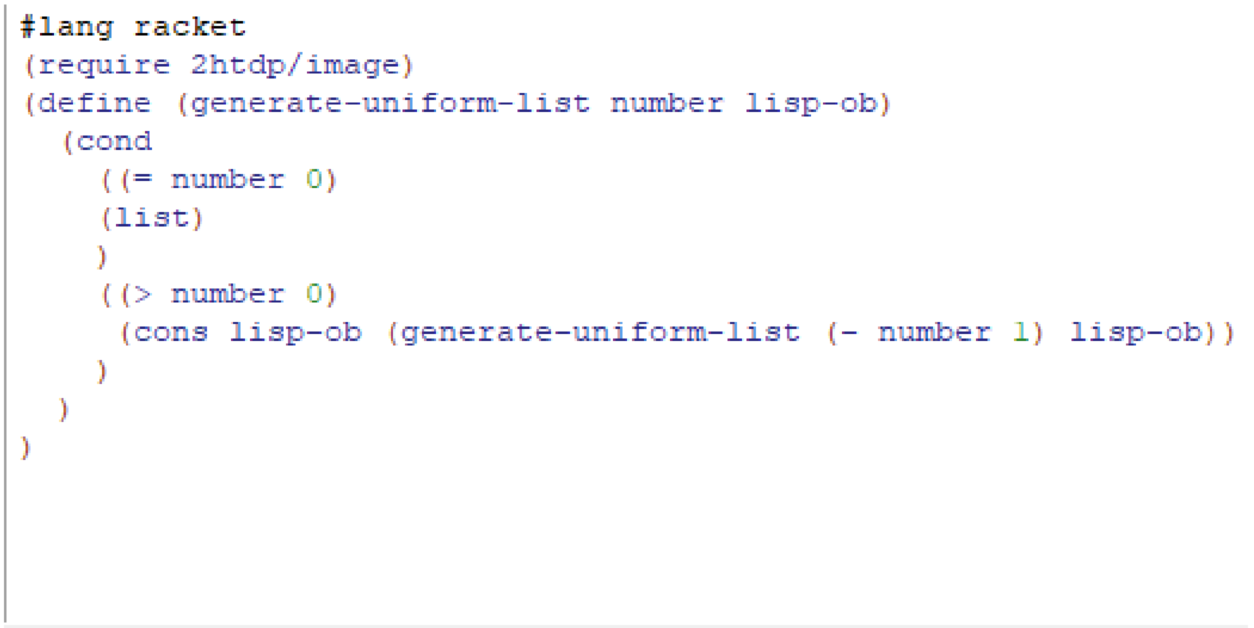
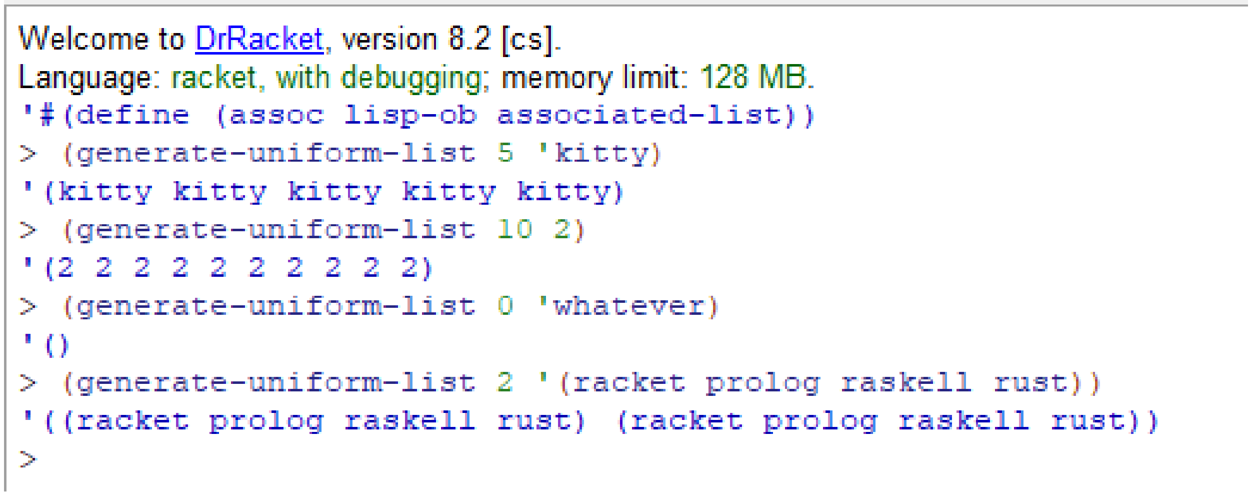
Fourth Racket Programming Assignment

Abstract – The purpose of this assignment is to demonstrate the abilities of recursive functions and higher order functions.

