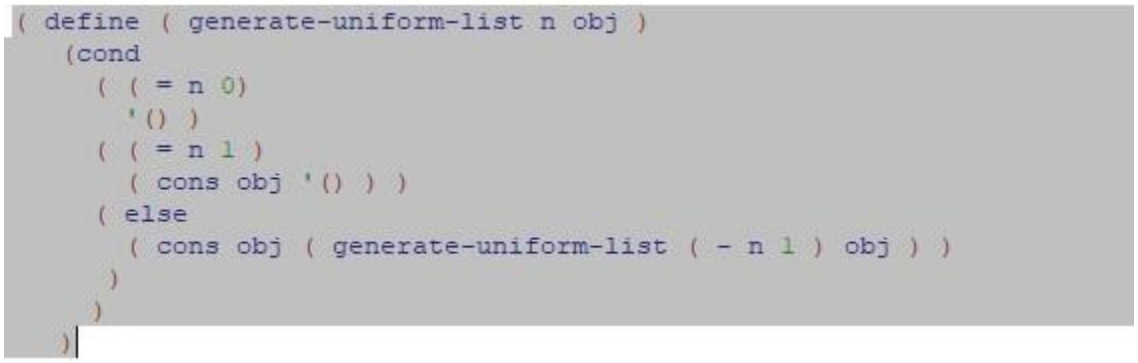
Kieran Finnegan

Programming Assignment 4

Learning Abstract: In this programming assignment I used recursive list processing and higher order functions to complete tasks. I also so familiar with using the functions maps, foldr and filter.

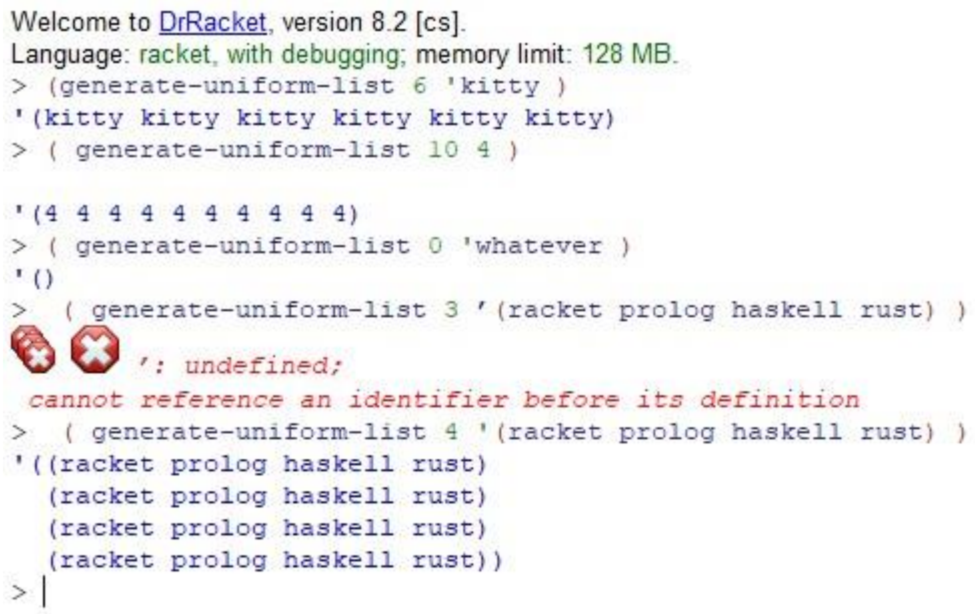
Task 1:

Code:



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

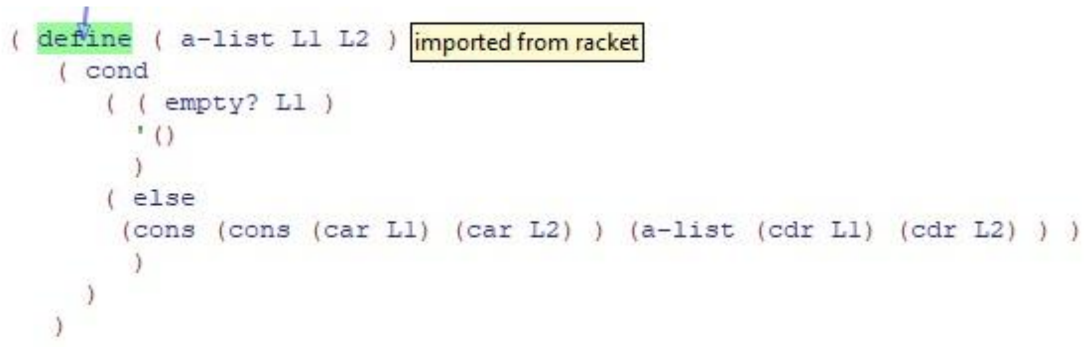
Demo:



```
Welcome to DrRacket, version 8.2 [cs].
Language: racket, with debugging; memory limit: 128 MB.
> (generate-uniform-list 6 'kitty )
'(kitty kitty kitty kitty kitty kitty)
> ( generate-uniform-list 10 4 )

'(4 4 4 4 4 4 4 4 4 4)
> ( generate-uniform-list 0 'whatever )
'()
>  ( generate-uniform-list 3 ’(racket prolog haskell rust) )
     ’: undefined;
 cannot reference an identifier before its definition
>  ( generate-uniform-list 4 '(racket prolog haskell rust) )
'((racket prolog haskell rust)
  (racket prolog haskell rust)
  (racket prolog haskell rust)
  (racket prolog haskell rust))
> 
```

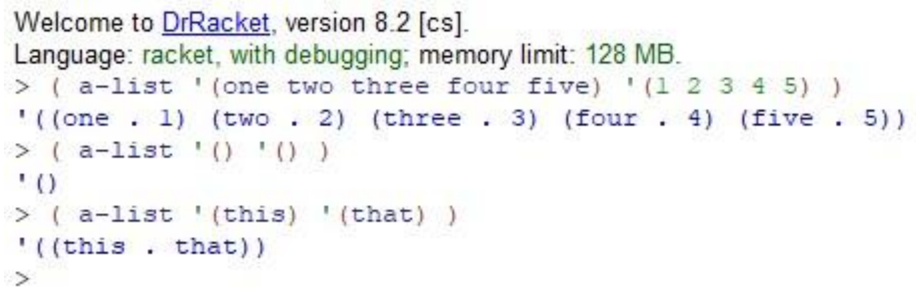
Task 2:

Code:



```
( define ( a-list L1 L2 )
  ( cond
     ( ( empty? L1 )
       '()
       )
     ( else
      (cons (cons (car L1) (car L2) ) (a-list (cdr L1) (cdr L2) ) )
       )
    )
  )
```

