack(Atk2,Dmg2))),nl,
evolves(X,Y), pokemon(name(Y),Type3,hp(J),attack(Atk3,Dmg3)),
write(pokemon(name(Y),Type3,hp(J),attack(Atk3,Dmg3))).
```

1. Query 1: Is picachu a “creatio ex nihilo” (created out of nothing) pokemon?

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

2. Query 2: Is raichu a “creatio ex nihilo” pokemon?

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

3. Query 3: By means of hand intervention, list all of the “creatio ex nihilo” pokemon.

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

4. Query 4: By means of the standard idiom of repetition, list all of the “creatio ex nihilo” pokemon.

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

5. Query 5: Does squirtle evolve into wartortle?

```
?- evolves(squirtle,wartortle).
true.
```

6. Query 6: Does wartortle evolve into squirtle?

```
?- evolves(wartortle,squirtle).
false.
```

7. Query 7: Does squirtle evolve into blastoise?

```
?- evolves(squirtle,blastoise).
false.
```

8. Query 8: By means of hand intervention, list all triples of pokemon such that the first evolves into the second and the second evolves into the third.

```
?- 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.
```

9. Query 9: By means of the standard idiom of repetition, list all pairs of pokemon such that the first evolves through an intermediary to the second - placing an arrow between each pair.

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

10. Query 10: By means of the standard idiom of repetition, list the names of all of the pokemon.

```
?- 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.
```

11. Query 11: By means of the standard idiom of repetition, list the names of all of the fire pokemon.

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

12. Query 12: By means of the standard idiom of repetition, provide a summary of each pokemon and its kind, representing each pairing of name and kind in the manner suggested by the redacted demo.

```
?- pokemon(N,Element,_,_),write(nks(N,kind(Element))),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.
```

13. Query 13: What is the name of the pokemon with the waterfall attack?

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

14. Query 14: What is the name of the pokemon with the poison-powder attack?

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

15. Query 15: By means of the standard idiom of repetition, list the names of the attacks of all of the water pokemon.

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

16. Query16: How much damage (hp count) can poliwhirl absorb?

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

17. Query 17: How much damage (hp count) can butterfree absorb?

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

18. Query 18: By means of the standard idiom of repetition, list the names of all of the pokemon that can absorb more than 85 units of damage.

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

19. Query 19: By means of the standard idiom of repetition, list the names of all of the pokemon that can dish out more than 60 units of damage with one instance of their attack.

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

20. Query 20: By means of the standard idiom of repetition, list the names and the hit point value for each of the "creation ex nihilo" pokemon, with the results formatted as the redacted demo suggests.

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

Task 4: Lisp Processing in Prolog Head/Tail Referencing Exercises

Head/Tail Referencing Exercises

?- [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)].
```

Prolog Code based on Example List Processors

```
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).
```

Demo for Example List Processors^

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

Prolog Code based on Specifications of List Processing Exercises

```
first([H|_], H).
last([H|[]], H).
```

```
last([_|T], Result) :- last(T, Result).
iota(0,[]).
nth(0,[H|_],H).
nth(N,[_|T],E) :- K is N - 1, nth(K,T,E).
iota(N,IotaN) :- K is N - 1, iota(K,IotaK), add_last(N,IotaK,IotaN).
add_first(X,L,[X|L]).
add_last(X,[],[X]).
add_last(X,[H|T],[H|TX]) :- add_last(X,T,TX).
pick(L,Item) :- length(L,Length), random(0,Length,RN), nth(RN,L,Item).
product([],1).
product([Head|Tail],Product) :- product(Tail,SumOfTail), Product is Head *
SumOfTail. factorial(0,0).
factorial(X,Y) :- iota(X,Iota), product(Iota,Product), Y is Product.
make_list(0,_,[]). make_list(Number,Anything,Name) :- X is Number - 1,
make_list(X,Anything,EleK), add_last(Anything,EleK,Name).
but_first([],[]).
but_first([_],[]).
but_first([_|E],E).
but_last([],[]).
but_last([_],[]).
but_last([H|T], Name) :- reverse(T, [_|X]),reverse(X,
RDC),add_first(H,RDC,Name). is_palindrome([]).
is_palindrome([_]).
is_palindrome(Index) :- first(Index,First), last(Index,Last), First = Last,
but_first(Index,X), but_last(X,Y), is_palindrome(Y).
noun_phrase(Name) :- pick([adorable, angry, adventurous, doubtful,
powerful, thoughtful],Adjective), pick([airport, pizza, island, dog, cup, ghost,
image, napkin],Noun), add_last(Adjective,[the],First),
add_last(Noun,First,Name).
sentence(Name) :- noun_phrase(A), noun_phrase(B), pick([ate, flew, wrote,
built, touched, became, saw],Verb), add_last(Verb,A,X), append(X,B,Name).
```

Demo based on Specifications of List Processing Exercises

```
?- 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, doubtful, airport] .
?- noun_phrase(NP).
NP = [the, doubtful, napkin] .
?- noun_phrase(NP).
NP = [the, angry, island] .
?- noun_phrase(NP).
NP = [the, thoughtful, airport] .
?- noun_phrase(NP).
NP = [the, doubtful, island] .
?- sentence(S).
S = [the, adorable, dog, wrote, the, powerful, image] .
?- sentence(S).
```

S = [the, powerful, island, touched, the, adventurous, ghost] .
?- sentence(S).
S = [the, adorable, cup, ate, the, adventurous, napkin] .
?- sentence(S).
S = [the, adorable, image, wrote, the, adventurous, pizza] .
?- sentence(S).
S = [the, thoughtful, image, flew, the, thoughtful, pizza] .
?- sentence(S).
S = [the, angry, cup, built, the, adorable, napkin] .
?- sentence(S).
S = [the, adorable, airport, built, the, adventurous, cup] .
?- sentence(S).
S = [the, doubtful, cup, touched, the, powerful, airport] .
?- sentence(S).
S = [the, adventurous, napkin, wrote, the, doubtful, airport] .
?- sentence(S).
S = [the, angry, pizza, became, the, powerful, image] .
?- sentence(S).
S = [the, powerful, cup, flew, the, doubtful, napkin] .
?- sentence(S).
S = [the, adventurous, pizza, became, the, adorable, ghost] .
?- sentence(S).
S = [the, doubtful, image, built, the, thoughtful, napkin] .
?- sentence(S).
S = [the, thoughtful, airport, touched, the, doubtful, airport] .
?- sentence(S).
S = [the, adorable, airport, became, the, doubtful, island] .
?- sentence(S).
S = [the, powerful, dog, built, the, powerful, pizza] .