Task 1 – Generate Uniform List

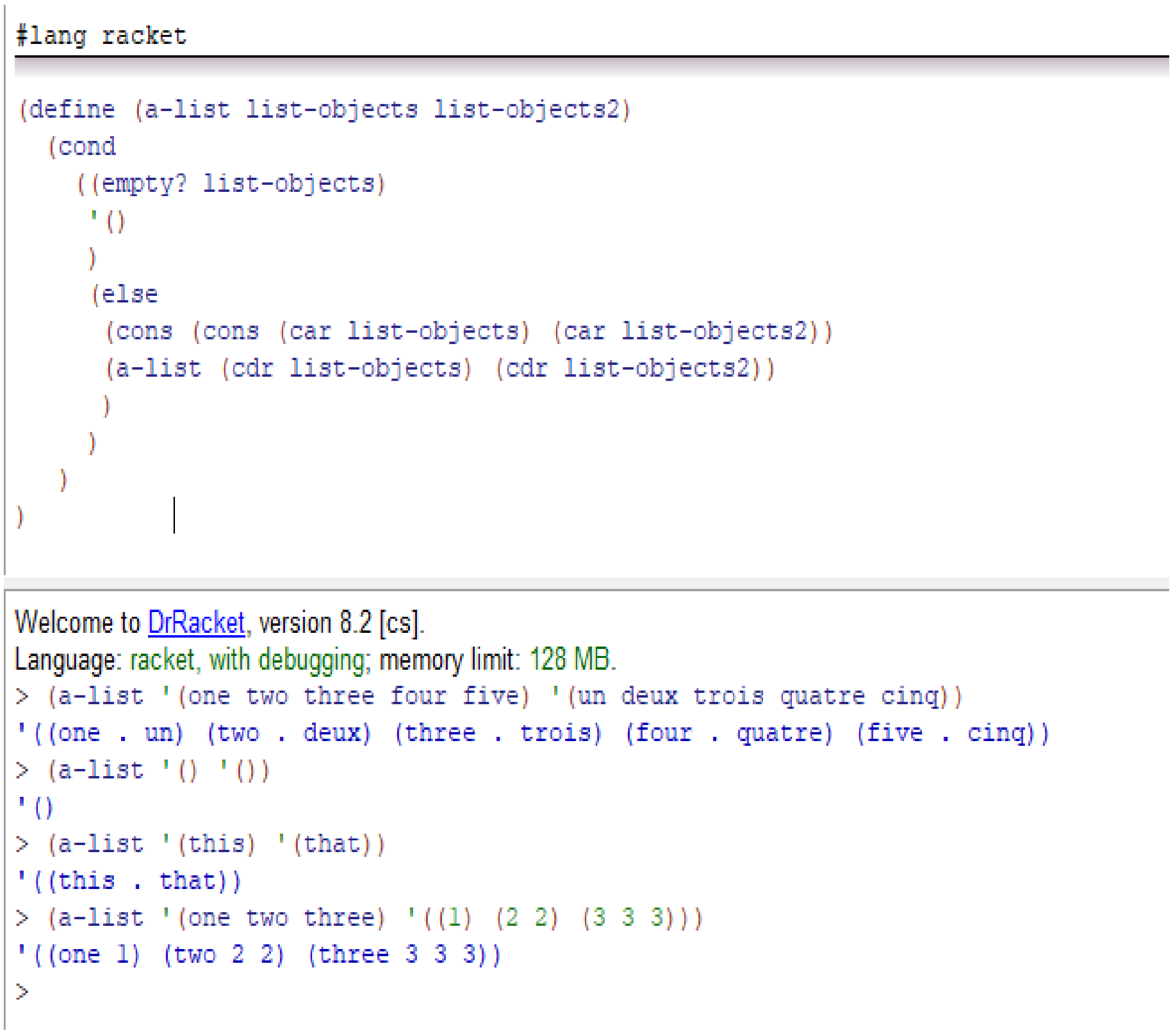


```
#lang racket
(require 2htdp/image)
(define (generate-uniform-list number lisp-ob)
  (cond
    ((= number 0)
    (list)
    )
    ((> number 0)
     (cons lisp-ob (generate-uniform-list (- number 1) lisp-ob))
    )
  )
)
```



```
Welcome to DrRacket, version 8.2 [cs].
