Harley Wakeman 03/21/2023

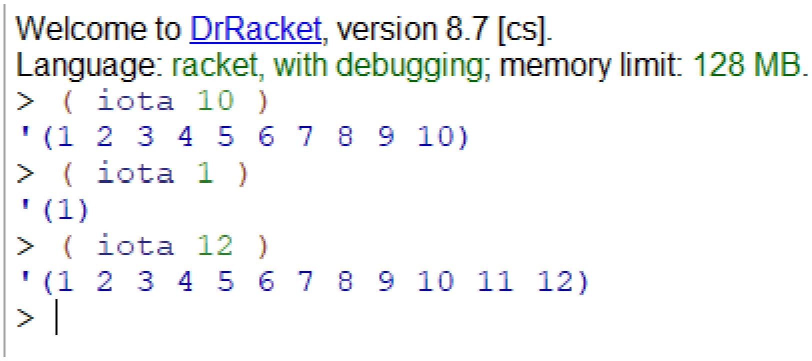
Abstract: This assignment will test my ability to use recursive list processing and higher order functions in Racket using 7 different tasks.

Task 1 Simple List Generators:

Task 1a – iota Function Definition:

```
( define ( iota integer )
   ( define ( snoc obj lst )
      ( cond
         ( ( empty? lst )
           ( list obj )
         )
         ( else
           ( cons ( car lst ) ( snoc obj ( cdr lst ) ) )
         )
       )
    )

  ( cond
     ( ( = integer 1 ) '( 1 ) )
     ( else
       ( snoc integer ( iota ( - integer 1 ) ) )
      )
   )
)
```

Task 1a – iota Demo:

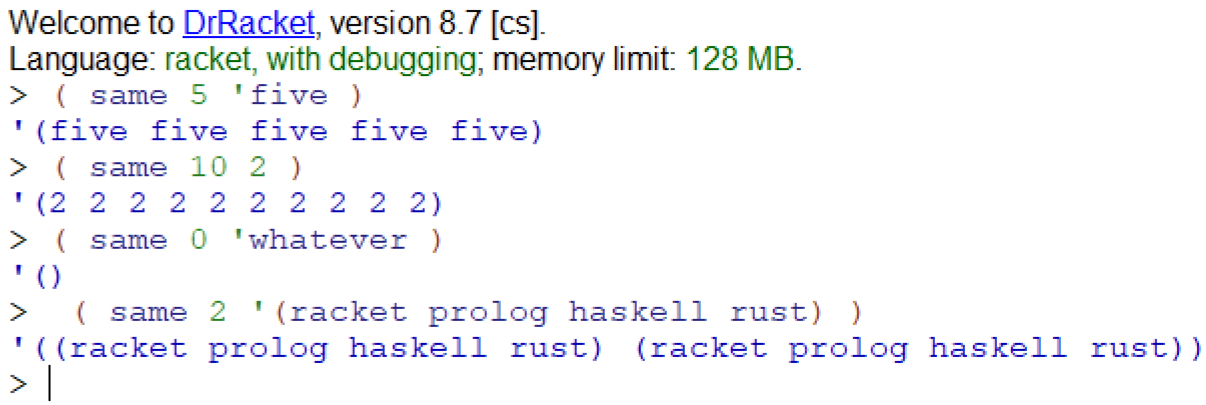


```
Welcome to DrRacket, version 8.7 [cs].
Language: racket, with debugging; memory limit: 128 MB.
> ( iota 10 )
'(1 2 3 4 5 6 7 8 9 10)
> ( iota 1 )
'(1)
> ( iota 12 )
'(1 2 3 4 5 6 7 8 9 10 11 12)
> 
```

Task 1b –same Function Definition:

```
( define ( same integer obj )
 ( cond
   ( ( zero? integer ) '( ) )
   ( else ( cons obj ( same ( - integer 1 ) obj ) ) )
 )
)
```

Task 1b –same Demo:

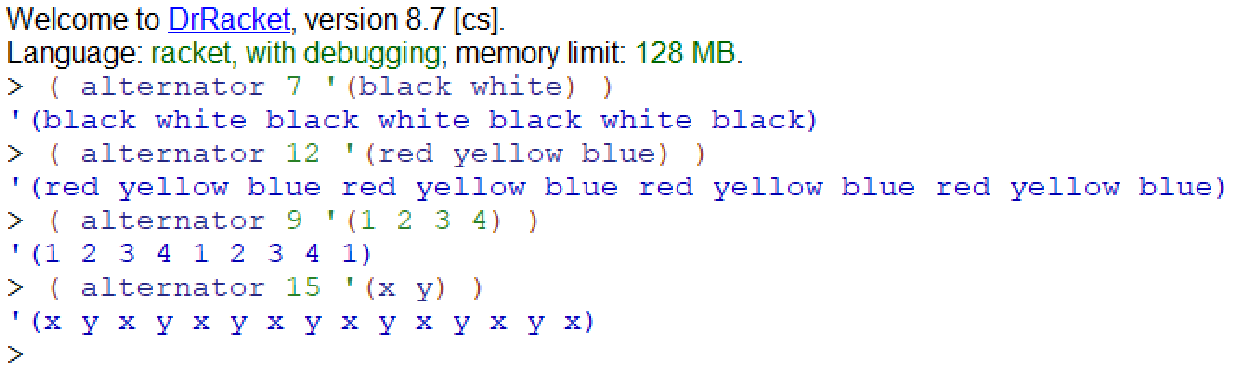


```
Welcome to DrRacket, version 8.7 [cs].
Language: racket, with debugging; memory limit: 128 MB.
> ( same 5 'five )
'(five five five five five)
> ( same 10 2 )
'(2 2 2 2 2 2 2 2 2 2)
> ( same 0 'whatever )
'()
>  ( same 2 '(racket prolog haskell rust) )
'((racket prolog haskell rust) (racket prolog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

Task 1c – Alternator Function Definition:

```
( define ( alternator integer lst )
 ( cond
   ( ( zero? integer ) '( ) )
   ( ( > ( length lst ) integer ) ( cons ( car lst ) (  alternator ( - integer 1 ) ( cdr lst ) ) ) )
   ( else ( append lst ( alternator ( - integer ( length lst ) ) lst ) ) )
 )
)
```

Task 1c – Alternator Demo:

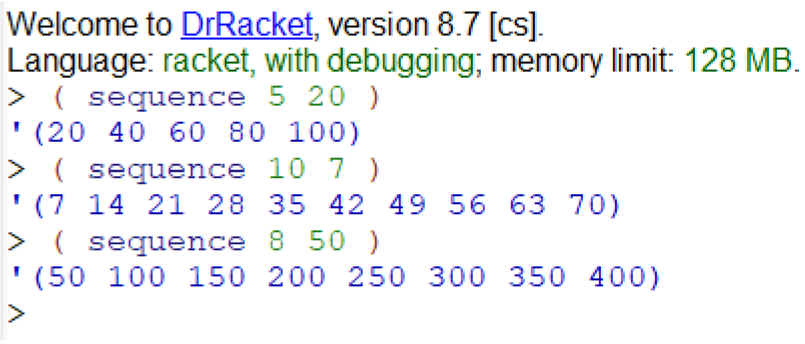


