# Racket Programming Assignment #4: RLP and HoFs

**Learning Abstract**

For this assignment, there are a total of nine tasks in which the first five tasks are based on recursive list processing from Lesson 7 and the remaining tasks are based on list processing with higher order functions from Lesson 8.

## Task 1 - Generate Uniform List

## Code for Generate Uniform List

```
#lang racket
( define ( generate-uniform-list n object )
  ( cond
     ( ( = n 0  )
       '()
       )
     ( else
       ( cons object ( generate-uniform-list ( - n 1 ) object ) )
       )
     )
  )
```

## Demo of Generate Uniform List Code

```
Welcome to DrRacket, version 8.6 [cs].
Language: racket, with debugging; memory limit: 256 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 2 - Association List Generator

## Code for Association List Generator

```
#lang racket
( define ( a-list list1 list2 )
  ( define x ( length list2 ) )
  ( cond
     ( ( = x 0 )
       '()
       )
     ( else
       ( cons ( cons ( car list1 ) ( car list2 ) ) ( a-list ( cdr list1 ) ( cdr list2 ) ) )
     )
  )
 )
```

## Demo of Association List Generator Code

```
Welcome to DrRacket, version 8.6 [cs].
Language: racket, with debugging; memory limit: 256 MB.
> ( a-list '(one two three four five) '(un deux trois quatre cinq) )
'((one . un) (two . deux) (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 3 - Assoc

## Code for Assoc

```
#lang racket
( define ( a-list list1 list2 )
  ( define x ( length list2 ) )
  ( cond
     ( ( = x 0 )
       '()
       )
     ( else
       ( cons ( cons ( car list1 ) ( car list2 ) ) ( a-list ( cdr list1 ) ( cdr list2 ) ) )
     )
  )
 )


( define ( assoc text any-list )
   ( cond
       ( ( eq? any-list '() )
         '()
         )
       ( ( equal? ( car (car any-list ) ) text )
           ( car any-list )
         )
         ( else
           ( assoc text ( cdr any-list ) )
           )
         )
   )
```

## Demo of Assoc Code

```
Welcome to DrRacket, version 8.6 [cs].
Language: racket, with debugging; memory limit: 256 MB.
> ( define al1 ( a-list '(one two three four) '(un deux trois quatre ) ) )
> ( define al2 ( a-list '(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 4 - Rassoc

## Code for Rassoc

```
#lang racket
( define ( a-list list1 list2 )
  ( define x ( length list2 ) )
  ( cond
     ( ( = x 0 )
       '()
     )
     ( else
       ( cons ( cons ( car list1 ) ( car list2 ) ) ( a-list ( cdr list1 ) ( cdr list2 ) ) )
     )
  )
 )

( define ( rassoc text any-list )
 ( cond
   ( ( eq? any-list '() )
     '( )
   )
   ( ( equal? ( cdr ( car any-list ) ) text )
     ( car any-list )
   )
   ( else
       ( rassoc text ( cdr any-list ) )
     )
   )
  )
```

## Demo for Rassoc Code

```
Welcome to DrRacket, version 8.6 [cs].
Language: racket, with debugging; memory limit: 256 MB.
> ( define al1 ( a-list '(one two three four) '(un deux trois quatre) ) )
> ( define al2 ( a-list '(one two three) '( (1) (2 2) (3 3 3) ) ) )
> al1
'((one . un) (two . deux) (three . trois) (four . quatre))
> ( rassoc 'three al1 )
'()
> ( rassoc 'trois al1 )
'(three . trois)
> al2
'((one 1) (two 2 2) (three 3 3 3))
> ( rassoc '(3 3 3) al2 )
'(three 3 3 3)
> ( rassoc 1 al2 )
'()
```

# Task 5 - Los->s

## Code for Los->s

```
#lang racket
( define ( los->s anything )
   ( cond
      ( ( equal? anything '())
        ""
        )
      ( ( = ( length anything ) 1)
        ( car anything )
        )
      ( else
        ( string-append ( car anything ) " " ( los->s ( cdr anything ) ) )
        )
      )
   )

( define ( generate-uniform-list n object )
  ( cond
     ( ( = n 0  )
       '()
       )
     ( else
       ( cons object ( generate-uniform-list ( - n 1 ) object ) )
       )
     )
  )
```

## Demo for Los->s code