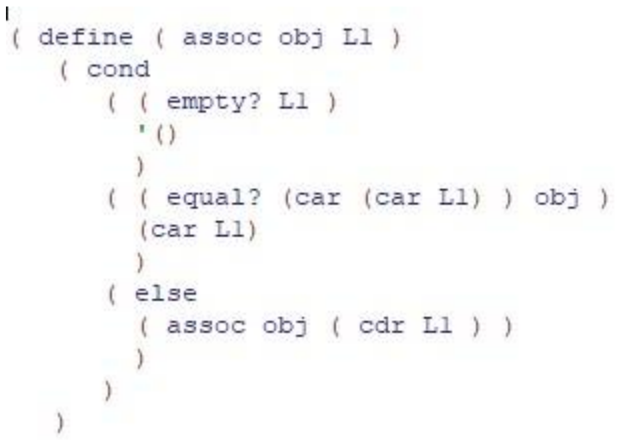
Demo:



```
Welcome to DrRacket, version 8.2 [cs].
Language: racket, with debugging; memory limit: 128 MB.
> ( a-list '(one two three four five) '(1 2 3 4 5) )
'((one . 1) (two . 2) (three . 3) (four . 4) (five . 5))
> ( a-list '() '() )
'()
> ( a-list '(this) '(that) )
'((this . that))
>
```

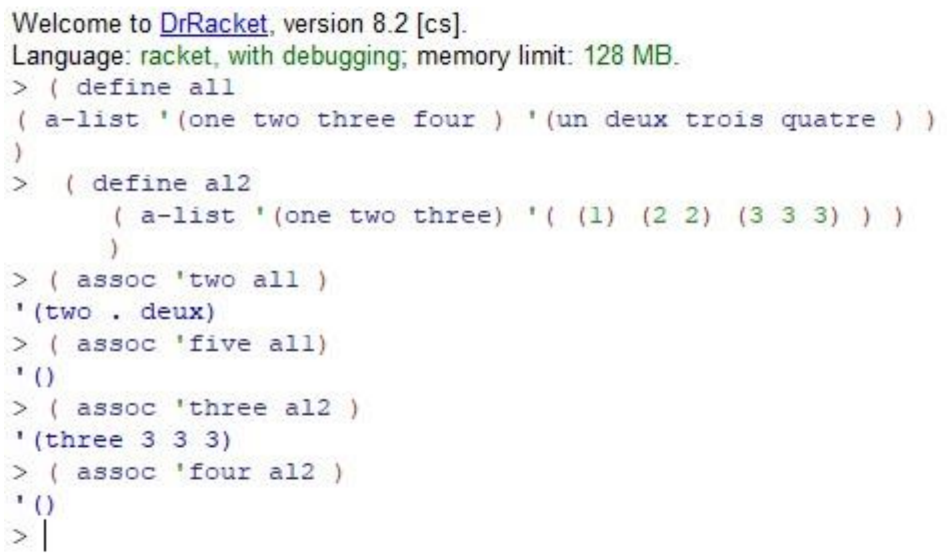
Task 3:

Code:



```
( define ( assoc obj L1 )
   ( cond
      ( ( empty? L1 )
        '()
        )
      ( ( equal? (car (car L1) ) obj )
        (car L1)
        )
      ( else
        ( assoc obj ( cdr L1 ) )
        )
      )
   )
```

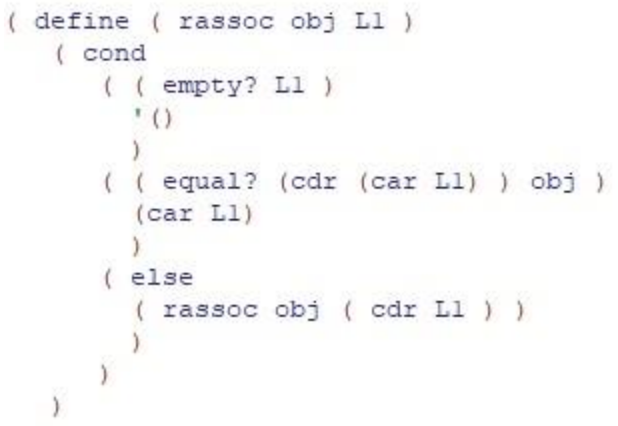
Demo:



```
Welcome to DrRacket, version 8.2 [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) ) )
      )
> ( assoc 'two al1 )
'(two . deux)
> ( assoc 'five al1)
'()
> ( assoc 'three al2 )
'(three 3 3 3)
> ( assoc 'four al2 )
'()
> 
```

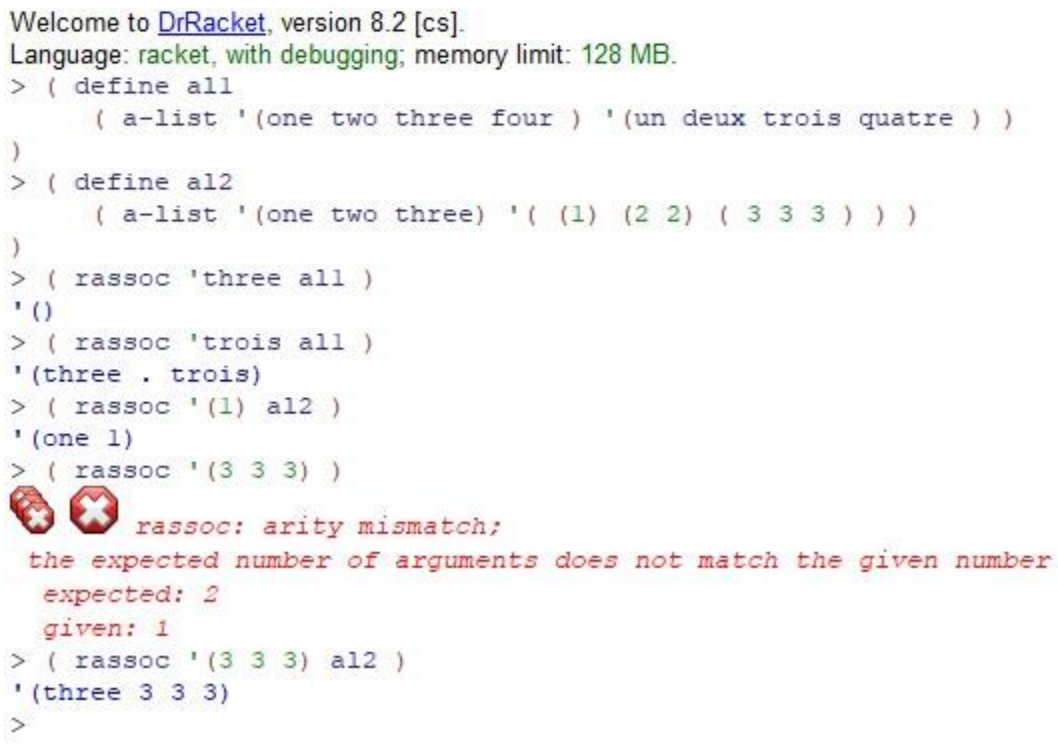
Task 4:

Code:



```
( define ( rassoc obj L1 )
   ( cond
      ( ( empty? L1 )
        '()
        )
      ( ( equal? (cdr (car L1) ) obj )
        (car L1)
        )
      ( else
        ( rassoc obj ( cdr L1 ) )
        )
      )
   )
```

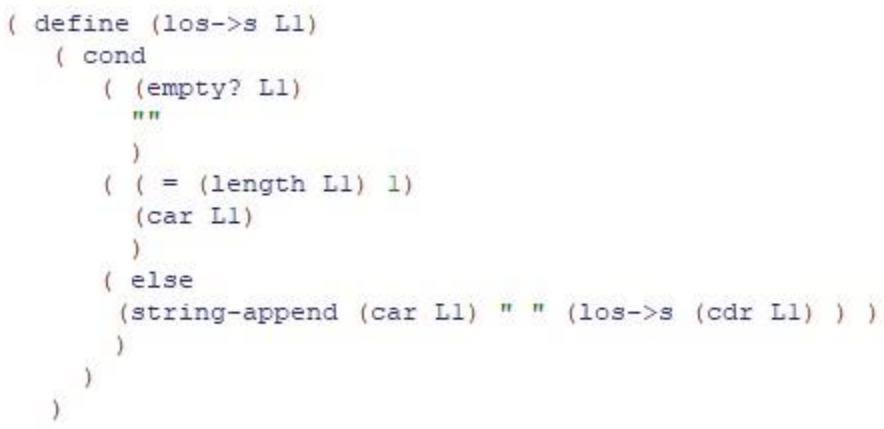
Demo:



```
Welcome to DrRacket, version 8.2 [cs].
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 ) ) )
)
> ( rassoc 'three all )
'()
> ( rassoc 'trois all )
'(three . trois)
> ( rassoc '(1) al2 )
'(one 1)
> ( rassoc '(3 3 3) )
rassoc: arity mismatch;
 the expected number of arguments does not match the given number
  expected: 2
  given: 1
> ( rassoc '(3 3 3) al2 )
'(three 3 3 3)
>
```

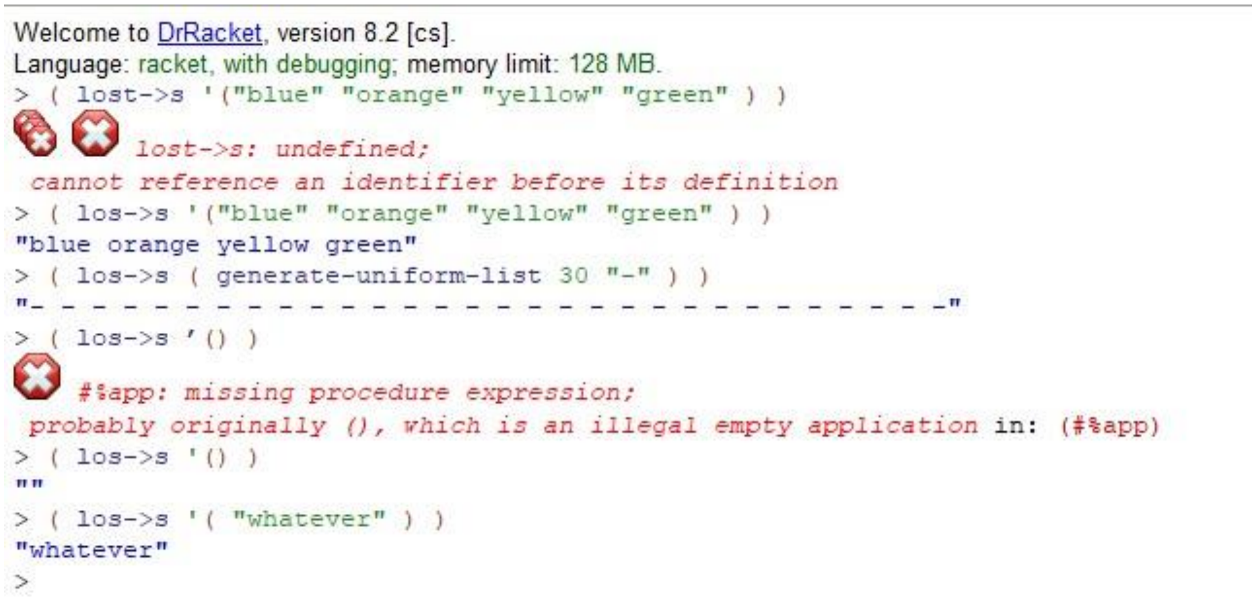
Task 5:

Code:



```
( define (los->s L1)
   ( cond
      ( (empty? L1)
        ""
        )
      ( ( = (length L1) 1)
        (car L1)
        )
      ( else
       (string-append (car L1) " " (los->s (cdr L1) ) )
       )
     )
   )
```