```
Welcome to DrRacket, version 8.7 [cs].
Language: racket, with debugging; memory limit: 128 MB.
> ( alternator 7 '(black white) )
'(black white black white black white black)
> ( alternator 12 '(red yellow blue) )
'(red yellow blue red yellow blue red yellow blue red yellow blue)
> ( alternator 9 '(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 integer num )
  ( define ( snoc obj lst )
    ( cond
      ( ( empty? lst )
        ( list obj )
      )
    ( else
     ( cons ( car lst ) ( snoc obj ( cdr lst ) ) )
    )
   )
  )

 ( cond
  ( ( = integer 1 ) ( list num ) )
  ( else
   ( snoc ( * integer num ) ( sequence ( - integer 1 ) num ) )
   )
  )
)
```

Task 1d –Sequence 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)
>
```

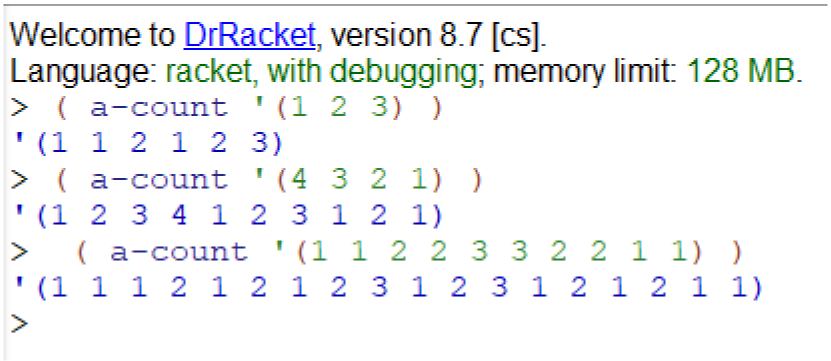
Task 2 Counting:

Task 2a – Accumulation Counting Function Definition:

```
( define ( a-count lst )
  ( define ( snoc obj lst )
    ( cond
      ( ( empty? lst )
       ( list obj )
      )
      ( else
       ( cons ( car lst ) ( snoc obj ( cdr lst ) ) )
      )
     )
   )


  ( define ( accumulation integer )
    ( cond
      ( ( zero? integer )
        '( )
       )
      ( else
       ( snoc integer ( accumulation ( - integer 1 ) ) )
      )
```

```
       )
    )


   ( cond
   ( ( empty? lst )
      '( )
    )
     ( else
     ( append  ( accumulation ( car lst ) ) ( a-count ( cdr lst ) ) )
    )
  )
)
```

Task 2a – Accumulation Counting Demo:

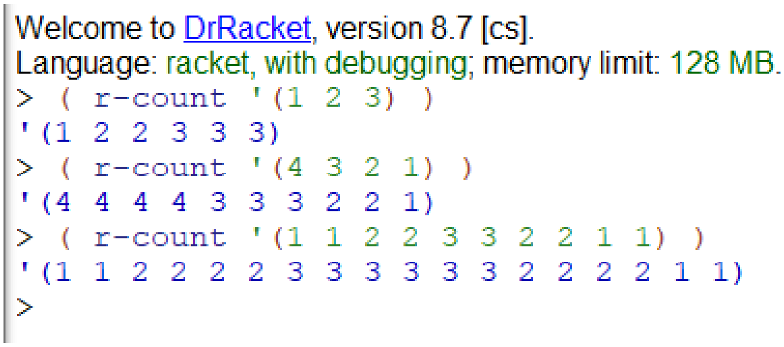


```
Welcome to DrRacket, version 8.7 [cs].
Language: racket, with debugging; memory limit: 128 MB.
> ( a-count '(1 2 3) )
'(1 1 2 1 2 3)
> ( a-count '(4 3 2 1) )
'(1 2 3 4 1 2 3 1 2 1)
>  ( a-count '(1 1 2 2 3 3 2 2 1 1) )
'(1 1 1 2 1 2 1 2 3 1 2 3 1 2 1 2 1 1)
>
```

Task 2b – Repetition Counting Function Definition:

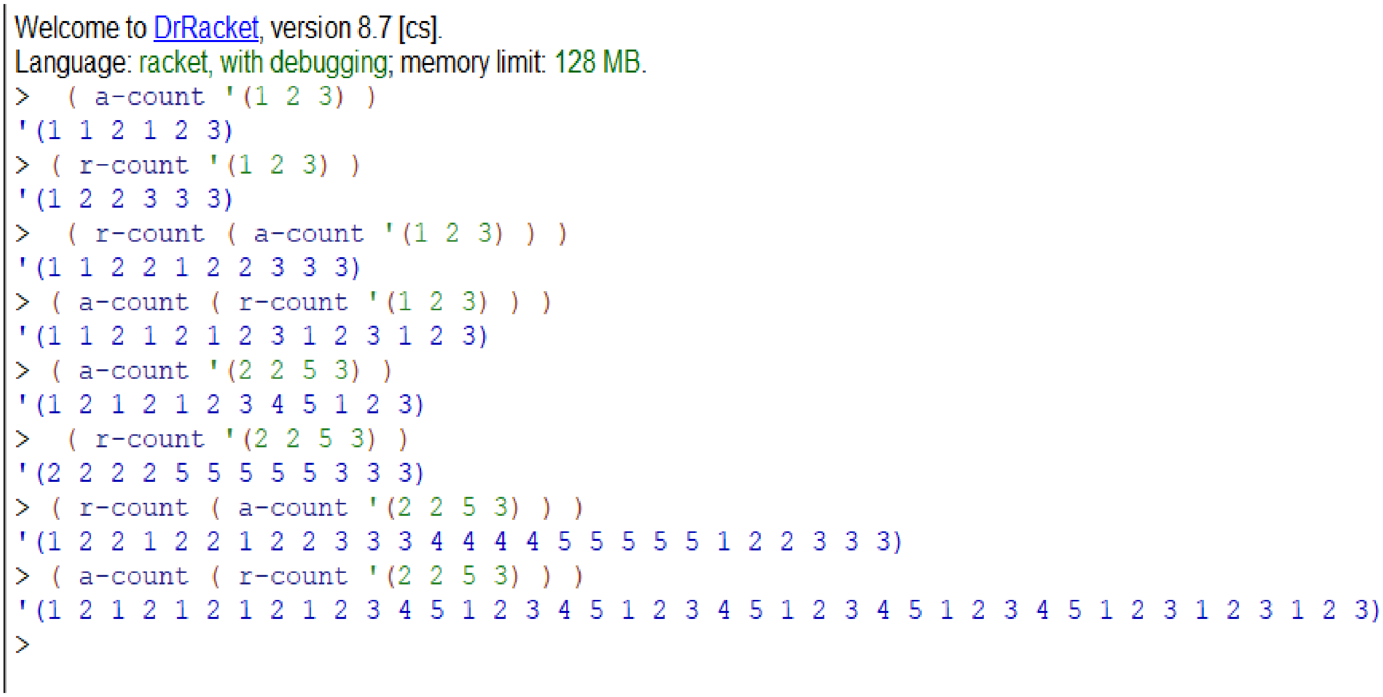
```
( define ( r-count lst )
    ( define ( snoc obj lst )
      ( cond
    ( ( empty? lst )
      ( list obj )
    )
    ( else
      ( cons ( car lst ) ( snoc obj ( cdr lst ) ) )
    )
   )
 )

( define ( repitition integer )
  ( define ( repitition-helper starting-num integer)
    ( cond
    ( ( zero? integer )
       '( )
     )
    ( else
      ( snoc starting-num ( repitition-helper starting-num ( - integer 1 ) ) )
    )
   )

  )
  ( cond
    ( ( zero? integer )
           '( )
         )
        ( else
          ( snoc integer ( repitition-helper integer ( - integer 1 ) ) )
        )
      )
   )

  ( cond
  ( ( empty? lst )
     '( )
   )
    ( else
    ( append ( repitition ( car lst ) ) ( r-count ( cdr lst ) ) )
   )
 )
)
```

Task 2b – Repetition Counting Demo:



```
Welcome to DrRacket, version 8.7 [cs].
Language: racket, with debugging; memory limit: 128 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 – Mixed Counting Demo:



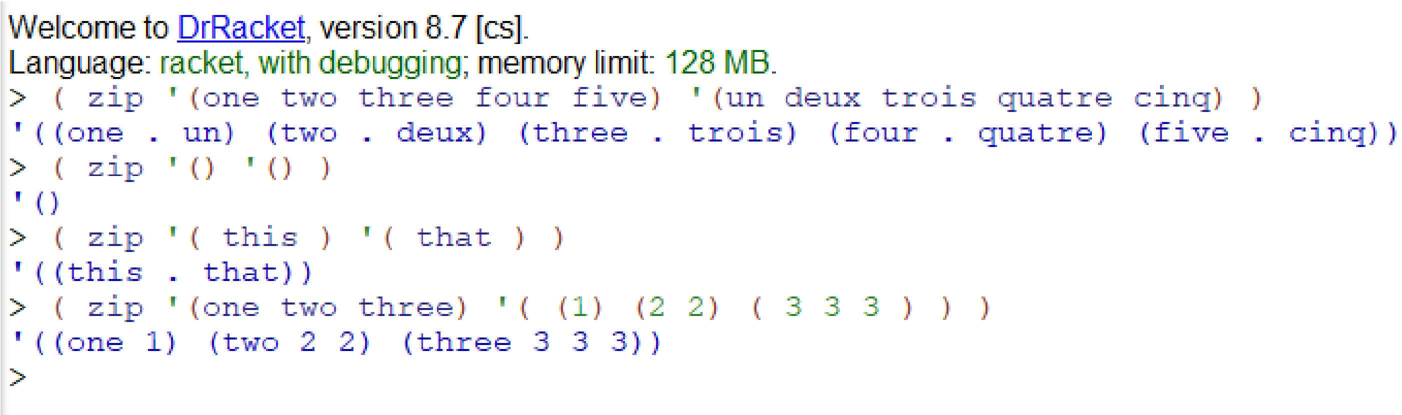
```
Welcome to DrRacket, version 8.7 [cs].
Language: racket, with debugging; memory limit: 128 MB.
>  ( a-count '(1 2 3) )
'(1 1 2 1 2 3)
> ( r-count '(1 2 3) )
'(1 2 2 3 3 3)
>  ( r-count ( a-count '(1 2 3) ) )
'(1 1 2 2 1 2 2 3 3 3)
> ( a-count ( r-count '(1 2 3) ) )
'(1 1 2 1 2 1 2 3 1 2 3 1 2 3)
> ( 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
       ( ( empty? lst1 )
         '( )
       )
       ( else
         ( cons ( cons ( car lst1 ) ( car lst2 ) ) ( zip ( cdr lst1 ) ( cdr lst2 ) ) )
       )
     )
  )
```

Task 3a – Zip Demo:

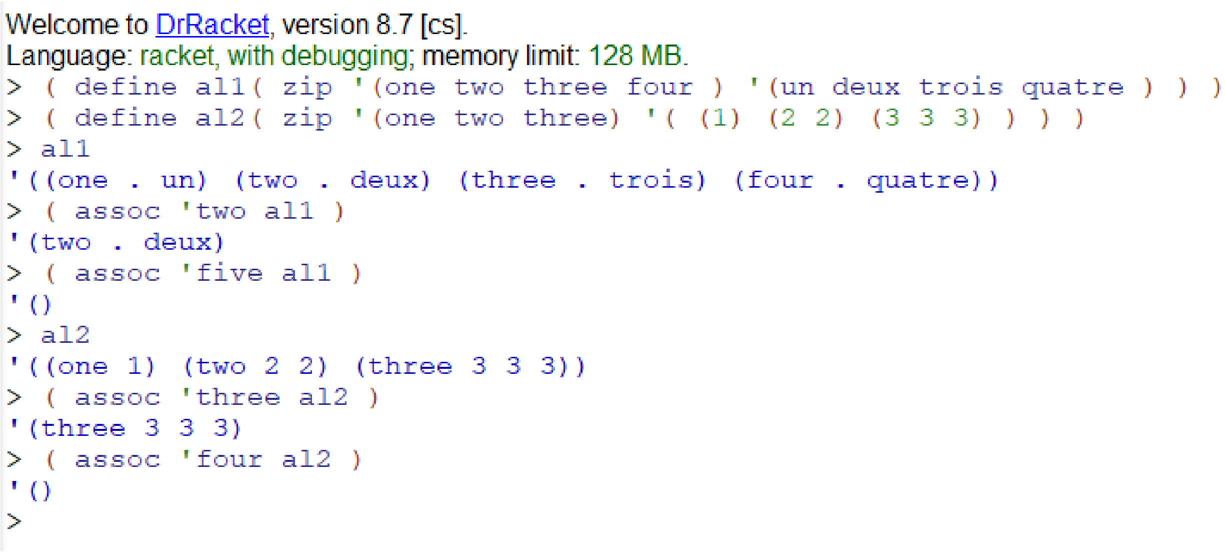


