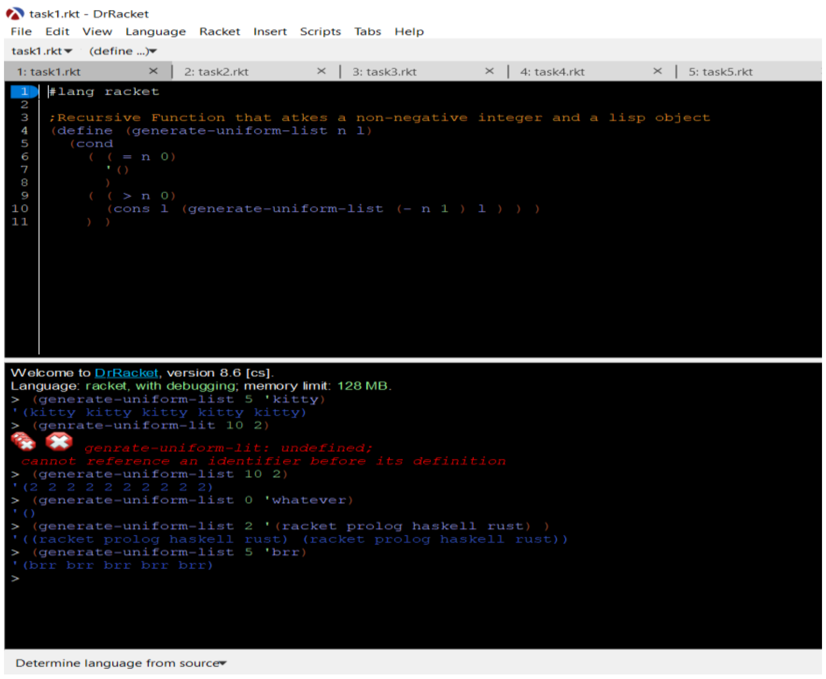
Racket Programming Assignment #4 RLP and HOF’s

Learning Abstract: For the first five tasks we were required to use relatively straightforward recursive methods for list processing, but the other half of the assignment required us to use higher order functions. The methods used in the other half of the assignment were also relatively straightforward, some borrowed code was also required to complete this task.

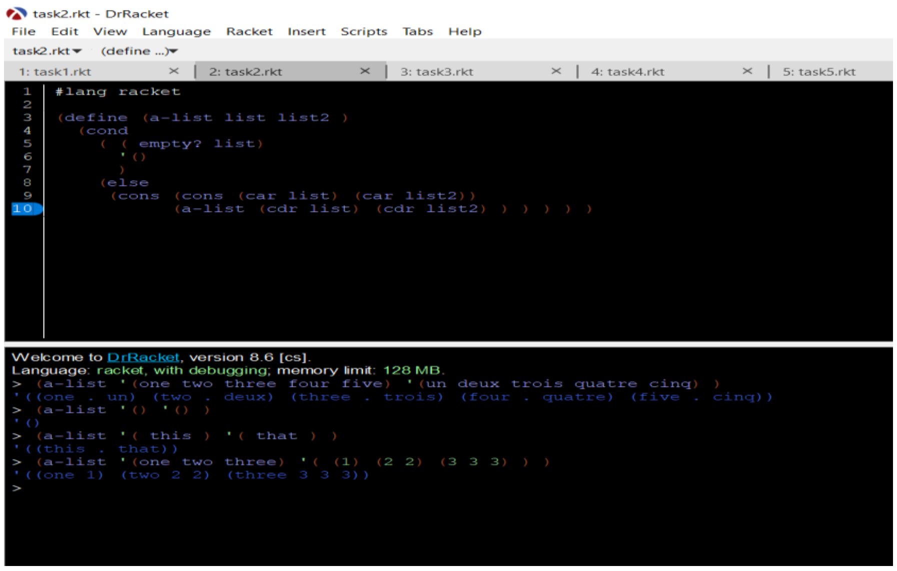
Task 1: Generate Uniform List



```racket
#lang racket

;Recursive Function that atkes a non-negative integer and a lisp object
(define (generate-uniform-list n l)
  (cond
    ( ( = n 0)
      '()
      )
    ( ( > n 0)
      (cons l (generate-uniform-list (- n 1 ) l ) ) )
    ) )
```

```
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)
> (genrate-uniform-lit 10 2)
  genrate-uniform-lit: undefined;
 cannot reference an identifier before its definition
> (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))
> (generate-uniform-list 5 'brr)
'(brr brr brr brr brr)
>
```

