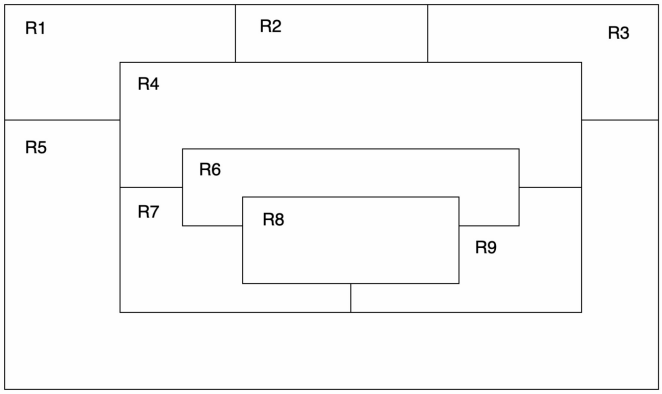
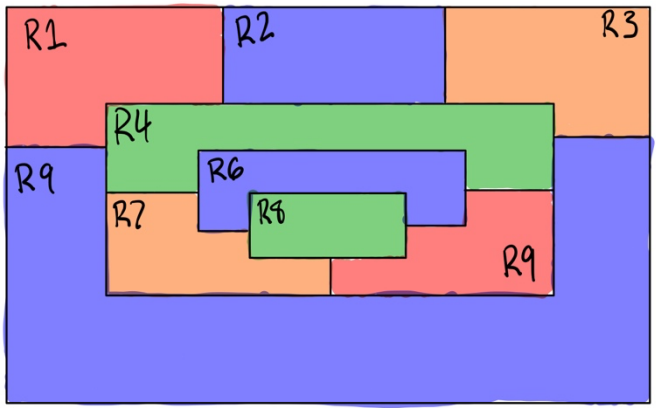
First Prolog Programming Assignment Solution

Task 1: Map Coloring






```
% ------------------------------------------------------------------------
% File: coloring.pro
% Line: Program to find a 4 color map rendering for the Task 1 Map.
% More: The colors used will be red, blue, green orange.
% ------------------------------------------------------------------------

% different(X,Y) :: X is not equal to Y
different(red,blue).
different(red,green).
different(red,orange).
different(green,blue).
different(green,orange).
different(green,red).
different(blue,green).
different(blue,orange).
different(blue,red).
different(orange,blue).
different(orange,green).
different(orange,red).
% ------------------------------------------------------------------------
coloring(R1, R2, R3, R4, R5, R6, R7, R8, R9) :-
        different(R1, R2),
        different(R1, R4),
        different(R1, R5),
        different(R2, R3),
        different(R2, R4),
        different(R3, R4),
        different(R3, R5),
        different(R4, R5),
        different(R4, R6),
        different(R4, R7),
        different(R4, R9),
        different(R5, R7),
        different(R5, R9),
        different(R6, R7),
        different(R6, R8),
        different(R6, R9),
        different(R7, R6),
        different(R7, R8),
        different(R7, R9),
        different(R8, R9).
```

```
?- consult("/Users/joseph/Documents/Workspace/Prolog Files/coloring.pro").
true.

?- coloring(R1,R2,R3,R4,R5,R6,R7,R8,R9).
R1 = R9, R9 = red,
R2 = R5, R5 = R6, R6 = blue,
R3 = R7, R7 = orange,
R4 = R8, R8 = green
```

Task 2: The Floating Shapes World

```
?- consult("/Users/joseph/Documents/Workspace/Prolog Files/FloatingShapes.pro").
true.

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

true.

?- squares.
sera
sara
sarah
true.

?- circles.
carla
cora
connie
claire
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 .
```

Task 3: Pokemon KB Interaction and Programming

Part 1: Queries

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

?- cen(raichu).
false.

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

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

```
?- pokemon(name(N), K, _,_), format("nks(name(~a), kind(~a))",[N, K]), 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(Atk,_)), write(Atk),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(_,_,_,attack(N,D)), D > 60, write(N),nl,fail.
thunder-shock
poison-powder
whirlwind
royal-blaze
fire-blast
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.
```

Part 2: Programs

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

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

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

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

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

dump_kind(T) :- pokemon(N,T,HP,ATK), write(pokemon(N, T, HP ,ATK)), nl, fail.

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

family(CEN) :- cen(CEN), write(CEN),write(" "), evolves(CEN, X), write(X), write(" "), evolves(X, Y),
write(Y).

families :- cen(CEN), family(CEN), nl, fail. families.

lineage(N) :- pokemon(name(N), T, HP, ATK), write(pokemon(name(N), T, HP, ATK)),nl, evolves(N,X),
pokemon(name(X), XT, XHP, XATK), write(pokemon(name(X), XT, XHP, XATK)),nl ,evolves(X, Y),
pokemon(name(Y), YT, YHP, YATK), write(pokemon(name(Y), YT, YHP, YATK)).
```

```
?- 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(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.

?- 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.

?- display_cen.
pikachu
bulbasaur
caterpie
charmander
vulpix
poliwag
squirtle
staryu
true.

?- family(pikachu).
pikachu raichu
false.

?- 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))
false.

Task 4: Lisp Processing in Prolog

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


factorial(N) :- iota(N, List), product(List, Product), write(Product).

make_list(0,_,[]).
make_list(LEN, WORD, List) :-
        NEWLEN is LEN - 1,
        make_list(NEWLEN, WORD,NEWList),
        add_first(WORD, NEWList, List).

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

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



is_palendrome([]).
is_palendrome([_]).
is_palendrome(List) :-
        first(List, A),
        last(List, B),
        A = B,
        but_first(List,X),
        but_last(X, Y),
        is_palendrome(Y).

noun_phrase(NP) :-
        pick([confused, tired, purple, creepy, helpful, cowardly], A),
        pick([ambulance, hair, pizza, dog, helmet, car, river, teacher], B),
        add_last(A, [the], Temp),
        add_last(B, Temp, NP).


sentence(S) :-
        noun_phrase(NP1),
        noun_phrase(NP2),
        pick([pushed, knew, helped, scared, walked, held], V),
        add_first(V, NP1, Temp),
        append(NP2, Temp, S).
```

```
?- make_list(0, hey, List).
List = [] .

?- make_list(5, hey, List).
List = [hey, hey, hey, hey, hey] .

?- product([3,5,2], Product).
Product = 30.

?- factorial(5).
120
true .

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

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

?- is_palendrome([]).
true .

?- is_palendrome([ab]).
true .

?- is_palendrome([a,b,c,d,e,d,c,b,a]).
true .

?- is_palendrome([a,c,c,d,e,d,c,b,a]).
false.

?- noun_phrase(NP).
NP = [the, creepy, pizza] .

?- sentence(S).
S = [the, tired, car, held, the, creepy, river] .
```

```
?- consult("/Users/joseph/Documents/Workspace/Prolog Files/list_processors.pro").
true.

?- first([apple,bannana, orange], 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].

?- 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([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].

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

?- 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] .
```