Racket Assignment #5: RLP and HoFs

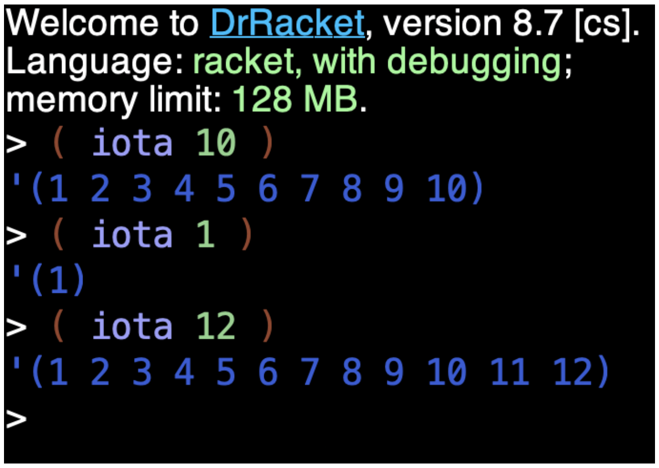
Abstract
Task 1 is dedicated to defining four simple list generators. Three of these require the use of recursion. One requires a classic application of higher order functions. Task 2 features programs generate number sequences by performing some interesting sorts of “counting.” These programs serve to channel one of Tom Johnson’s many “automantic composition” techniques. Task 3 affords you an opportunity to get acquainted with “association lists,” which are a classic data structure introduced in McCarthy’s original Lisp. This task also serves as a segue into Task 4, which pertains to the transformation of number sequences to musical notes represented in ABC notation. Task 5 channels Frank Stella, famous for (among other things) his nested squares. Task 6 simulates a cognitive phenomenon known as chromesthesia, the mapping of musical pitchs to colors. Task 7 simulations grapheme to color synesthesia, in which letters are mapped to colors.

Task 1: Simple List Generators

Task 1a - iota

```
#lang racket


( define ( iota n )
   ( define ( count-up i )

         ( if ( <= i n )
           ( cons i (count-up ( + i 1)))
          '()))
      ( count-up 1 )
)
```

Demo



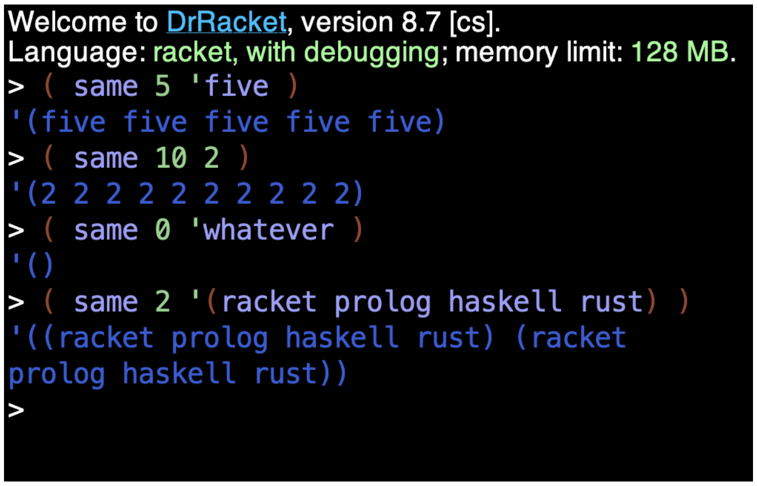
```
Welcome to DrRacket, version 8.7 [cs].
Language: racket, with debugging;
memory limit: 128 MB.
> ( iota 10 )
'(1 2 3 4 5 6 7 8 9 10)
> ( iota 1 )
'(1)
> ( iota 12 )
'(1 2 3 4 5 6 7 8 9 10 11 12)
>
```

Task 1b - Same

Function Definition

```
(define (same n liobj)
  (cond
    ((<= n 0) '())
    ((> n 0)
     (cons liobj (same (- n 1) liobj)))))
```

Demo



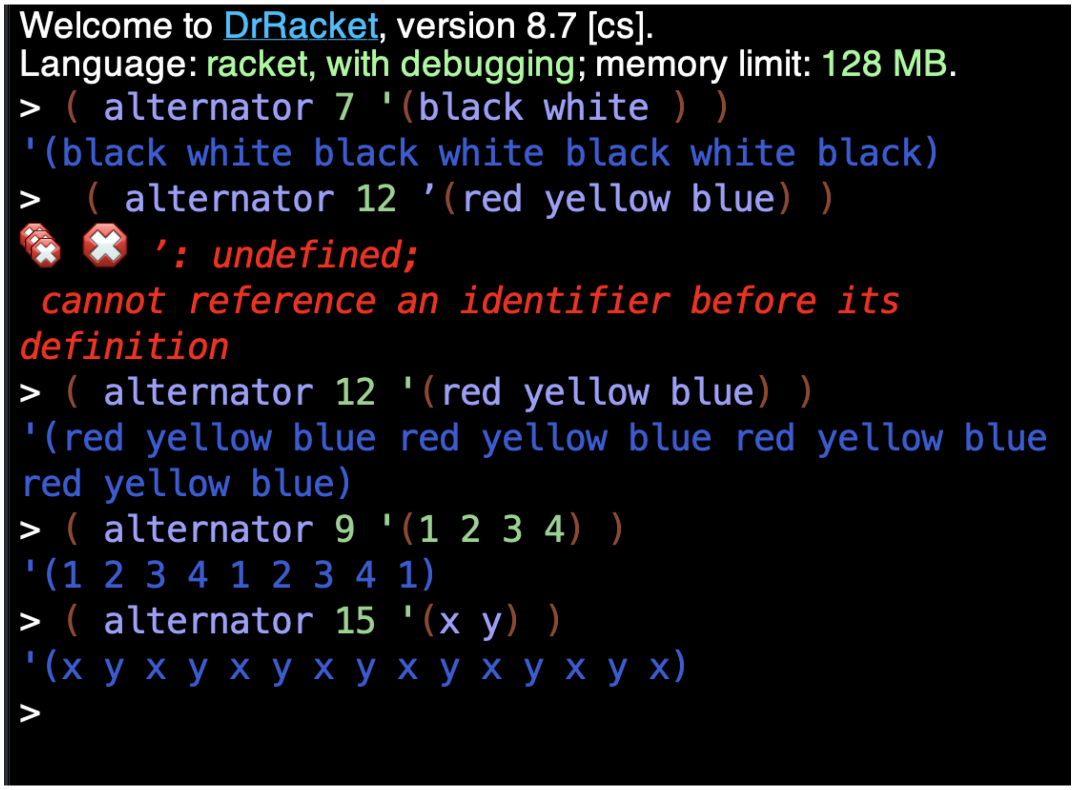
```
Welcome to DrRacket, version 8.7 [cs].
Language: racket, with debugging; memory limit: 128 MB.
> ( same 5 'five )
'(five five five five five)
> ( same 10 2 )
'(2 2 2 2 2 2 2 2 2 2)
> ( same 0 'whatever )
'()
> ( same 2 '(racket prolog haskell rust) )
'((racket prolog haskell rust) (racket
prolog haskell rust))
>
```

