Sven Kappeler - Prolog Assignment 1:

Learning Abstract

This assignment is an introduction to the Racket programming language. The purpose of this assignment is to get myself acquainted with recursive functions and lambda functions whilst using the Racket programming language.

Task 1: Map Coloring

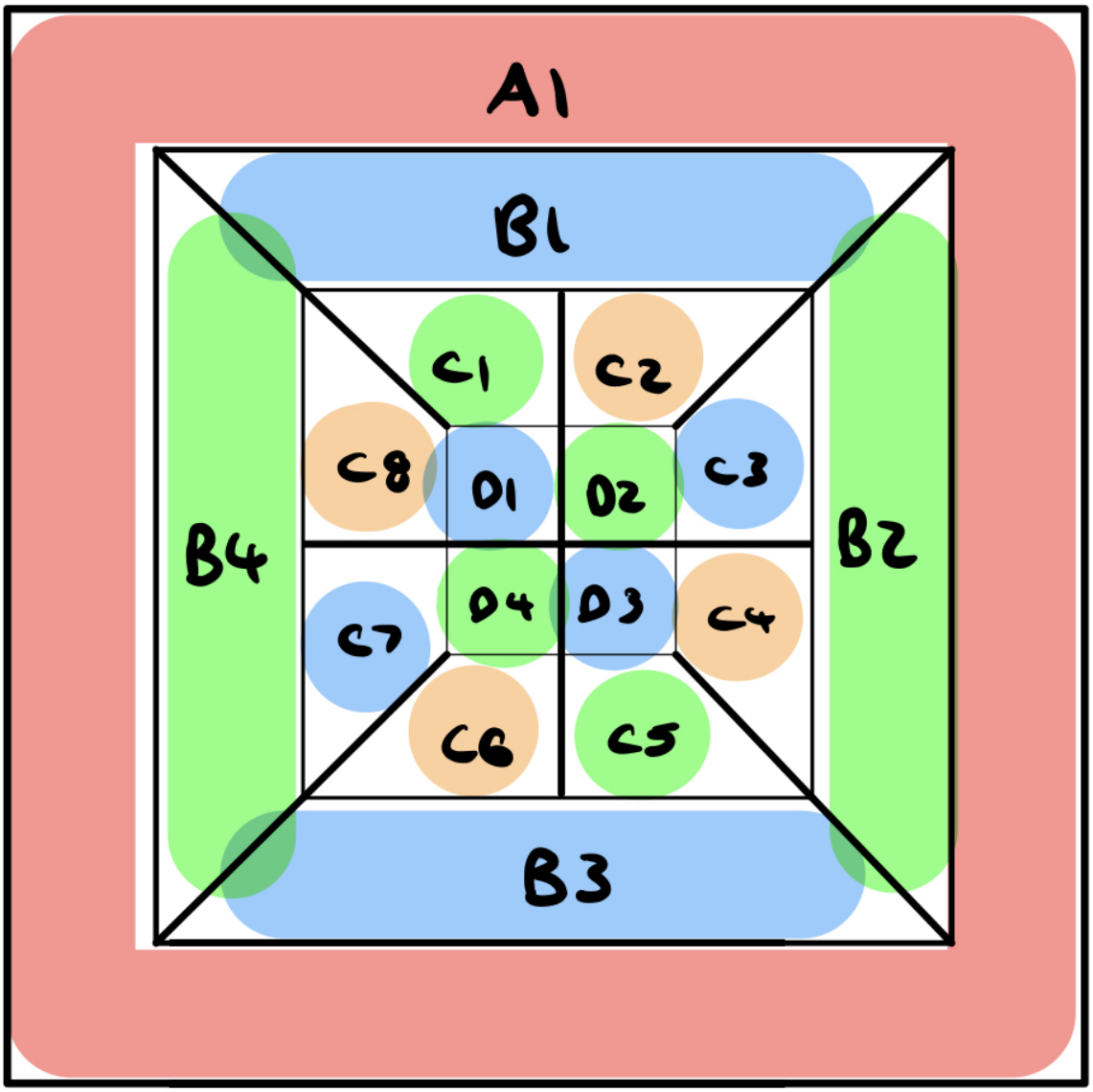
```
% Different Colors

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).

% Touching Sides

coloring(A1,B1,B2,B3,B4,C1,C2,C3,C4,C5,C6,C7,C8,D1,D2,D3,D4) :-
    different(A1,B1),
    different(A1,B2),
    different(A1,B3),
    different(A1,B4),

    different(B1,B2),
    different(B1,B4),
    different(B1,C1),
    different(B1,C2),

    different(B2,C3),
    different(B2,C4),
    different(B2,B3),

    different(B3,C5),
    different(B3,C6),
    different(B3,B4),

    different(B4,C7),
    different(B4,C8),

    different(C1,C2),
    different(C1,D1),
    different(C1,C8),
```

```
    different(C2,D2),
    different(C2,C3),

    different(C3,D2),
    different(C3,C4),

    different(C4,D3),
    different(C4,C5),

    different(C5,D3),
    different(C5,C6),

    different(C6,C7),
    different(C6,D4),

    different(C7,D4),
    different(C7,C8),

    different(C8,D1),

    different(D1,D2),
    different(D1,D4),

    different(D2,D3),

    different(D3,D4).
```

