Racket Programming Assignment #4: RLP and HoFs

<><><><><><><><><><><><><><><><><><><><><>

Learning Abstract:

In this assignment I did recursive list processing and other tasks that pertain to list processing with higher order functions.

<><><><><><><><><><><><><><><><><><><><><><><><><><><><><><><><><><><><><><><><><><><><><><>

Task 1 - Generate Uniform List:

Untitled (define ...) Debug Macro Stepper Run Stop

```
#lang racket
(define (generate-uniform-list number Lobject)
  (cond
    ((= number 0)
     (list))
    ((> number 0)
     (define unilist (append (list Lobject) (generate-uniform-list (- number 1)
                                                                   Lobject)))

     unilist)))
```

```
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))
>
```

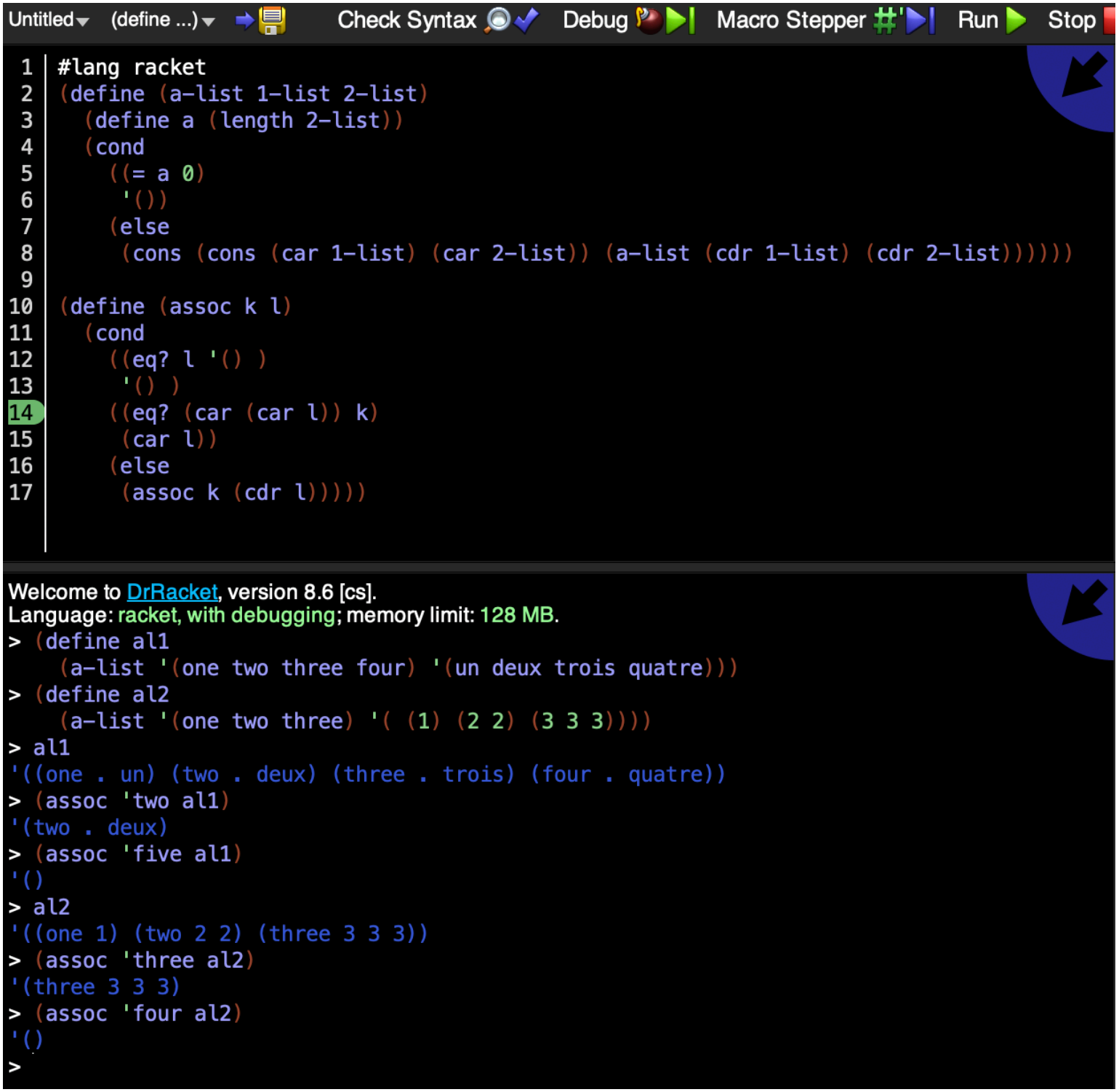
Task 2 - Association List Generator:

Untitled (define ...) Check Syntax Debug Macro

```
1  #lang racket
2  (define (a-list obj pair)
3    (cond
4      (( > (length obj) 0)
5       (cons
6       (cons (list-ref obj 0) (list-ref pair 0))
7       (a-list (cdr obj) (cdr pair))
8       )
9      )
10   ((= (length obj) 0)
11    '()
12    )))
```

```
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:


Untitled
(define ...)
Check Syntax
Debug
Macro Stepper
Run
Stop
1 #lang racket
2 (define (a-list 1-list 2-list)
3 (define a (length 2-list))
4 (cond
5 ((= a 0)
6 '())
7 (else
8 (cons (cons (car 1-list) (car 2-list)) (a-list (cdr 1-list) (cdr 2-list))))))
9
10 (define (assoc k l)
11 (cond
12 ((eq? l '())
13 '())
14 ((eq? (car (car l)) k)
15 (car l))
16 (else
17 (assoc k (cdr l)))))
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))
> (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:

assoc.rkt (define ...) Check Syntax Debug Macro Stepper Run Stop

```
1  #lang racket
2  (define (a-list 1-list 2-list)
3    (define a (length 2-list))
4    (cond
5      ((= a 0)
6       '())
7      (else
8       (cons (cons (car 1-list) (car 2-list)) (a-list (cdr 1-list) (cdr 2-list))))))
9
10 (define (rassoc k l)
11   (cond
12     ((eq? l '() )
13      '() )
14     ((equal? (cdr (car l)) k)
15      (car l))
16     (else
17      (rassoc k (cdr l)))))
```

```
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:

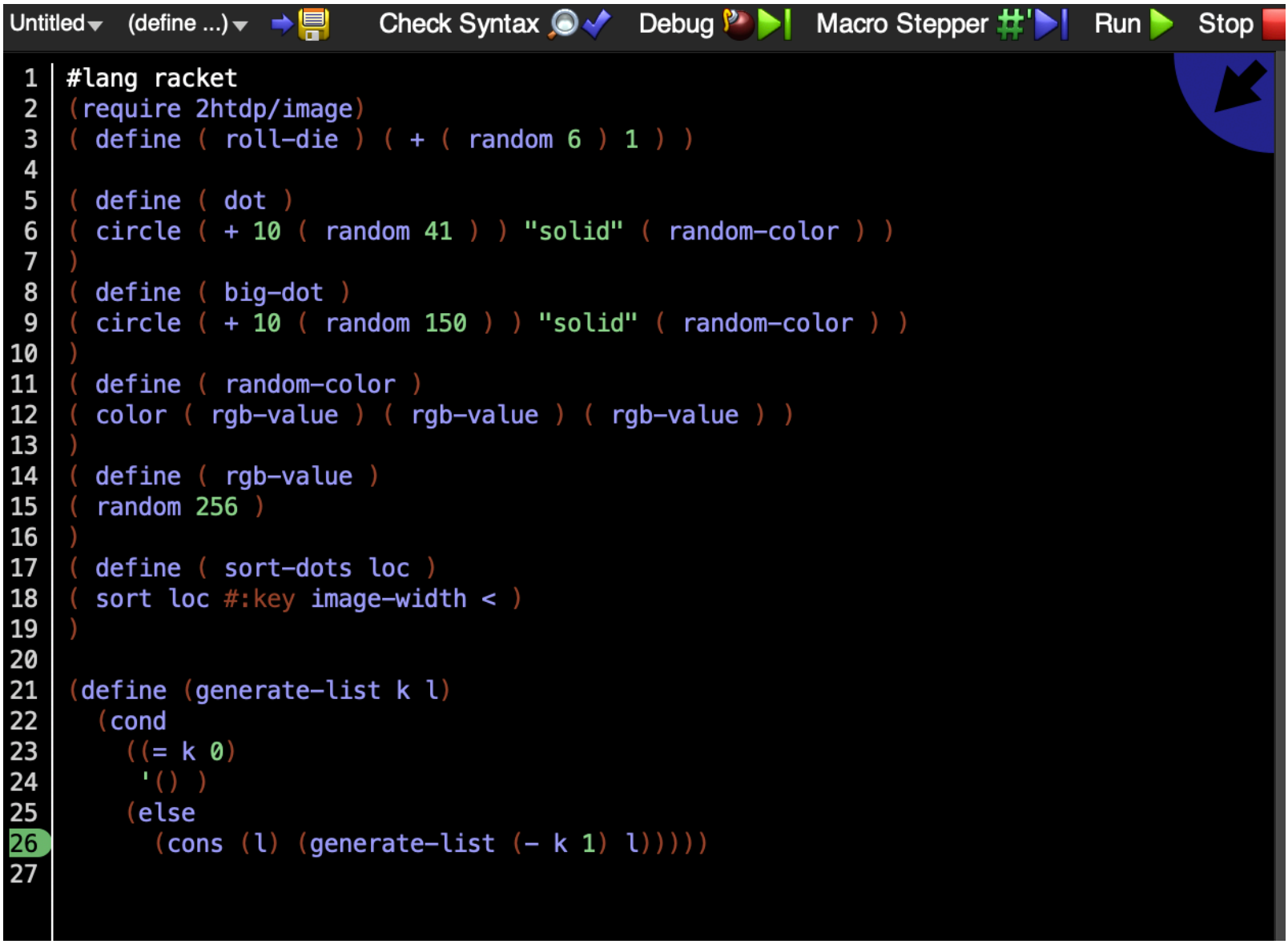
Untitled (define ...) Check Syntax Debug Macro Stepper

```
1  #lang racket
2  (define (los->s slist)
3    (cond
4      ((> (length slist) 1)
5       (string-append (list-ref slist 0) " " (los->s (cdr slist))))
6      ((= (length slist) 1)
7       (string-append (list-ref slist 0) (los->s (cdr slist))))
8      (else
9       " "
10      )))
11
12 (define (generate-uniform-list k l)
13   (cond
14     ((= k 0)
15      '() )
16     (else
17      (cons l (generate-uniform-list (- k 1) l)))))
```

```
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:

Code -