```
Welcome to DrRacket, version 8.7 [cs].
Language: racket, with debugging; memory limit: 128 MB.
> ( zip '(one two three four five) '(un deux trois quatre cinq) )
'((one . un) (two . deux) (three . trois) (four . quatre) (five . cinq))
> ( zip '() '() )
'()
> ( zip '( this ) '( that ) )
'((this . that))
> ( 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 assoc-list )
  ( cond
    (  ( = ( length assoc-list ) 0 )
      '()
    )
    ( ( equal? ( car ( car assoc-list ) ) obj )
      ( car assoc-list )
    )
    ( else
      ( assoc obj ( cdr assoc-list ) )
    )
  )
)
```

Task 3b –Assoc Demo:

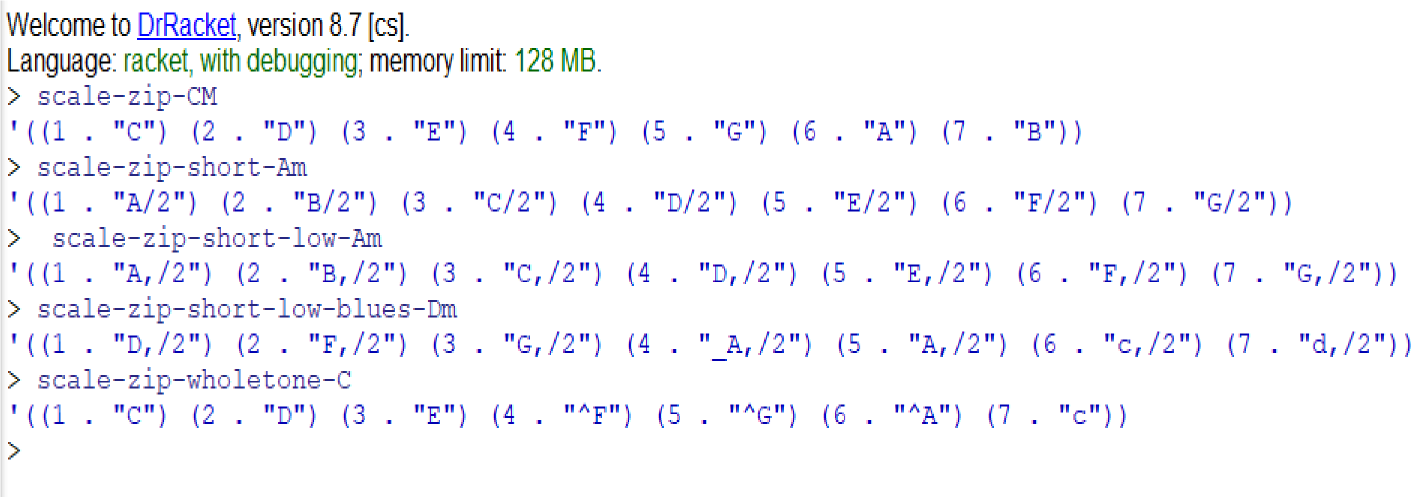