```
Welcome to DrRacket, version 8.6 [cs].
Language: racket, with debugging; memory limit: 256 MB.
> ( los->s '( "red" "yellow" "blue" "purple" ) )
"red yellow blue purple"
> ( los->s ( generate-uniform-list 20 "-" ) )
"- - - - - - - - - - - - - - - - - - - -"
> ( los->s '() )
""
> ( los->s '( "whatever" ) )
"whatever"
```

# Task 6 - Generate List

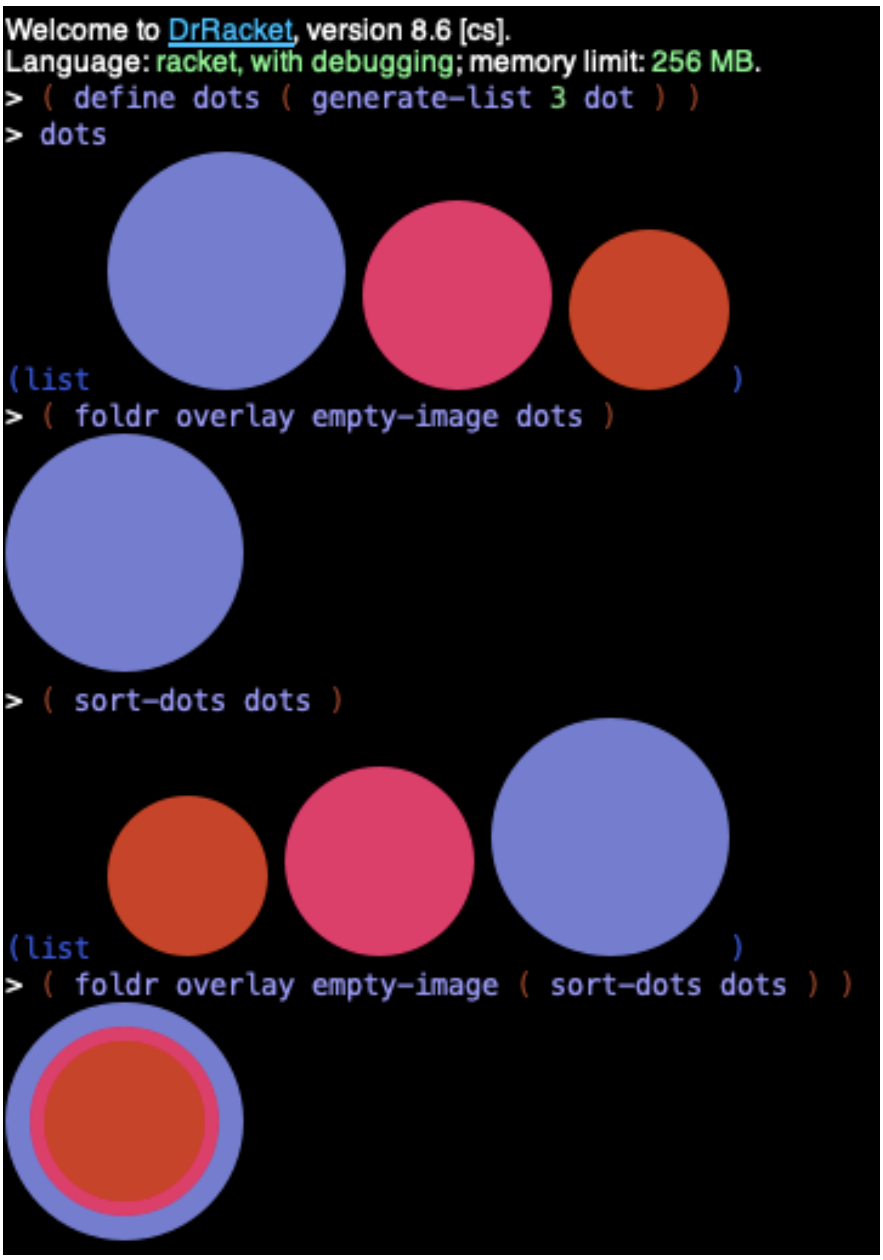
## Code for Generate List

```
#lang racket
( require 2htdp/image )
( define ( roll-die ) ( + ( random 6 ) 1 ) )

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

( define ( big-dot ) ( circle ( + 10 ( random 150 ) ) "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 < ) )



