Racket Programming Assignment #4: RLP and HOFs

Brandon LaPointe

Learning Abstract

This assignment includes ten separate tasks produced within the Dr. Racket coding environment focused on Recursive List Processing (RLP) and Higher Order Functions (HOFs). The first five tasks pertain to straight-forward recursive list processing. The remaining tasks pertain to list processing with higher order functions.

Task 1: Generate Uniform List

```
Welcome to DrRacket, version 8.6 [cs].
Language: racket, with debugging; memory limit: 128 MB.
> ( generate-uniform-list 5 'kitty )
'(kitty kitty kitty kitty kitty)
> ( generate-uniform-list 10 2 )
'(2 2 2 2 2 2 2 2 2 2)
> ( generate-uniform-list 0 'whatever )
'()
> ( generate-uniform-list 2 '(racket prolog haskell rust) )
'((racket prolog haskell rust) (racket prolog haskell rust))
>
```

Task 1: Source Code

```
#lang racket

; -----------------------------------------------------------------------
; Generate Uniform List Function
; EXAMPLE COMMANDS
;
; ( generate-uniform-list 5 'kitty )
; ( generate-uniform-list 0 'kitty )
; ( generate-uniform-list 2 '(racket prolog haskell rust) )

( define ( generate-uniform-list num obj )
  ( cond
    ( ( zero? num )
      ( list )
    )
    ( ( = num 1 )
      ( list obj )
    )
    ( else
      ( append ( list obj ) ( generate-uniform-list ( - num 1 ) obj ) )
    )
  )
)
```

Task 2: Association List Generator

```
Welcome to DrRacket, version 8.6 [cs].
Language: racket, with debugging; memory limit: 128 MB.
> ( a-list '(one two three four five) '(un duex trois quatre cinq) )
'((one . un) (two . duex) (three . trois) (four . quatre) (five . cinq))
> ( a-list '() '() )
'()
> ( a-list '( this ) '( that ) )
'((this . that))
> ( a-list '(one two three) '( (1) (2 2) ( 3 3 3 ) ) )
'((one 1) (two 2 2) (three 3 3 3))
>
```

Task 2: Source Code

```
#lang racket

; -----------------------------------------------------------------------
; Association List Generator Function
; EXAMPLE COMMANDS
;
; ( a-list '(one two three four five) '(un duex trois quatre cinq) )
; ( a-list '() '() )
; ( a-list '( this ) '( that ) )
; ( a-list '(one two three) '( (1) (2 2) ( 3 3 3 ) ) )
;

( define ( a-list eq-list1 eq-list2 )
  ( cond
    ( ( empty? eq-list1 )
      ( list )
    )
    ( else
      ( append ( list ( cons ( car eq-list1 ) ( car eq-list2 ) ) ) ( a-list ( cdr eq-list1 ) ( cdr eq-list2 ) ) )
    )
  )
)
```

Task 3: Assoc

```
Welcome to DrRacket, version 8.6 [cs].
Language: racket, with debugging; memory limit: 128 MB.
> ( define all ( a-list '(one two three four) '(un duex trois quatre) ) )
> ( define al2 ( a-list '(one two three) '( (1) (2 2) (3 3 3) ) ) )
> all
'((one . un) (two . duex) (three . trois) (four . quatre))
> ( assoc 'two all )
'(two . duex)
> ( assoc 'five all )
'()
> al2
'((one 1) (two 2 2) (three 3 3 3))
> ( assoc 'three al2 )
'(three 3 3 3)
> ( assoc 'four al2 )
'()
>
```

Task 3: Source Code

```
#lang racket

; -----------------------------------------------------------------------
; Assoc Functions ( a-list, assoc )
; EXAMPLE COMMANDS
;
; ( define al1 ( a-list '(one two three four) '(un duex trois quatre) ) )
; ( define al2 ( a-list '(one two three) '( (1) (2 2) (3 3 3) ) ) )
; al1
; ( assoc 'two al1 )
; ( assoc 'five al1 )
; al2
; ( assoc 'three al2 )
; ( assoc 'four al2 )
;

( define ( assoc obj assoc-list )
  ( cond
    ( ( empty? assoc-list )
      ( list )
    )
    ( ( equal? ( caar assoc-list ) obj )
      ( car assoc-list )
    )
    ( else
      ( assoc obj ( cdr assoc-list ) )
    )
  )
)
```