Language: racket, with debugging; memory limit: 128 MB.
'#(define (assoc lisp-ob associated-list))
> (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 raskell rust))
'((racket prolog raskell rust) (racket prolog raskell rust))
>
```

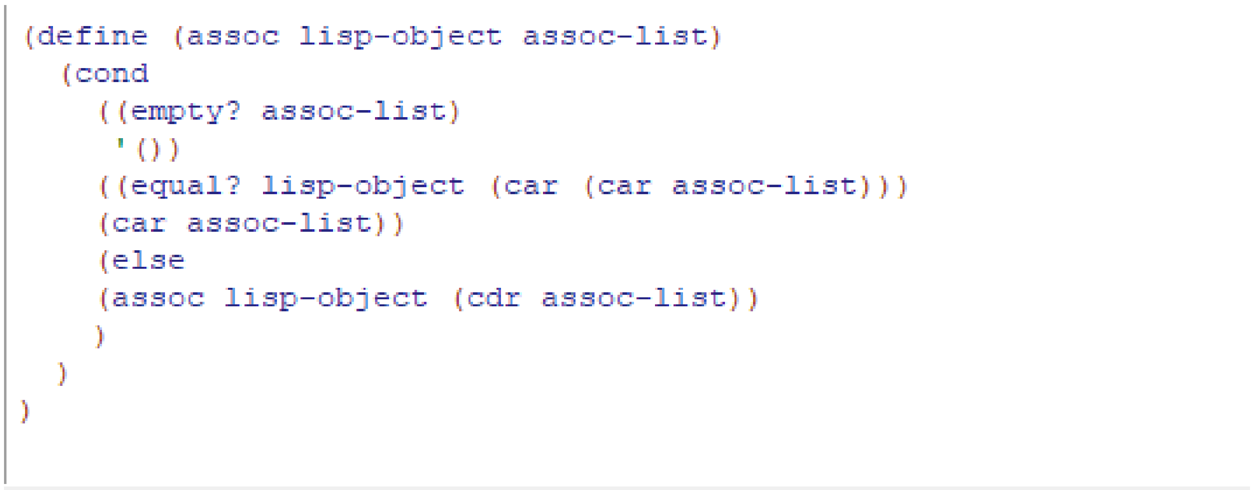
Task 2 – Association List Generator



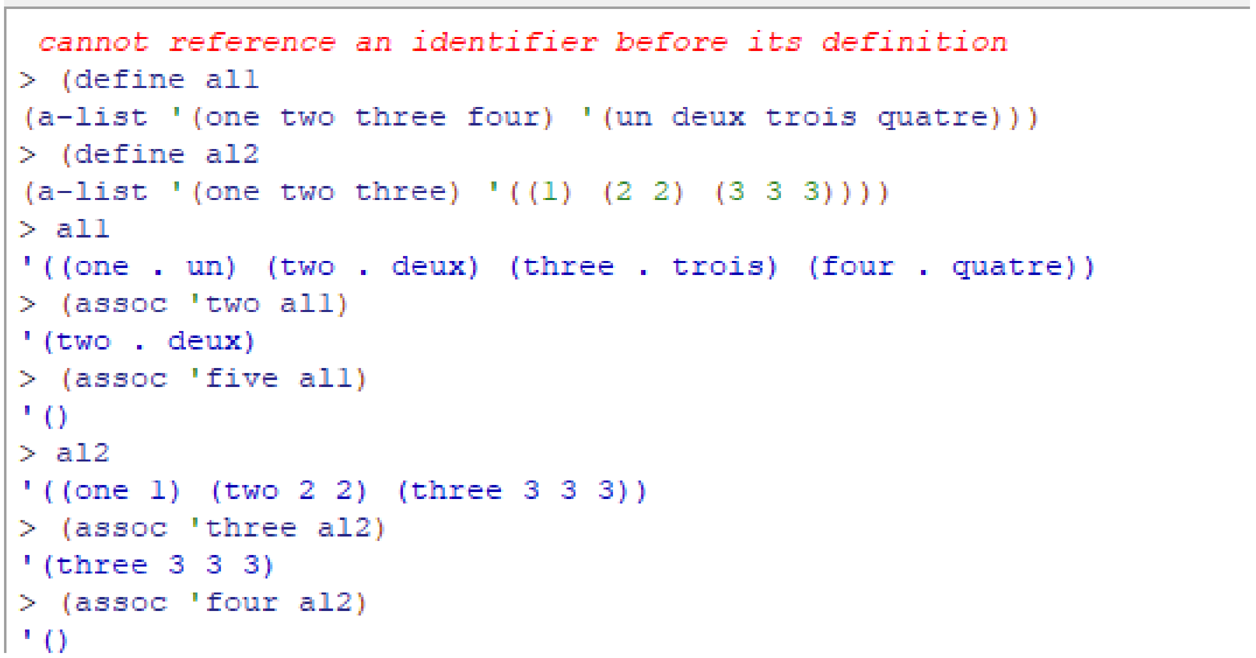
```
#lang racket

(define (a-list list-objects list-objects2)
  (cond
    ((empty? list-objects)
     '()