( define ( generate-list n object )
  ( cond
     ( ( = n 0  )
       '()
       )
     ( else
      ( cons (object) ( generate-list ( - n 1 ) object ) )
       )
     )
  )
```

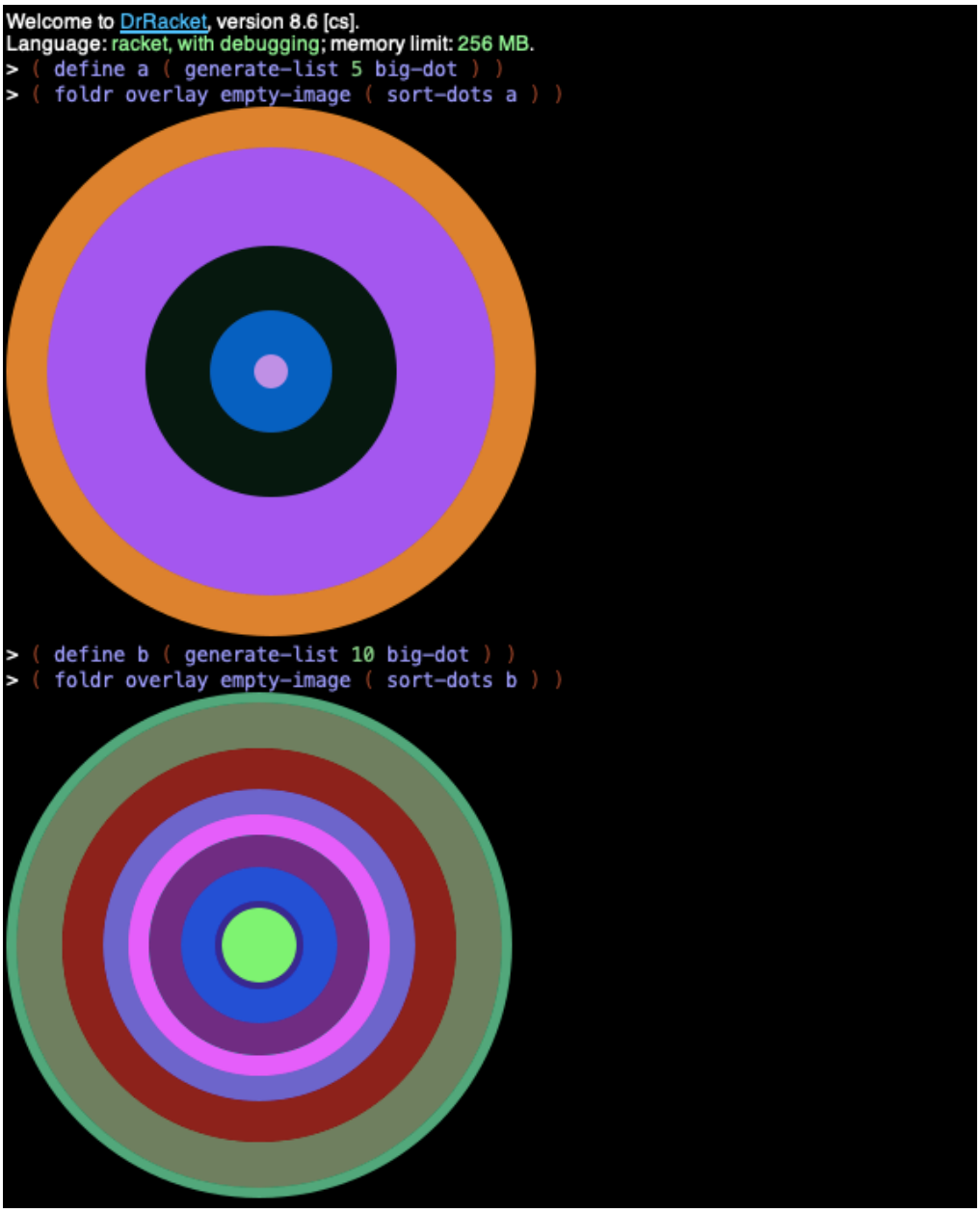
## Demo of Generate List Code (1)

