Racket Assignment #5: RLP and HoFs

Learning 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 pitches to colors. Task 7 simulations grapheme to color synesthesia, in which letters are mapped to colors.

Task 1: Simple List Generators

Task 1a: iota

Function Definition

```
#lang racket
( define ( iota n )
  ( if ( <= n 0 )
    '()
    ( append ( iota ( - n 1 ) ) ( list n ) )
  )
)
```

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

```
#lang racket
( define ( same i lst )
   ( if ( <= i 0 )
      '()
      ( cons lst ( same ( - i 1 ) lst ) )
   )
)
```

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

```
#lang racket
( define ( alternator m lst )
   ( if ( = m 0 )
     '()
     ( if ( > ( length lst ) m )
        ( cons ( car lst ) ( alternator ( - m 1 ) ( cdr lst ) ) )
        ( append lst ( alternator ( - m ( length lst ) ) lst ) )
     )
   )
)
```

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

```
#lang racket
( define ( sequence x y )
   ( map ( lambda (z) (* z y) ) ( iota x ) )
)
```

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

```
#lang racket
(define (a-count lst)
  (define (add-to-end obj lst)
    (cond
      ((empty? lst)
       (list obj))
      (else
       (cons (car lst) (add-to-end obj (cdr lst))))))

  (define (make-list num)
    (cond
      ((zero? num)
       '())
      (else
       (add-to-end num (make-list (- num 1))))))

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

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

```
#lang racket
(define (r-count lst)
  (define (add-to-end obj lst)
    (cond
      ((empty? lst)
       (list obj))
      (else
       (cons (car lst) (add-to-end obj (cdr lst))))))

  (define (repetition num)
    (define (repetition-helper start num)
      (cond
        ((zero? num)
         '())
        (else
         (add-to-end start (repetition-helper start (- num 1))))))

    (cond
      ((zero? num)
       '())
      (else
       (add-to-end num (repetition-helper num (- num 1))))))

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

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

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

```
#lang racket
(define (zip lst1 lst2)
  (cond
    ((empty? 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 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

```
#lang racket
(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)))))
```

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

```
#lang racket
( define ( iota n )
  ( if ( <= n 0 )
    '()
    ( append ( iota ( - n 1 ) ) ( list n ) )
  )
)

( 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: 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

```
#lang racket
(define (nr->note smallInt assocList)
  (cond
    ((= (length assocList) 0)
     '())
    ((= (car (car assocList)) smallInt)
     (cdr (car assocList)))
    (else
     (nr->note smallInt (cdr assocList)))))
```

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

```
#lang racket
( define ( nrs->notes smallIntList assocList )
  ( map ( lambda (n) ( nr->note n assocList ) ) smallIntList )
)
```

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

```
#lang racket
(define (nrs->abc lst assocList)
  (string-join
    (map (lambda (n) (nr->note n assocList)) lst)
    " "))
```

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 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"
>
```

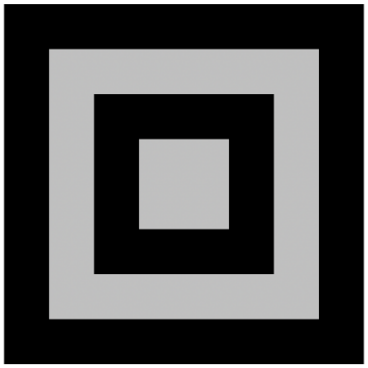
Task 5: Stella

Function Definition

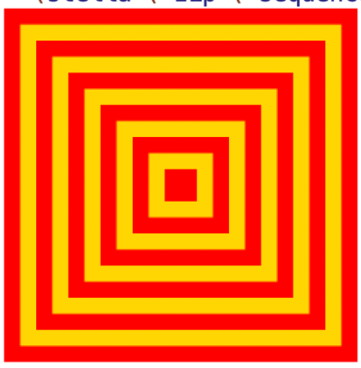
```
#lang racket
(require 2htdp/image)

