Temitope Emokpae 3/28/2023

CSC 344

Racket Programming Assignment #5: RLP and HoFs

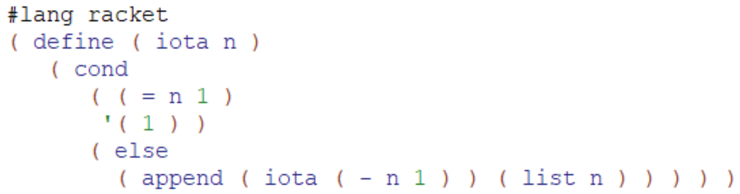
Learning Abstract

This racket programming assignment goes in-depth with practicing higher-order functions in 7 tasks. Task 1 involves simple list generators. Task 2 consists in generating number sequences by performing some interesting sorts of "counting." Task 3 involves "association lists." Task 4 transforms number sequences into musical notes represented in ABC notation. Task 5 consists in creating Frank Stella's nested squares. Task 6 involves chromesthesia, the mapping of musical pitches to colors. Task 7 requires grapheme to the color synesthesia, in which letters map to colors.

Task 1 - Simple List Generators

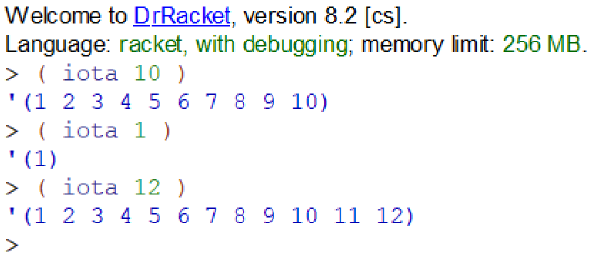
Task 1a - iota

Function Definition:



```
#lang racket
( define ( iota n )
   ( cond
      ( ( = n 1 )
       '( 1 ) )
      ( else
        ( append ( iota ( - n 1 ) ) ( list n ) ) ) ) ) )
```

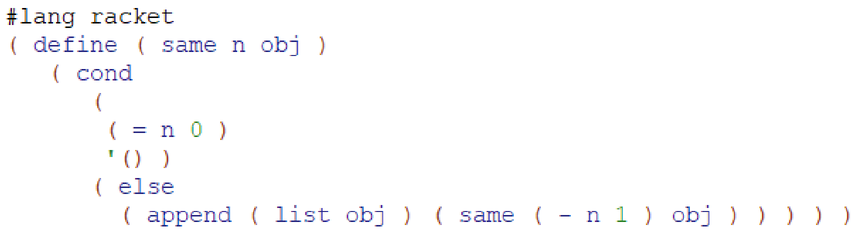
Demo:



```
Welcome to DrRacket, version 8.2 [cs].
Language: racket, with debugging; memory limit: 256 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)
>
```

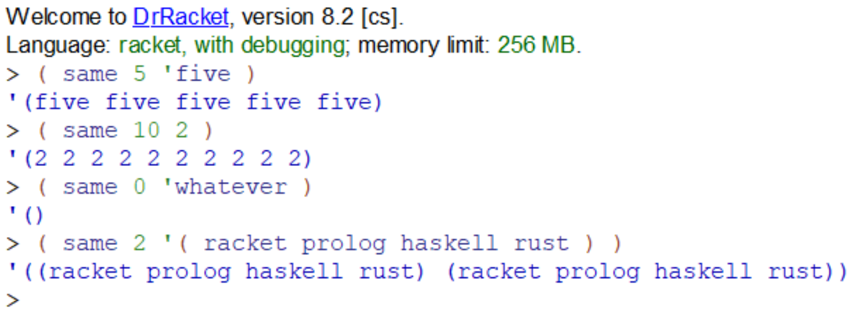
Task 1b - Same

Function Definition:



```
#lang racket
( define ( same n obj )
   ( cond
      (
       ( = n 0 )
       '() )
      ( else
        ( append ( list obj ) ( same ( - n 1 ) obj ) ) ) ) )
```