Task 4: Rassoc

```
Welcome to DrRacket, version 8.6 [cs].
Language: racket, with debugging; memory limit: 128 MB.
> ( define all ( a-list '(one two three four) '(un duex trois quatre) ) )
> ( define al2 ( a-list '(one two three) '( (1) (2 2) (3 3 3) ) ) )
> all
'((one . un) (two . duex) (three . trois) (four . quatre))
> ( rassoc 'three all )
'()
> ( rassoc 'trois all )
'(three . trois)
> al2
'((one 1) (two 2 2) (three 3 3 3))
> ( rassoc '(1) al2 )
'(one 1)
> ( rassoc '(3 3 3) al2 )
'(three 3 3 3)
> ( rassoc 1 al2 )
'()
>
```

Task 4: Source Code

```
#lang racket

; -----------------------------------------------------------------------
; Rassoc Functions ( a-list, rassoc )
; EXAMPLE COMMANDS
;
; ( define al1 ( a-list '(one two three four) '(un duex trois quatre) ) )
; ( define al2 ( a-list '(one two three) '( (1) (2 2) (3 3 3) ) ) )
; al1
; ( rassoc 'three al1 )
; ( rassoc 'trois al1 )
; al2
; ( rassoc '(1) al2 )
; ( rassoc '(3 3 3) al2 )
; ( rassoc 1 al2 )
;

( define ( rassoc obj rassoc-list )
  ( cond
    ( ( empty? rassoc-list )
      ( list )
    )
    ( ( equal? ( cdr ( car rassoc-list ) ) obj )
      ( car rassoc-list )
    )
    ( else
      ( rassoc obj ( cdr rassoc-list ) )
    )
  )
)
```

Task 5: Los->s

```
Welcome to DrRacket, version 8.6 [cs].
Language: racket, with debugging; memory limit: 128 MB.
> ( los->s '( "red" "yellow" "blue" "purple" ) )
"red yellow blue purple"
> ( los->s ( generate-uniform-list 20 "-" ) )
"- - - - - - - - - - - - - - - - - - - -"
> ( los->s '() )
""
> ( los->s '( "whatever" ) )
"whatever"
>
```

Task 5: Source Code

```
#lang racket

; -----------------------------------------------------------------------
; Los->S Functions ( generate-uniform-list, los->s )
; EXAMPLE COMMANDS
;
; ( los->s '( "red" "yellow" "blue" "purple" ) )
; ( los->s ( generate-uniform-list 20 "-" ) )
; ( los->s '() )
; ( los->s '( "whatever" ) )

( define ( los->s char-string-list )
  ( cond
    ( ( empty? char-string-list )
      ( string-append "" )
    )
    ( ( empty? ( cdr char-string-list ) )
      ( string-append ( car char-string-list ) )
    )
    ( else
      ( string-append ( car char-string-list ) " " ( los->s ( cdr char-string-list ) ) )
    )
  )
)
```

Task 6: Generate List – Demo 1

```
Welcome to DrRacket, version 8.6 [cs].
Language: racket, with debugging; memory limit: 128 MB.
> ( generate-list 10 roll-die )
'(3 1 2 1 5 3 2 6 6 2)
> ( generate-list 20 roll-die )
'(6 4 1 4 5 1 3 1 2 1 1 2 1 5 3 1 5 4 5 5)
> ( generate-list 12 ( lambda () ( list-ref '( red yellow blue ) ( random 3 ) ) ) )
'(red yellow red yellow blue red yellow yellow blue red blue red)
>
```

Task 6: Generate List – Demo 2

```
Welcome to DrRacket, version 8.6 [cs].
Language: racket, with debugging; memory limit: 128 MB.
> ( define dots ( generate-list 3 dot ) )
> dots
```

```
(list                                                  )
> ( foldr overlay empty-image dots )
```