(define (stella assocList)
  (foldr overlay empty-image
    (map (lambda (pair) (square (car pair) "solid" (cdr pair)))
assocList)))
```

The Five Demos

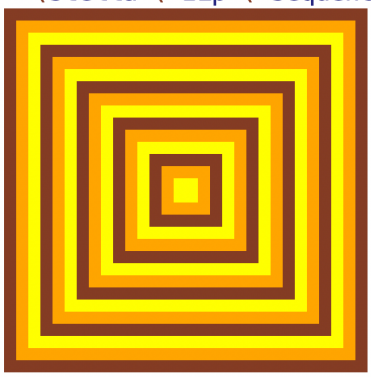
```
Welcome to DrRacket, version 8.7 [cs].
Language: racket, with debugging; memory limit: 128 MB.
> (stella '( ( 70 . silver ) ( 140 . black ) ( 210 . silver ) ( 280 . black ) ) )
```



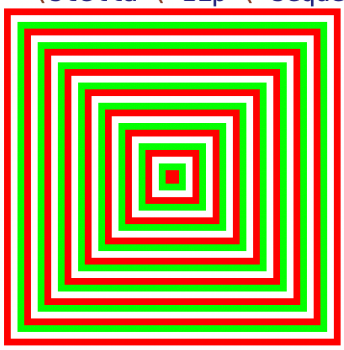
```
> (stella ( zip ( sequence 11 25 ) ( alternator 11 '( red gold ) ) ) )
```



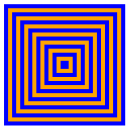
```
> (stella ( zip ( sequence 15 18 ) ( alternator 15 '( yellow orange brown ) ) ) )
```



```
> (stella ( zip ( sequence 25 10 ) ( alternator 25 '( red green white ) ) ) )
```



```
> (stella ( zip ( sequence 15 6 ) ( alternator 15 '( blue orange ) ) ) )
```



```
>
```

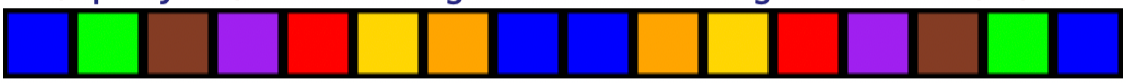
Task 6: Chromesthetic Renderings

Code

```
#lang racket
(require 2htdp/image)
( 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-input )
   ( foldr beside empty-image
       ( map ( lambda (c) ( color->box c ) )
       ( map ( lambda (pitch) ( pc->color pitch ) ) pitch-input ) )
   )
)
```

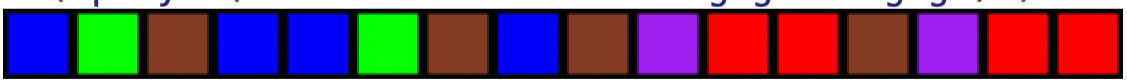
Demo