Task 1c - Alternator

```
( define ( alternator n liobj )
   ( cond
      (( <= n 0) '())
      ((> n 0 )
       ( cons ( car liobj)
              ( alternator ( - n 1 )
                          (append (cdr liobj) (list (car
liobj))))))))
```

Function Definition

Demo



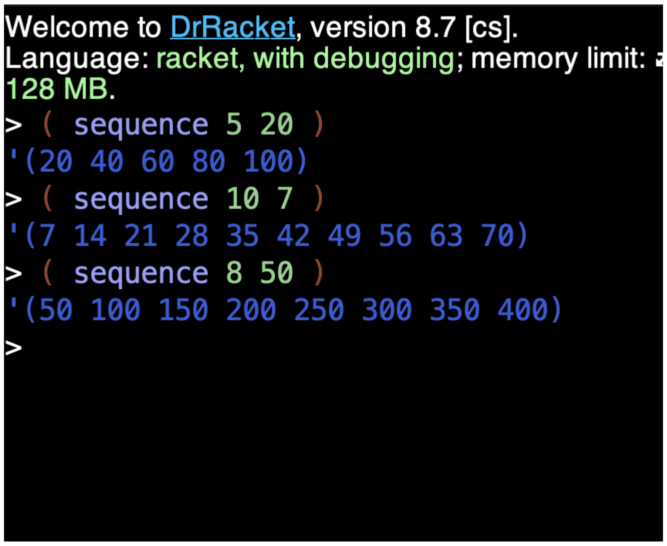
```
Welcome to DrRacket, version 8.7 [cs].
Language: racket, with debugging; memory limit: 128 MB.
> ( alternator 7 '(black white ) )
'(black white black white black white black)
>  ( alternator 12 '(red yellow blue) )
 ': undefined;
 cannot reference an identifier before its
definition
> ( alternator 12 '(red yellow blue) )
'(red yellow blue red yellow blue red yellow blue
red yellow blue)
> ( alternator 9 '(1 2 3 4) )
'(1 2 3 4 1 2 3 4 1)
> ( alternator 15 '(x y) )
'(x y x y x y x y x y x y x y x)
>
```

Task 1d - Sequence

Function Definition

```
( define (sequence n num )
   (cond
     (( <= n 0 ) '())
     (else
      ( map ( lambda(x)(* x num ))(iota n)))))
```

Demo



```
Welcome to DrRacket, version 8.7 [cs].
Language: racket, with debugging; memory limit:
128 MB.
> ( sequence 5 20 )
'(20 40 60 80 100)
> ( sequence 10 7 )
'(7 14 21 28 35 42 49 56 63 70)
> ( sequence 8 50 )
'(50 100 150 200 250 300 350 400)
>
```

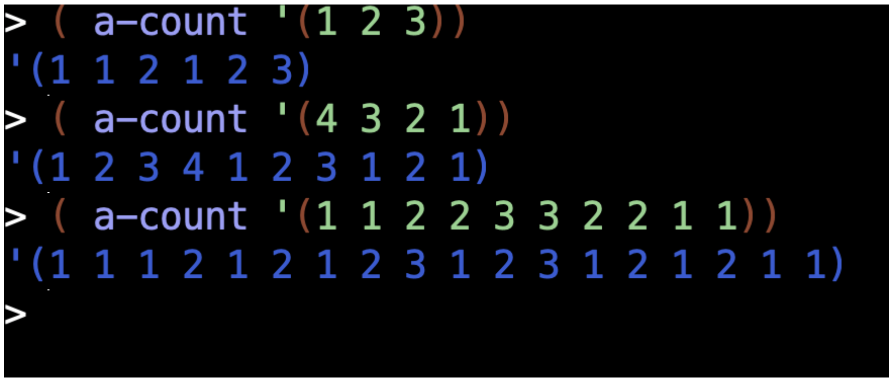
Task 2: Counting

Task 2a - Accumulation Counting

Function Definition

```
(define (a-count lst)
    (cond
      ((null? lst) '())
      ( else
        (append ( iota ( car lst) )(a-count (cdr lst))))))
```

Demo



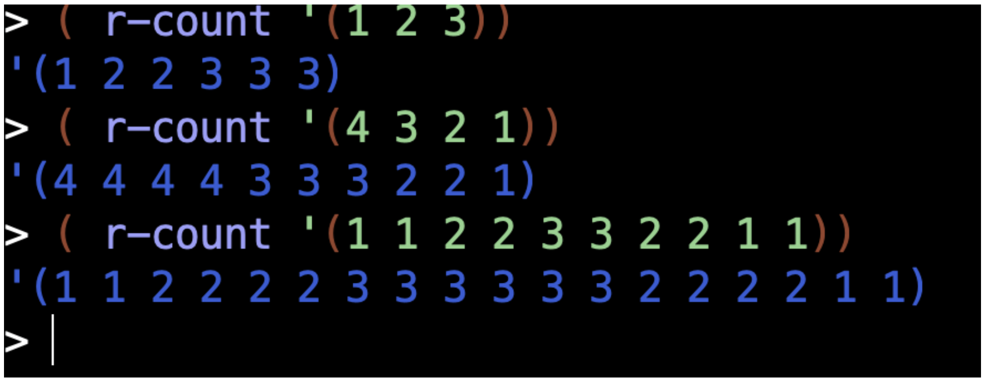
```
> ( a-count '(1 2 3))
'(1 1 2 1 2 3)
> ( a-count '(4 3 2 1))
'(1 2 3 4 1 2 3 1 2 1)
> ( a-count '(1 1 2 2 3 3 2 2 1 1))
'(1 1 1 2 1 2 1 2 3 1 2 3 1 2 1 2 1 1)
>
```

Task 2b - Repetition Counting

Function Definition

```
(define (r-count lst)
    (cond
      ((null? lst) '())
      ( else
        (append ( same ( car lst) (car lst) )(r-count (cdr
lst))))))
```