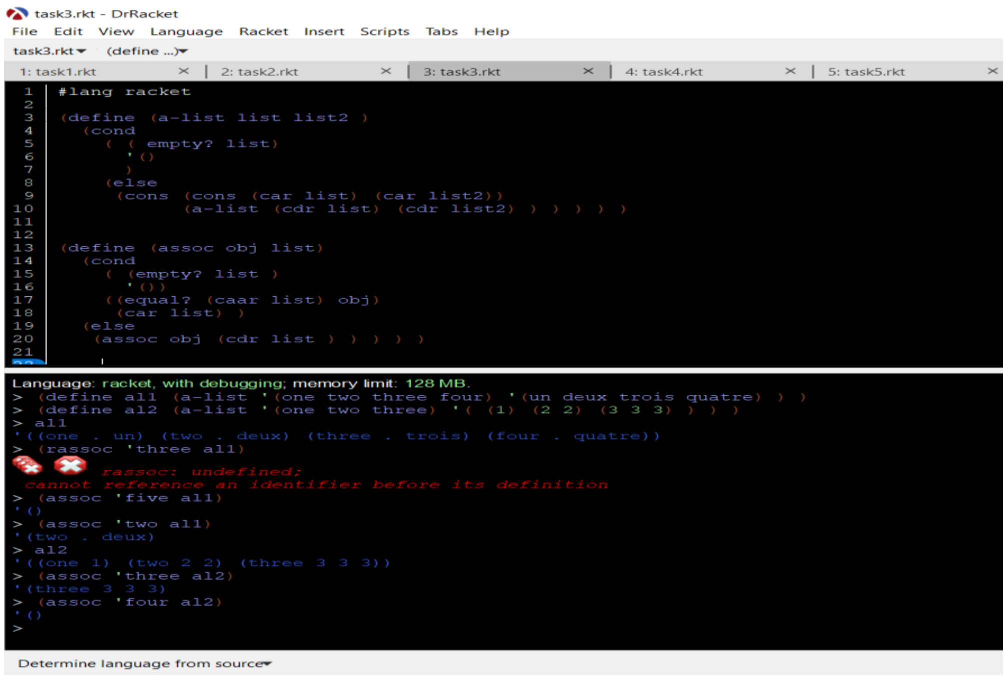
Task 2: Association List Generator



```
#lang racket

(define (a-list list list2 )
  (cond
    ( ( empty? list)
      '()
      )
    (else
     (cons (cons (car list) (car list2))
           (a-list (cdr list) (cdr list2) ) ) ) ) )
```

```
Welcome to DrRacket, version 8.6 [cs].
Language: racket, with debugging; memory limit: 128 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))
>
```

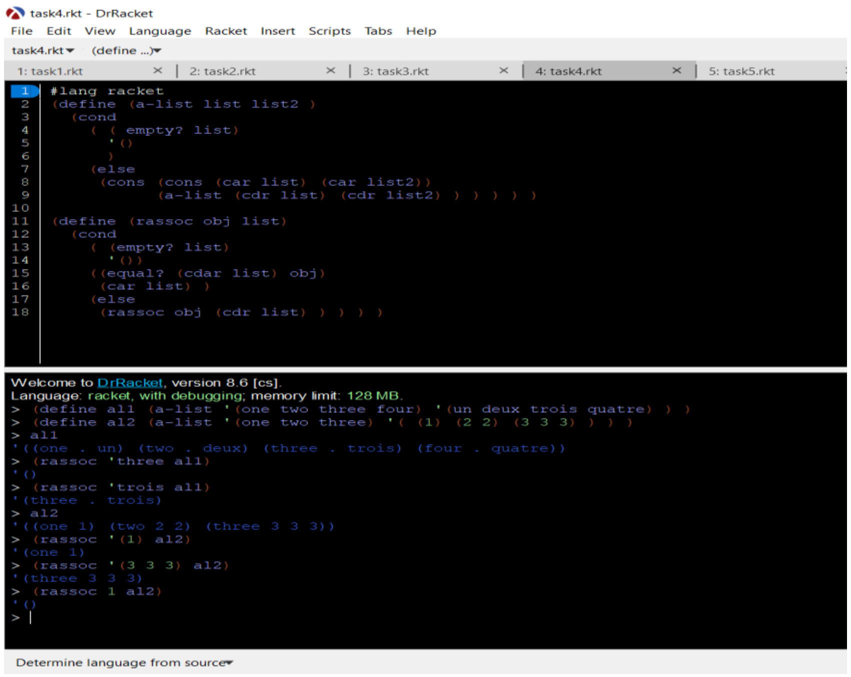
Task 3: Assoc



```
#lang racket

(define (a-list list list2 )
  (cond
    ( ( empty? list)
      '()
      )
    (else
     (cons (cons (car list) (car list2))
           (a-list (cdr list) (cdr list2) ) ) ) ) )


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

```

```
Language: racket, with debugging; memory limit: 128 MB.
> (define all (a-list '(one two three four) '(un deux trois quatre) ) )
> (define al2 (a-list '(one two three) '( (1) (2 2) (3 3 3) ) ) )
> all
'((one . un) (two . deux) (three . trois) (four . quatre))
> (rassoc 'three all)
      rassoc: undefined;
 cannot reference an identifier before its definition
> (assoc 'five all)
'()
> (assoc 'two all)
'(two . deux)
> al2
'((one 1) (two 2 2) (three 3 3 3))
> (assoc 'three al2)
'(three 3 3 3)
> (assoc 'four al2)
'()
>
```

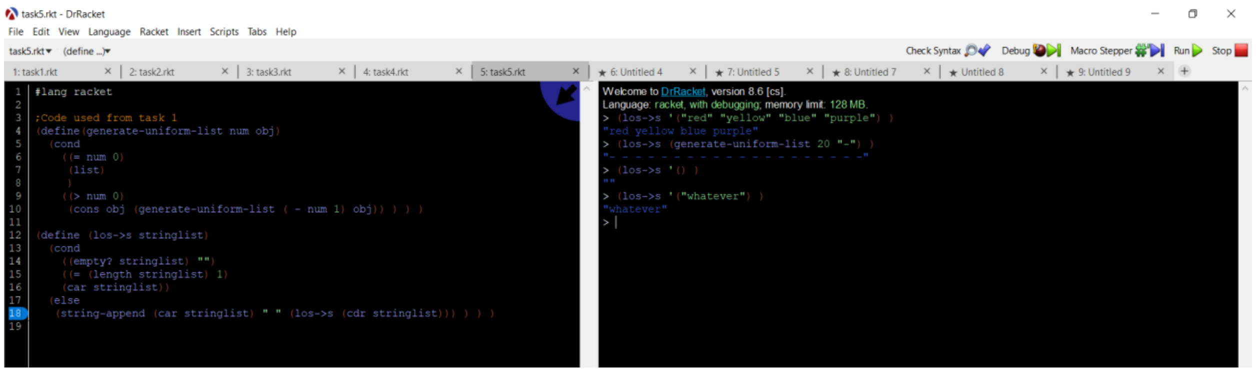
Task 4 Rassoc



```
#lang racket
(define (a-list list list2 )
  (cond
    ( ( empty? list)
      '()
      )
    (else
     (cons (cons (car list) (car list2))
           (a-list (cdr list) (cdr list2) ) ) ) ) )

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