```
Welcome to DrRacket, version 8.7 [cs].
Language: racket, with debugging; memory limit: 128 MB.
> ( define al1( zip '(one two three four ) '(un deux trois quatre ) ) )
> ( define al2( zip '(one two three) '( (1) (2 2) (3 3 3) ) ) )
> al1
'((one . un) (two . deux) (three . trois) (four . quatre))
> ( 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 –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") )
)
```

Task 3c – Establishing some Association Lists Demo:



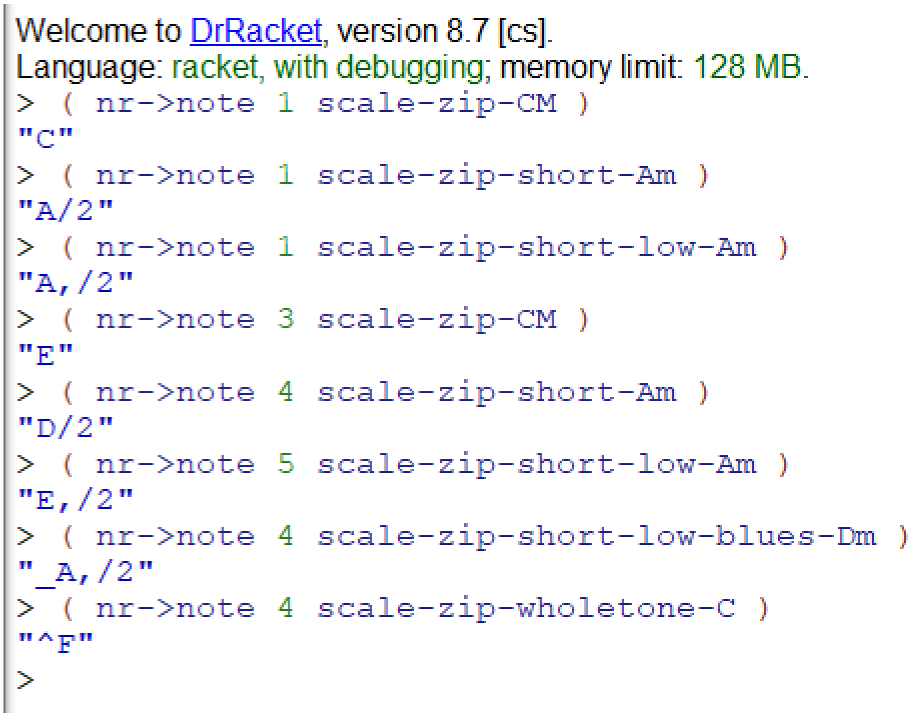
```
Welcome to DrRacket, version 8.7 [cs].
Language: racket, with debugging; memory limit: 128 MB.
> scale-zip-CM
'((1 . "C") (2 . "D") (3 . "E") (4 . "F") (5 . "G") (6 . "A") (7 . "B"))
> scale-zip-short-Am
'((1 . "A/2") (2 . "B/2") (3 . "C/2") (4 . "D/2") (5 . "E/2") (6 . "F/2") (7 . "G/2"))
>  scale-zip-short-low-Am
'((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 small-int assoc-list )
    ( cond
      (  ( = ( length assoc-list ) 0 )
        '()
      )
      ( ( = ( car ( car assoc-list ) ) small-int )
        ( cdr ( car assoc-list ) )
      )
      ( else
         ( nr->note small-int ( cdr assoc-list ) )
    )
  )
)
```

Task 4a – nr->note Demo:

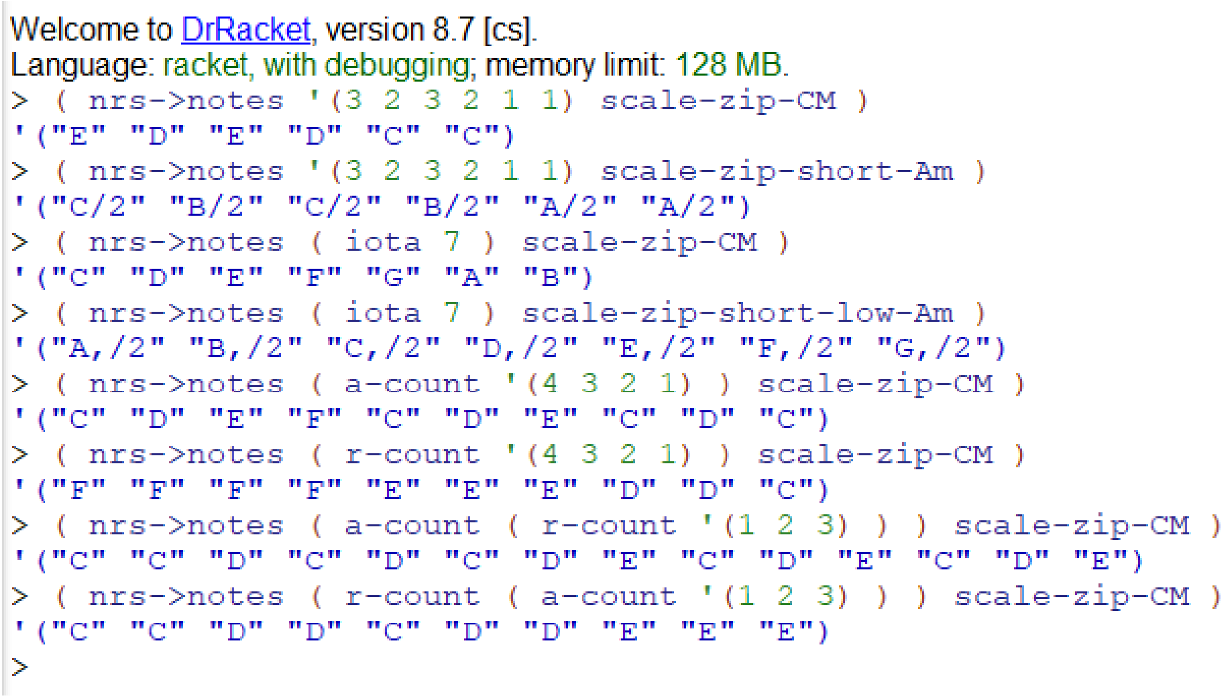


```
Welcome to DrRacket, version 8.7 [cs].
Language: racket, with debugging; memory limit: 128 MB.
> ( nr->note 1 scale-zip-CM )
"C"
> ( nr->note 1 scale-zip-short-Am )
"A/2"
> ( nr->note 1 scale-zip-short-low-Am )
"A,/2"
> ( nr->note 3 scale-zip-CM )
"E"
> ( nr->note 4 scale-zip-short-Am )
"D/2"
> ( 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 small-int-list assoc-list )
  ( map ( lambda (n) ( nr->note n assoc-list ) ) small-int-list )
)
```

Task 4b – nrs->notes Demo:

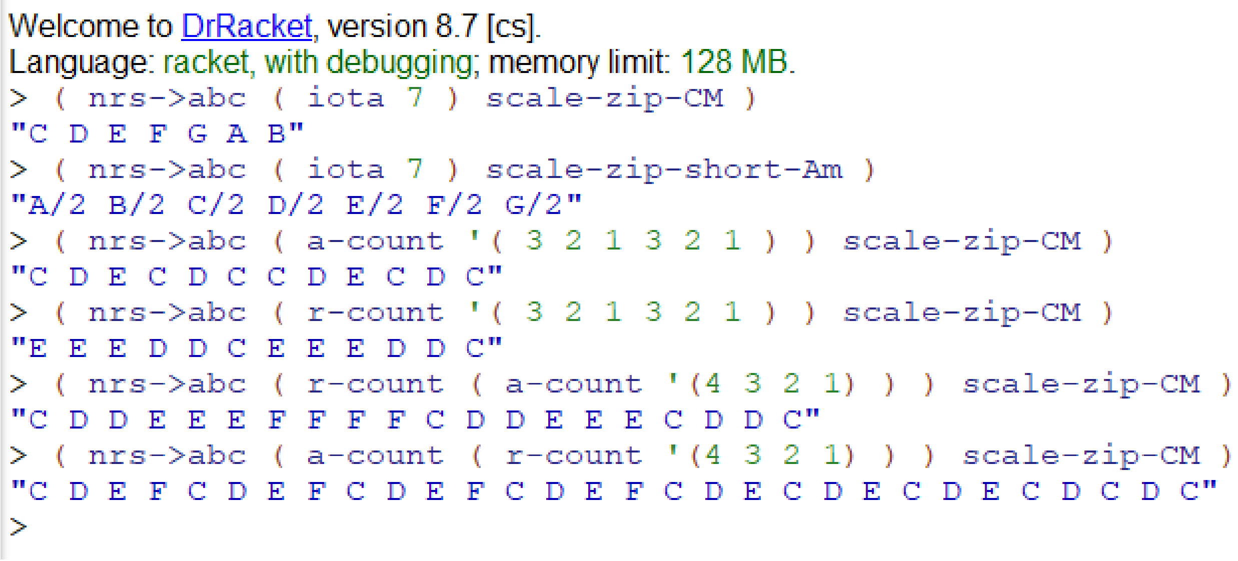