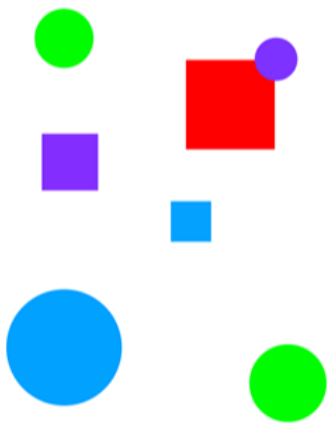
coloring(A1,B1,B2,B3,B4,C1,C2,C3,C4,C5,C6,C7,C8,D1,D2,D3,D4).

A1 = red,
B1 = B3, **B3** = C3, **C3** = C7, **C7** = D1, **D1** = D3, **D3** = blue,
B2 = B4, **B4** = C1, **C1** = C5, **C5** = D2, **D2** = D4, **D4** = green,
C2 = C4, **C4** = C6, **C6** = C8, **C8** = orange

```
?- coloring(A1,B1,B2,B3,B4,C1,C2,C3,C4,C5,C6,C7,C8,D1,D2
,D3,D4).
```


A1
B1
B2
B3
B4
C1
C2
C3
C4
C5
C6
C7
C8
D1
D2
D3
D4

Task 2: Floating Shapes World



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

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

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.
```

```
?- listing(squares),
   squares,
   listing(circles),
   circles,
   listing(shapes),
   shapes.
```

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

sera
sara
sarah

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

carla
cora
connie
claire

```
shapes :-
    circles,
    squares.
```

carla
cora
connie
claire
sera
sara
sarah
true 1

```
Shape = sara
Shape = cora
```

```
?- blue(Shape).
```

```
cora
sarah
false
```

```
?- large(Name),write(Name),nl,fail.
```

```
carla
connie
claire
sera
sara
false
```

```
?- small(Name),write(Name),nl,fail.
```

```
A = 153.86
Next | 10 | 100 | 1,000 | Stop
```

```
?- area(cora,A).
```

```
A = 50.24
Next | 10 | 100 | 1,000 | Stop
```

```
?- area(carla,A).
```

Task 3: Pokemon KB Interaction and Programming

A)

```
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(Name,Name2), evolves(Name2,Name3).
```

Name = bulbasaur,
Name2 = ivysaur,
Name3 = venusaur
Name = caterpie,
Name2 = metapod,
Name3 = butterfree
Name = charmander,
Name2 = charmeleon,
Name3 = charizard
Name = poliwag,
Name2 = poliwhirl,
Name3 = poliwrath
Name = squirtle,
Name2 = wartortle,
Name3 = blastoise
false

```
evolves(Name,Name2), evolves(Name2,Name3), write(Name), write("->"), write(Name3), nl, fail.
```

bulbasaur->venusaur
caterpie->butterfree
charmander->charizard
poliwag->poliwrath
squirtle->blastoise
false

10)

```
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(Name),_,_,attack(waterfall,_)).
```

Name = wartortle

```
pokemon(name(Name),_,_,attack(poison-powder,_)).
```

Name = venusaur

```
pokemon(_,water,_,attack(NameA,_)), write(NameA), 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(X),_), X >= 85, write(Name), nl, fail.
```

raichu
venusaur
butterfree
charizard
ninetails
poliwrath
blastoise
false

```
pokemon(_,_,_,attack(Name,X)), X > 60, write(Name), nl, fail.
```

thunder-shock
poison-powder
whirlwind
royal-blaze
fire-blast
false

```
pokemon(name(Name),_,hp(X),_), cen(Name), write(Name), write(": "),
write(X), nl, fail.
```

pikachu: 60
bulbasaur: 40
caterpie: 50
charmander: 50
vulpix: 60
poliwag: 60
squirtle: 40
staryu: 40
false

B)

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

display-names :-
    pokemon(name(Name),_,_,_),
    write(Name),
    nl,
    true.

display-attacks :-
    pokemon(_,_,_,attack(Name,_)),
    write(Name),
    nl,
    true.

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

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

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

dump_kind(T) :-
    pokemon(Name,T,HP,Attack),
    write(pokemon(Name,T,HP,Attack)),
    nl,
    fail.

dump_name(N) :-
    pokemon(name(N),Type,HP,Attack),
    write(pokemon(name(N),Type,HP,Attack)),
    nl,
    true.

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


family(Name) :-
    evolves(Name,Name2),
    evolves(Name2,Name3),
    write(Name),
    write(" "),
    write(Name2),
    write(" "),
    write(Name3),
    true.

family(Name):-
    evolves(Name,Name2),
    \+ evolves(Name2,_),
    write(Name),
    write(" "),
    write(Name2),
    true.
```

```
families :-
    cen(N),
    family(N),
    nl,
    true.

lineage(Name) :-
    evolves(Name,Name2),
    evolves(Name2,Name3),
    dump_name(Name),
    dump_name(Name2),
    dump_name(Name3),
    true.

lineage(Name) :-
    evolves(Name,Name2),
    \+ evolves(Name2,_),
    dump_name(Name),
    dump_name(Name2),
    true.

lineage(Name) :-
    \+ evolves(Name,_),
    dump_name(Name).
```