```
Welcome to DrRacket, version 8.6 [cs].
Language: racket, with debugging; memory limit: 128 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 '(1) al2)
'(one 1)
> (rassoc '(3 3 3) al2)
'(three 3 3 3)
> (rassoc 1 al2)
'()
> |
```

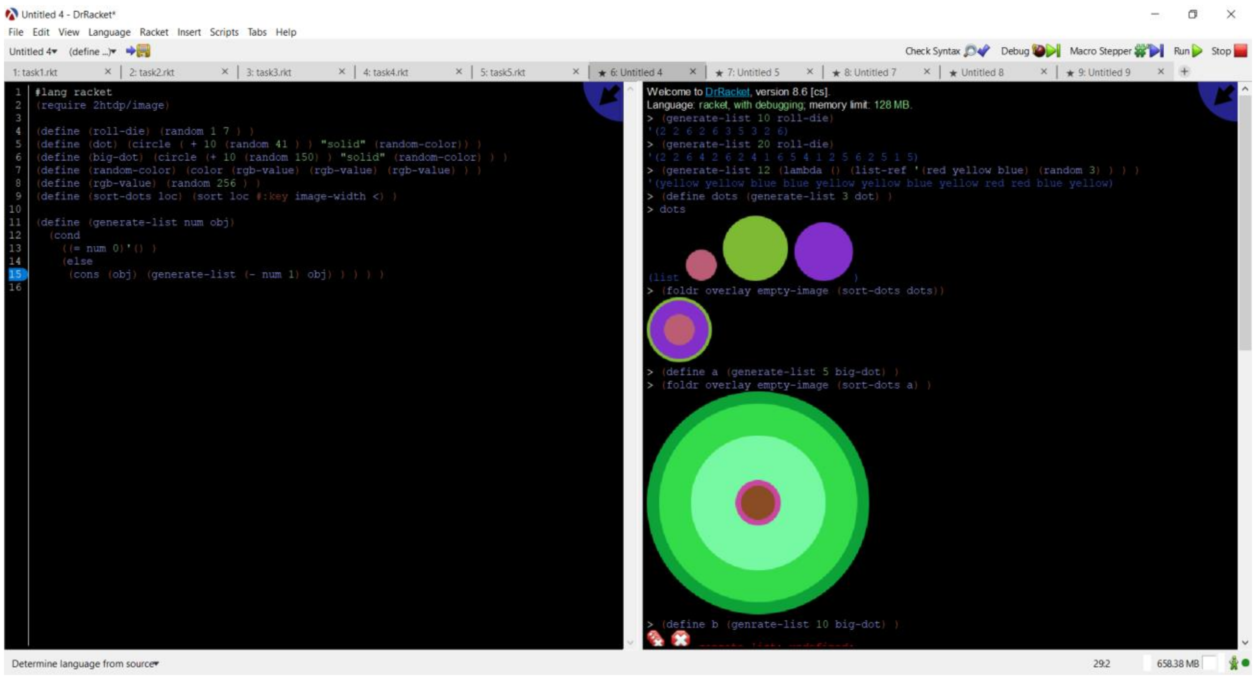
Task 5: Los->s



```
#lang racket

;Code used from task 1
(define(generate-uniform-list num obj)
  (cond
    ((= num 0)
     (list)
     )
    ((> num 0)
     (cons obj (generate-uniform-list ( - num 1) obj)) ) ) )

(define (los->s stringlist)
  (cond
    ((empty? stringlist) "")
    ((= (length stringlist) 1)
    (car stringlist))
  (else
   (string-append (car stringlist) " " (los->s (cdr stringlist))) ) ) )
```

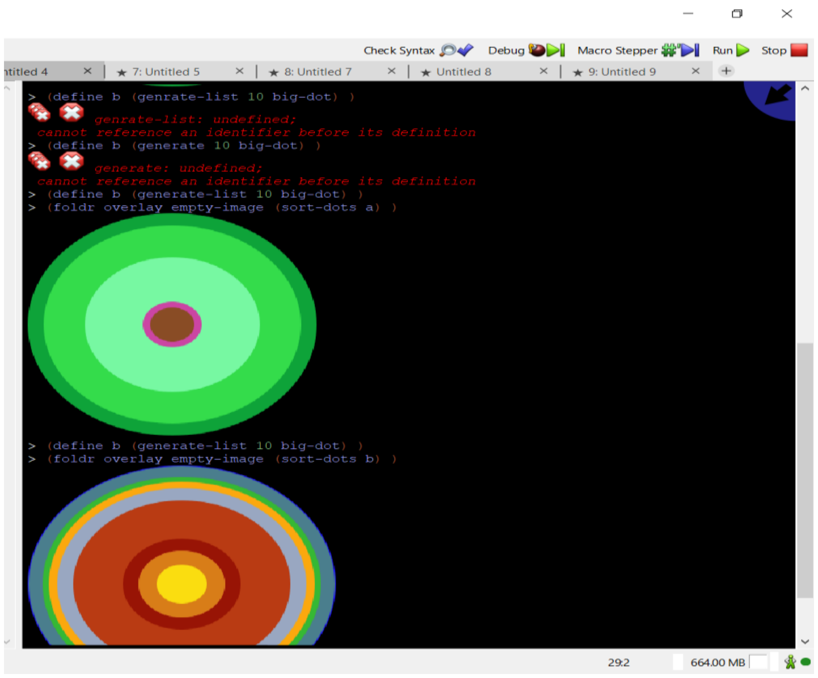
```
Welcome to DrRacket, version 8.6 [cs].
Language: racket, with debugging; memory limit: 128 MB.
> (los->s '("red" "yellow" "blue" "purple") )
"red yellow blue purple"
> (los->s (generate-uniform-list 20 "-") )
"- - - - - - - - - - - - - - - - - - - -"
> (los->s '() )
""
> (los->s '("whatever") )
"whatever"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