```
Welcome to DrRacket, version 8.7 [cs].
Language: racket, with debugging; memory limit: 128 MB.
> ( nrs->notes '(3 2 3 2 1 1) scale-zip-CM )
'("E" "D" "E" "D" "C" "C")
> ( nrs->notes '(3 2 3 2 1 1) scale-zip-short-Am )
'("C/2" "B/2" "C/2" "B/2" "A/2" "A/2")
> ( nrs->notes ( iota 7 ) scale-zip-CM )
'("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 small-int-list assoc-list )
  ( string-join ( map ( lambda ( n ) ( nr->note n assoc-list ) ) small-int-list ) " " )
 )
```

Task 4c – nrs->abc Demo:



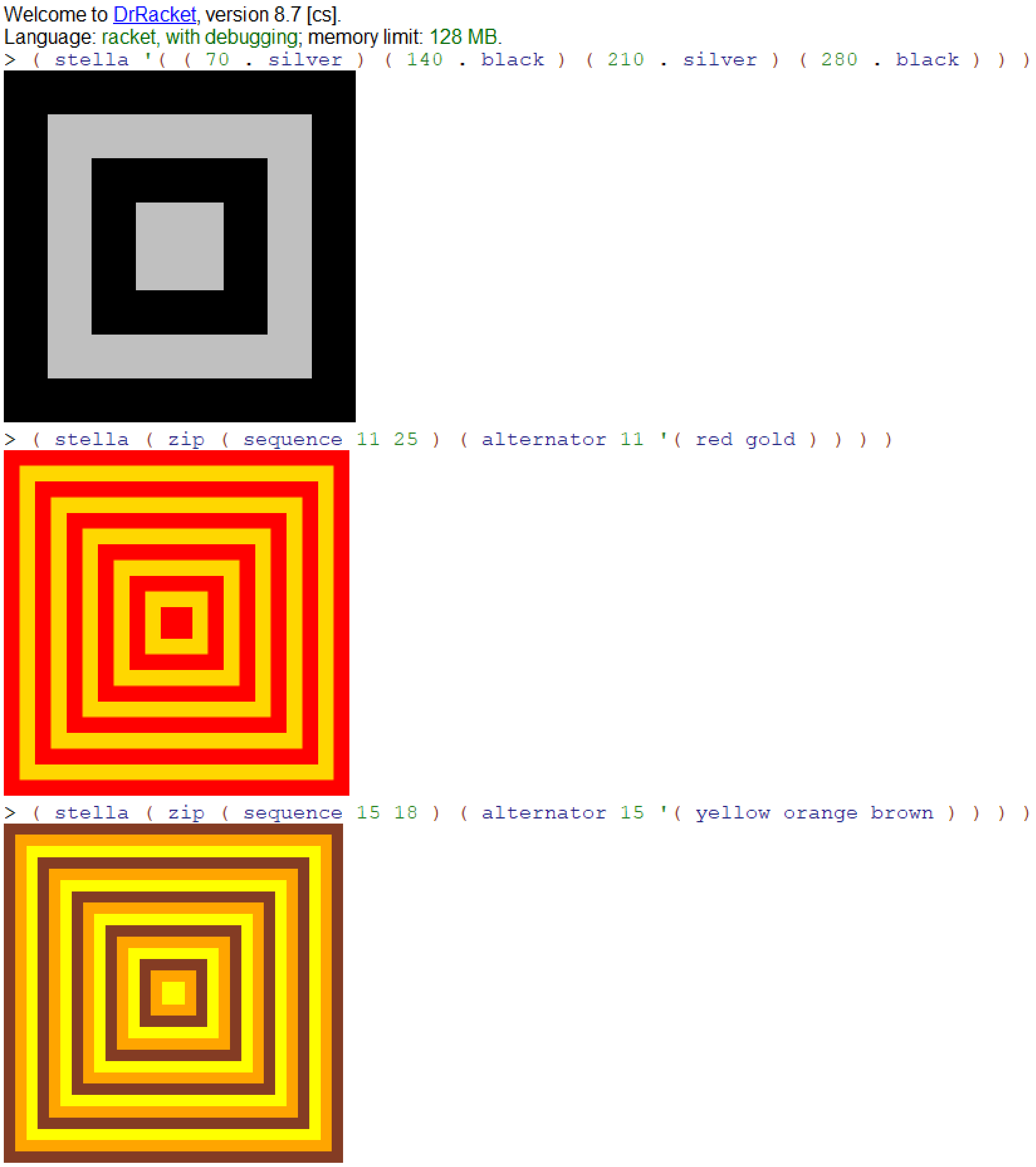
```
Welcome to DrRacket, version 8.7 [cs].
Language: racket, with debugging; memory limit: 128 MB.
> ( nrs->abc ( iota 7 ) scale-zip-CM )
"C D E F G A B"
> ( nrs->abc ( iota 7 ) scale-zip-short-Am )
"A/2 B/2 C/2 D/2 E/2 F/2 G/2"
> ( nrs->abc ( a-count '( 3 2 1 3 2 1 ) ) scale-zip-CM )
"C D E C D C C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

Task 5 Stella:

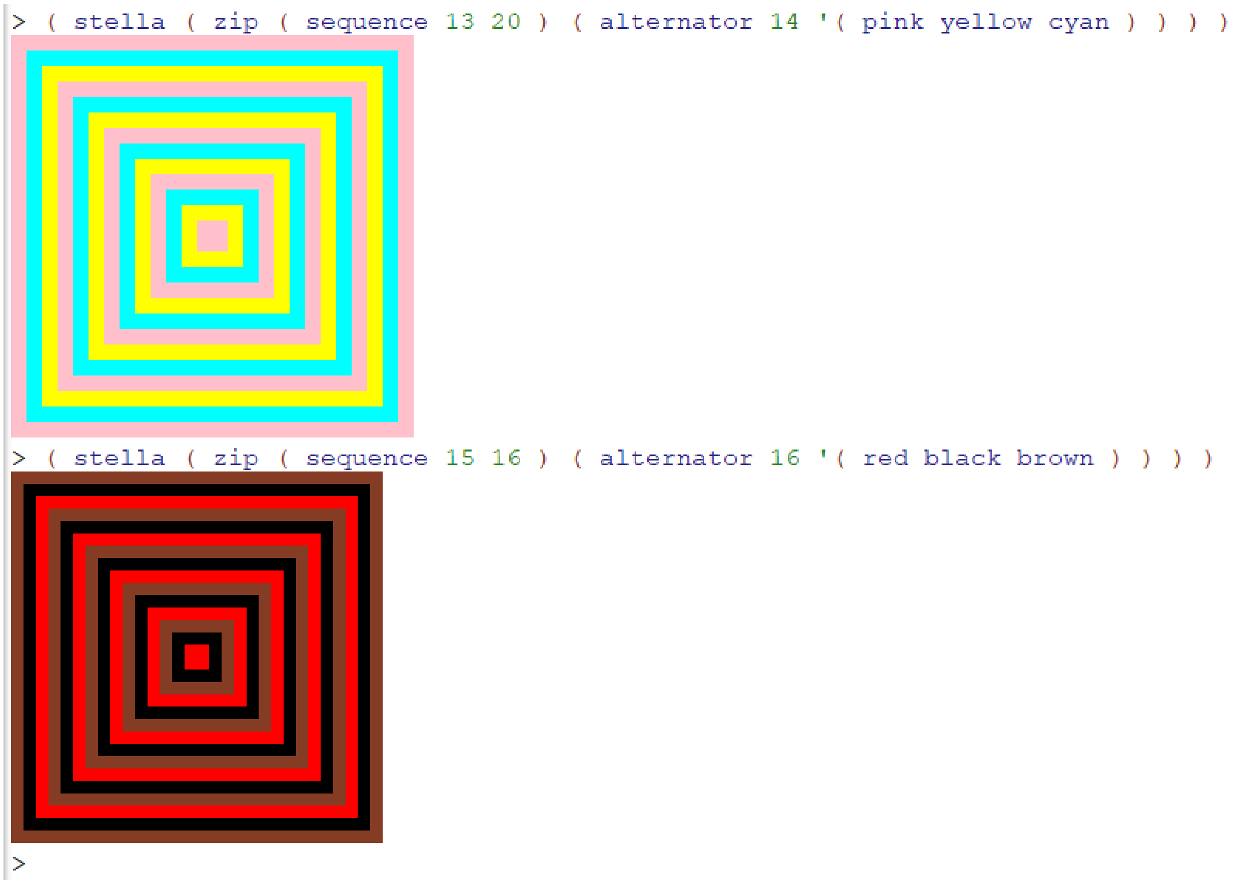
Function Definition:

```
( define ( stella assoc-list )
  ( foldr overlay empty-image
     ( map ( lambda ( n ) ( square ( car n ) "solid" ( cdr n ) ) ) assoc-list )
   )
 )
```

The Five Demos:



```
Welcome to DrRacket, version 8.7 [cs].
Language: racket, with debugging; memory limit: 128 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 ) ) ) )
```



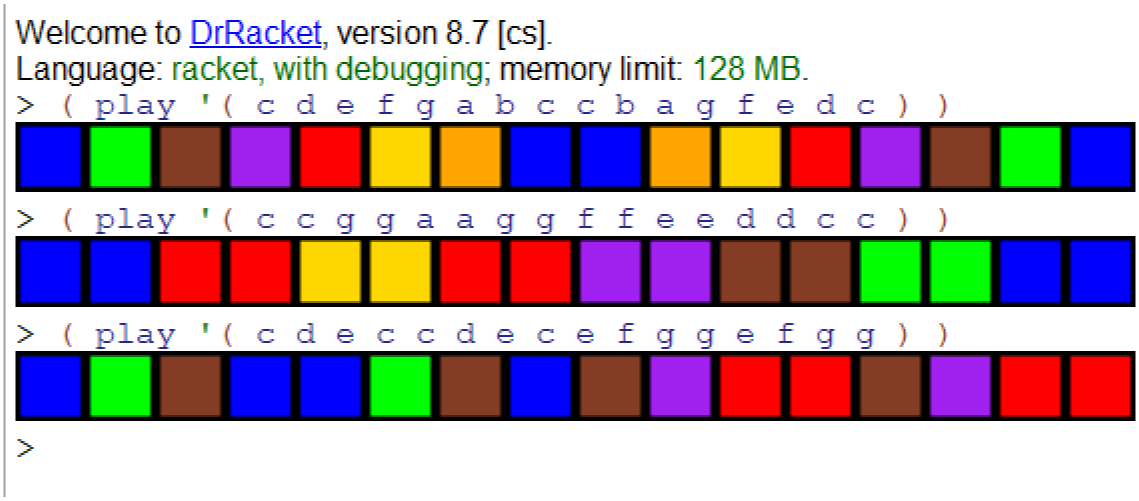
```
> ( stella ( zip ( sequence 13 20 ) ( alternator 14 '( pink yellow cyan ) ) ) )
> ( stella ( zip ( sequence 15 16 ) ( alternator 16 '( red black brown ) ) ) )
>
```

Task 6 Chromesthetic Renderings:

Code:

```
( define ( play list )
   ( define assoc-list ( zip pitch-classes boxes ) )


   ( foldr beside empty-image
     ( map ( lambda ( n ) ( cdr ( assoc n assoc-list ) ) ) list )
   )
)
```

Demo:



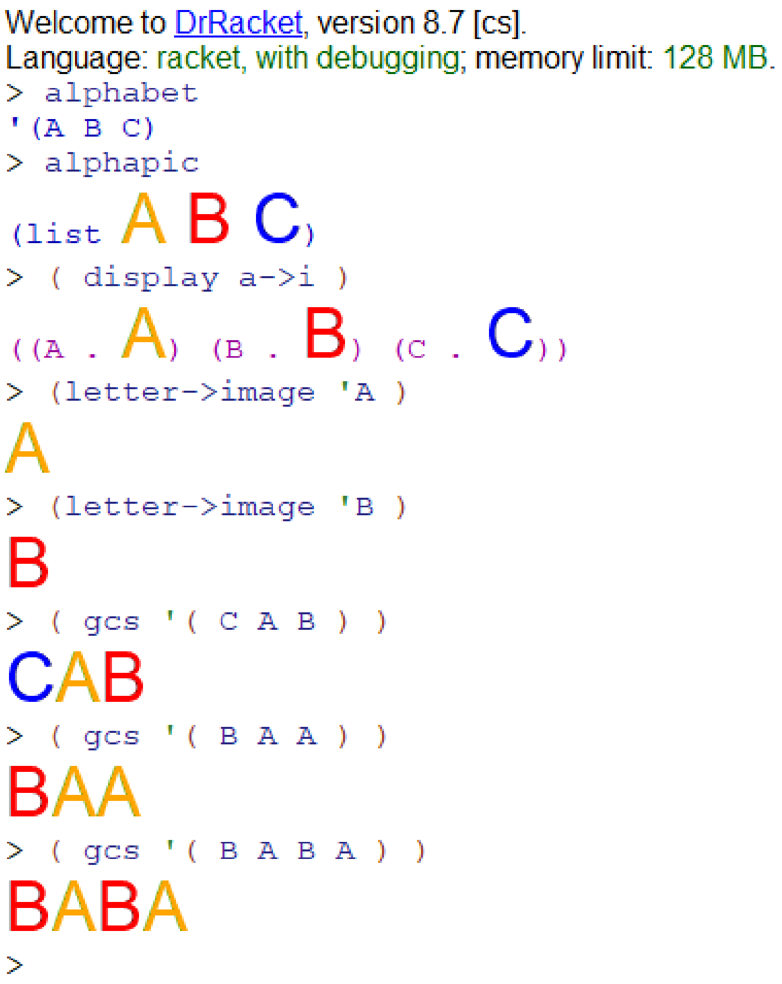
```
Welcome to DrRacket, version 8.7 [cs].
Language: racket, with debugging; memory limit: 128 MB.
> ( play '( c d e f g a b c c b a g f e d c ) )
🟦🟩🟫🟪🟥🟨🟧🟦🟦🟧🟨🟥🟪🟫🟩🟦
> ( play '( c c g g a a g g f f e e d d c c ) )
🟦🟦🟥🟥🟨🟨🟥🟥🟪🟪🟫🟫🟩🟩🟦🟦
> ( play '( c d e c c d e c e f g g e f g g ) )
🟦🟩🟫🟦🟦🟩🟫🟦🟫🟪🟥🟥🟫🟪🟥🟥
>
```