```
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 ) )
```



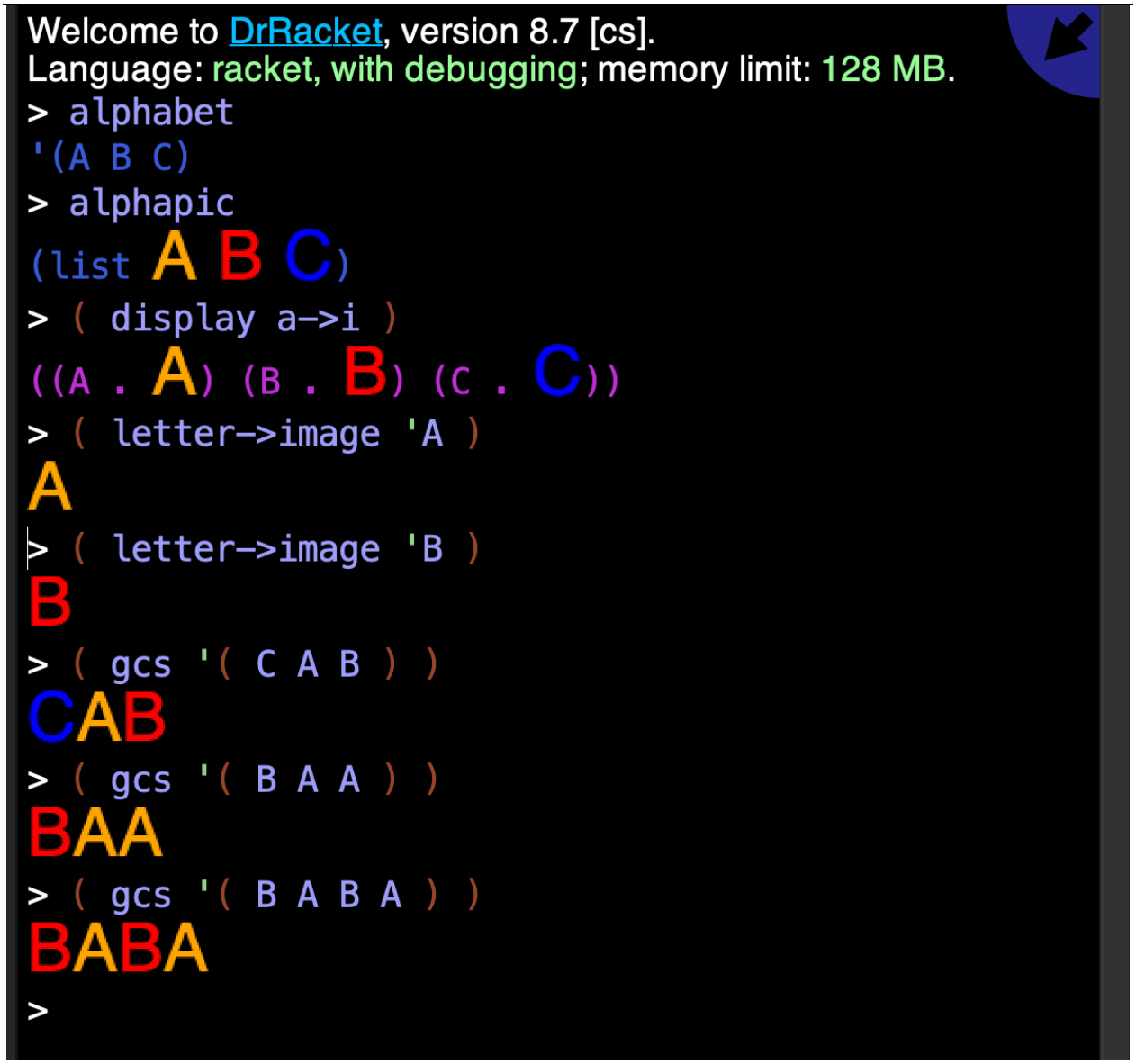
```
>
```

Task 7: Grapheme to Color Synesthesia

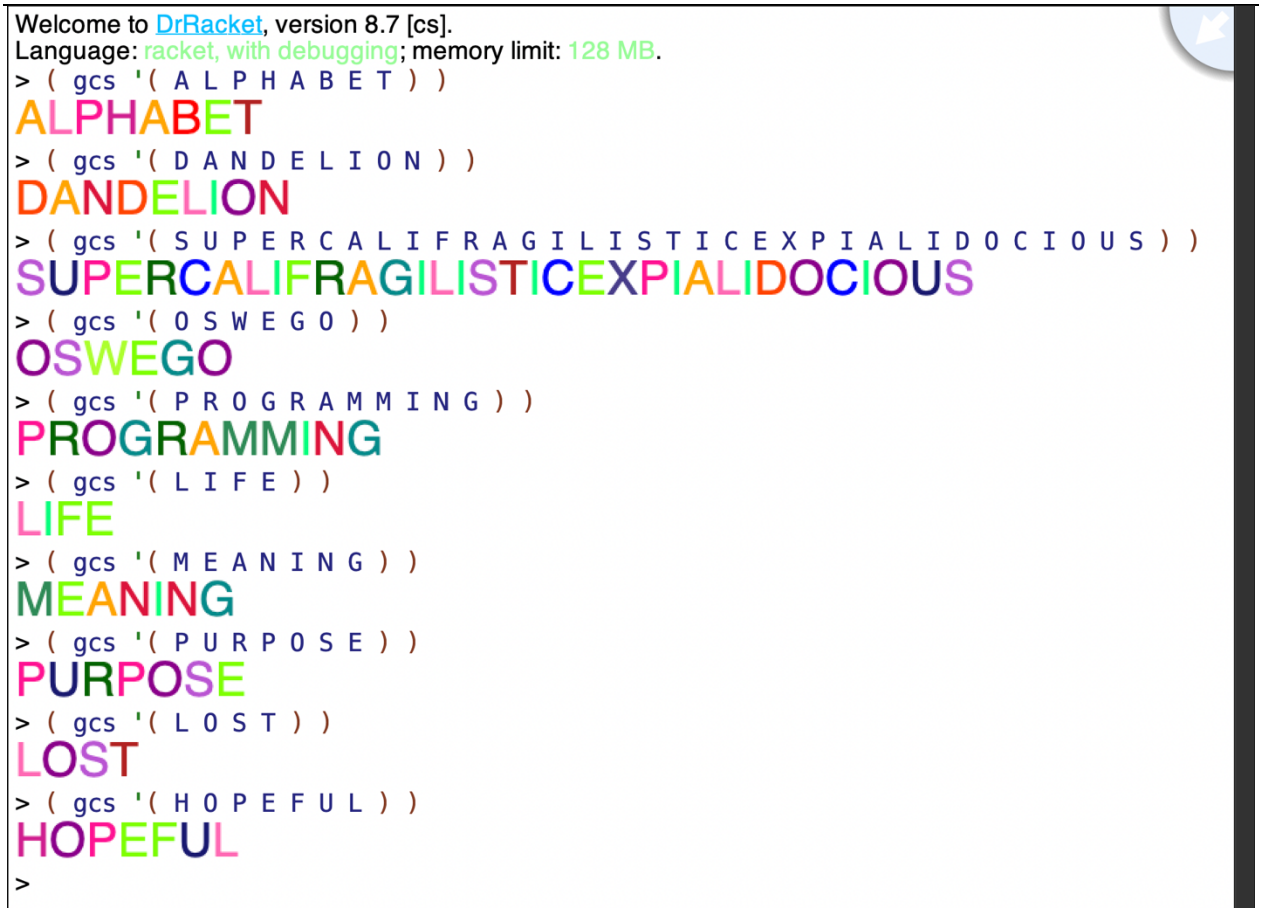
Code