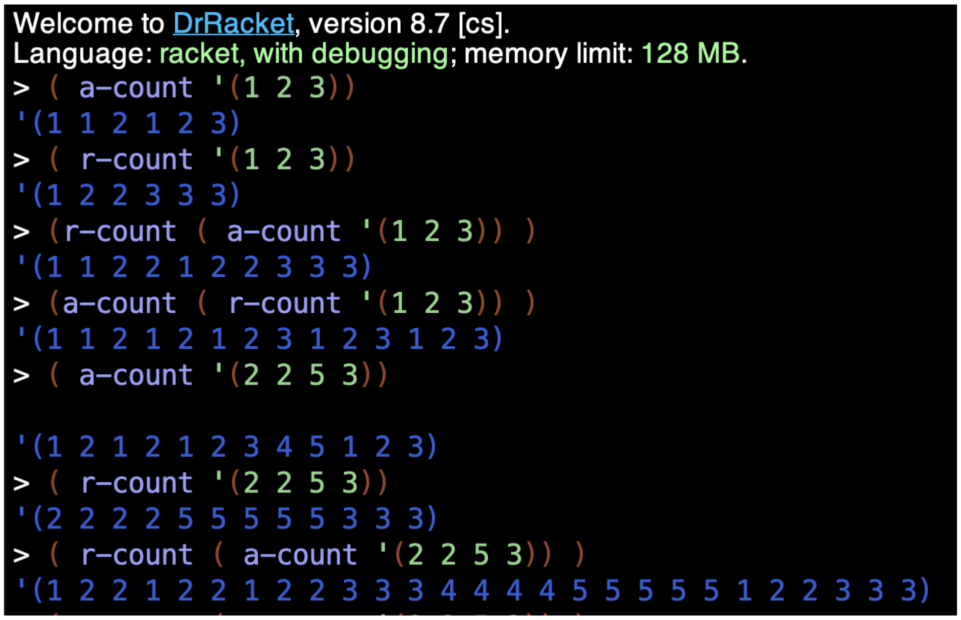
Demo



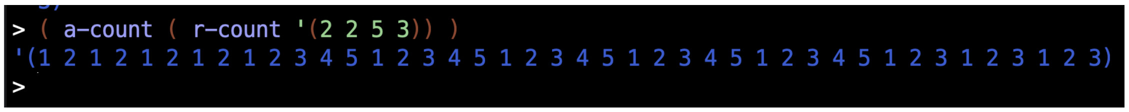
```
> ( r-count '(1 2 3))
'(1 2 2 3 3 3)
> ( r-count '(4 3 2 1))
'(4 4 4 4 3 3 3 2 2 1)
> ( r-count '(1 1 2 2 3 3 2 2 1 1))
'(1 1 2 2 2 2 3 3 3 3 3 3 2 2 2 2 1 1)
> 
```

Task 2c - Mixed Counting Demo

Demo



```
Welcome to DrRacket, version 8.7 [cs].
Language: racket, with debugging; memory limit: 128 MB.
> ( a-count '(1 2 3))
'(1 1 2 1 2 3)
> ( r-count '(1 2 3))
'(1 2 2 3 3 3)
> (r-count ( a-count '(1 2 3)) )
'(1 1 2 2 1 2 2 3 3 3)
> (a-count ( r-count '(1 2 3)) )
'(1 1 2 1 2 1 2 3 1 2 3 1 2 3)
> ( a-count '(2 2 5 3))

'(1 2 1 2 1 2 3 4 5 1 2 3)
> ( r-count '(2 2 5 3))
'(2 2 2 2 5 5 5 5 5 3 3 3)
> ( r-count ( a-count '(2 2 5 3)) )
'(1 2 2 1 2 2 1 2 2 3 3 3 4 4 4 4 5 5 5 5 5 1 2 2 3 3 3)
```



```
> ( a-count ( r-count '(2 2 5 3)) )
'(1 2 1 2 1 2 1 2 1 2 3 4 5 1 2 3 4 5 1 2 3 4 5 1 2 3 4 5 1 2 3 4 5 1 2 3 1 2 3 1 2 3)
> 
```

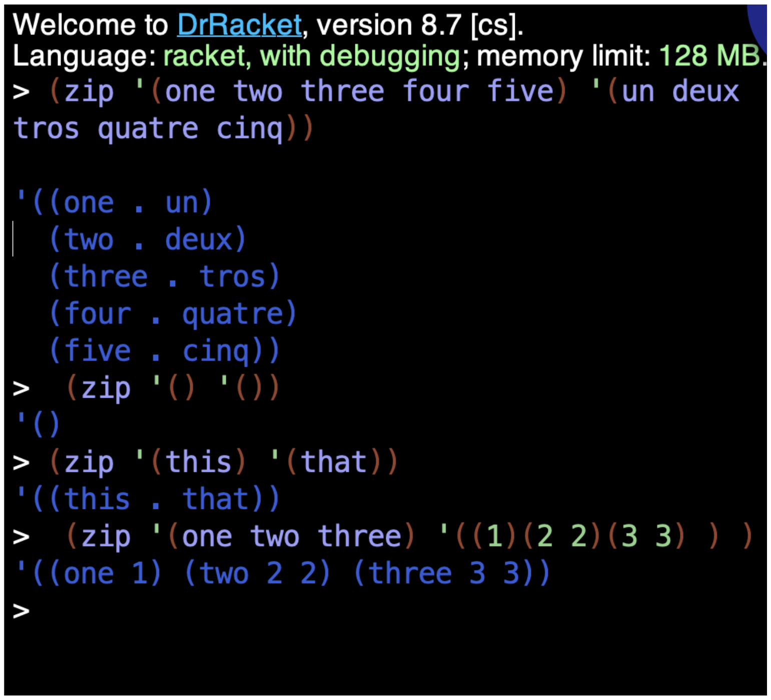
Task 3: Association Lists

Task 3a - Zip

Function Definition

```
(define (zip lst1 lst2)
  ( cond
     ( ( null? lst1) '())
     ( else
       ( cons ( cons ( car lst1 )(car lst2))(zip (cdr lst1)(cdr
lst2))))))
```

Demo



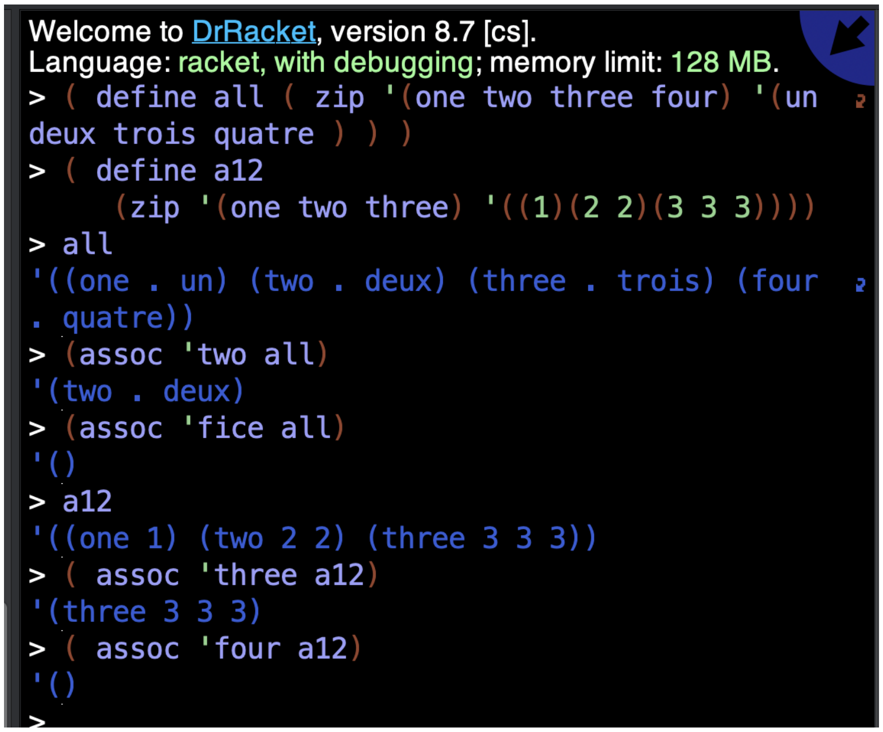
```
Welcome to DrRacket, version 8.7 [cs].
Language: racket, with debugging; memory limit: 128 MB.
> (zip '(one two three four five) '(un deux
tros quatre cinq))

'((one . un)
  (two . deux)
  (three . tros)
  (four . quatre)
  (five . cinq))
>  (zip '() '())
'()
> (zip '(this) '(that))
'((this . that))
>  (zip '(one two three) '((1)(2 2)(3 3) ) )
'((one 1) (two 2 2) (three 3 3))
>
```