Task 6: Generate list



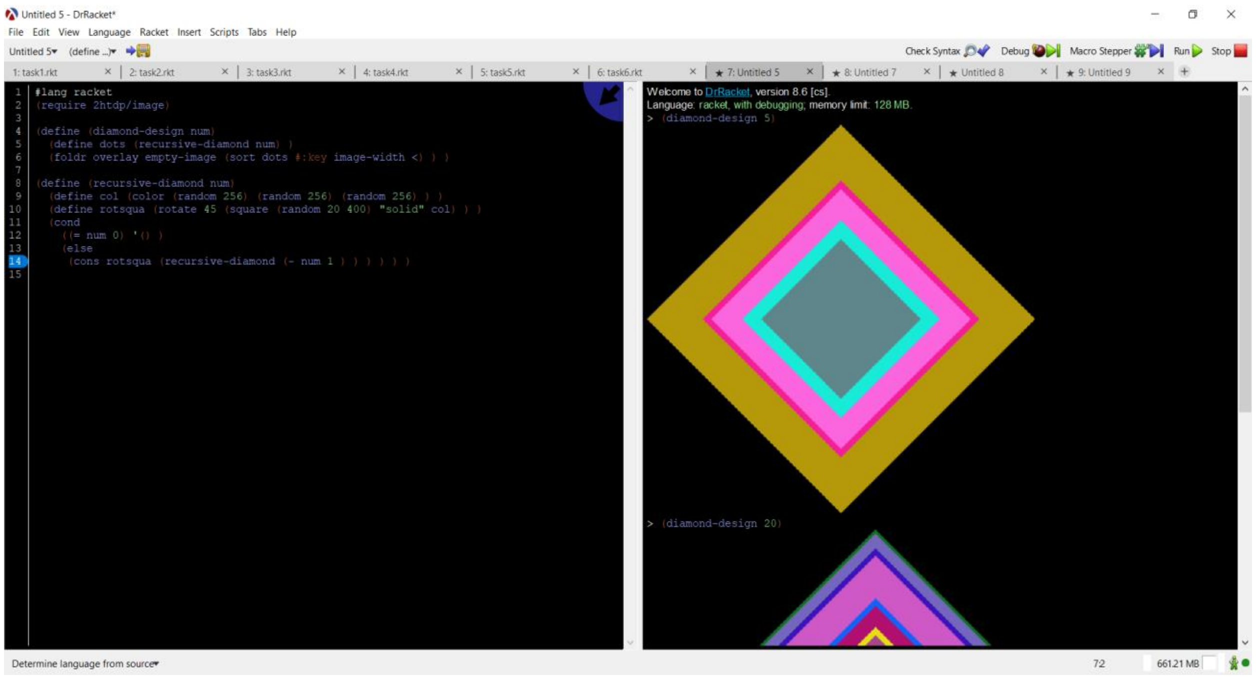
```
#lang racket
(require 2htdp/image)

(define (roll-die) (random 1 7 ) )
(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 num obj)
  (cond
    ((= num 0)'() )
    (else
     (cons (obj) (generate-list (- num 1) obj) ) ) ) ) )
```

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

> (define a (generate-list 5 big-dot) )
> (foldr overlay empty-image (sort-dots a) )

> (define b (genrate-list 10 big-dot) )
```



```
> (define b (genrate-list 10 big-dot) )
   genrate-list: undefined;
 cannot reference an identifier before its definition
> (define b (generate 10 big-dot) )
   generate: undefined;
 cannot reference an identifier before its definition
> (define b (generate-list 10 big-dot) )
> (foldr overlay empty-image (sort-dots a) )