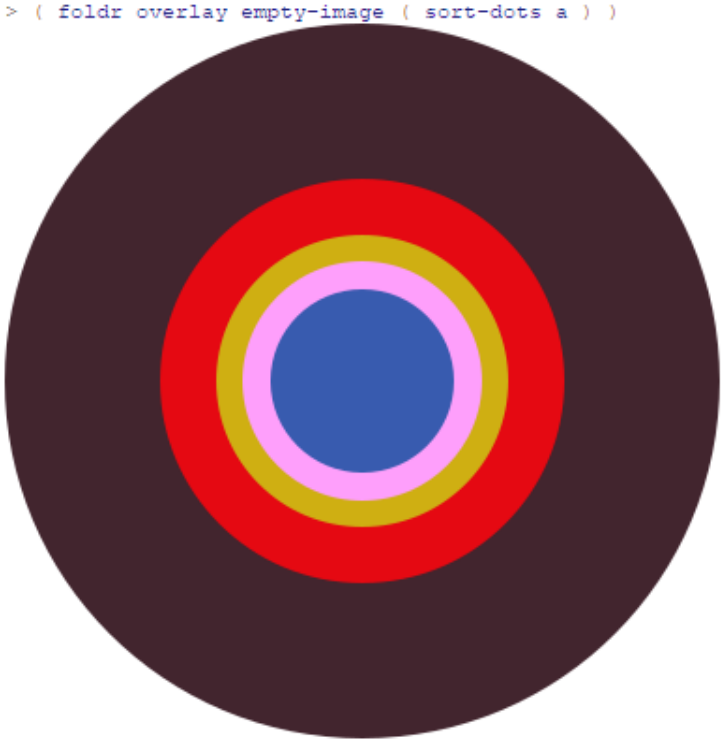
```
> ( sort-dots dots )
```

```
(list                                                  )
> ( foldr overlay empty-image ( sort-dots dots ) )
```

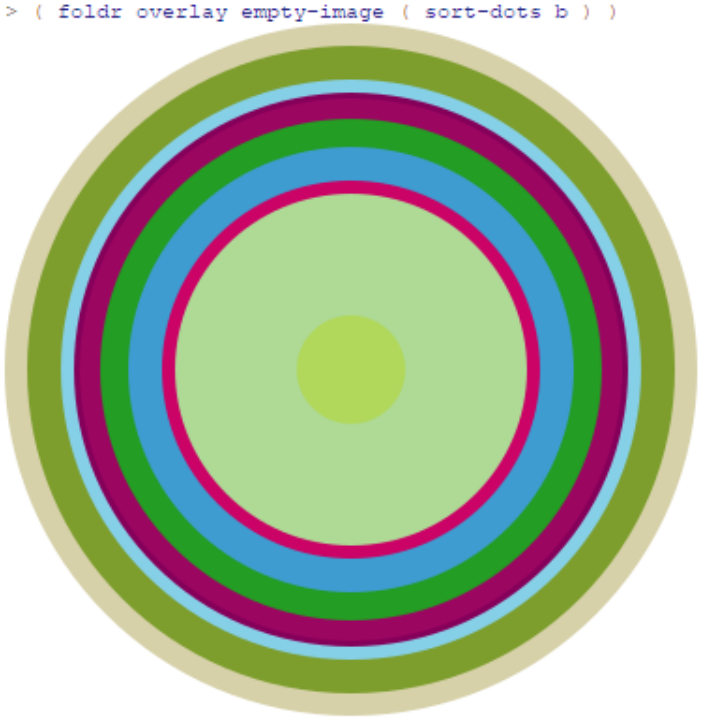
```
>
```

Task 6: Generate List – Demo 3

```
Welcome to DrRacket, version 8.6 [cs].
Language: racket, with debugging; memory limit: 128 MB.
> ( define a ( generate-list 5 big-dot ) )
> ( foldr overlay empty-image ( sort-dots a ) )
```



```
> ( define b ( generate-list 10 big-dot ) )
> ( foldr overlay empty-image ( sort-dots b ) )
```