Task 3b - Assoc

Function Definition

```
(define (assoc lst-obj lst )
  ( cond
     ( ( null? lst) '())
     ( ( eq? ( car (car lst ) ) lst-obj)(car lst))
     ( else
       ( assoc lst-obj ( cdr lst) ) ) ) )
```

Demo



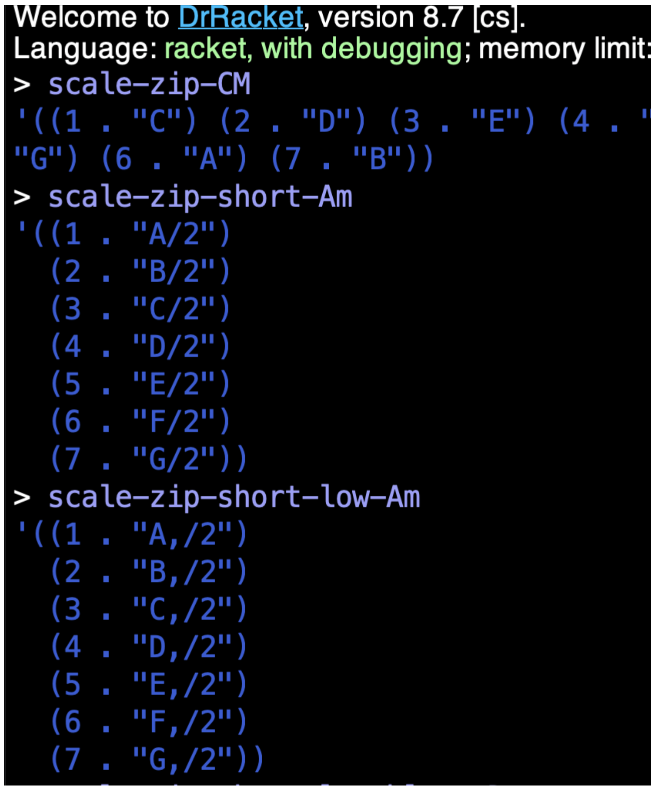
```
Welcome to DrRacket, version 8.7 [cs].
Language: racket, with debugging; memory limit: 128 MB.
> ( define all ( zip '(one two three four) '(un
deux trois quatre ) ) )
> ( define a12
     (zip '(one two three) '((1)(2 2)(3 3 3))))
> all
'((one . un) (two . deux) (three . trois) (four
. quatre))
> (assoc 'two all)
'(two . deux)
> (assoc 'fice all)
'()
> a12
'((one 1) (two 2 2) (three 3 3 3))
> ( assoc 'three a12)
'(three 3 3 3)
> ( assoc 'four a12)
'()
>
```

Task 3c - Establishing some Association Lists

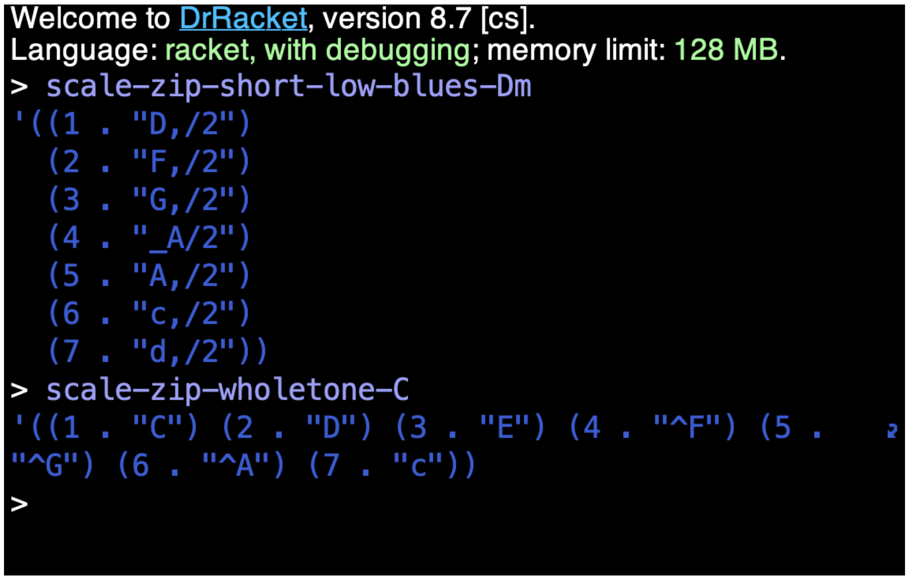
Code

```
( define scale-zip-CM (zip (iota 7) '("C" "D" "E" "F" "G" "A"
"B" ) ))
( define scale-zip-short-Am (zip (iota 7) '("A/2" "B/2" "C/2"
"D/2" "E/2" "F/2" "G/2" ) ))
( define scale-zip-short-low-Am (zip (iota 7) '("A,/2" "B,/2"
"C,/2" "D,/2" "E,/2" "F,/2" "G,/2" ) ))
( define scale-zip-short-low-blues-Dm (zip (iota 7) '("D,/2"
"F,/2" "G,/2" "_A/2" "A,/2" "c,/2" "d,/2" ) ))
( define scale-zip-wholetone-C ( zip ( iota 7)'("C" "D" "E" "^F"
"^G" "^A" "c" ) ))
```

Demo



```
Welcome to DrRacket, version 8.7 [cs].
Language: racket, with debugging; memory limit:
> scale-zip-CM
'((1 . "C") (2 . "D") (3 . "E") (4 . '
"G") (6 . "A") (7 . "B"))
> scale-zip-short-Am
'((1 . "A/2")
  (2 . "B/2")
  (3 . "C/2")
  (4 . "D/2")
  (5 . "E/2")
  (6 . "F/2")
  (7 . "G/2"))
> scale-zip-short-low-Am
'((1 . "A,/2")
  (2 . "B,/2")
  (3 . "C,/2")
  (4 . "D,/2")
  (5 . "E,/2")
  (6 . "F,/2")
  (7 . "G,/2"))
```



```
Welcome to DrRacket, version 8.7 [cs].
Language: racket, with debugging; memory limit: 128 MB.
> scale-zip-short-low-blues-Dm
'((1 . "D,/2")
  (2 . "F,/2")
  (3 . "G,/2")
  (4 . "_A/2")
  (5 . "A,/2")
  (6 . "c,/2")
  (7 . "d,/2"))
> scale-zip-wholetone-C
'((1 . "C") (2 . "D") (3 . "E") (4 . "^F") (5 .
"^G") (6 . "^A") (7 . "c"))
>
```