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

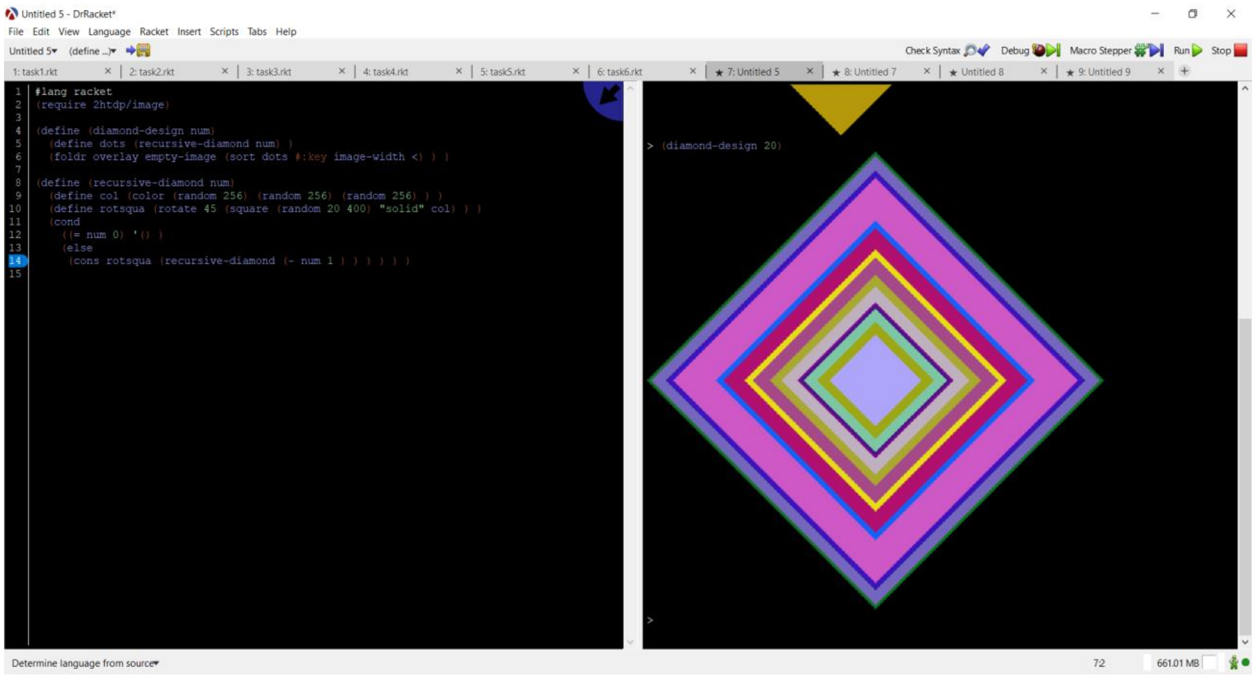
Task 7: The Diamond



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

(define (diamond-design num)
  (define dots (recursive-diamond num) )
  (foldr overlay empty-image (sort dots #:key image-width <) ) )

(define (recursive-diamond num)
  (define col (color (random 256) (random 256) (random 256) ) )
  (define rotsqua (rotate 45 (square (random 20 400) "solid" col) ) )
  (cond
    ((= num 0) '() )
    (else
     (cons rotsqua (recursive-diamond (- num 1 ) ) ) ) ) )
```

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

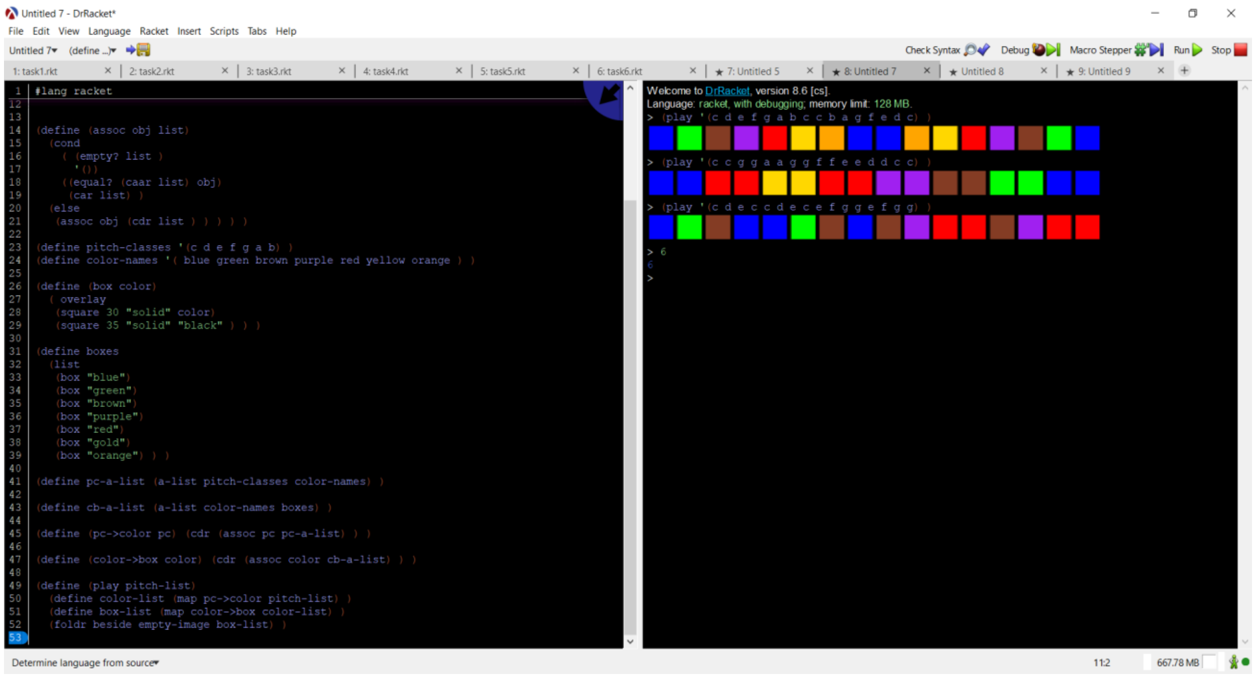
```
> (diamond-design 20)
```



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

(define (diamond-design num)
  (define dots (recursive-diamond num) )
  (foldr overlay empty-image (sort dots #:key image-width <) ) )

(define (recursive-diamond num)
  (define col (color (random 256) (random 256) (random 256) ) )
  (define rotsqua (rotate 45 (square (random 20 400) "solid" col) ) )
  (cond
    ((= num 0) '() )
    (else
     (cons rotsqua (recursive-diamond (- num 1 ) ) ) ) ) )
```

```
> (diamond-design 20)

>
```