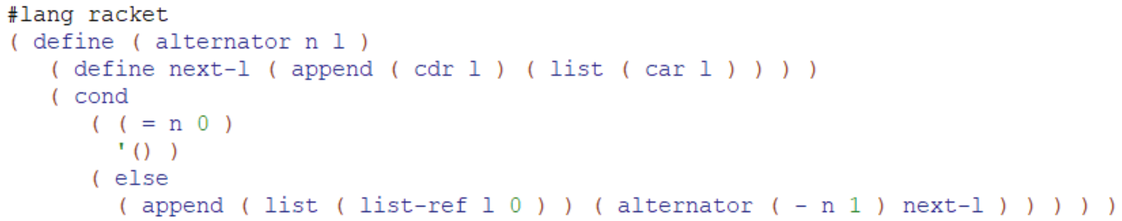
Demo:



```
Welcome to DrRacket, version 8.2 [cs].
Language: racket, with debugging; memory limit: 256 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))
>
```

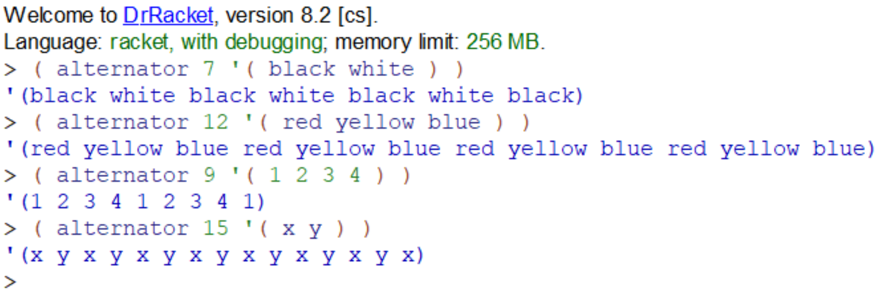
Task 1c - Alternator

Function Definition:



```
#lang racket
( define ( alternator n l )
   ( define next-l ( append ( cdr l ) ( list ( car l ) ) ) )
   ( cond
      ( ( = n 0 )
        '() )
      ( else
        ( append ( list ( list-ref l 0 ) ) ( alternator ( - n 1 ) next-l ) ) ) ) )
```

Demo:



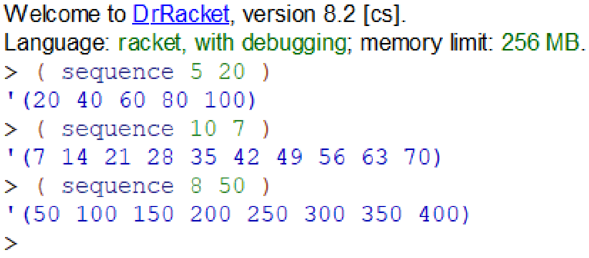
```
Welcome to DrRacket, version 8.2 [cs].
Language: racket, with debugging; memory limit: 256 MB.
> ( alternator 7 '( black white ) )
'(black white black white black white black)
> ( alternator 12 '( red yellow blue ) )
'(red yellow blue red yellow blue red yellow blue red yellow blue)
> ( alternator 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 x )
   ( map ( lambda ( i )
            ( * i x ) ) ( iota n ) ) )
```

Demo:



```
Welcome to DrRacket, version 8.2 [cs].
Language: racket, with debugging; memory limit: 256 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)
>
```

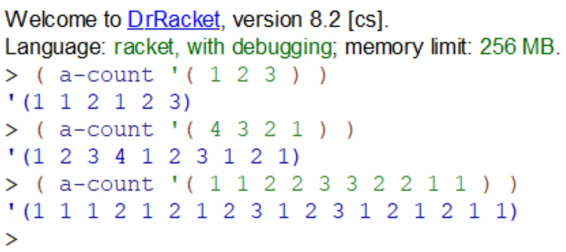
Task 2 - Counting

Task 2a - Accumulation Counting

Function Definition:

```
( define ( a-count l )
   ( cond
      ( ( empty? l )
        '() )
      ( else
        ( append ( iota ( list-ref l 0 ) ) ( a-count ( cdr l ) ) ) ) ) )
```

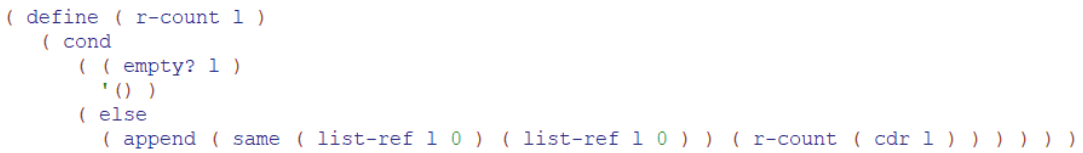
Demo:



```
Welcome to DrRacket, version 8.2 [cs].
Language: racket, with debugging; memory limit: 256 MB.
> ( 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)
>
```