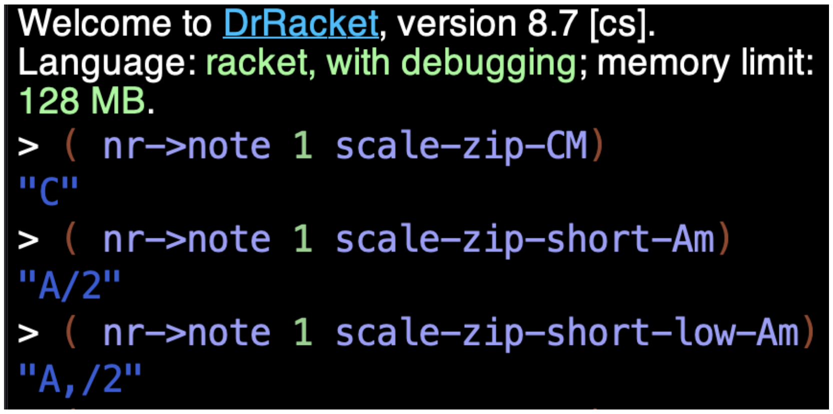
Task 4 - Numbers to Notes to ABC

Task 4a - nr -> notes

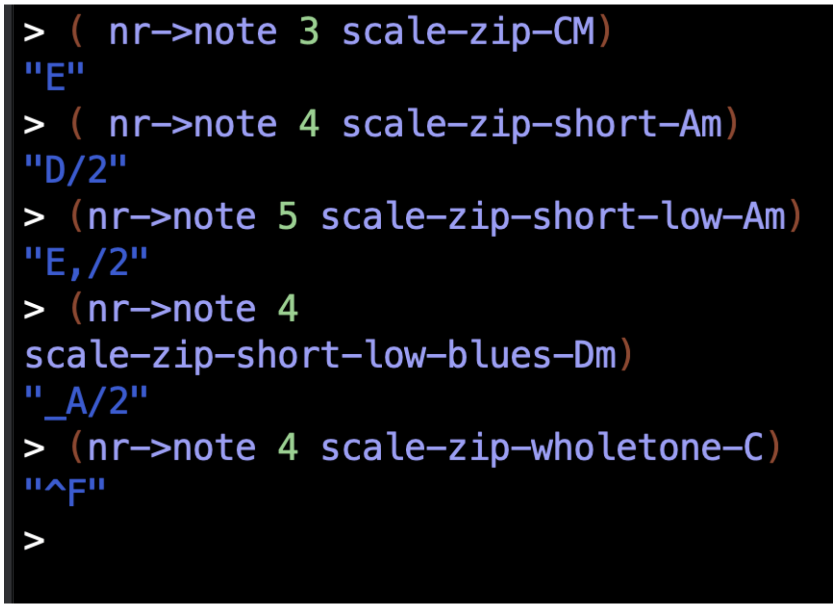
Function Definition

```
(define (nr->note n assoclst)
  (cond
    ((null? assoclst) "")
    ((= n (caar assoclst)) (cdar assoclst))
    (else (nr->note n (cdr assoclst)))))
```

Demo



```
Welcome to DrRacket, version 8.7 [cs].
Language: racket, with debugging; memory limit:
128 MB.
> ( nr->note 1 scale-zip-CM)
"C"
> ( nr->note 1 scale-zip-short-Am)
"A/2"
> ( nr->note 1 scale-zip-short-low-Am)
"A,/2"
```



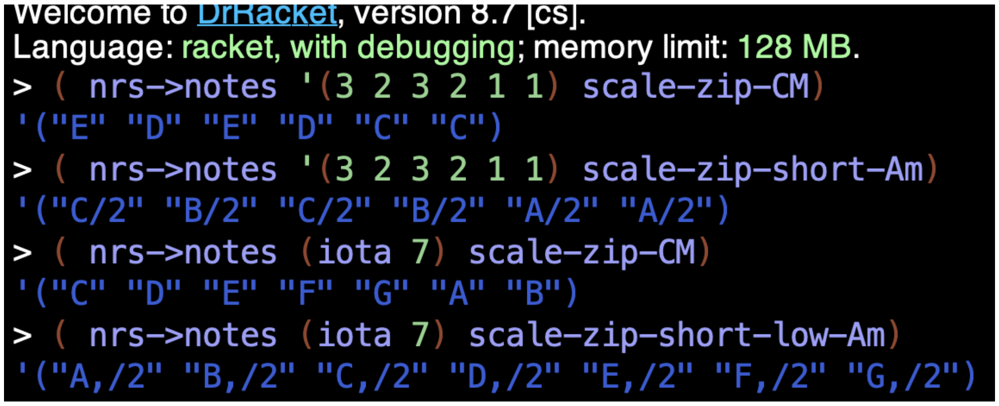
```
> ( nr->note 3 scale-zip-CM)
"E"
> ( nr->note 4 scale-zip-short-Am)
"D/2"
> (nr->note 5 scale-zip-short-low-Am)
"E,/2"
> (nr->note 4
scale-zip-short-low-blues-Dm)
"_A/2"
> (nr->note 4 scale-zip-wholetone-C)
"^F"
>
```

Task 4b - nrs->notes

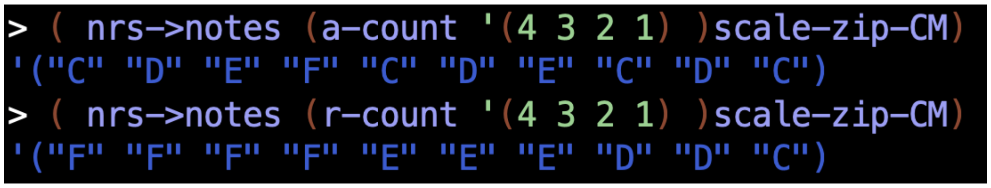
Function Definition

```
(define (nrs->notes assoclst1 assoclst2)
  (cond ((null? assoclst1) '())
        (else (map (lambda (x) (nr->note x assoclst2))
assoclst1))
    ))
```