```
#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 k l)
  (cond
    ((= k 0)
     '() )
    (else
      (cons (l) (generate-list (- k 1) l)))))

```

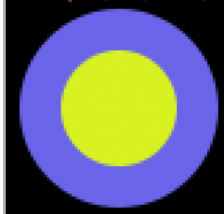
Demo 1:

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

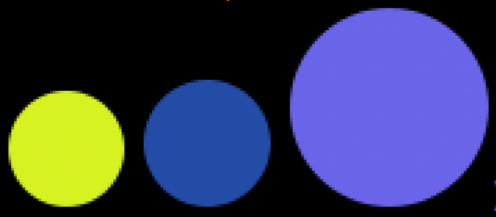
Demo 2:

```
> (define dots (generate-list 3 dot))
> dots
```

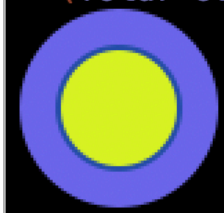
```
(list                 )
> (foldr overlay empty-image dots)
```



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

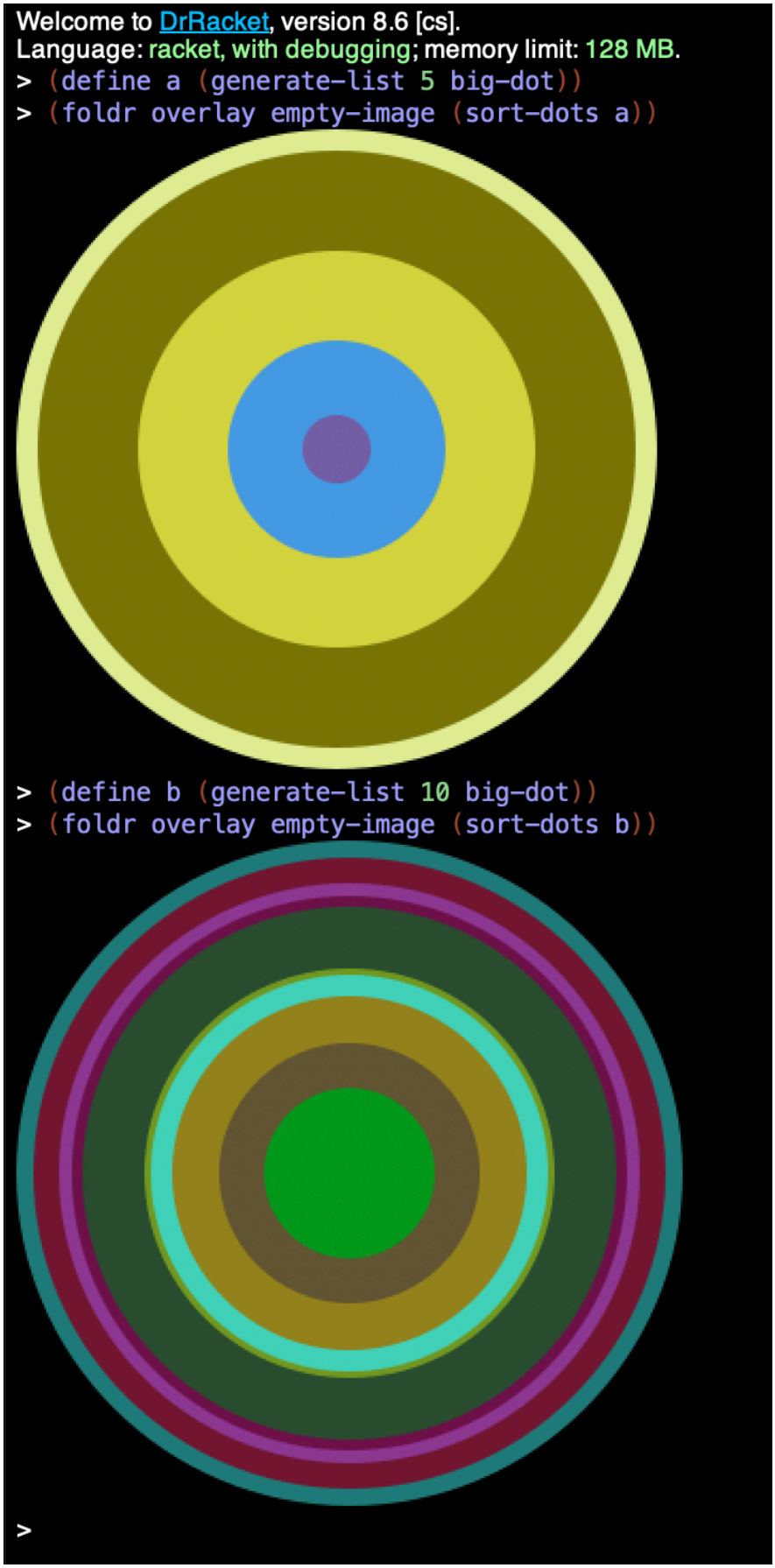