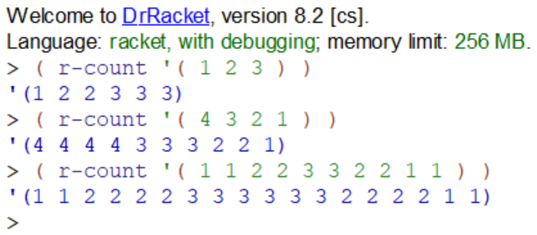
Task 2b - Repetition Counting

Function Definition:



```
( define ( r-count l )
   ( cond
      ( ( empty? l )
        '() )
      ( else
        ( append ( same ( list-ref l 0 ) ( list-ref l 0 ) ) ( r-count ( cdr l ) ) ) ) ) )
```

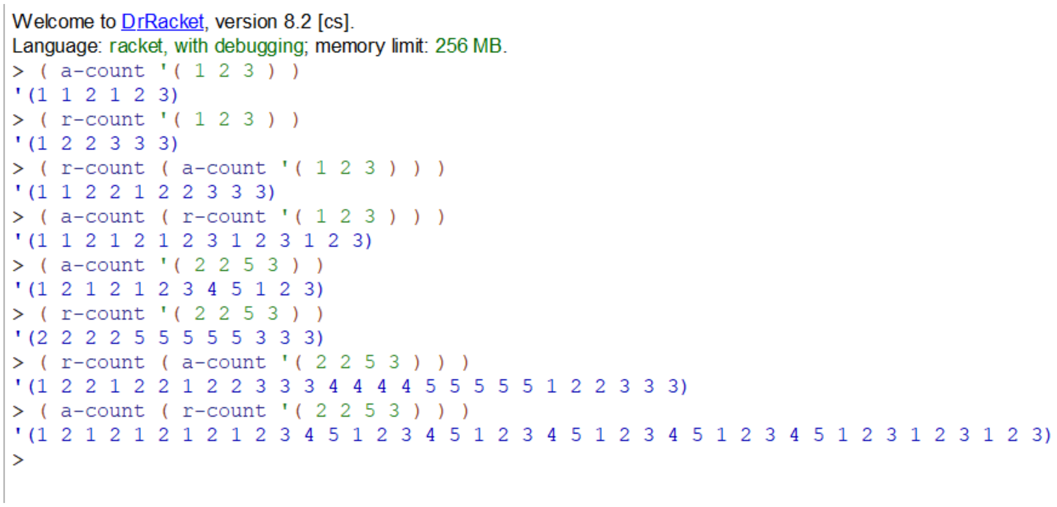
Demo :



```
Welcome to DrRacket, version 8.2 [cs].
Language: racket, with debugging; memory limit: 256 MB.
> ( 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 - Repetition Counting

Demo:



```
Welcome to DrRacket, version 8.2 [cs].
Language: racket, with debugging; memory limit: 256 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)
>
```

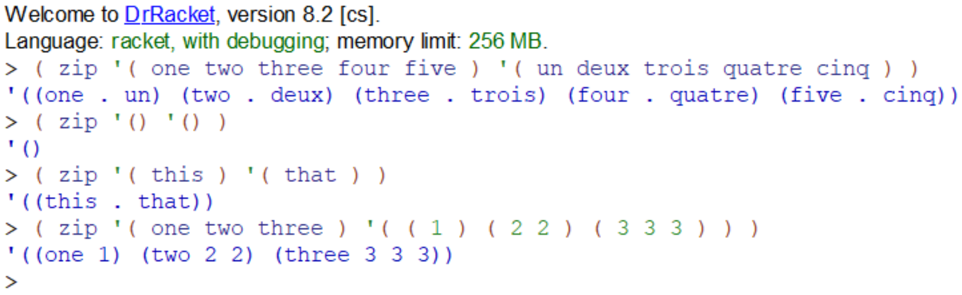
Task 3 - Association Lists

Task 3a - Zip

Function Definition:

```
#lang racket
( define ( zip l1 l2 )
( cond
   ( ( or ( empty? l1 ) ( empty? l2 ) )
     '() )
   ( else
     ( append ( list ( cons ( list-ref l1 0 ) ( list-ref l2 0 ) ) ) ( zip ( cdr l1 ) ( cdr l2 ) ) ) ) ) )
```

Demo:



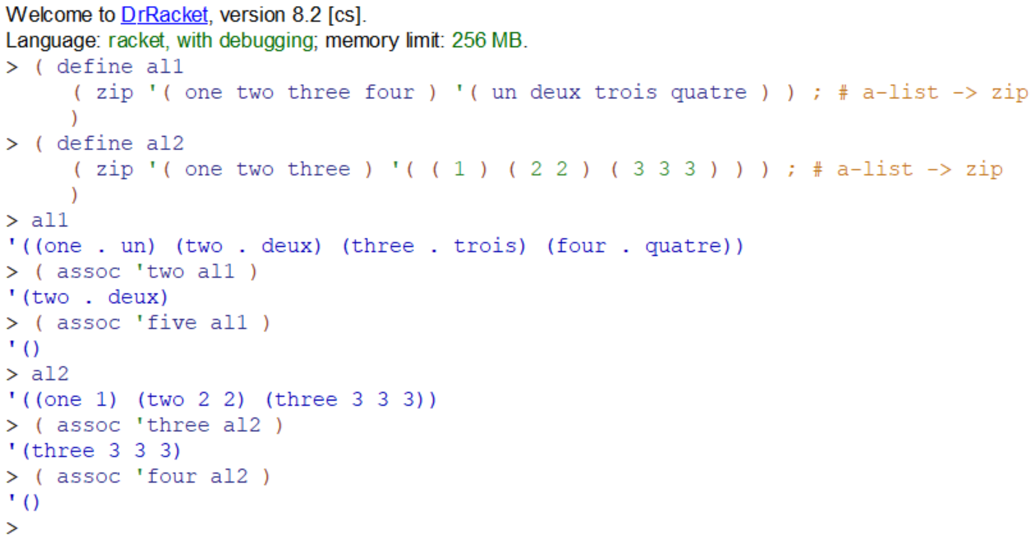
```
Welcome to DrRacket, version 8.2 [cs].
Language: racket, with debugging; memory limit: 256 MB.
> ( zip '( one two three four five ) '( un deux trois quatre cinq ) )
'((one . un) (two . deux) (three . trois) (four . quatre) (five . cinq))
> ( zip '() '() )
'()
> ( zip '( this ) '( that ) )
'((this . that))
> ( zip '( one two three ) '( ( 1 ) ( 2 2 ) ( 3 3 3 ) ) )
'((one 1) (two 2 2) (three 3 3 3))
>
```

Task 3b - Assoc

Function Definition:

```
( define ( assoc obj l )
   ( cond
      ( ( empty? l )
        '() )
      ( else
        ( cond
           ( ( equal? obj ( car ( list-ref l 0 ) ) )
             ( list-ref l 0 ) )
           ( else
             ( assoc obj ( cdr l ) ) ) ) ) ) ) )
```

Demo:



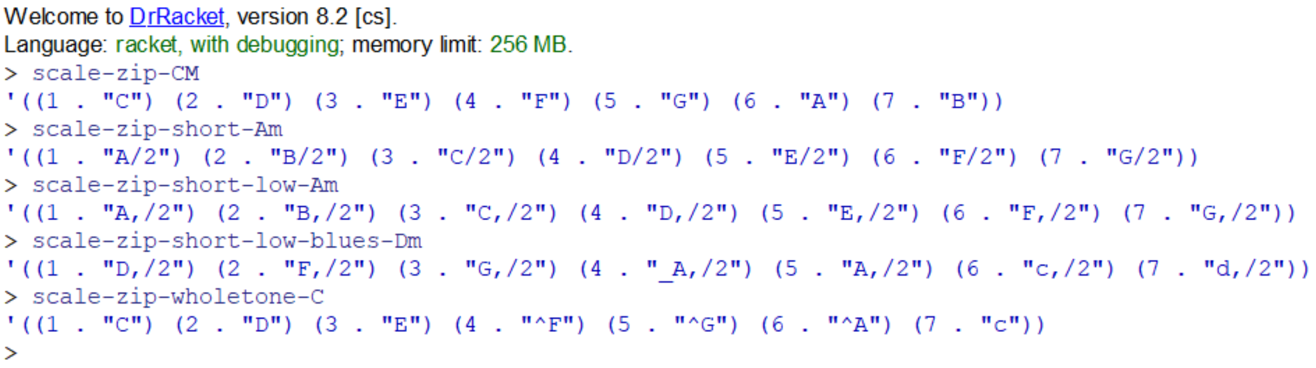
```
Welcome to DrRacket, version 8.2 [cs].
Language: racket, with debugging; memory limit: 256 MB.
> ( define al1
     ( zip '( one two three four ) '( un deux trois quatre ) ) ; # a-list -> zip
     )
> ( define al2
     ( zip '( one two three ) '( ( 1 ) ( 2 2 ) ( 3 3 3 ) ) ) ) ; # a-list -> zip
     )
> al1
'((one . un) (two . deux) (three . trois) (four . quatre))
> ( assoc 'two al1 )
'(two . deux)
> ( assoc 'five al1 )
'()
> al2
'((one 1) (two 2 2) (three 3 3 3))
> ( assoc 'three al2 )
'(three 3 3 3)
> ( assoc 'four al2 )
'()
>
```

Task 3c - Assoc

Function Definition:

```
( 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 8.2 [cs].
Language: racket, with debugging; memory limit: 256 MB.
> scale-zip-CM
'((1 . "C") (2 . "D") (3 . "E") (4 . "F") (5 . "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"))
> 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"))
>
```

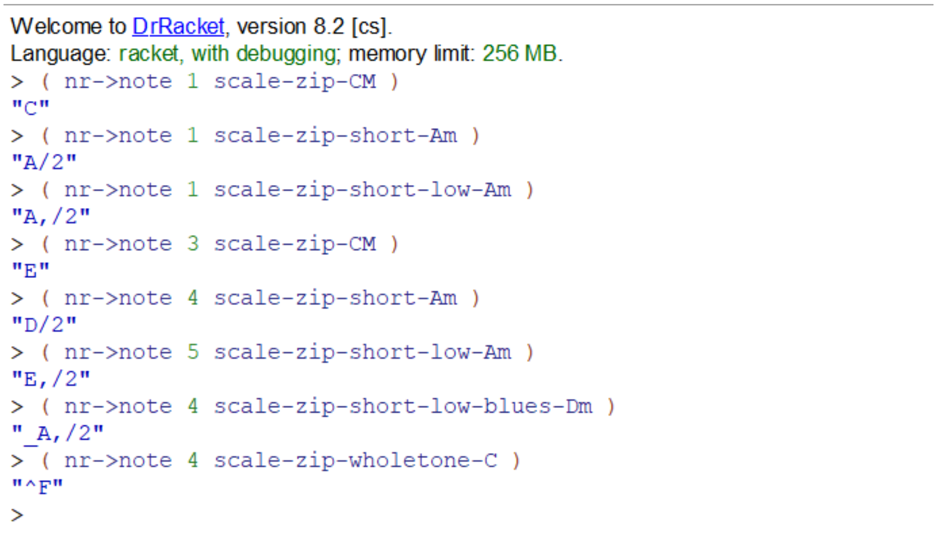
Task 4 - Numbers to Notes to ABC

Task 4a - nr->note

Function Definition:

```
( define ( nr->note n assoc )
   ( cdr ( list-ref assoc ( - n 1 ) ) ) )
```

Demo:



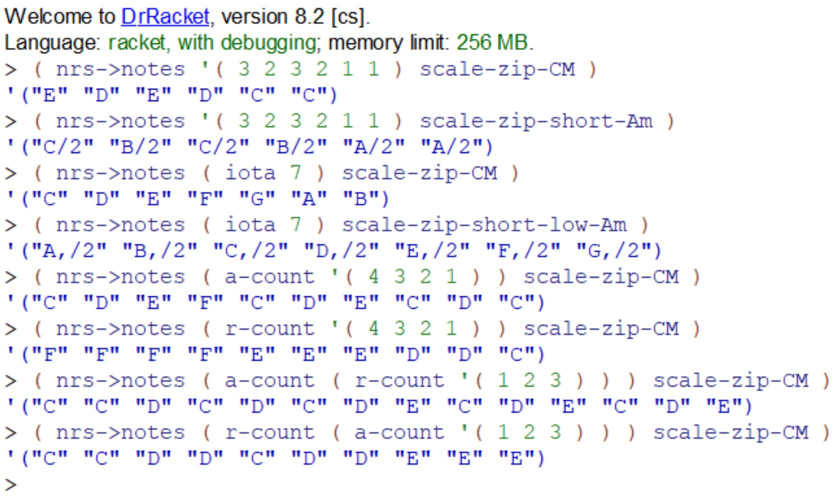
```
Welcome to DrRacket, version 8.2 [cs].
Language: racket, with debugging; memory limit: 256 MB.
> ( nr->note 1 scale-zip-CM )
"C"
> ( nr->note 1 scale-zip-short-Am )
"A/2"
> ( nr->note 1 scale-zip-short-low-Am )
"A,/2"
> ( nr->note 3 scale-zip-CM )
"E"
> ( nr->note 4 scale-zip-short-Am )
"D/2"
> ( 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 n abc )
   ( map ( lambda ( num ) ( nr->note num abc ) ) n ) )
```

Demo:



```
Welcome to DrRacket, version 8.2 [cs].
Language: racket, with debugging; memory limit: 256 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 n l )
   ( cond
      ( ( empty? n )
        "" )
      ( else
        ( string-join ( list ( nr->note ( list-ref n 0 ) l ) ( nrs->abc ( cdr n ) l ) ) ) ) ) ) )
```

Demo:

```
Welcome to DrRacket, version 8.2 [cs].
Language: racket, with debugging; memory limit: 256 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 '( 4 3 2 1 ) ) ) scale-zip-CM )
"C D D E E E F F F F 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 D C "
>
```

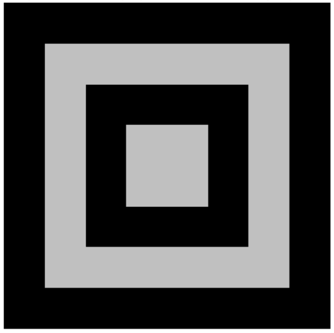
Task 5 - Stella

Function Definition:

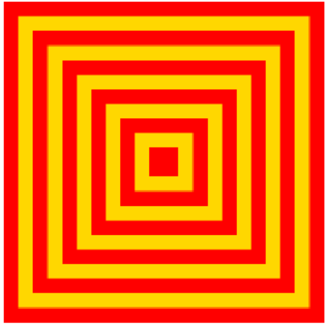
```
( define ( stella spec )
   ( cond
      ( ( empty? spec ) empty-image )
      ( else
        ( foldr overlay empty-image
                ( map ( lambda ( pair )
                         ( square ( car pair ) 'solid ( cdr pair ) ) ) spec ) ) ) ) ) )

( define ( make-square pair )
   ( square ( car pair ) 'solid ( cdr pair ) ) )
```

Demo:

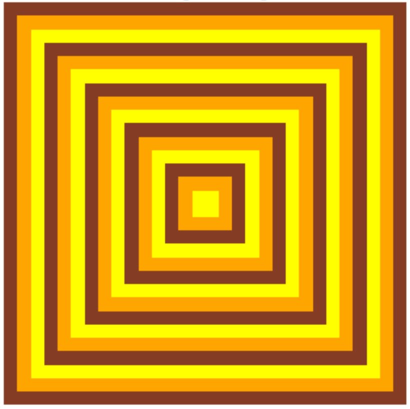
```
Welcome to DrRacket, version 8.2 [cs].
Language: racket, with debugging; memory limit: 256 MB.
> ( stella '( ( 70 . silver ) ( 140 . black ) ( 210 . silver ) ( 280 . black ) ) )
```



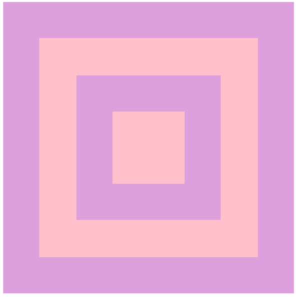
```
> ( stella ( zip ( sequence 11 25 ) ( alternator 11 '( red gold ) ) ) )
```



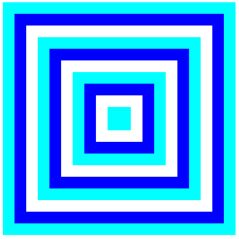
```
> ( stella ( zip ( sequence 15 18 ) ( alternator 15 '( yellow orange brown ) ) ) )
```



```
>
> ( stella '( ( 50 . pink ) ( 100 . plum ) ( 150 . pink ) ( 200 . plum ) ) )
```