```
Welcome to DrRacket, version 8.6 [cs].
Language: racket, with debugging; memory limit: 256 MB.
> ( generate-list 10 roll-die )
'(1 1 5 6 3 6 1 2 3 4)
> ( generate-list 20 roll-die )
'(3 1 2 5 5 1 5 2 3 3 1 2 3 3 4 3 5 1 1 5)
> ( generate-list 12 ( lambda () ( list-ref '( red yellow blue ) ( random 3 ) ) ) )
'(red yellow red yellow yellow yellow yellow red blue red yellow blue)
```

# Demo of Generate List Code (2)



# Demo of Generate List Code (3)



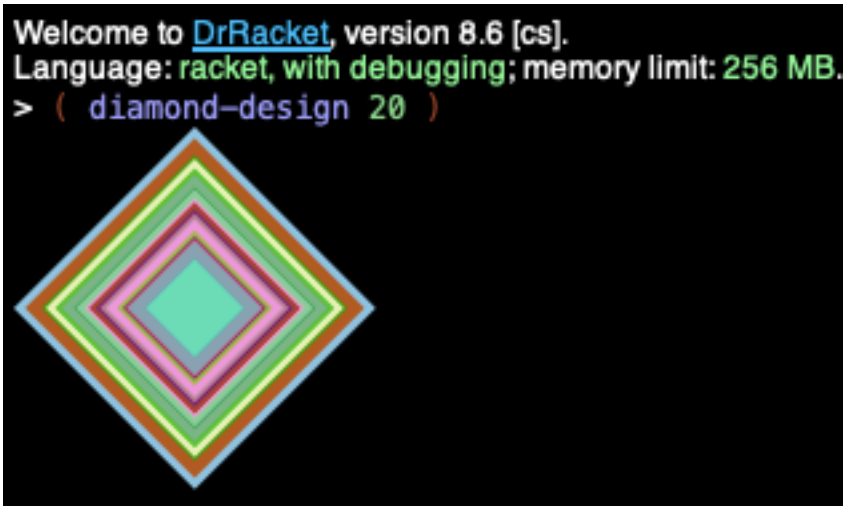
# Task 7 - The Diamond

## Code for The Diamond

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

( define ( diamond ) ( rotate 45 ( square (+ 1 ( random 19 100 ) ) "solid" ( random-color ) ) ) )

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

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

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

( define ( diamonds n )
   ( cond
      ( ( = n 0 )
         '()
        )
      ( else
        ( cons ( diamond ) ( diamonds ( - n 1 ) ) )
        )
      )
   )

( define ( diamond-design n )
  ( foldr overlay empty-image ( sort-diamonds ( diamonds n ) ) )
 )
```

## Demo of The Diamond code (1)

## Demo of The Diamond code (2)



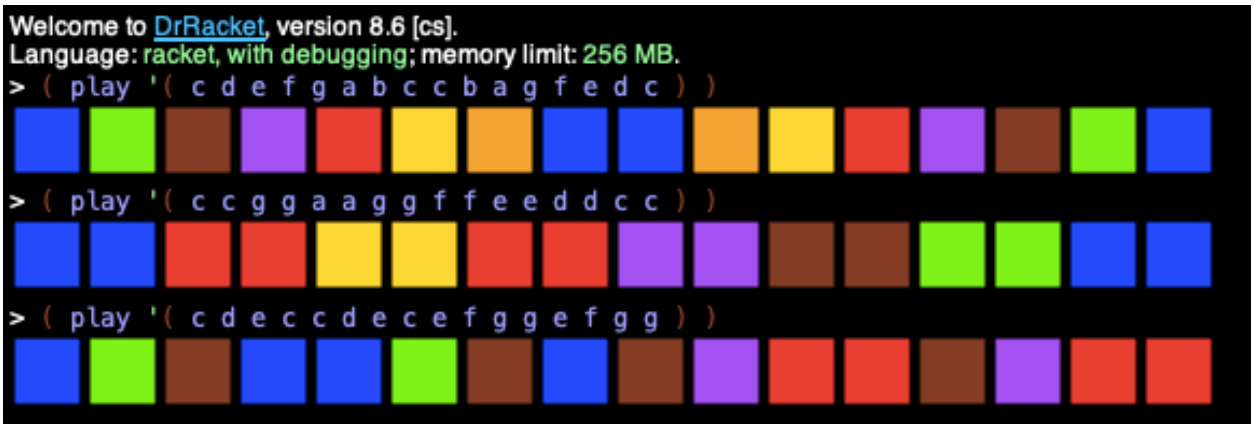

## Task 8 - Chromesthetic renderings

## Code for Chromesthetic renderings

```
#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 ( a-list list1 list2 )
  ( define x ( length list2 ) )
  ( cond
     ( ( = x 0 )
       '()
       )
     ( else
       ( cons ( cons ( car list1 ) ( car list2 ) ) ( a-list ( cdr
list1 ) ( cdr list2 ) ) )
     )
  )
 )


( 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 ) )
 )

( define ( play notes )
   ( define colors
      ( map ( lambda (a) ( pc->color a ) ) notes )
      )
   ( define rainbow-squares
      ( map ( lambda (a) ( color->box a ) ) colors )
      )
   ( foldr beside empty-image rainbow-squares )
   )
```