---

display-names.

pikachu
true
raichu
bulbasaur
ivysaur
venusaur
caterpie
metapod
butterfree
charmander
charmeleon
charizard
vulpix
ninetails
poliwag
poliwhirl
poliwrath
squirtle
wartortle
blastoise
staryu
starmie

display-attacks.

gnaw
true
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

powerful(pikacu).

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
false

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
false

family(pikachu).

pikachu raichu

true 1

family(squirtle).

squirtle wartortle blastoise

true 1

false

families.

pikachu raichu

true 1

bulbasaur ivysaur venusaur

caterpie metapod butterfree

charmander charmeleon charizard

vulpix ninetails

poliwag poliwhirl poliwrath

squirtle wartortle blastoise

staryu starmie

lineage(metapod).

*pokemon(name(*metapod),grass,*hp(*70),*attack(*stun-spore,20))

*pokemon(name(*butterfree),grass,*hp(*130),*attack(*whirlwind,80))

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(butterfree).

*pokemon(name(*butterfree),grass,*hp(*130),*attack(*whirlwind,80))

true

Task 4: List Processing in Prolog

```
[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,
R = [blue, green],
S = yellow

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

[a,[b, c]] = [a, b, c].

false

[cell(Row,Column)|Rest] = [cell(1,1), cell(3,2), cell(1,3)].

Column = Row, Row = 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)]
```

```
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

Next 10 100 1,000 Stop

```
last([c,d,e,f,g,a,b],P).
```

P = b

false

```
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]

```
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

false

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

Pie = blueberry

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

Pie = blueberry

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

Pie = peach

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

Pie = apple

```
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]

C)

sentence(S).

S = [the, red, woman, robbed, the, tall, boy]

sentence(S).

S = [the, short, man, ate, the, short, chair]

sentence(S).

S = [the, tall, boy, loved, the, small, dog]

sentence(S).

S = [the, red, girl, ate, the, blue, boy]

sentence(S).

S = [the, blue, chair, liked, the, blue, boy]

sentence(S).

S = [the, short, dog, hugged, the, short, boy]

```
?- sentence(S).
```

noun_phrase(NP).

NP = [the, small, tables]

noun_phrase(NP).

NP = [the, tall, tables]

noun_phrase(NP).

NP = [the, blue, boy]

noun_phrase(NP).

NP = [the, blue, cat]

noun_phrase(NP).

NP = [the, small, woman]

```
?- noun_phrase(NP).
```

```
is_palindrome([x]).
```

true

```
is_palindrome([a,b,c]).
```

false

```
is_palindrome([a,b,b,a]).
```

true

```
but_first([a,b,c,d],X).
```

X = [b, c, d]

```
but_last([a,b,c,d],X).
```

X = [a, b, c]

```
make_list(10,2,List).
```

List = [2, 2, 2, 2, 2, 2, 2, 2, 2, 2]

```
make_list(10,'two',List).
```

List = [two, two, two, two, two, two, two, two, two, two]

```
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).

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

factorial(0,1).
factorial(Number,Factorial) :-
    iota(Number,Iota),
    product(Iota,Factorial).

make_list(0,_,[]).
make_list(Number,String,List) :-
    I is Number - 1,
    make_list(I,String,ListI),
    add_last(String,ListI,List).

but_first([_|T],List) :-
    List = T.
```

```
but_first(TL,List) :-
    length(TL,Length),
    Length = 1,
    List = [].

but_last(TL,List) :-
    reverse(TL,ReverseTL),
    but_first(ReverseTL,ReverseList),
    reverse(ReverseList,List).

is_palindrome(List) :-
    first(List,H),
    last(List,L),
    H == L,
    but_first(List,HList),
    but_last(HList,EList),
    is_palindrome(EList).

is_palindrome([]).
is_palindrome([_]).

noun_phrase(NP) :-
    pick([small,large,red,blue,tall,short], Adj),
    pick([boy,girl,man,woman,chair,tables,cat,dog],Noun),
    NP = [the,Adj,Noun].

sentence(S) :-
    pick([liked,high-fived,ate,hated,loved,robbed,hugged],PAdj),
    noun_phrase(NP1),
    noun_phrase(NP2),
    append(NP1,[PAdj],S1),
    append(S1,NP2,S).
```