```
> ( stella ( zip ( sequence 13 16 ) ( alternator 10 '( cyan white blue ) ) ) )
```



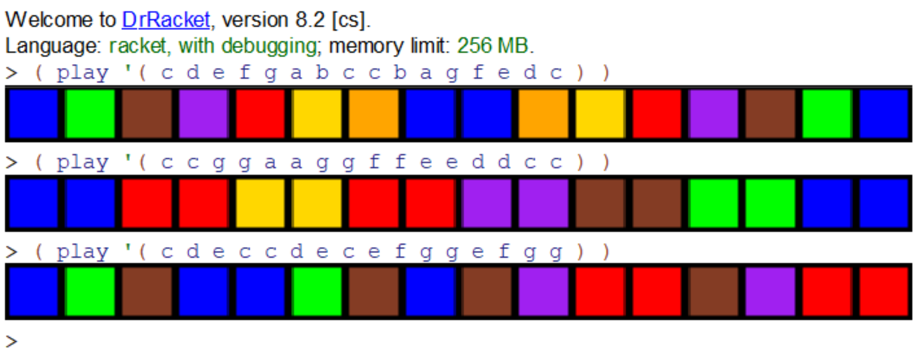
```
>
```

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 ) ) ; a-list -> zip
( define cb-a-list ( zip color-names boxes ) ) ; a-list -> zip

( define ( pc->color pc )
   ( cdr ( assoc pc pc-a-list ) ) )

( define ( color->box color )
   ( cdr ( assoc color cb-a-list ) ) )

( define ( play pitches )
   ( foldr beside empty-image
           ( map ( lambda ( color ) ( color->box color ) )
                 ( map ( lambda ( pitch ) ( pc->color pitch ) ) pitches ) ) ) )
```

Demo:



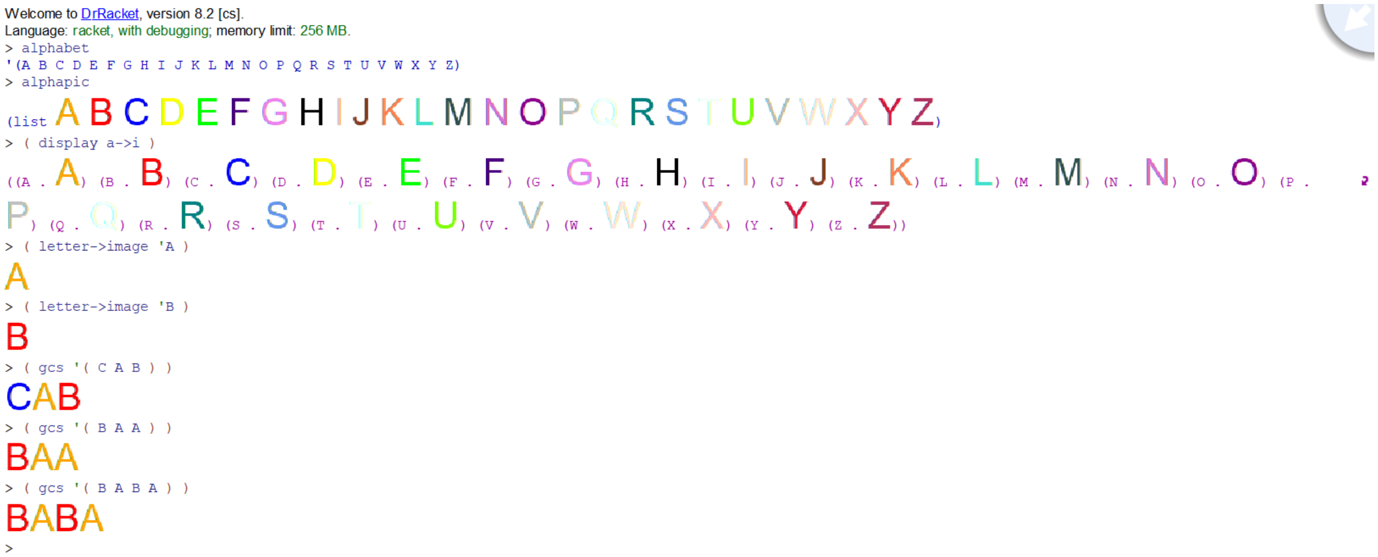
```
Welcome to DrRacket, version 8.2 [cs].
Language: racket, with debugging; memory limit: 256 MB.
> ( play '( c d e f g a b c c b a g f e d c ) )
> ( play '( c c g g a a g g f f e e d d c c ) )
> ( play '( c d e c c d e c e f g g e f g g ) )
>
```