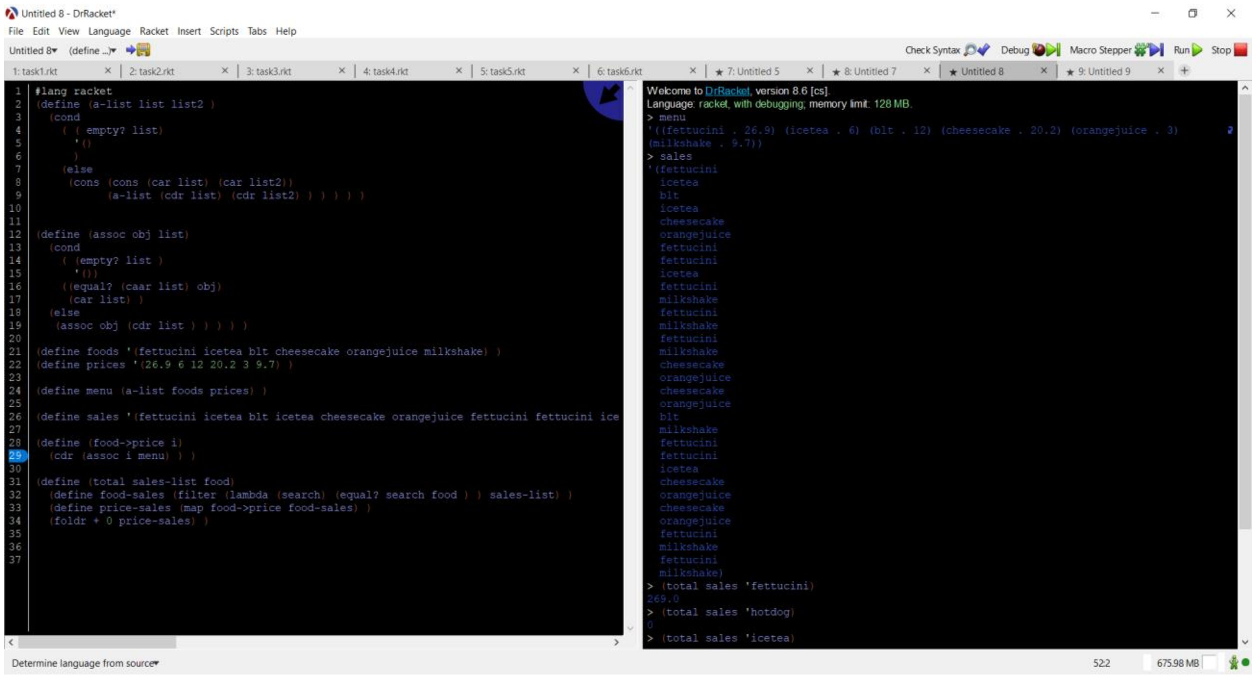
Task 8: Chromesthetic Renderings


Untitled 7 - DrRacket*
File Edit View Language Racket Insert Scripts Tabs Help
Untitled 7 (define ...)
Check Syntax Debug Macro Stepper Run Stop
1: task1.rkt 2: task2.rkt 3: task3.rkt 4: task4.rkt 5: task5.rkt 6: task6.rkt 7: Untitled 5 8: Untitled 7 Untitled 8 9: Untitled 9
Determine language from source
11:2
667.78 MB

```
#lang racket


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

(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) ) )

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

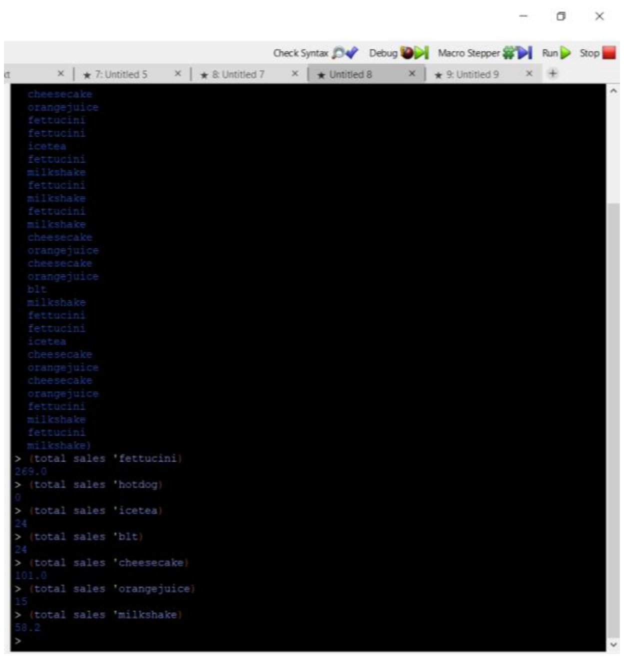
```
Welcome to DrRacket, version 8.6 [cs].
Language: racket, with debugging; memory limit: 128 MB.
> (play '(c d e f g a b c c b a g f e d c) )