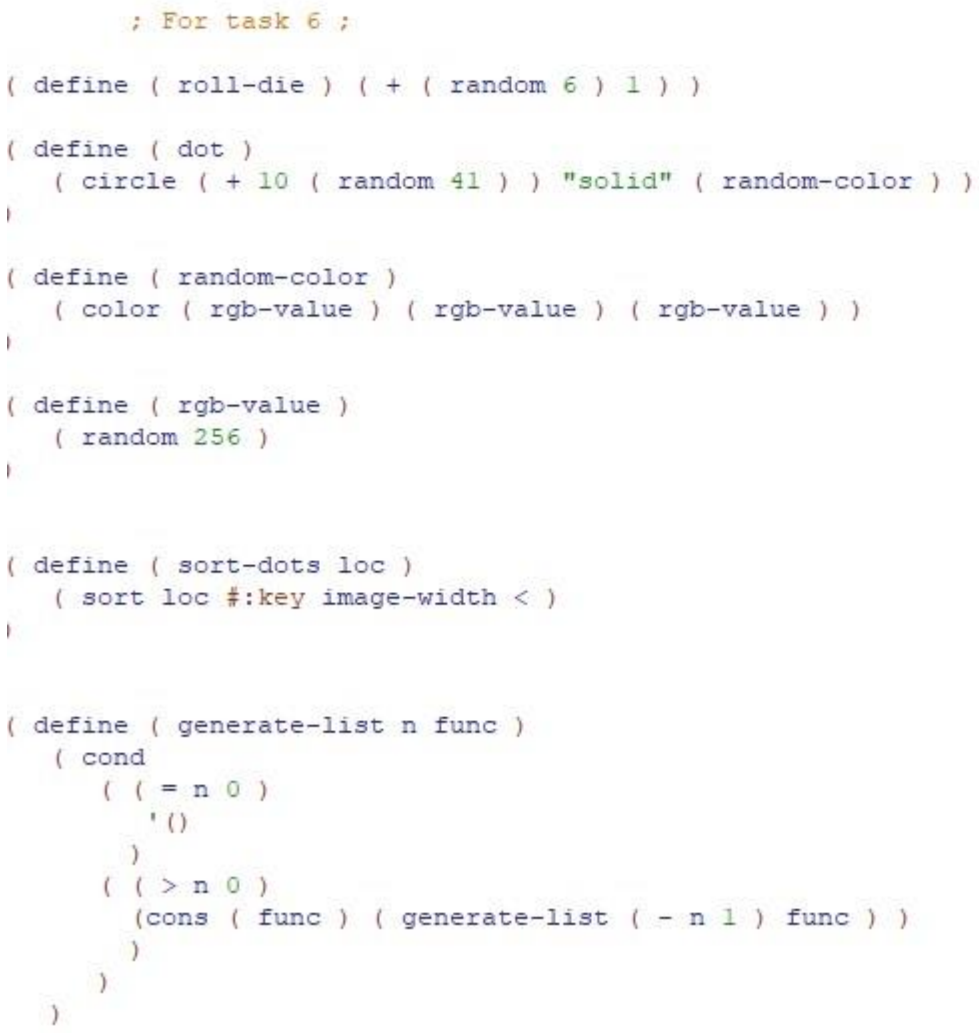
Demo:



```
Welcome to DrRacket, version 8.2 [cs].
Language: racket, with debugging; memory limit: 128 MB.
> ( lost->s '("blue" "orange" "yellow" "green" ) )
lost->s: undefined;
 cannot reference an identifier before its definition
> ( los->s '("blue" "orange" "yellow" "green" ) )
"blue orange yellow green"
> ( los->s ( generate-uniform-list 30 "-" ) )
"- - - - - - - - - - - - - - - - - - - - - - - - - - - - - -"
> ( los->s ’() )
#%app: missing procedure expression;
 probably originally (), which is an illegal empty application in: (#%app)
> ( los->s '() )
""
> ( los->s '( "whatever" ) )
"whatever"
>
```

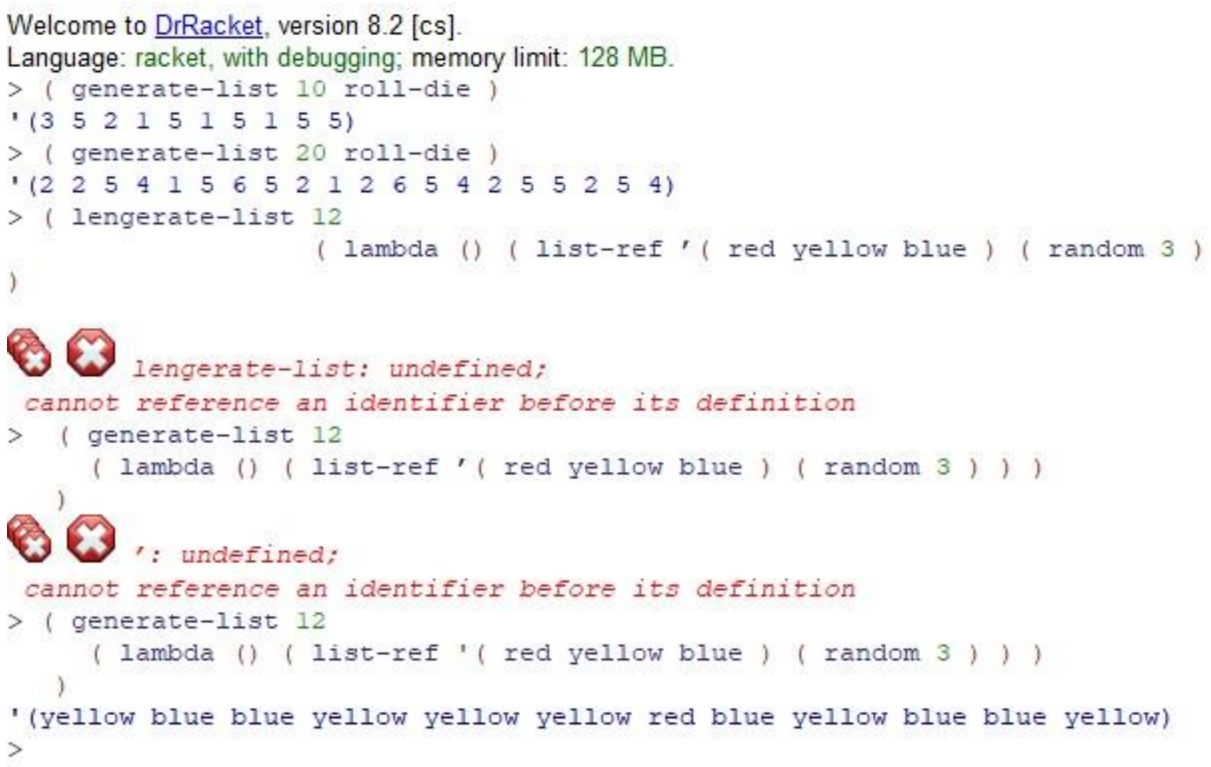
Task 6:

Code:

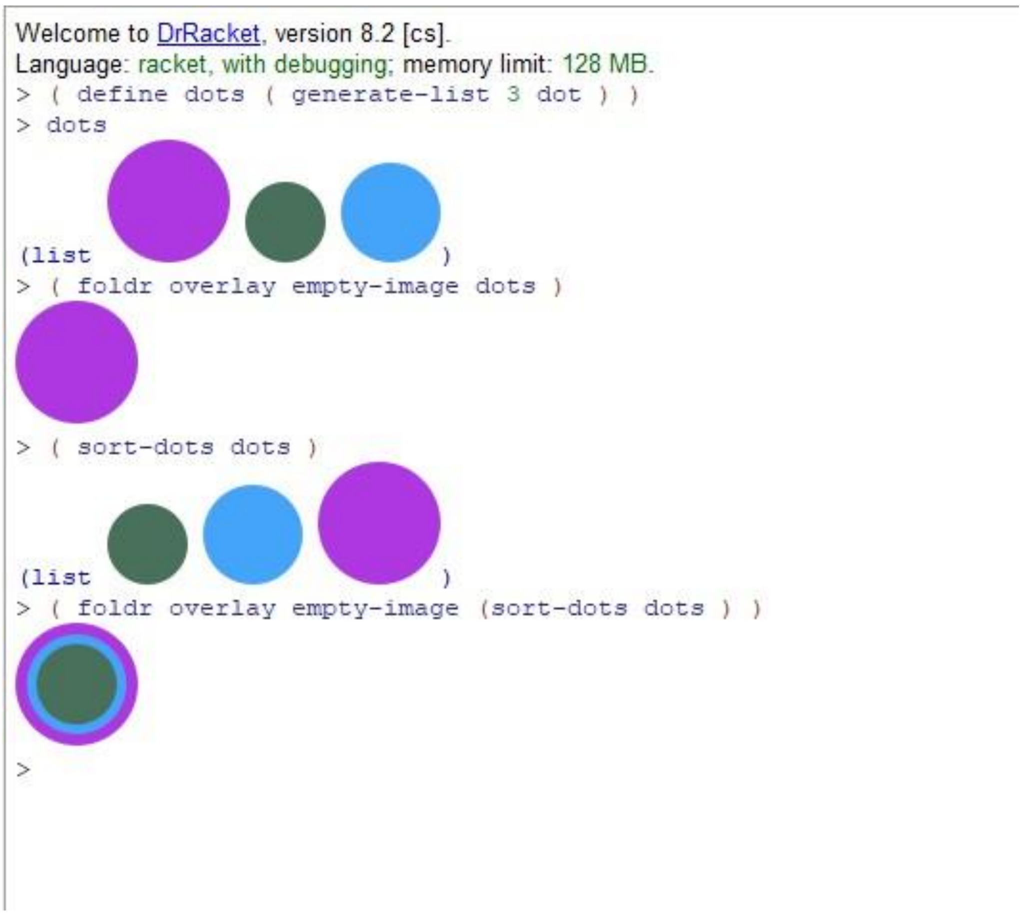


```
        ; For task 6 ;

( 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 < )
)


( define ( generate-list n func )
   ( cond
      ( ( = n 0 )
         '()
       )
      ( ( > n 0 )
        (cons ( func ) ( generate-list ( - n 1 ) func ) )
        )
      )
   )
```

Demo:



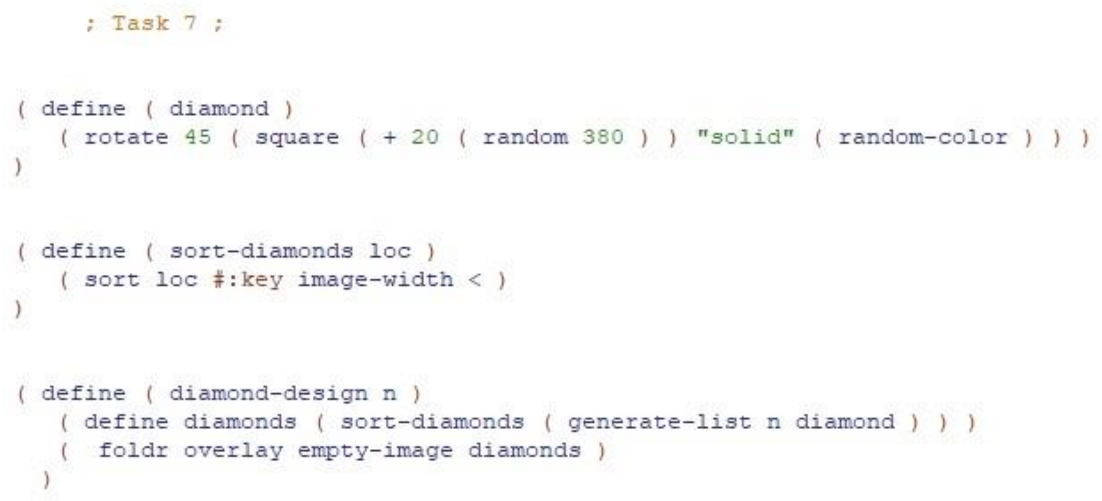
```
Welcome to DrRacket, version 8.2 [cs].
Language: racket, with debugging; memory limit: 128 MB.
> ( generate-list 10 roll-die )
'(3 5 2 1 5 1 5 1 5 5)
> ( generate-list 20 roll-die )
'(2 2 5 4 1 5 6 5 2 1 2 6 5 4 2 5 5 2 5 4)
> ( lengerate-list 12
                    ( lambda () ( list-ref ’( red yellow blue ) ( random 3 )
)
lengerate-list: undefined;
 cannot reference an identifier before its definition
>  ( generate-list 12
     ( lambda () ( list-ref ’( red yellow blue ) ( random 3 ) ) )
   )
’: undefined;
 cannot reference an identifier before its definition
> ( generate-list 12
     ( lambda () ( list-ref '( red yellow blue ) ( random 3 ) ) )
   )
'(yellow blue blue yellow yellow yellow red blue yellow blue blue yellow)
>
```



```
Welcome to DrRacket, version 8.2 [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 ) )

>
```