```
#lang racket
(require 2htdp/image)
( define AI ( text "A" 36 "orange" ) )
( define BI ( text "B" 36 "red" ) )
( define CI ( text "C" 36 "blue" ) )
( define DI ( text "D" 36 "Orange Red" ) )
( define EI ( text "E" 36 "Chartreuse" ) )
( define FI ( text "F" 36 "Lawn Green" ) )
( define GI ( text "G" 36 "Dark Cyan" ) )
( define HI ( text "H" 36 "Violet Red" ) )
( define II ( text "I" 36 "Spring Green" ) )
( define JI ( text "J" 36 "Maroon" ) )
( define KI ( text "K" 36 "Dark Red" ) )
( define LI ( text "L" 36 "Hot Pink" ) )
( define MI ( text "M" 36 "Sea Green" ) )
( define NI ( text "N" 36 "Crimson" ) )
( define OI ( text "O" 36 "Dark Magenta" ) )
( define PI ( text "P" 36 "Deep Pink" ) )
( define QI ( text "Q" 36 "Tomato" ) )
( define RI ( text "R" 36 "Dark Green" ) )
( define SI ( text "S" 36 "Medium Orchid" ) )
( define TI ( text "T" 36 "Firebrick" ) )
( define UI ( text "U" 36 "Midnight Blue" ) )
( define VI ( text "V" 36 "Royal Blue" ) )
( define WI ( text "W" 36 "Green Yellow" ) )
( define XI ( text "X" 36 "Dark Slate Blue" ) )
( define YI ( text "Y" 36 "Light Coral" ) )
( define ZI ( text "Z" 36 "Lime" ) )

( 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 letters)
  (foldr beside empty-image (map (lambda (letter) (letter->image
letter)) letters)))
```

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 '(S U P E R C A L I F R A G I L I S T I C E X P I A L I D O C I O U S))
SUPERCALIFRAGILISTICEXPIALIDOCIOUS
> (gcs '(O S W E G O))
OSWEGO
> (gcs '(P R O G R A M M I N G))
PROGRAMMING
> (gcs '(L I F E))
LIFE
> (gcs '(M E A N I N G))
MEANING
> (gcs '(P U R P O S E))
PURPOSE
> (gcs '(L O S T))
LOST
> (gcs '(H O P E F U L))
HOPEFUL
>