🟦🟩🟫🟪🟥🟨🟧🟦🟦🟧🟨🟥🟪🟫🟩🟦
> (play '(c c g g a a g g f f e e d d c c) )
🟦🟦🟥🟥🟨🟨🟥🟥🟪🟪🟫🟫🟩🟩🟦🟦
> (play '(c d e c c d e c e f g g e f g g) )
🟦🟩🟫🟦🟦🟩🟫🟦🟫🟪🟥🟥🟫🟪🟥🟥
> 6
6
>
```

Task 9: Diner


Untitled 8 - DrRacket*
File Edit View Language Racket Insert Scripts Tabs Help
Untitled 8 (define ...)
Check Syntax Debug Macro Stepper Run Stop
1: task1.rkt 2: task2.rkt 3: task3.rkt 4: task4.rkt 5: task5.rkt 6: task6.rkt 7: Untitled 5 8: Untitled 7 Untitled 8 9: Untitled 9
Determine language from source
52:2
675.98 MB

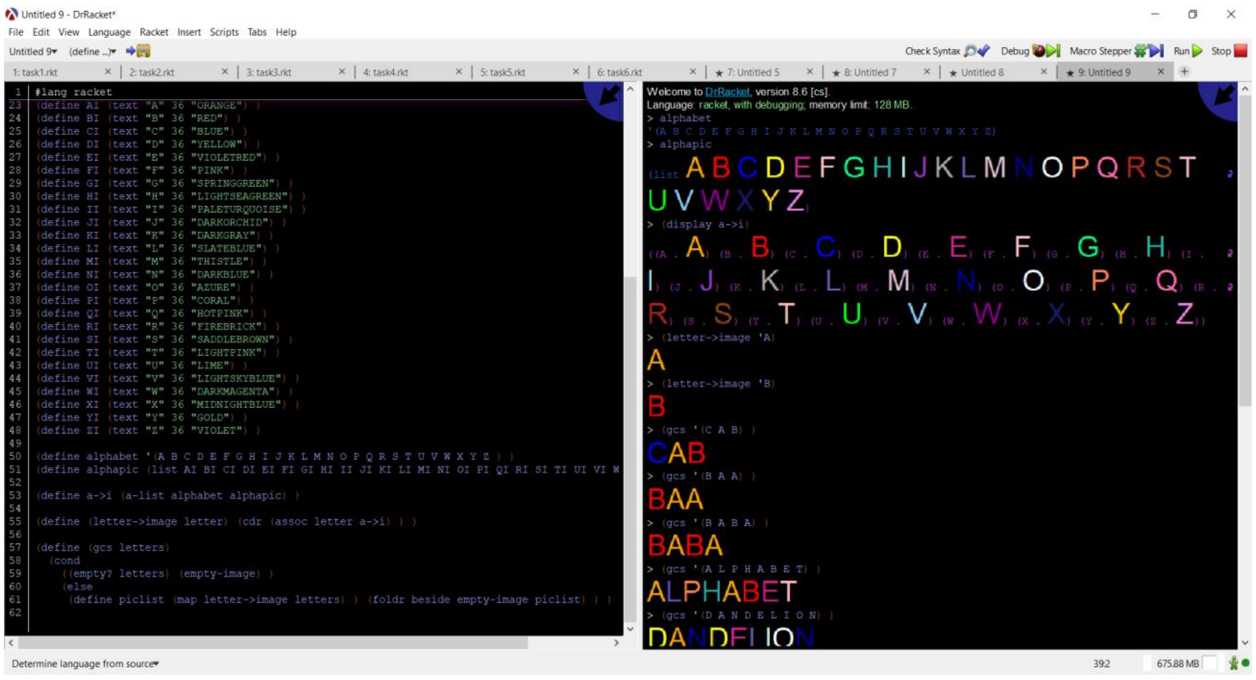
```
#lang racket
(define (a-list list list2 )
  (cond
    ( ( empty? list)
      '()
      )
    (else
     (cons (cons (car list) (car list2))
           (a-list (cdr list) (cdr list2) ) ) ) ) )