Task 7 - Grapheme to Color Synesthesia

Code:

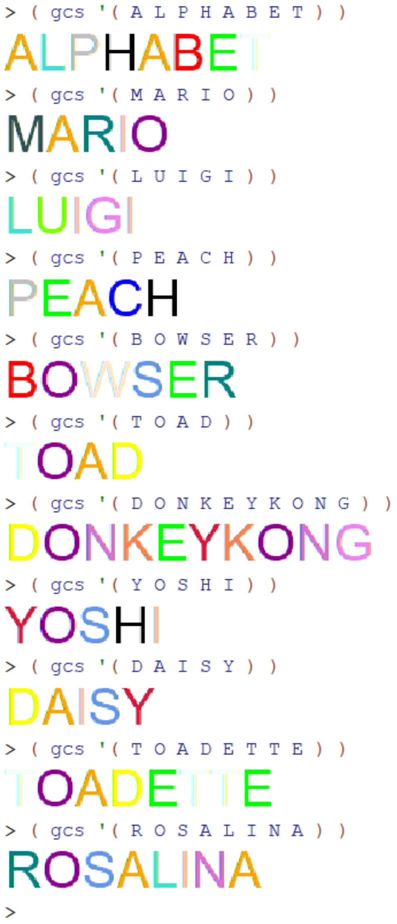
```
( define AI ( text "A" 36 "orange" ) )
( define BI ( text "B" 36 "red" ) )
( define CI ( text "C" 36 "blue" ) )
( define DI ( text "D" 36 "yellow" ) )
( define EI ( text "E" 36 "green" ) )
( define FI ( text "F" 36 "indigo" ) )
( define GI ( text "G" 36 "violet" ) )
( define HI ( text "H" 36 "black" ) )
( define II ( text "I" 36 "pink" ) )
( define JI ( text "J" 36 "brown" ) )
( define KI ( text "K" 36 "coral" ) )
( define LI ( text "L" 36 "turquoise" ) )
( define MI ( text "M" 36 "Dark Slate Gray" ) )
( define NI ( text "N" 36 "orchid" ) )
( define OI ( text "O" 36 "Dark Magenta" ) )
( define PI ( text "P" 36 "Silver" ) )
( define QI ( text "Q" 36 "Azure" ) )
( define RI ( text "R" 36 "Teal" ) )
( define SI ( text "S" 36 "CornflowerBlue" ) )
( define TI ( text "T" 36 "Mint Cream" ) )
( define UI ( text "U" 36 "Chartreuse" ) )
( define VI ( text "V" 36 "Grey" ) )
( define WI ( text "W" 36 "Misty Rose" ) )
( define XI ( text "X" 36 "Light Pink" ) )
( define YI ( text "Y" 36 "Crimson" ) )
( define ZI ( text "Z" 36 "Maroon" ) )

( 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 Z ) )
( 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 ZI ) )

( define a->i ( zip alphabet alphapic ) )

( define ( letter->image letter )
   ( cdr ( assoc letter a->i ) ) )

( define ( gcs list )
   ( foldr beside empty-image ( map ( lambda ( letter ) ( letter->image letter ) ) list ) ) )
```

Demo 1:



```
Welcome to DrRacket, version 8.2 [cs].
Language: racket, with debugging; memory limit: 256 MB.
> 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 Z)
> alphapic
(list 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 Z)
> ( display a->i )
((A . A) (B . B) (C . C) (D . D) (E . E) (F . F) (G . G) (H . H) (I . I) (J . J) (K . K) (L . L) (M . M) (N . N) (O . O) (P .
P) (Q . Q) (R . R) (S . S) (T . T) (U . U) (V . V) (W . W) (X . X) (Y . Y) (Z . Z))
> ( 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:



```
> ( gcs '( A L P H A B E T ) )
ALPHABET
> ( gcs '( M A R I O ) )
MARIO
> ( gcs '( L U I G I ) )
LUIGI
> ( gcs '( P E A C H ) )
PEACH
> ( gcs '( B O W S E R ) )
BOWSER
> ( gcs '( T O A D ) )
TOAD
> ( gcs '( D O N K E Y K O N G ) )
DONKEYKONG
> ( gcs '( Y O S H I ) )
YOSHI
> ( gcs '( D A I S Y ) )
DAISY
> ( gcs '( T O A D E T T E ) )
TOADETTE
> ( gcs '( R O S A L I N A ) )
ROSALINA
>
```