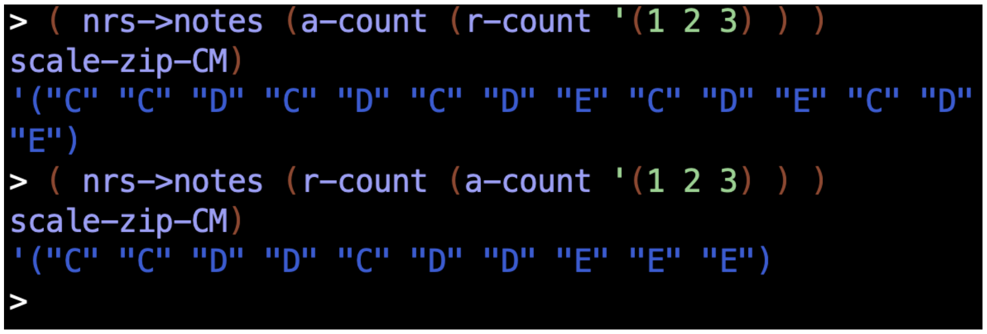
Demo



```
Welcome to DrRacket, version 8.7 [cs].
Language: racket, with debugging; memory limit: 128 MB.
> ( nrs->notes '(3 2 3 2 1 1) scale-zip-CM)
'("E" "D" "E" "D" "C" "C")
> ( nrs->notes '(3 2 3 2 1 1) scale-zip-short-Am)
'("C/2" "B/2" "C/2" "B/2" "A/2" "A/2")
> ( nrs->notes (iota 7) scale-zip-CM)
'("C" "D" "E" "F" "G" "A" "B")
> ( nrs->notes (iota 7) scale-zip-short-low-Am)
'("A,/2" "B,/2" "C,/2" "D,/2" "E,/2" "F,/2" "G,/2")
```



```
> ( nrs->notes (a-count '(4 3 2 1) )scale-zip-CM)
'("C" "D" "E" "F" "C" "D" "E" "C" "D" "C")
> ( nrs->notes (r-count '(4 3 2 1) )scale-zip-CM)
'("F" "F" "F" "F" "E" "E" "E" "D" "D" "C")
```



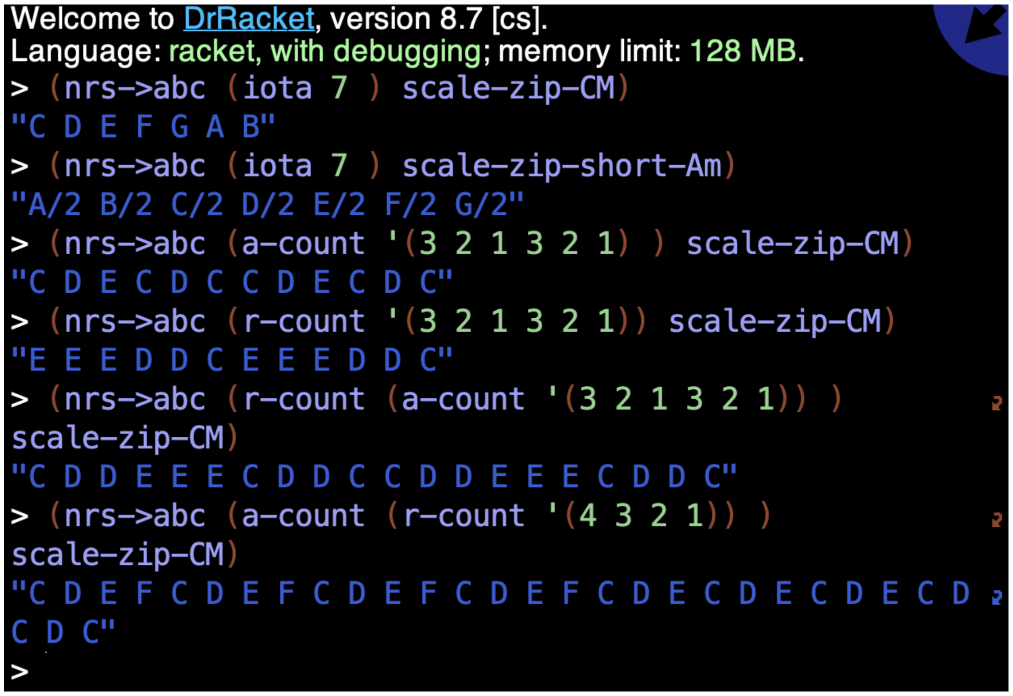
```
> ( nrs->notes (a-count (r-count '(1 2 3) ) )
scale-zip-CM)
'("C" "C" "D" "C" "D" "C" "D" "E" "C" "D" "E" "C" "D"
"E")
> ( nrs->notes (r-count (a-count '(1 2 3) ) )
scale-zip-CM)
'("C" "C" "D" "D" "C" "D" "D" "E" "E" "E")
>
```

Task 4c - nrs->abc

Function Definition

```
(define (nrs->abc assoclst1 assoclst2)
  ( string-join (nrs->notes assoclst1 assoclst2 ) ))
```

Demo



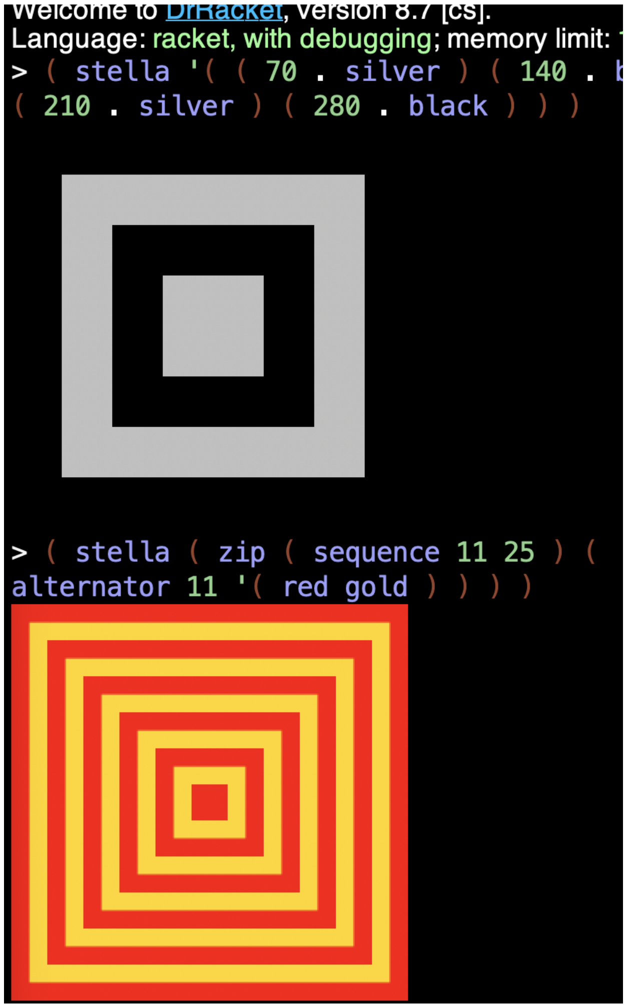
```
Welcome to DrRacket, version 8.7 [cs].
Language: racket, with debugging; memory limit: 128 MB.
> (nrs->abc (iota 7 ) scale-zip-CM)
"C D E F G A B"
> (nrs->abc (iota 7 ) scale-zip-short-Am)
"A/2 B/2 C/2 D/2 E/2 F/2 G/2"
> (nrs->abc (a-count '(3 2 1 3 2 1) ) scale-zip-CM)
"C D E C D C C D E C D C"
> (nrs->abc (r-count '(3 2 1 3 2 1)) scale-zip-CM)
"E E E D D C E E E D D C"
> (nrs->abc (r-count (a-count '(3 2 1 3 2 1)) )
scale-zip-CM)
"C D D E E E C D D C C D D E E E C D D C"
> (nrs->abc (a-count (r-count '(4 3 2 1)) )
scale-zip-CM)
"C D E F C D E F C D E F C D E F C D E C D E C D E C D
C D C"
>
```