Task 7 Grapheme to Color Synesthesia:

Code:

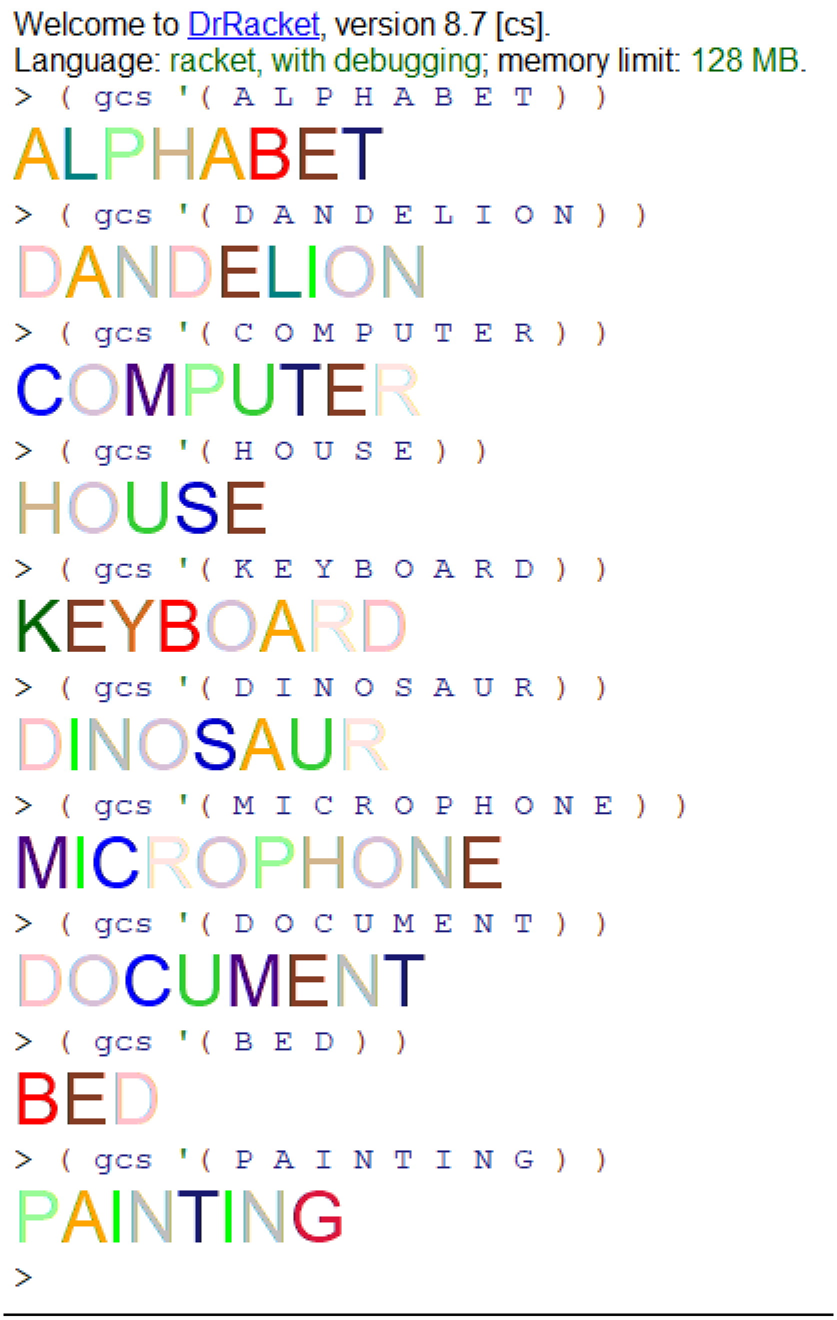
```
( define AI (text "A" 36 "orange") )
( define BI (text "B" 36 "red") )
( define CI (text "C" 36 "blue") )
( define DI (text "D" 36 "pink") )
( define EI (text "E" 36 "brown") )
( define FI (text "F" 36 "orchid") )
( define GI (text "G" 36 "crimson") )
( define HI (text "H" 36 "tan") )
( define II (text "I" 36 "green") )
( define JI (text "J" 36 "cyan") )
( define KI (text "K" 36 "dark green") )
( define LI (text "L" 36 "teal") )
( define MI (text "M" 36 "indigo") )
( define NI (text "N" 36 "gray") )
( define OI (text "O" 36 "thistle") )
( define PI (text "P" 36 "pale green") )
( define QI (text "Q" 36 "olive drab") )
( define RI (text "R" 36 "misty rose") )
( define SI (text "S" 36 "medium blue") )
( define TI (text "T" 36 "midnight blue") )
( define UI (text "U" 36 "lime green") )
( define VI (text "V" 36 "gold") )
( define WI (text "W" 36 "yellow") )
( define XI (text "X" 36 "sienna") )
( define YI (text "Y" 36 "chocolate") )
( 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 lst )
 ( foldr beside empty-image
   ( map ( lambda ( n ) ( cdr ( assoc n a->i ) ) ) lst )
  )
)
```

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 '( C O M P U T E R ) )
COMPUTER
> ( gcs '( H O U S E ) )
HOUSE
> ( gcs '( K E Y B O A R D ) )
KEYBOARD
> ( gcs '( D I N O S A U R ) )
DINOSAUR
> ( gcs '( M I C R O P H O N E ) )
MICROPHONE
> ( gcs '( D O C U M E N T ) )
DOCUMENT
> ( gcs '( B E D ) )
BED
> ( gcs '( P A I N T I N G ) )
PAINTING
>
```