```
(list                 )
> (foldr overlay empty-image (sort-dots dots))
```



```
>
```

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))
>

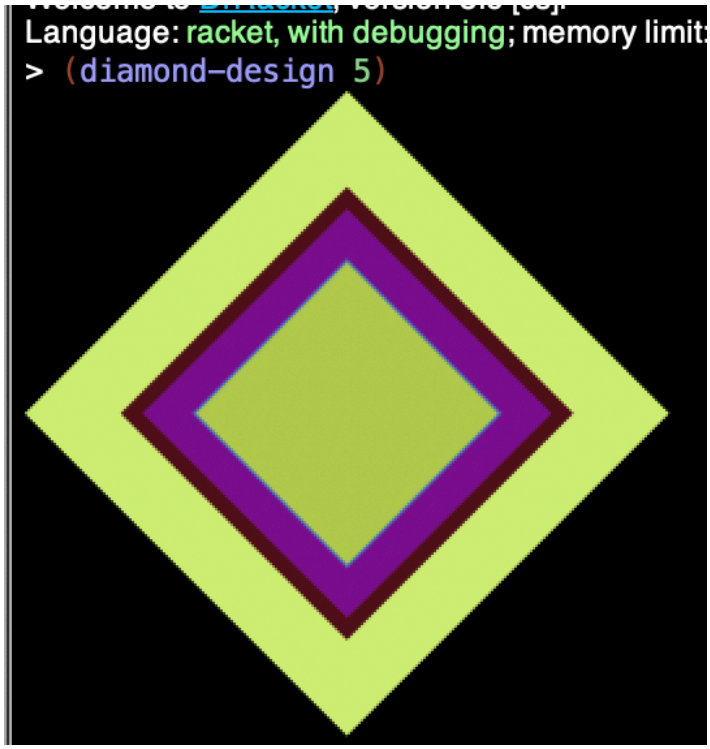
Task 7 - The Diamond:

Code -

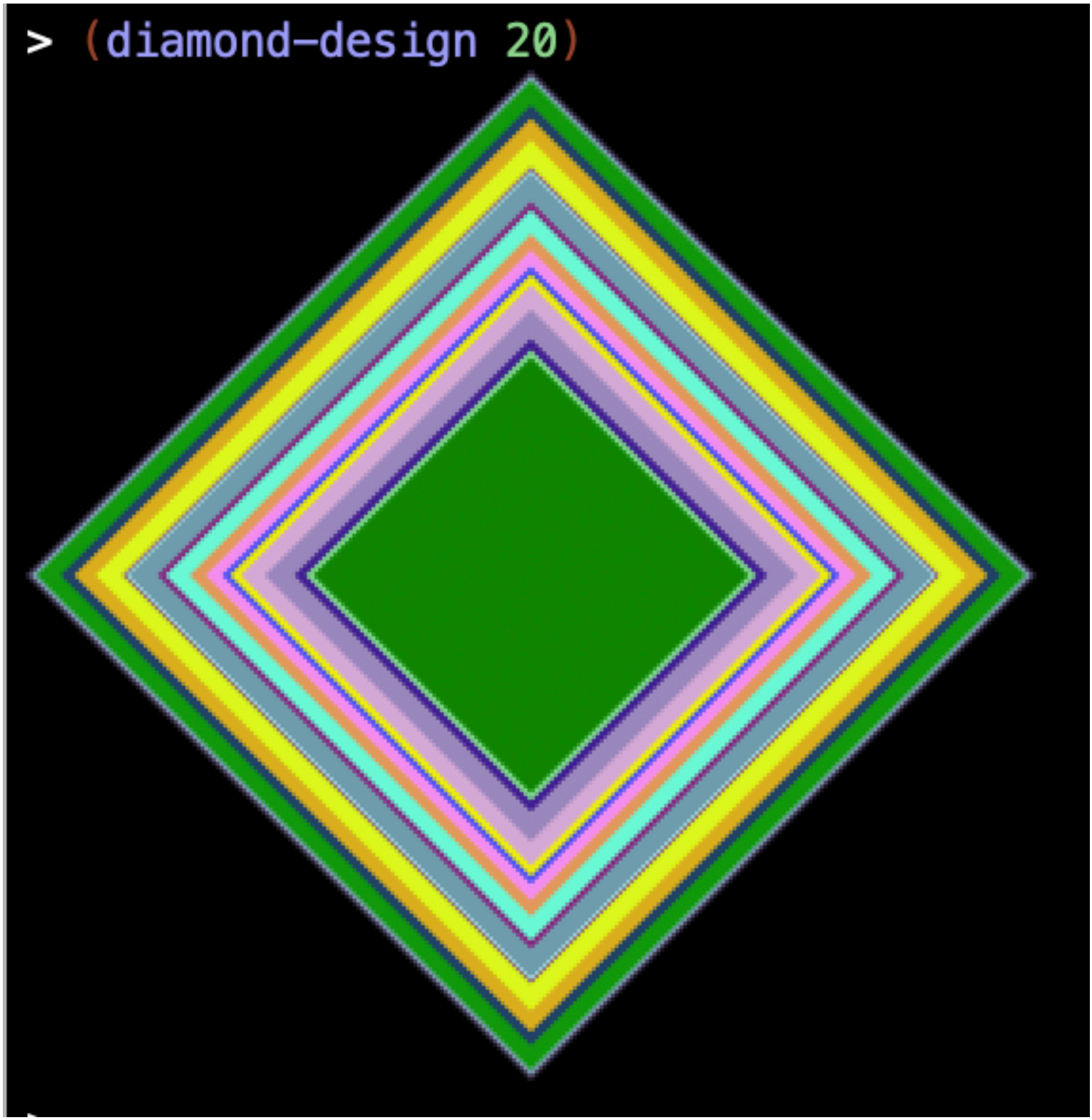
Untitled (define ...) Check Syntax Debug Macro Stepper Run Stop