     )
     (else
      (cons (cons (car list-objects) (car list-objects2))
      (a-list (cdr list-objects) (cdr list-objects2))
      )
     )
   )
)
```

```
Welcome to DrRacket, version 8.2 [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

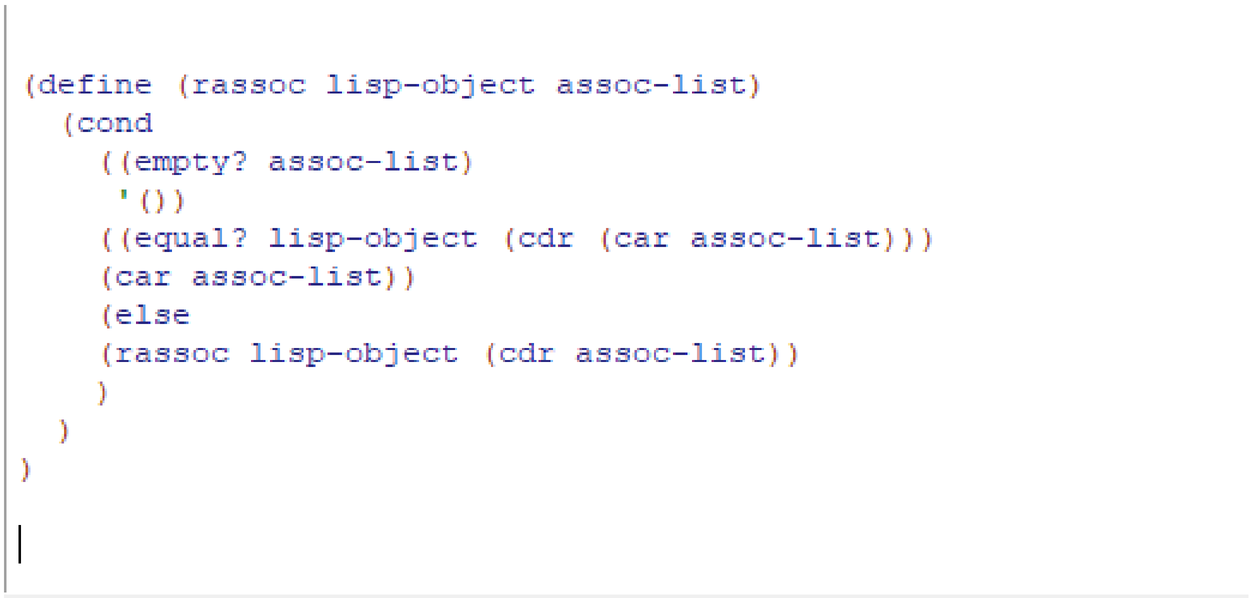


```
(define (assoc lisp-object assoc-list)
  (cond
    ((empty? assoc-list)
     '())
    ((equal? lisp-object (car (car assoc-list)))
    (car assoc-list))
    (else