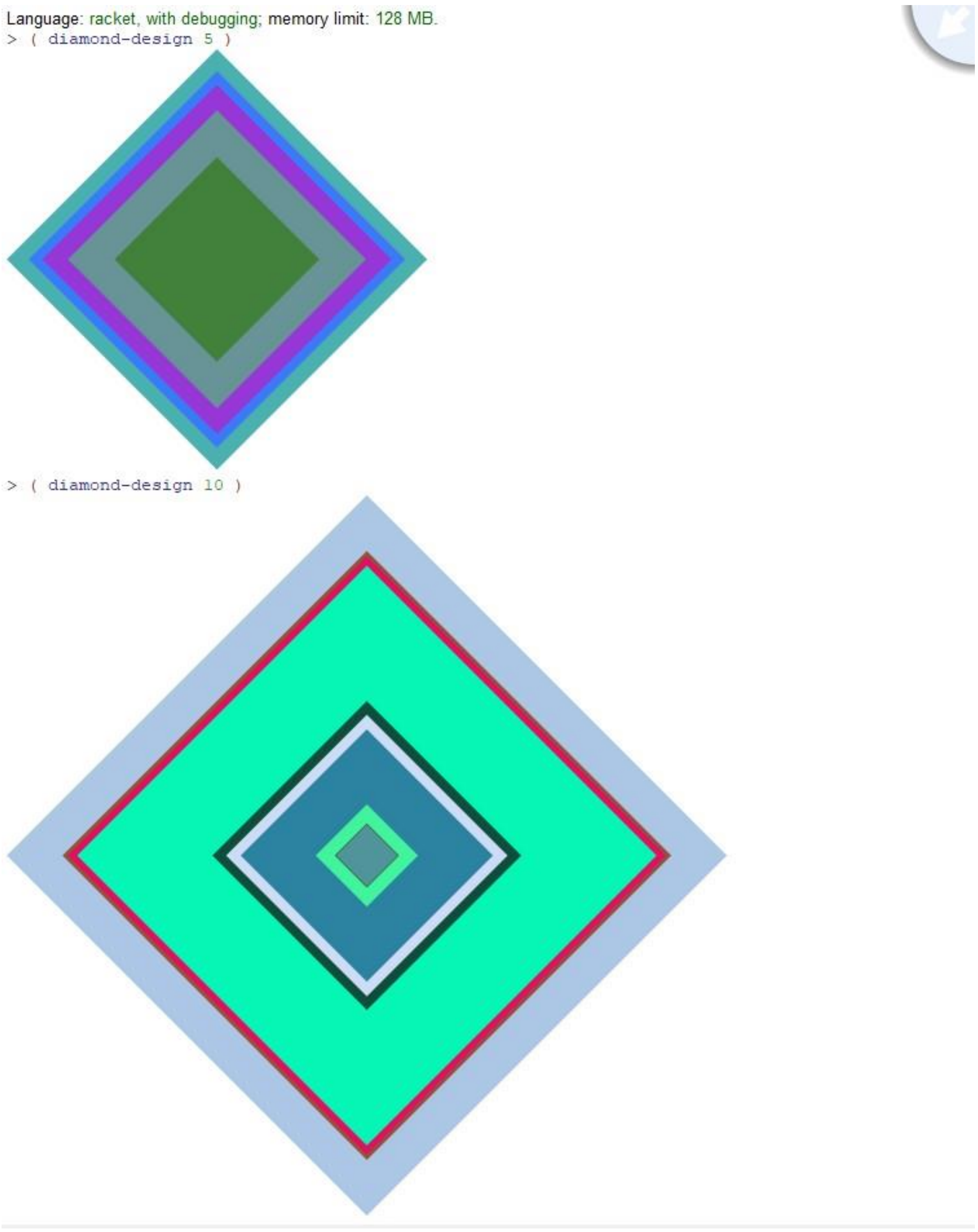
Task 7:

Code:



```
; Task 7 ;

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

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

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

Demo:



```
Language: racket, with debugging; memory limit: 128 MB.
> ( diamond-design 5 )
```

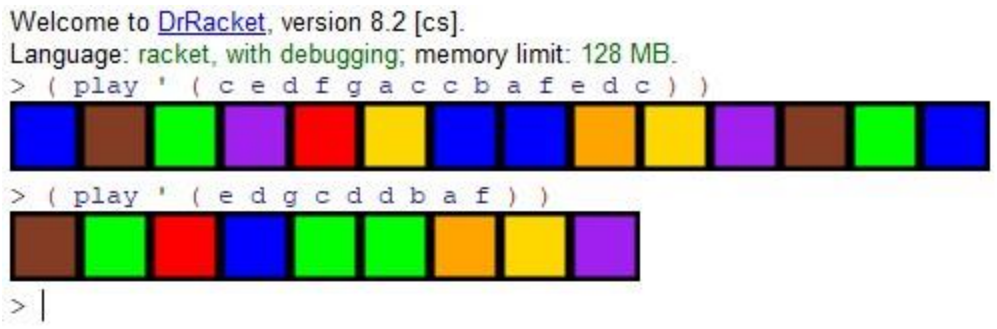
```
> ( diamond-design 10 )
```

Task 8:

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 ) )
)


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

Demo:



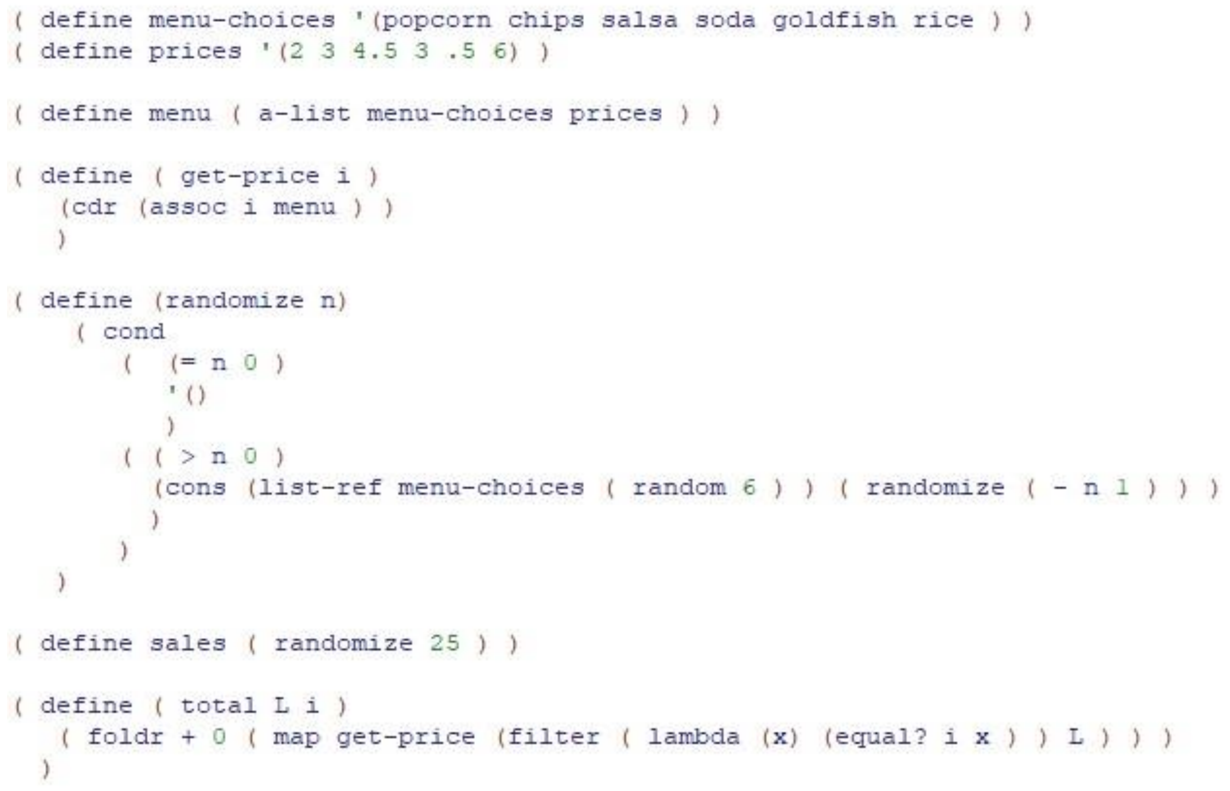
```
Welcome to DrRacket, version 8.2 [cs].
Language: racket, with debugging; memory limit: 128 MB.
> ( play ' ( c e d f g a c c b a f e d c ) )
```