Task 5 - Stella

Function Definition

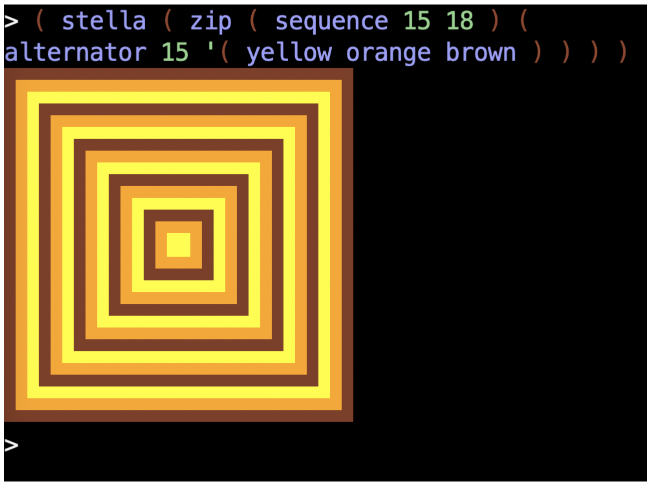
```
(define (stella asscolst)
  (if (null? asscolst) ""
      (let ((boxes (map (lambda (x) (square (car x) 'solid (cdr
x))) asscolst)))
        (foldr overlay empty-image boxes))))
```

Demo



```
Welcome to DrRacket, version 8.7 [cs].
Language: racket, with debugging; memory limit:
> ( stella '( ( 70 . silver ) ( 140 . b
( 210 . silver ) ( 280 . black ) ) )
```

```
> ( stella ( zip ( sequence 11 25 ) (
alternator 11 '( red gold ) ) ) )
```



```
> ( stella ( zip ( sequence 15 18 ) (
alternator 15 '( yellow orange brown ) ) ) )
```

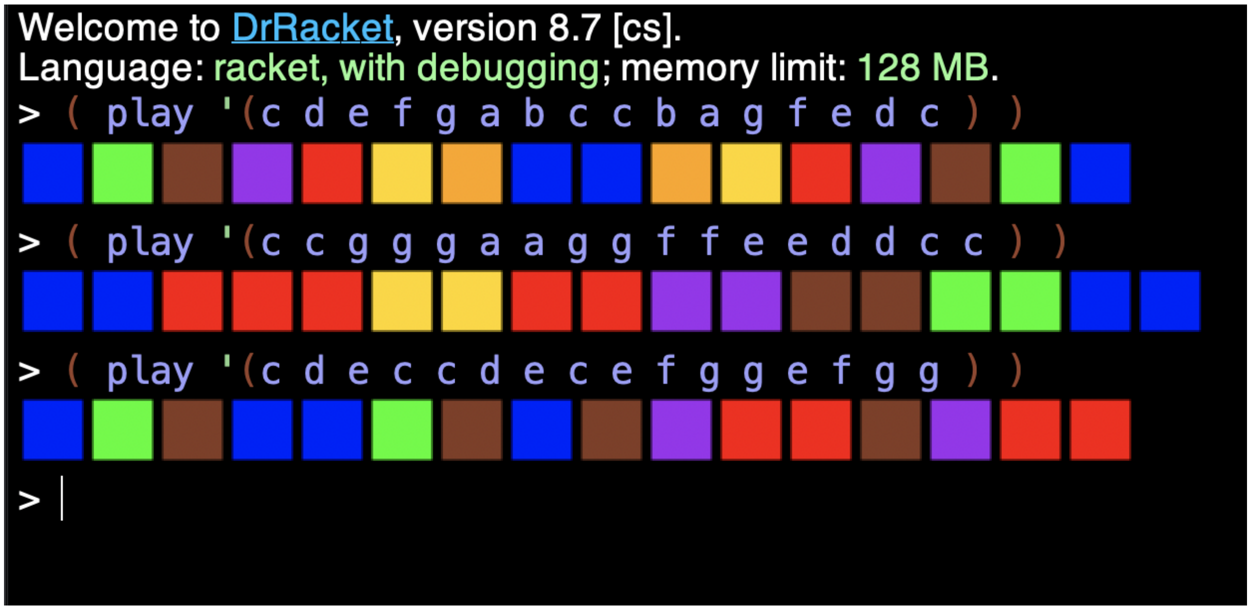
```
>
```

Task 6 - Chromesthetic Renderings

Code

```
( define pitch-classes '( c d e f g a b ) )
( define color-names '( blue green brown purple red yellow
orange) )
( define ( box color )
( overlay
( square 30 "solid" color )
( square 35 "solid" "black" )
)
)
( define boxes
( list
( box "blue" )
( box "green" )
( box "brown" )
( box "purple" )
( box "red" )
( box "gold" )
( box "orange" )
)
)
( define pc-a-list ( zip pitch-classes color-names ) )
( define cb-a-list ( zip color-names boxes ) )
( define ( pc->color pc )
( cdr ( assoc pc pc-a-list ) )
)
( define ( color->box color )
( cdr ( assoc color cb-a-list ) )
)
(define (play pitch-list)
  (define color-list (map pc->color pitch-list))
  (define box-list (map color->box color-list))
  (foldr beside empty-image box-list))
```

Demo



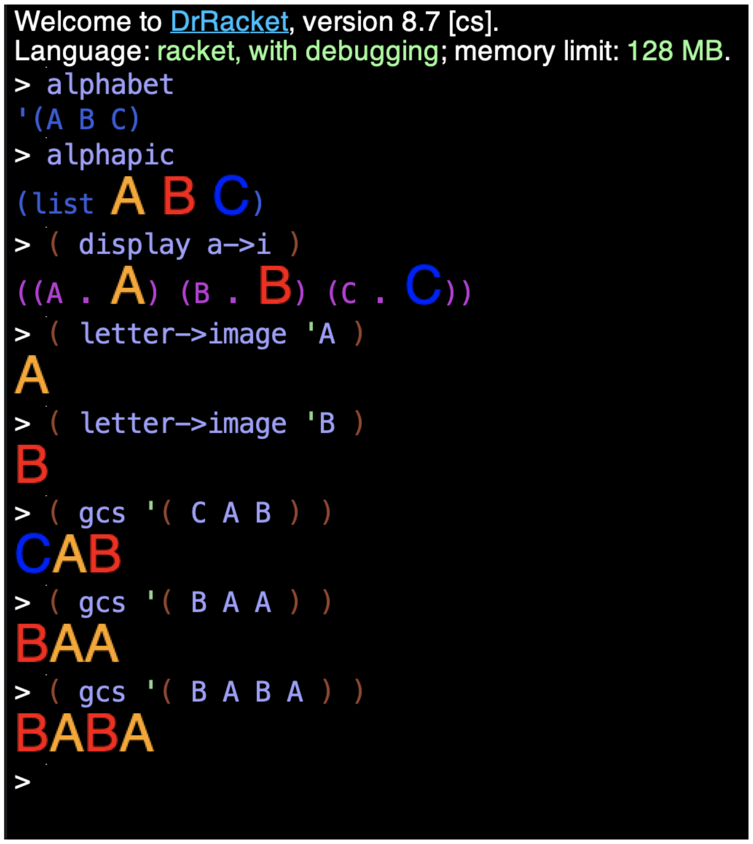
```
Welcome to DrRacket, version 8.7 [cs].
Language: racket, with debugging; memory limit: 128 MB.
> ( play '(c d e f g a b c c b a g f e d c ) )
🟦🟩🟫🟪🟥🟨🟧🟦🟦🟧🟨🟥🟪🟫🟩🟦
> ( play '(c c g g g a a g g f f e e d d c c ) )
🟦🟦🟥🟥🟥🟨🟨🟥🟥🟪🟪🟫🟫🟩🟩🟦🟦
> ( play '(c d e c c d e c e f g g e f g g ) )
🟦🟩🟫🟦🟦🟩🟫🟦🟫🟪🟥🟥🟫🟪🟥🟥
> |
```