    (assoc lisp-object (cdr assoc-list))
    )
  )
)
```

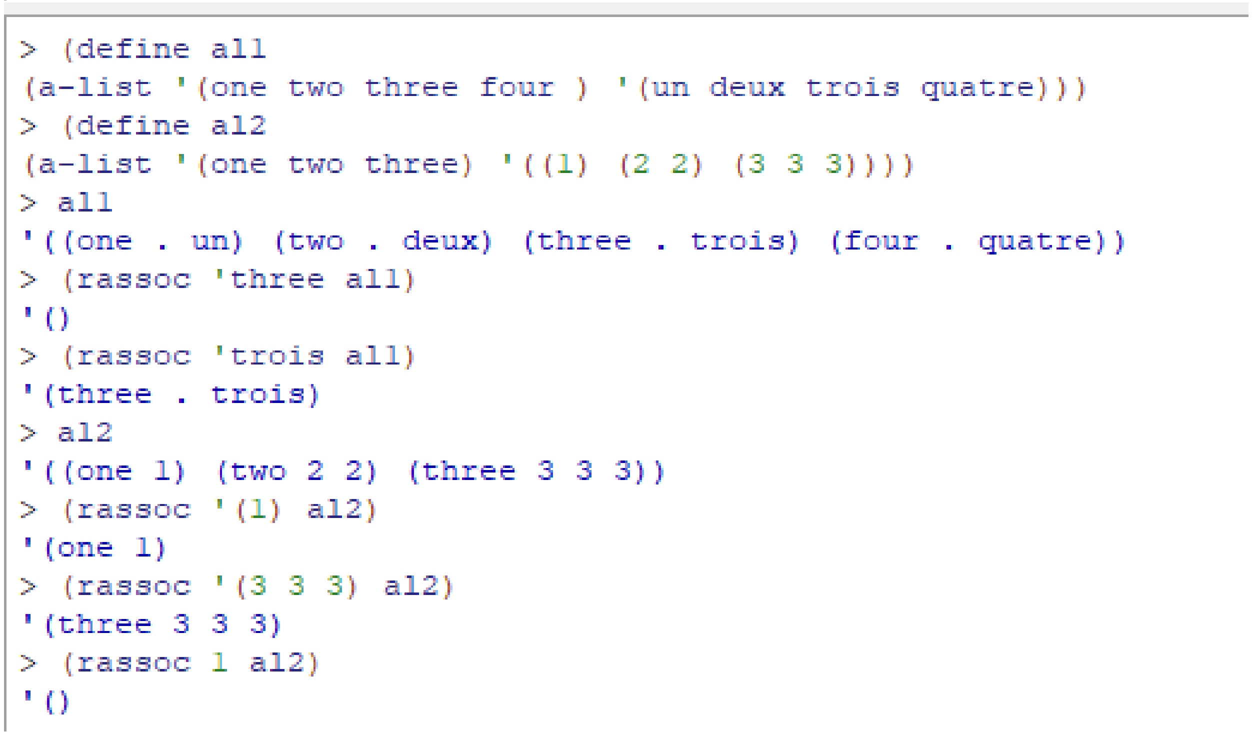


```
 cannot reference an identifier before its definition
> (define al1
(a-list '(one two three four) '(un deux trois quatre)))
> (define al2
(a-list '(one two three) '((1) (2 2) (3 3 3))))
> al1
'((one . un) (two . deux) (three . trois) (four . quatre))
> (assoc 'two al1)
'(two . deux)
> (assoc 'five al1)
'()
> al2
'((one 1) (two 2 2) (three 3 3 3))
> (assoc 'three al2)