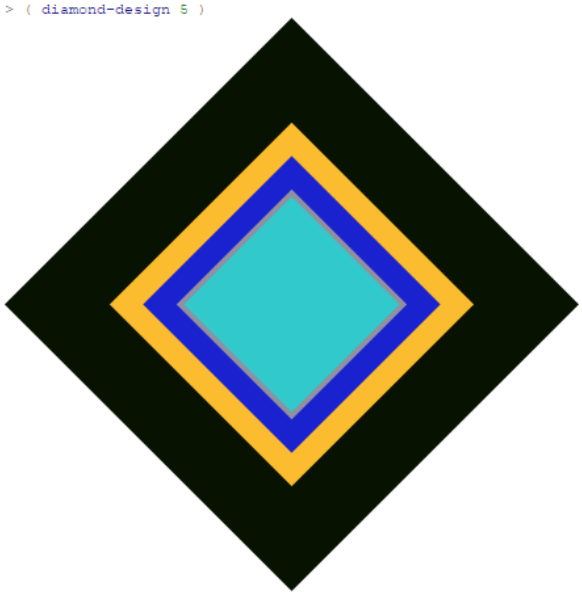
```
>
```

Task 6: Source Code

```
#lang racket

; -----------------------------------------------------------------------
; Generate List Functions and Definitions ( Auxiliary Code, big-dot, generate-list )
; EXAMPLE COMMANDS
; DEMO 1
; ( generate-list 10 roll-die )
; ( generate-list 20 roll-die )
; ( generate-list 12 ( lambda () ( list-ref '( red yellow blue ) ( random 3 ) ) ) )
; DEMO 2
; ( define dots ( generate-list 3 dot ) )
; dots
; ( foldr overlay empty-image dots )
; ( sort-dots dots )
; ( foldr overlay empty-image ( sort-dots dots ) )
; DEMO 3
; ( define a ( generate-list 5 big-dot ) )
; ( foldr overlay empty-image ( sort-dots a ) )
; ( define b ( generate-list 10 big-dot ) )
; ( foldr overlay emptry-image ( sort-dots b ) )

; -----------------------------------------------------------------------
; Import the image library from Version 2 of "How to Design Programs"
;
(require 2htdp/image)

; -----------------------------------------------------------------------
; Preliminary Auxilary Code
;

( define ( roll-die ) ( + ( random 6 ) 1 ) )

( define ( dot )
  ( circle ( + 10 ( random 41 ) ) "solid" ( random-color ) )
)

( define ( random-color )
  ( color ( rgb-value ) ( rgb-value ) ( rgb-value ) )
)

( define ( rgb-value )
  ( random 256 )
)

( define ( sort-dots loc )
  ( sort loc #:key image-width < )
)

; -----------------------------------------------------------------------
; Big-dot Function
;

( define ( big-dot )
  ( circle ( + 100 ( random 60 ) ) "solid" ( random-color ) )
)

; -----------------------------------------------------------------------
; Generate-list Function
;

( define ( generate-list num funct )
  ( cond
    ( ( zero? num )
      ( list )
    )
    ( else
      ( append ( list ( funct ) ) ( generate-list ( - num 1 ) funct ) )
    )
  )
)
```

Task 7: The Diamond – Demo 1

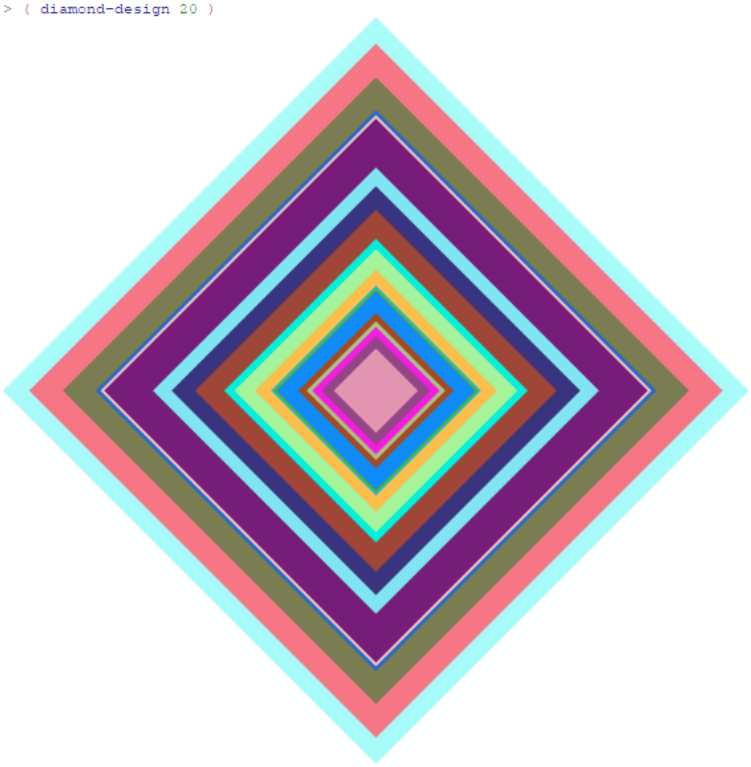
```
Welcome to DrRacket, version 8.6 [cs].
Language: racket, with debugging; memory limit: 128 MB.
> ( diamond-design 5 )
```



```
>
```

Task 7: The Diamond – Demo 2

```
Welcome to DrRacket, version 8.6 [cs].
Language: racket, with debugging; memory limit: 128 MB.
> ( diamond-design 20 )
```



```
>
```