```
> ( play ' ( e d g c d d b a f ) )
```

```
>
```

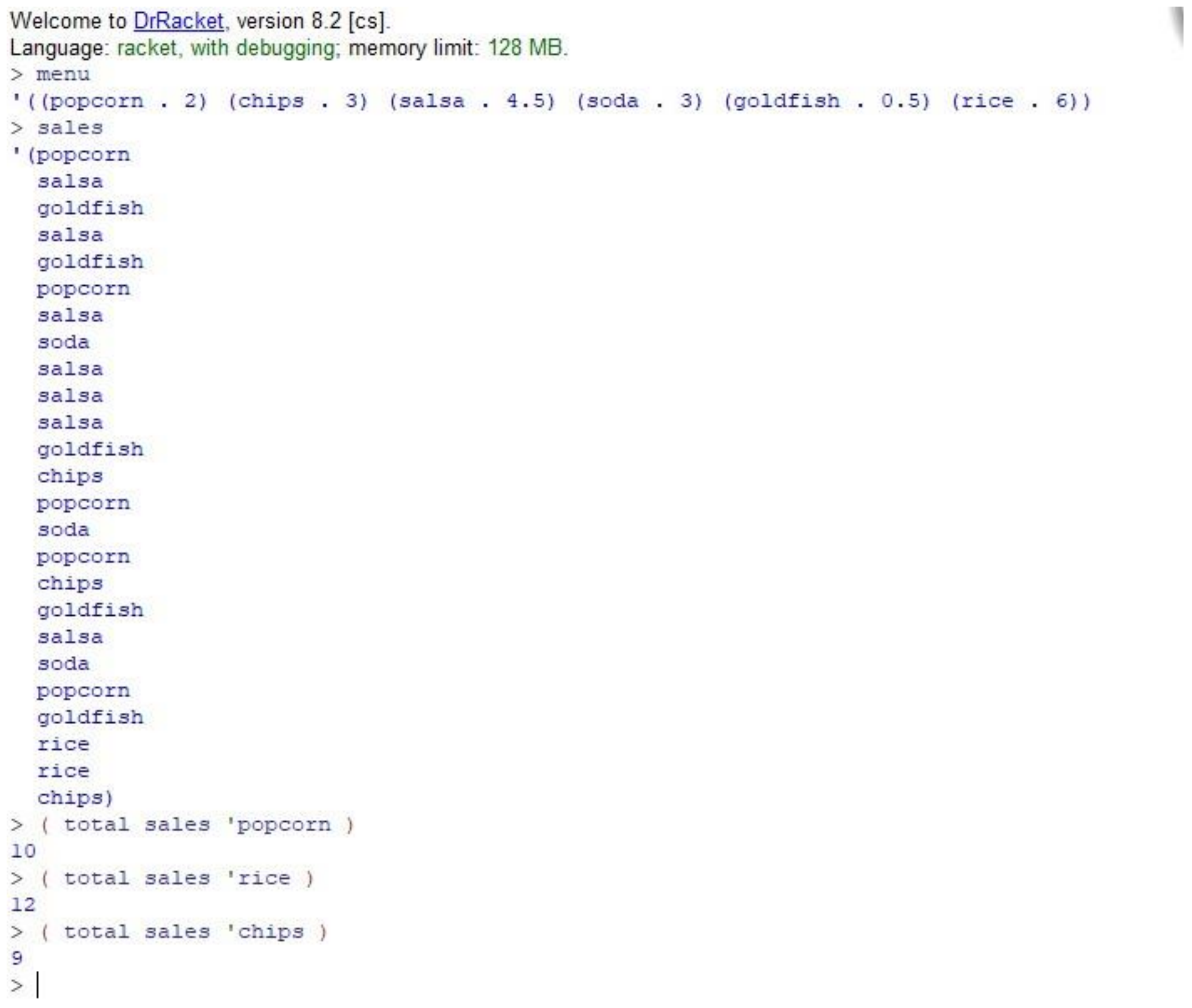
Task 9:

Code:



```
( define menu-choices '(popcorn chips salsa soda goldfish rice ) )
( define prices '(2 3 4.5 3 .5 6) )

( define menu ( a-list menu-choices prices ) )

( define ( get-price i )
   (cdr (assoc i menu ) )
   )

( define (randomize n)
    ( cond
       (   (= n 0 )
           '()
           )
       ( ( > n 0 )
         (cons (list-ref menu-choices ( random 6 ) ) ( randomize ( - n 1 ) ) )
         )
       )
   )

( define sales ( randomize 25 ) )