Task 7 - Grapheme to Color Synesthesia

Code

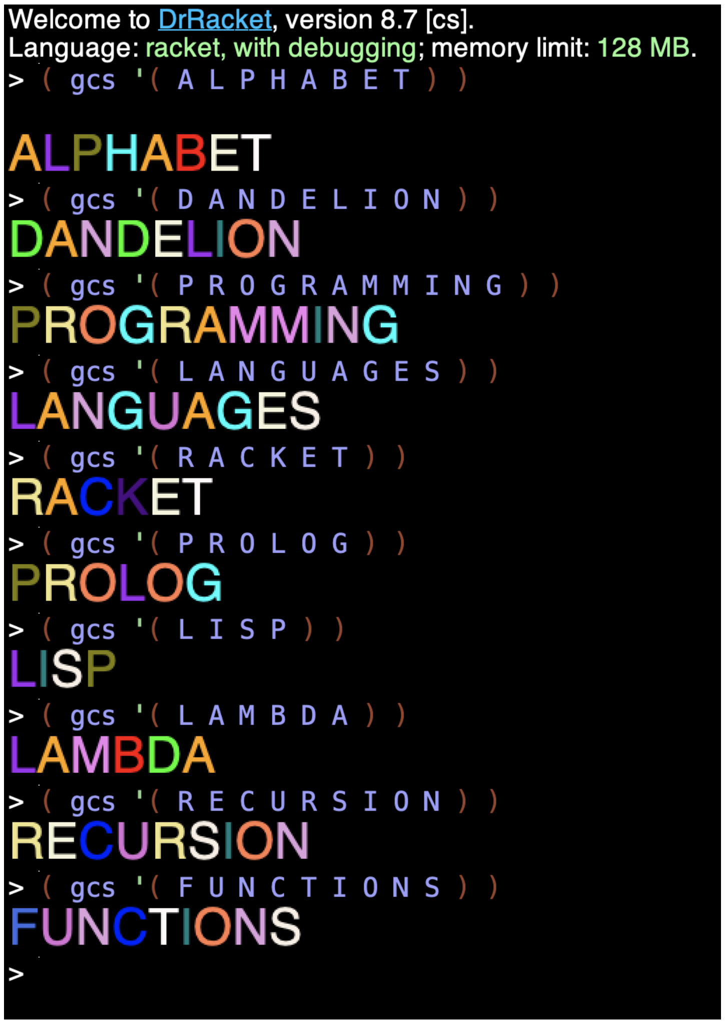
```
( define pitch-classes '( c d e f g a b ) )
( define color-names '( blue green brown purple red yellow
orange) )
( define ( box color )
( overlay
( square 30 "solid" color )
( square 35 "solid" "black" )
)
)
( define boxes
( list
( box "blue" )
( box "green" )
( box "brown" )
( box "purple" )
( box "red" )
( box "gold" )
( box "orange" )))
( define pc-a-list ( zip pitch-classes color-names ) )
( define cb-a-list ( zip color-names boxes ) )
( define ( pc->color pc )
( cdr ( assoc pc pc-a-list ) )
)
( define ( color->box color )
( cdr ( assoc color cb-a-list ) )
)
(define (play pitch-list)
  (define color-list (map pc->color pitch-list))
  (define box-list (map color->box color-list))
  (foldr beside empty-image box-list))
( define AI (text "A" 36 "orange") )
( define BI (text "B" 36 "red") )
( define CI (text "C" 36 "blue") )
( define DI (text "D" 36 "green") )
( define EI (text "E" 36 "beige") )
( define FI (text "F" 36 "royal blue") )
( define GI (text "G" 36 "cyan") )
( define HI (text "H" 36 "aqua") )
( define II (text "I" 36 "teal") )
( define JI (text "J" 36 "navy") )
( define KI (text "K" 36 "indigo") )
( define LI (text "L" 36 "purple") )
( define MI (text "M" 36 "violet") )
( define NI (text "N" 36 "plum") )
( define OI (text "O" 36 "coral") )
( define PI (text "P" 36 "olive") )
( define QI (text "Q" 36 "silver") )
( define RI (text "R" 36 "khaki") )
( define SI (text "S" 36 "linen") )
( define TI (text "T" 36 "snow") )
( define UI (text "U" 36 "orchid") )
( define VI (text "V" 36 "gold") )
( define WI (text "W" 36 "wheat") )
( define XI (text "X" 36 "sienna") )
( define YI (text "Y" 36 "crimson") )

( define alphabet '( A B C D E F G H I J K L M N O P Q R S T U V
W X Y) )
( define alphapic ( list AI BI CI DI EI FI GI HI II JI KI LI MI
NI OI PI QI RI SI TI UI VI WI XI YI) )
( define a->i ( zip alphabet alphapic ) )

( define ( letter->image lol )
   ( cdr ( assoc lol a->i ) ) )
(define (gcs lols)
  (define images (map (lambda (lol) (letter->image lol)) lols))
  (foldr beside empty-image images))
```

Demo 1



```
Welcome to DrRacket, version 8.7 [cs].
Language: racket, with debugging; memory limit: 128 MB.
> alphabet
'(A B C)
> alphapic
(list A B C)
> ( display a->i )
((A . A) (B . B) (C . C))
> ( letter->image 'A )
A
> ( letter->image 'B )
B
> ( gcs '( C A B ) )
CAB
> ( gcs '( B A A ) )
BAA
> ( gcs '( B A B A ) )
BABA
>
```

Demo 2



```
Welcome to DrRacket, version 8.7 [cs].
Language: racket, with debugging; memory limit: 128 MB.
> ( gcs '( A L P H A B E T ) )

ALPHABET
> ( gcs '( D A N D E L I O N ) )
DANDELION
> ( gcs '( P R O G R A M M I N G ) )
PROGRAMMING
> ( gcs '( L A N G U A G E S ) )
LANGUAGES
> ( gcs '( R A C K E T ) )
RACKET
> ( gcs '( P R O L O G ) )
PROLOG
> ( gcs '( L I S P ) )
LISP
> ( gcs '( L A M B D A ) )
LAMBDA
> ( gcs '( R E C U R S I O N ) )
RECURSION
> ( gcs '( F U N C T I O N S ) )
FUNCTIONS
>
```