Task 7: Source Code

```
#lang racket

; -----------------------------------------------------------------------
; Import the image library from Version 2 of "How to Design Programs"
;

(require 2htdp/image)

; -----------------------------------------------------------------------
; Random-color Function
;

( define ( rgb-value )
  ( random 256 )
)

( define ( random-color )
  ( color ( rgb-value ) ( rgb-value ) ( rgb-value ) )
)

; -----------------------------------------------------------------------
; Generate-list Function
;

( define ( generate-list num funct )
  ( cond
    ( ( zero? num )
      ( list )
    )
    ( else
      ( append ( list ( funct ) ) ( generate-list ( - num 1 ) funct ) )
    )
  )
)

; -----------------------------------------------------------------------
; Diamond Function
;

( define ( diamond )
  ( rotate 45 ( square ( + 20 ( random 380 ) ) "solid" ( random-color ) ) )
)

; -----------------------------------------------------------------------
; Sort-diamonds Function
;

( define ( sort-diamonds loc )
  ( sort loc #:key image-width < )
)

; -----------------------------------------------------------------------
; Diamond-design Function
;

( define ( diamond-design num )
  ( define group ( generate-list num diamond ) )
  ( foldr overlay empty-image ( sort-diamonds group ) )
)
```

Task 8: Chromesthetic Renderings

Welcome to DrRacket, version 8.6 [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 8: Source Code

```
#lang racket

; -----------------------------------------------------------------------
; Chromesthetic Renderings Functions and Definitions ( Auxiliary Code, play )
; EXAMPLE COMMANDS
;
; ( 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 ) )
;

; -----------------------------------------------------------------------
; Import the image library from Version 2 of "How to Design Programs"
;
(require 2htdp/image)

; -----------------------------------------------------------------------
; Auxiliary 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 ( a-list pitch-classes color-names ) )

( define cb-a-list ( a-list color-names boxes ) )

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

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

; -----------------------------------------------------------------------
; Play Function
;

( define ( play pitch-list )
  ( cond
    ( ( empty? pitch-list )
      ( display empty-image )
    )
    ( else
      ( foldr beside empty-image ( map color->box ( map pc->color pitch-list ) ) )
    )
  )
)
```

Task 9: Diner

```
Welcome to DrRacket, version 8.6 [cs].
Language: racket, with debugging; memory limit: 128 MB.
> menu
'((steak . 23) (grilledsalmon . 19) (draftbeer . 7) (clubsoda . 3) (cappuccino . 4.5) (beignets . 7.5))
> sales
'(steak
  clubsoda
  grilledsalmon
  draftbeer
  grilledsalmon
  clubsoda
  beignets
  cappuccino
  steak
  steak
  clubsoda
  steak
  draftbeer
  beignets
  cappuccino
  beignets
  cappuccino
  grilledsalmon
  draftbeer
  steak
  steak
  clubsoda
  beignets
  cappuccino
  beignets
  cappuccino
  steak
  draftbeer
  steak
  draftbeer)
> ( total sales 'steak )
184
> ( total sales 'lemonade )
0
> ( total sales 'grilledsalmon )
57
> ( total sales 'draftbeer )
35
> ( total sales 'clubsoda )
12
> ( total sales 'cappuccino )
22.5
> ( total sales 'beignets )
37.5
>
```

Task 9: Source Code

```
#lang racket

; -----------------------------------------------------------------------
; Diner Functions and Definitions ( menu-items, item-prices, menu, sales, item->price, total )
; EXAMPLE COMMANDS
;
; menu
; sales
; ( total sales 'steak )
; ( total sales 'lemonade )
; ( total sales 'grilledsalmon )
; ( total sales 'draftbeer )
; ( total sales 'clubsoda )
; ( total sales 'cappuccino )
; ( total sales 'beignets )
;

; -----------------------------------------------------------------------
; Menu-items Definition
;

( define menu-items '( steak grilledsalmon draftbeer clubsoda cappuccino beignets ) )

; -----------------------------------------------------------------------
; Item-prices Definition
;

( define item-prices '( 23 19 7 3 4.5 7.5 ) )

; -----------------------------------------------------------------------
; Menu Definition
;

( define menu ( a-list menu-items item-prices ) )

; -----------------------------------------------------------------------
; Sales Definition
;

( define sales '( steak clubsoda grilledsalmon draftbeer grilledsalmon clubsoda beignets cappuccino steak steak clubsoda steak draftbeer beignets cappuccino beignets
cappuccino grilledsalmon draftbeer steak steak clubsoda beignets cappuccino beignets cappuccino steak draftbeer steak draftbeer ) )

; -----------------------------------------------------------------------
; Item->Price Function
;

( define ( item->price item )
  ( cdr ( assoc item menu ) )
)

; -----------------------------------------------------------------------
; Total Function
;

( define ( total sales-list menu-item )
  ( define match-list ( filter ( lambda ( search-item ) ( equal? search-item menu-item ) ) sales-list ) )
  ( cond
    ( ( empty? match-list )
      0
    )
    ( else
      ( define item-sales ( map item->price match-list ) )
      ( foldr + 0 item-sales )
    )
  )
)
```