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

(define foods '(fettucini icetea blt cheesecake orangejuice milkshake) )
(define prices '(26.9 6 12 20.2 3 9.7) )

(define menu (a-list foods prices) )

(define sales '(fettucini icetea blt icetea cheesecake orangejuice fettucini fettucini ice

(define (food->price i)
  (cdr (assoc i menu) ) )

(define (total sales-list food)
  (define food-sales (filter (lambda (search) (equal? search food ) ) sales-list) )
  (define price-sales (map food->price food-sales) )
  (foldr + 0 price-sales) )
```

```
Welcome to DrRacket, version 8.6 [cs].
Language: racket, with debugging; memory limit: 128 MB.
> menu
'((fettucini . 26.9) (icetea . 6) (blt . 12) (cheesecake . 20.2) (orangejuice . 3)
(milkshake . 9.7))
> sales
'(fettucini
  icetea
  blt
  icetea
  cheesecake
  orangejuice
  fettucini
  fettucini
  icetea
  fettucini
  milkshake
  fettucini
  milkshake
  fettucini
  milkshake
  cheesecake
  orangejuice
  cheesecake
  orangejuice
  blt
  milkshake
  fettucini
  fettucini
  icetea
  cheesecake
  orangejuice
  cheesecake
  orangejuice
  fettucini
  milkshake
  fettucini
  milkshake)
> (total sales 'fettucini)
269.0
> (total sales 'hotdog)
0
> (total sales 'icetea)
```



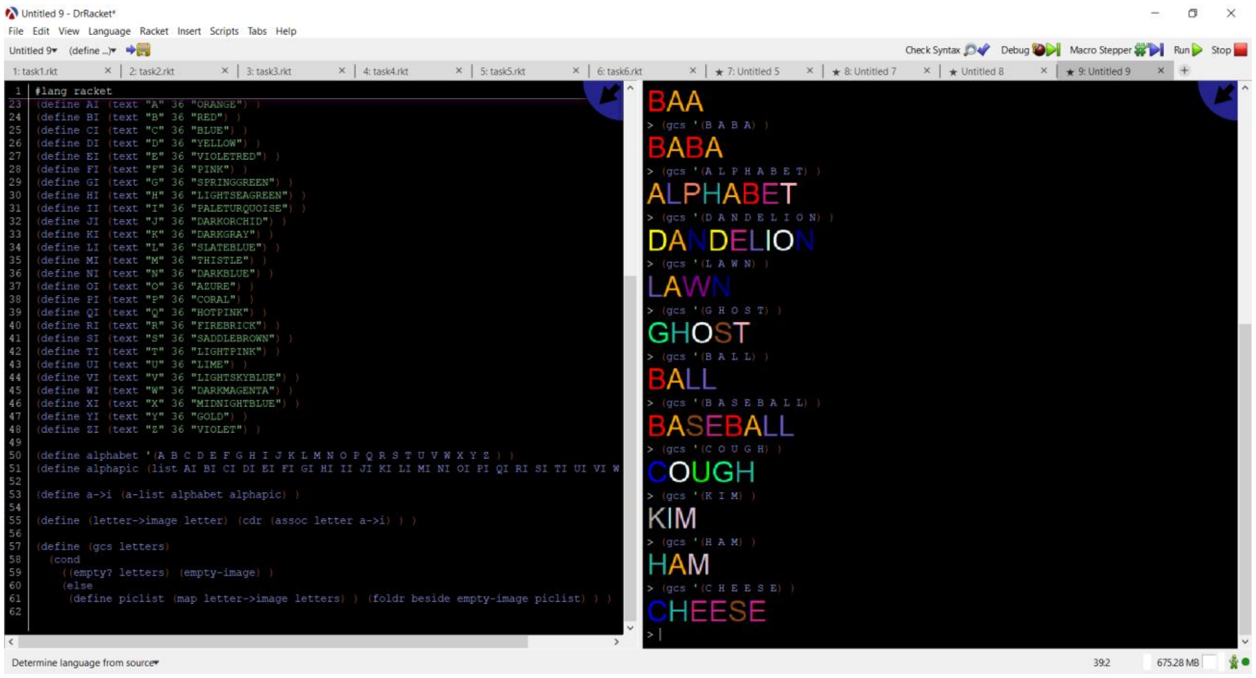
```
  cheesecake
  orangejuice
  fettucini
  fettucini
  icetea
  fettucini
  milkshake
  fettucini
  milkshake
  fettucini
  milkshake
  cheesecake
  orangejuice
  cheesecake
  orangejuice
  blt
  milkshake
  fettucini
  fettucini
  icetea
  cheesecake
  orangejuice
  cheesecake
  orangejuice
  fettucini
  milkshake
  fettucini
  milkshake)
> (total sales 'fettucini)
269.0
> (total sales 'hotdog)
0
> (total sales 'icetea)
24
> (total sales 'blt)
24
> (total sales 'cheesecake)
101.0
> (total sales 'orangejuice)
15
> (total sales 'milkshake)
58.2
>
```

Task 10: Grapheme Color Synesthesia



```racket
#lang racket
(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 "VIOLETRED") )
(define FI (text "F" 36 "PINK") )
(define GI (text "G" 36 "SPRINGGREEN") )
(define HI (text "H" 36 "LIGHTSEAGREEN") )
(define II (text "I" 36 "PALETURQUOISE") )
(define JI (text "J" 36 "DARKORCHID") )
(define KI (text "K" 36 "DARKGRAY") )
(define LI (text "L" 36 "SLATEBLUE") )
(define MI (text "M" 36 "THISTLE") )
(define NI (text "N" 36 "DARKBLUE") )
(define OI (text "O" 36 "AZURE") )
(define PI (text "P" 36 "CORAL") )
(define QI (text "Q" 36 "HOTPINK") )
(define RI (text "R" 36 "FIREBRICK") )
(define SI (text "S" 36 "SADDLEBROWN") )
(define TI (text "T" 36 "LIGHTPINK") )
(define UI (text "U" 36 "LIME") )
(define VI (text "V" 36 "LIGHTSKYBLUE") )
(define WI (text "W" 36 "DARKMAGENTA") )
(define XI (text "X" 36 "MIDNIGHTBLUE") )
(define YI (text "Y" 36 "GOLD") )
(define ZI (text "Z" 36 "VIOLET") )

(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 W

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

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

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

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



```racket
#lang racket
(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 "VIOLETRED") )
(define FI (text "F" 36 "PINK") )
(define GI (text "G" 36 "SPRINGGREEN") )
(define HI (text "H" 36 "LIGHTSEAGREEN") )
(define II (text "I" 36 "PALETURQUOISE") )
(define JI (text "J" 36 "DARKORCHID") )
(define KI (text "K" 36 "DARKGRAY") )
(define LI (text "L" 36 "SLATEBLUE") )
(define MI (text "M" 36 "THISTLE") )
(define NI (text "N" 36 "DARKBLUE") )
(define OI (text "O" 36 "AZURE") )
(define PI (text "P" 36 "CORAL") )
(define QI (text "Q" 36 "HOTPINK") )
(define RI (text "R" 36 "FIREBRICK") )
(define SI (text "S" 36 "SADDLEBROWN") )
(define TI (text "T" 36 "LIGHTPINK") )
(define UI (text "U" 36 "LIME") )
(define VI (text "V" 36 "LIGHTSKYBLUE") )
(define WI (text "W" 36 "DARKMAGENTA") )
(define XI (text "X" 36 "MIDNIGHTBLUE") )
(define YI (text "Y" 36 "GOLD") )
(define ZI (text "Z" 36 "VIOLET") )

(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 W

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

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

(define (gcs letters)
  (cond
    ((empty? letters) (empty-image) )
    (else
     (define piclist (map letter->image letters) ) (foldr beside empty-image piclist) ) )

```

```
BAA
> (gcs '(B A B A) )
BABA
> (gcs '(A L P H A B E T) )
ALPHABET
> (gcs '(D A N D E L I O N) )
DANDELION
> (gcs '(L A W N) )
LAWN
> (gcs '(G H O S T) )
GHOST
> (gcs '(B A L L) )
BALL
> (gcs '(B A S E B A L L) )
BASEBALL
> (gcs '(C O U G H) )
COUGH
> (gcs '(K I M) )
KIM
> (gcs '(H A M) )
HAM
> (gcs '(C H E E S E) )
CHEESE
> |
```