## Demo of Chromesthetic renderings code



# Task 9 - Diner

# Code for Diner

```
#lang racket
( define ( total sales item )
   ( foldr + 0
    ( map totalprice
    ( filter ( lambda (x)
                ( cond
                    ( ( equal? item x )
                         #t )
                       ( else
                           #f )
                       )
                )
             sales
             )
    )
    )
   )
( define ( totalprice item )
   ( cdr ( assoc item menu ) )
   )


( define ( a-list list1 list2 )
  ( define x ( length list2 ) )
  ( cond
     ( ( = x 0 )
       '()
       )
     ( else
       ( cons ( cons ( car list1 ) ( car list2 ) ) ( a-list ( cdr
list1 ) ( cdr list2 ) ) )
     )
  )
 )


( define menu ( a-list '(hamburger grilledcheese malt coke coffee pie)
'(5.5 4.5 3 1 1 3.5) ) )
( define sales
   '(hamburger
     coke
     grilledcheese
     malt
     grilledcheese
     coke
     pie
     coffee
     hamburger
```

```
    hamburger
    coke
    hamburger
    malt
    hamburger
    malt
    pie
    coffee
    pie
    coffee
    grilledcheese
    malt
    hamburger
    hamburger
    coke
    pie
    coffee
    pie
    coffee
    hamburger
    malt
    hamburger
    malt) )
```

## Demo of Diner code

```
Welcome to DrRacket, version 8.6 [cs].
Language: racket, with debugging; memory limit: 256 MB.

> menu
'((hamburger . 5.5) (grilledcheese . 4.5) (malt . 3) (coke . 1)
(coffee . 1) (pie . 3.5))
> sales
'(hamburger
  coke
  grilledcheese
  malt
  grilledcheese
  coke
  pie
  coffee
  hamburger
  hamburger
  coke
  hamburger
  malt
  hamburger
  malt
  pie
  coffee
  pie
  coffee
  grilledcheese
```

```
  malt
  hamburger
  hamburger
  coke
  pie
  coffee
  pie
  coffee
  hamburger
  malt
  hamburger
  malt)
> ( total sales 'hamburger )
49.5
> ( total sales 'hotdog )
0
> ( total sales 'grilledcheese )
13.5
> ( total sales 'malt )
18
> ( total sales 'coke )
4
> ( total sales 'coffee )
5
```

# Task 10 - Grapheme Color Synesthesia

## Task 10a - ABC

```
#lang racket
( require 2htdp/image )
( define AI ( text "A" 36 "orange" ) )
( define BI ( text "B" 36 "red" ) )
( define CI ( text "C" 36 "blue" ) )

( define alphabet '(A B C) )
( define alphapic ( list AI BI CI ) )

( define ( a-list list1 list2 )
  ( define x ( length list2 ) )
  ( cond
     ( ( = x 0 )
       '()
       )
     ( else
       ( cons ( cons ( car list1 ) ( car list2 ) ) ( a-list ( cdr
list1 ) ( cdr list2 ) ) )
     )
  )
 )
```

```
( define ( assoc text any-list )
   ( cond
       ( ( eq? any-list '() )
         '()
         )
       ( ( equal? ( car (car any-list ) ) text )
            ( car any-list )
         )
         ( else
           ( assoc text ( cdr any-list ) )
           )
         )
   )


( define a->i ( a-list alphabet alphapic ) )
( define ( letter->image alphabet) ( cdr ( assoc alphabet a->i ) ) )
( define ( gcs letters )
   ( cond
      ( ( empty? letters ) ( empty-image ) )
      (else
       ( foldr beside empty-image ( map letter->image letters  ) ) )
      )
   )
```