Task 10: Grapheme Color Synethesia

```
Welcome to DrRacket, version 8.6 [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 )
```

```
> ( letter->image 'B )
```

```
> ( gcs '( C A B ) )
```

```
> ( gcs '( B A A ) )
```

```
> ( gcs '( B A B A ) )
```



```
>
```

```
Welcome to DrRacket, version 8.6 [cs].
Language: racket, with debugging; memory limit: 128 MB.
> ( gcs '( A L P H A B E T ) )
```

```
> ( gcs '( D A N D E L I O N ) )
```

```
> ( gcs '( E X C L U S I V E ) )
```

```
> ( gcs '( I N I T I A T I V E ) )
```

```
> ( gcs '( A M B I D E X T R O U S ) )
```

```
> ( gcs '( E Q U I L I B R I U M ) )
```

```
> ( gcs '( Z U G Z W A N G ) )
```

```
> ( gcs '( S O C I E T Y ) )
```

```
> ( gcs '( F I G H T E R ) )
```

```
> ( gcs '( P U M P K I N S ) )
```

```
>
```

Task 10: Extended Source Code

```
; ---------------------------------------------------------------------
; Grapheme Color Synesthesia Functions
; EXAMPLE COMMANDS
;
; alphabet
; alphapic
; ( display a->i )
; ( letter->image 'A )
; ( letter->image 'B )
; ( gcs '( C A B ) )
; ( gcs '( B A A ) )
; ( gcs '( B A B A ) )
; ( gcs '( A L P H A B E T ) )
; ( gcs '( D A N D E L I O N ) )
; ( gcs '( E X C L U S I V E ) )
; ( gcs '( I N I T I A T I V E ) )
; ( gcs '( A M B I D E X T R O U S ) )
; ( gcs '( E Q U I L I B R I U M ) )
; ( gcs '( Z U G Z W A N G ) )
; ( gcs '( S O C I E T Y ) )
; ( gcs '( F I G H T E R ) )
; ( gcs '( P U M P K I N S ) )

; ---------------------------------------------------------------------
; Auxiliary Code
;

( 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 "violetred") )
( define FI (text "F" 36 "peachpuff") )
( define GI (text "G" 36 "olivedrab") )
( define HI (text "H" 36 "deepskyblue") )
( define II (text "I" 36 "indigo") )
( define JI (text "J" 36 "darkcyan") )
( define KI (text "K" 36 "goldenrod") )
( define LI (text "L" 36 "lightcoral") )
( define MI (text "M" 36 "yellow") )
( define NI (text "N" 36 "springgreen") )
( define OI (text "O" 36 "slateblue") )
( define PI (text "P" 36 "plum") )
( define QI (text "Q" 36 "sienna") )
( define RI (text "R" 36 "mediumvioletred") )
( define SI (text "S" 36 "darkorange") )
( define TI (text "T" 36 "chartreuse") )
( define UI (text "U" 36 "darkturquoise") )
( define VI (text "V" 36 "gold") )
( define WI (text "W" 36 "navy") )
( define XI (text "X" 36 "cornflowerblue") )
( define YI (text "Y" 36 "cadetblue") )
( define ZI (text "Z" 36 "chocolate") )

;( define alphabet '(A B C) )

;( define alphapic ( list AI BI CI ) )

;( define a->i ( a-list alphabet alphapic ) )

( 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 alphabet-image-a-list ( a-list alphabet alphapic ) )
```

```
; -----------------------------------------------------------------------
; Letter->Image Function
;

( define ( letter->image letter )
   ( cdr ( assoc letter alphabet-image-a-list ) )
)

; -----------------------------------------------------------------------
; GCS Function
;

( define ( gcs letter-list )
  ( cond
     ( ( empty? letter-list )
       ( empty-image )
     )
     ( else
       ( define alphapic-list ( map letter->image letter-list ) )
       ( foldr beside empty-image alphapic-list )
     )
  )
)
```