( define ( total L i )
   ( foldr + 0 ( map get-price (filter ( lambda (x) (equal? i x ) ) L ) ) )
  )
```

Demo:

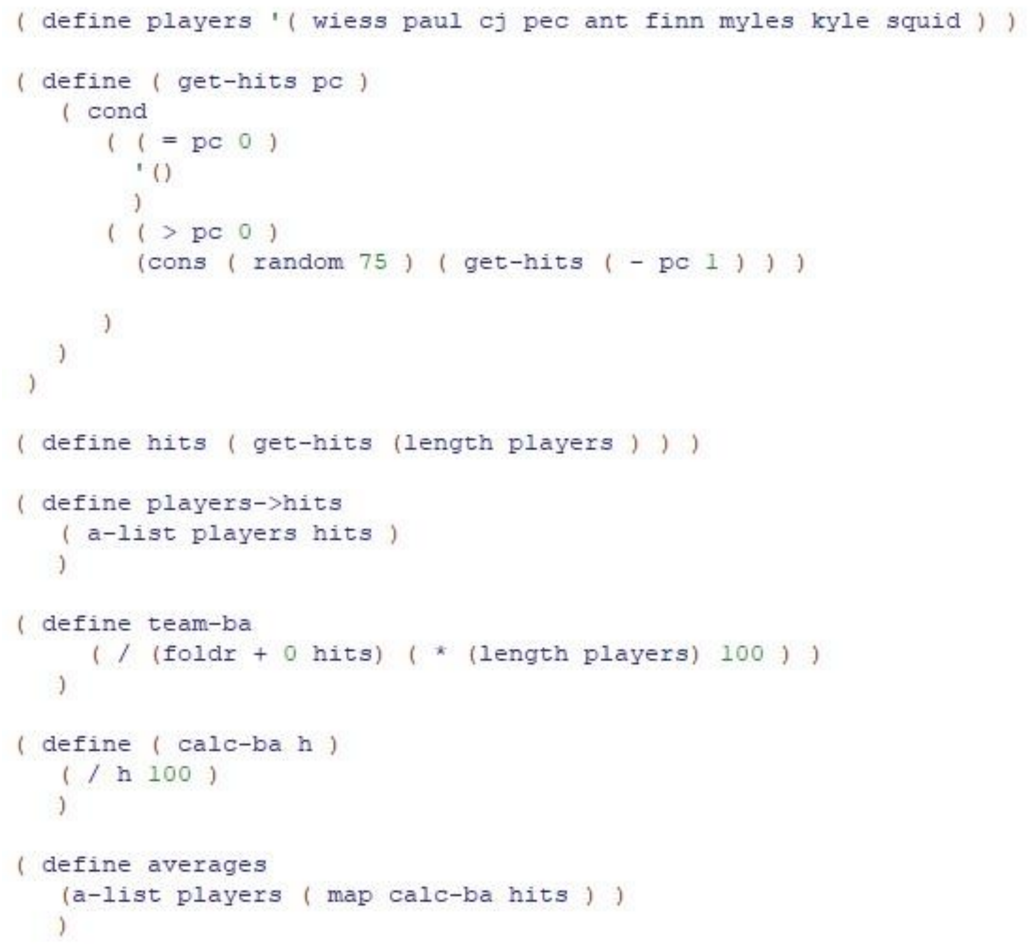


```
Welcome to DrRacket, version 8.2 [cs].
Language: racket, with debugging; memory limit: 128 MB.
> menu
'((popcorn . 2) (chips . 3) (salsa . 4.5) (soda . 3) (goldfish . 0.5) (rice . 6))
> sales
'(popcorn
  salsa
  goldfish
  salsa
  goldfish
  popcorn
  salsa
  soda
  salsa
  salsa
  salsa
  goldfish
  chips
  popcorn
  soda
  popcorn
  chips
  goldfish
  salsa
  soda
  popcorn
  goldfish
  rice
  rice
  chips)
> ( total sales 'popcorn )
10
> ( total sales 'rice )
12
> ( total sales 'chips )
9
> |
```

Task 10:

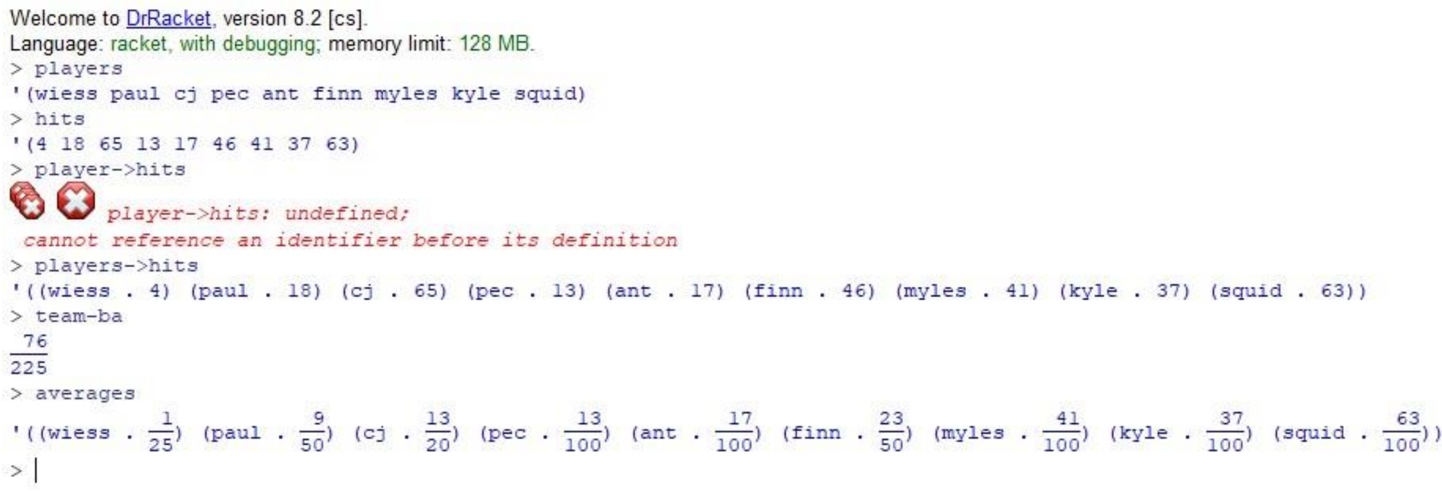
Specification: For my task 10. I created a list of players and a list that represents the hits each player has in 100 at bats. I then created a function that calculated the batting average for all the players individually and one function for the entire team.

Code:



```
( define players '( wiess paul cj pec ant finn myles kyle squid ) )

( define ( get-hits pc )
   ( cond
      ( ( = pc 0 )
        '()
        )
      ( ( > pc 0 )
        (cons ( random 75 ) ( get-hits ( - pc 1 ) ) )

      )
   )
 )

( define hits ( get-hits (length players ) ) )

( define players->hits
   ( a-list players hits )
   )

( define team-ba
     ( / (foldr + 0 hits) ( * (length players) 100 ) )
   )

( define ( calc-ba h )
   ( / h 100 )
   )

( define averages
   (a-list players ( map calc-ba hits ) )
   )
```

Demo:



```
Welcome to DrRacket, version 8.2 [cs].
Language: racket, with debugging; memory limit: 128 MB.
> players
'(wiess paul cj pec ant finn myles kyle squid)
> hits
'(4 18 65 13 17 46 41 37 63)
> player->hits
player->hits: undefined;
 cannot reference an identifier before its definition
> players->hits
'((wiess . 4) (paul . 18) (cj . 65) (pec . 13) (ant . 17) (finn . 46) (myles . 41) (kyle . 37) (squid . 63))
> team-ba
76/225
> averages
'((wiess . 1/25) (paul . 9/50) (cj . 13/20) (pec . 13/100) (ant . 17/100) (finn . 23/50) (myles . 41/100) (kyle . 37/100) (squid . 63/100))
> 
```