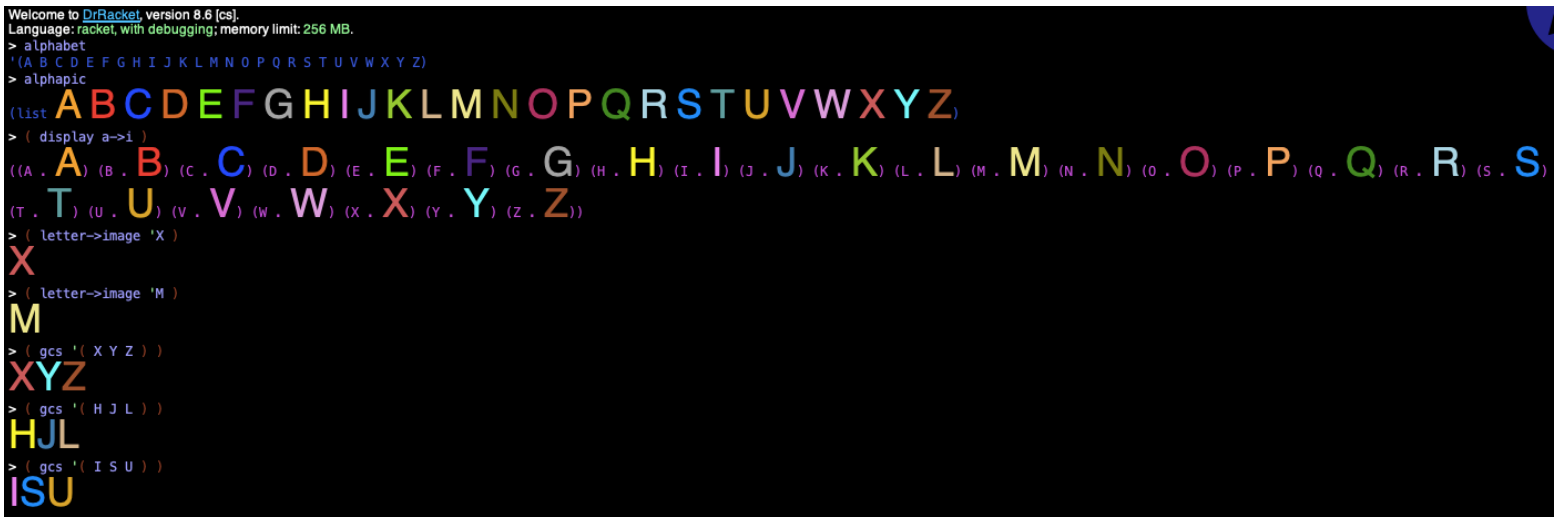
## Task 10b - ABC Demo

# Task 10c - Extended ABC code for the full alphabet

```
#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 "chocolate" ) )
( define EI ( text "E" 36 "green" ) )
( define FI ( text "F" 36 "indigo" ) )
( define GI ( text "G" 36 "dark gray" ) )
( define HI ( text "H" 36 "yellow" ) )
( define II ( text "I" 36 "violet" ) )
( define JI ( text "J" 36 "steel blue" ) )
( define KI ( text "K" 36 "yellow green" ) )
( define LI ( text "L" 36 "tan" ) )
( define MI ( text "M" 36 "khaki" ) )
( define NI ( text "N" 36 "olive" ) )
( define OI ( text "O" 36 "maroon" ) )
( define PI ( text "P" 36 "sandy brown" ) )
( define QI ( text "Q" 36 "forest green" ) )
( define RI ( text "R" 36 "light blue" ) )
( define SI ( text "S" 36 "dodger blue" ) )
( define TI ( text "T" 36 "cadetblue" ) )
( define UI ( text "U" 36 "goldenrod" ) )
( define VI ( text "V" 36 "orchid" ) )
( define WI ( text "W" 36 "plum" ) )
( define XI ( text "X" 36 "indian red" ) )
( define YI ( text "Y" 36 "aqua" ) )
( define ZI ( text "Z" 36 "sienna" ) )


( 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-list list1 list2 )
  ( define x ( length list2 ) )
  ( cond
     ( ( = x 0 )
       '()
      )
     ( else
       ( cons ( cons ( car list1 ) ( car list2 ) ) ( a-list ( cdr
list1 ) ( cdr list2 ) ) )
     )
  )
 )

( define ( assoc text any-list )
   ( cond
```

```
        ( ( eq? any-list '() )
          '()
          )
        ( ( equal? ( car (car any-list ) ) text )
             ( car any-list )
          )
          ( else
            ( assoc text ( cdr any-list ) )
            )
          )
   )


( define a->i ( a-list alphabet alphapic ) )
( define ( letter->image alphabet) ( cdr ( assoc alphabet a->i ) ) )
( define ( gcs letters )
   ( cond
      ( ( empty? letters ) ( empty-image ) )
      (else
       ( foldr beside empty-image ( map letter->image letters  ) ) )
      )
   )
```

## Task 10c - Demo of Extended ABC code for the full alphabet



# Task 10d - Alphabet Demo