```
1  #lang racket
2  (require 2htdp/image)
3  (define (sort-diamonds d) (sort d #:key image-width <))
4  (define (rgb-value) (random 256))
5  (define (random-color) (color (rgb-value) (rgb-value) (rgb-value)))
6  (define (diamond) (rotate 45 (square (+ 1 (random 80 200)) "solid" (random-color))))
7
8  (define (multi-diamonds k)
9    (cond
10     ((= k 0)
11      '() )
12     (else
13      (cons (diamond) (multi-diamonds (- k 1))))))
14
15 (define (diamond-design k)
16   (foldr overlay empty-image (sort-diamonds (multi-diamonds k ))))
```

Demo 1:



Demo 2:



Task 8 - 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 ( a-list list1 list2 )
 ( define k ( length list2 ) )
 ( cond
 ( ( = k 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 pitchList)
  (define colorList
    (map (lambda (x) (pc->color x)) pitchList))
  (define boxList
    (map (lambda (x) (color->box x)) colorList))
  (foldr beside empty-image boxList))
```

Demo:

```
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 9 - Diner:

```
#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 (total sales item)
  (foldr + 0
         (map totalsales
              (filter (lambda (x)
                        (cond
                          ((equal? item x)
                           #t)
                          (else
                           #f)))
                      sales))))
(define menu (a-list '(pizza wings calzone empanada soda tacos)
                     '(3.50 10.50 7 4 3 5)))
(define (totalsales item)
  (cdr (assoc item menu)))
(define sales
  '(pizza pizza wings calzone empanada soda tacos soda soda tacos empanada
          pizza wings calzone soda calzone tacos wings wings wings wings tacos
          empanada calzone calzone pizza))
```

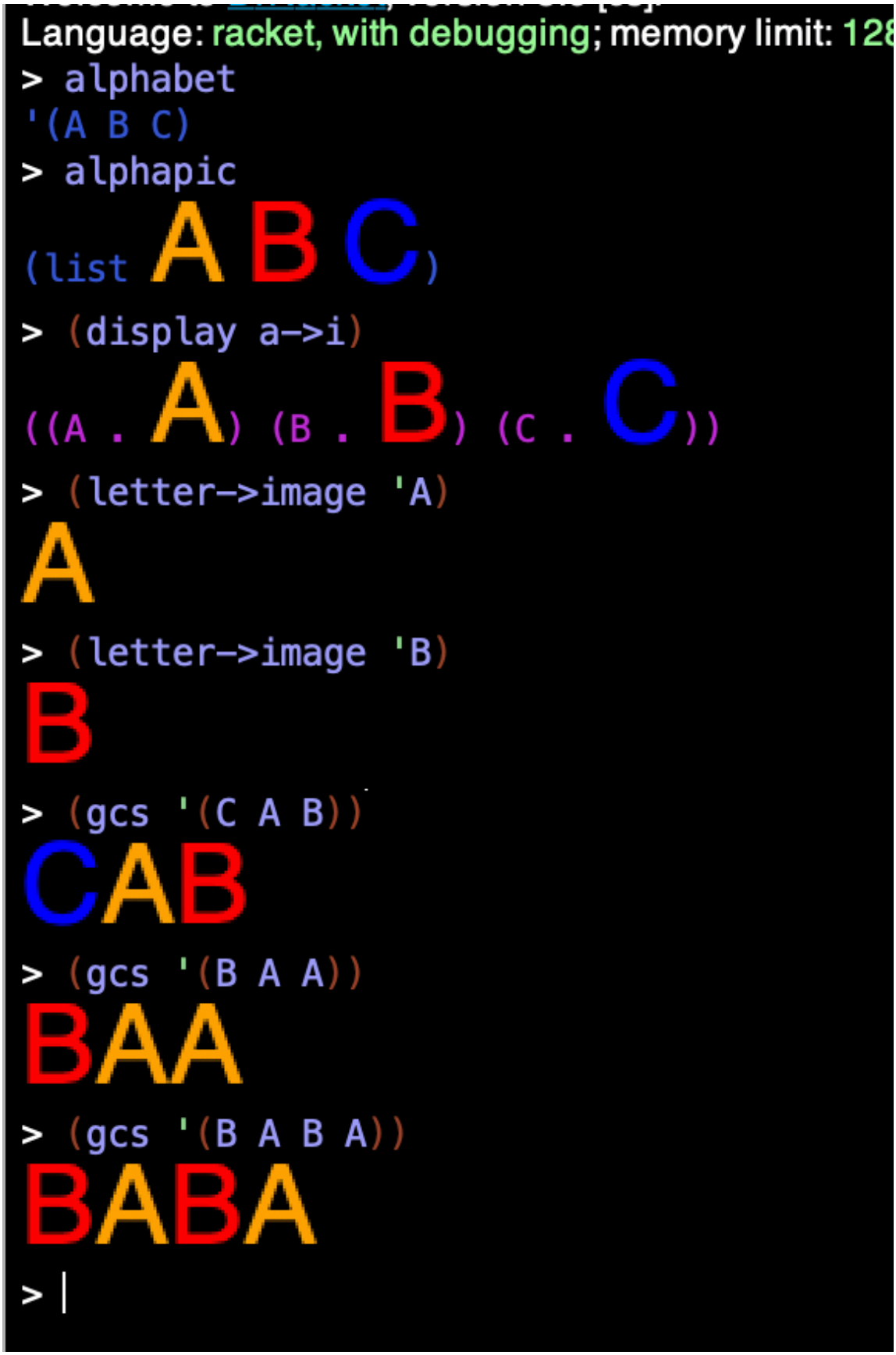
Demo:

```
Language: racket, with debugging; memory limit: 128 MB.
> menu
'((pizza . 3.5) (wings . 10.5) (calzone . 7) (empanada . 4) (soda . 3) (tacos . 5))
>  sales
'(pizza
  pizza
  wings
  calzone
  empanada
  soda
  tacos
  soda
  soda
  tacos
  empanada
  pizza
  wings
  calzone
  soda
  calzone
  tacos
  wings
  wings
  wings
  wings
  tacos
  empanada
  calzone
  calzone
  pizza)
> (total sales 'pizza)
14.0
> (total sales 'wings)
63.0
> (total sales 'calzone)
35
> (total sales 'empanada)
12
> (total sales 'soda)
12
> (total sales 'tacos)
20
>
```

Task 10 - Grapheme Color Synesthesia:
10a -

```
#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 a->i ( a-list alphabet alphapic ) )

( define ( assoc t l )
 ( cond
 ( ( eq? l '() )
 '()
 )
 ( ( equal? ( car (car l ) ) t)
 ( car l )
 )
 ( else
 ( assoc t ( cdr l ) )
 )
 )
 )

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

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

10b -


Language: racket, with debugging; memory limit: 128
> 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
>

10c -

```
#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 "maroon" ) )
( define EI ( text "E" 36 "pink" ) )
( define FI ( text "F" 36 "snow" ) )
( define GI ( text "G" 36 "chocolate" ) )
( define HI ( text "H" 36 "coral" ) )
( define II ( text "I" 36 "sienna" ) )
( define JI ( text "J" 36 "salmon" ) )
( define KI ( text "K" 36 "peru" ) )
( define LI ( text "L" 36 "gold" ) )
( define MI ( text "M" 36 "yellow" ) )
( define NI ( text "N" 36 "olive" ) )
( define OI ( text "O" 36 "green" ) )
( define PI ( text "P" 36 "turquoise" ) )
( define QI ( text "Q" 36 "teal" ) )
( define RI ( text "R" 36 "azure" ) )
( define SI ( text "S" 36 "navy" ) )
( define TI ( text "T" 36 "purple" ) )
( define UI ( text "U" 36 "lavender" ) )
( define VI ( text "V" 36 "white" ) )
( define WI ( text "W" 36 "gray" ) )
( define XI ( text "X" 36 "honeydew" ) )
( define YI ( text "Y" 36 "magenta" ) )
( define ZI ( text "Z" 36 "beige" ) )

( 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) )
```

```
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)
>
```

10d -


Welcome to DrRacket, version 8.6 [cs].
Language: racket, with debugging; memory limit:
> (gcs '(A L P H A B E T))
ALPHABET
> (gcs '(D A N D E L I O N))
DANDELION
> (gcs '(C H I C K E N))
CHICKEN
> (gcs '(S U N F L O W E R))
SUNFLOWER
> (gcs '(C O M P U T E R))
COMPUTER
> (gcs '(R A C K E T))
RACKET
> (gcs '(H E L L O))
HELLO
> (gcs '(M U S I C))
MUSIC
> (gcs '(S O C C E R))
SOCCER
> (gcs '(J D S B R I M S))
JDSBRIMS
>