'(three 3 3 3)
> (assoc 'four al2)
'()
```

Task 4 – Rassoc

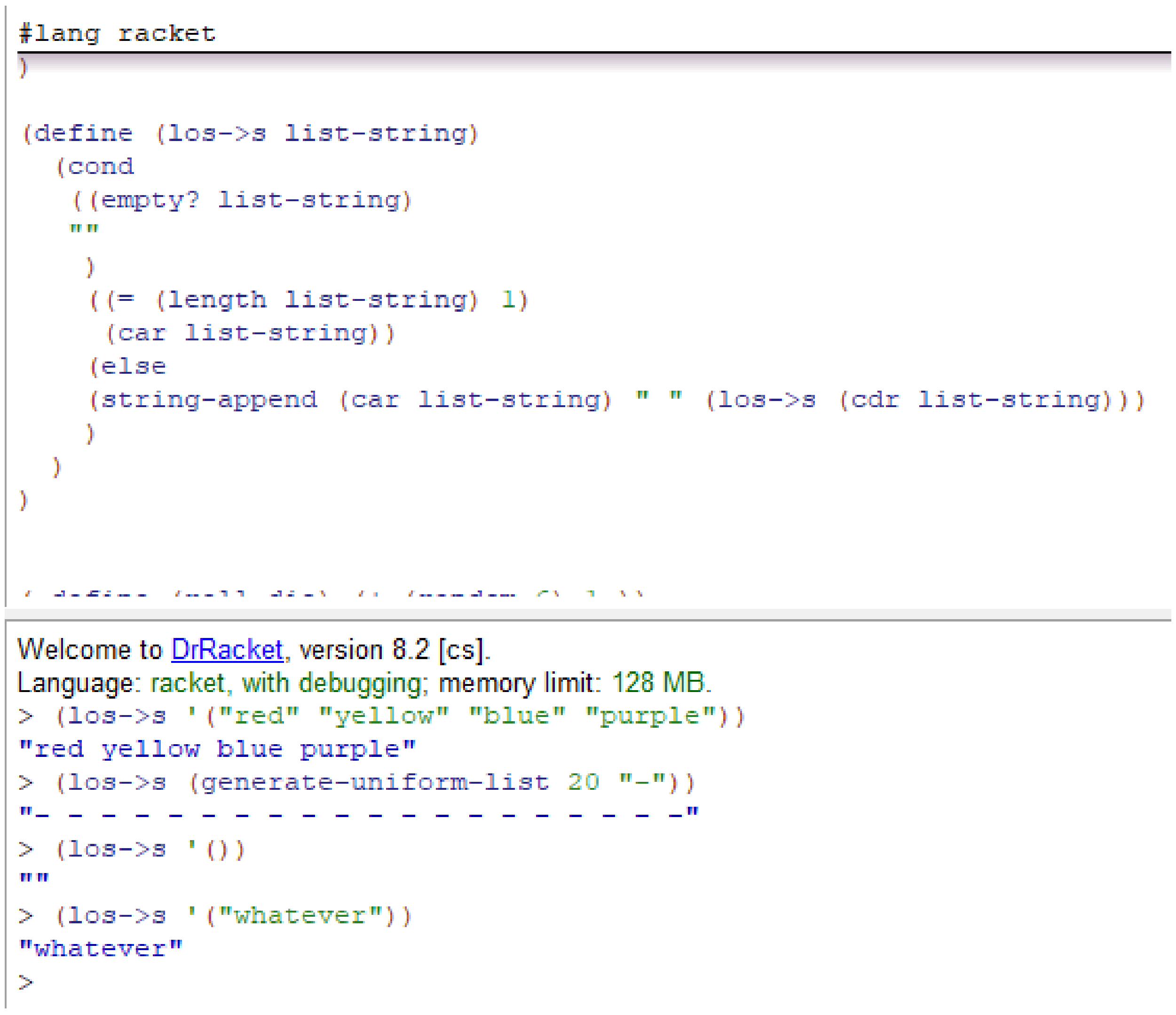


```
(define (rassoc lisp-object assoc-list)
  (cond
    ((empty? assoc-list)
     '())
    ((equal? lisp-object (cdr (car assoc-list)))
    (car assoc-list))
    (else
    (rassoc lisp-object (cdr assoc-list))
    )
  )
)
```



```
> (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 '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)
'()
```

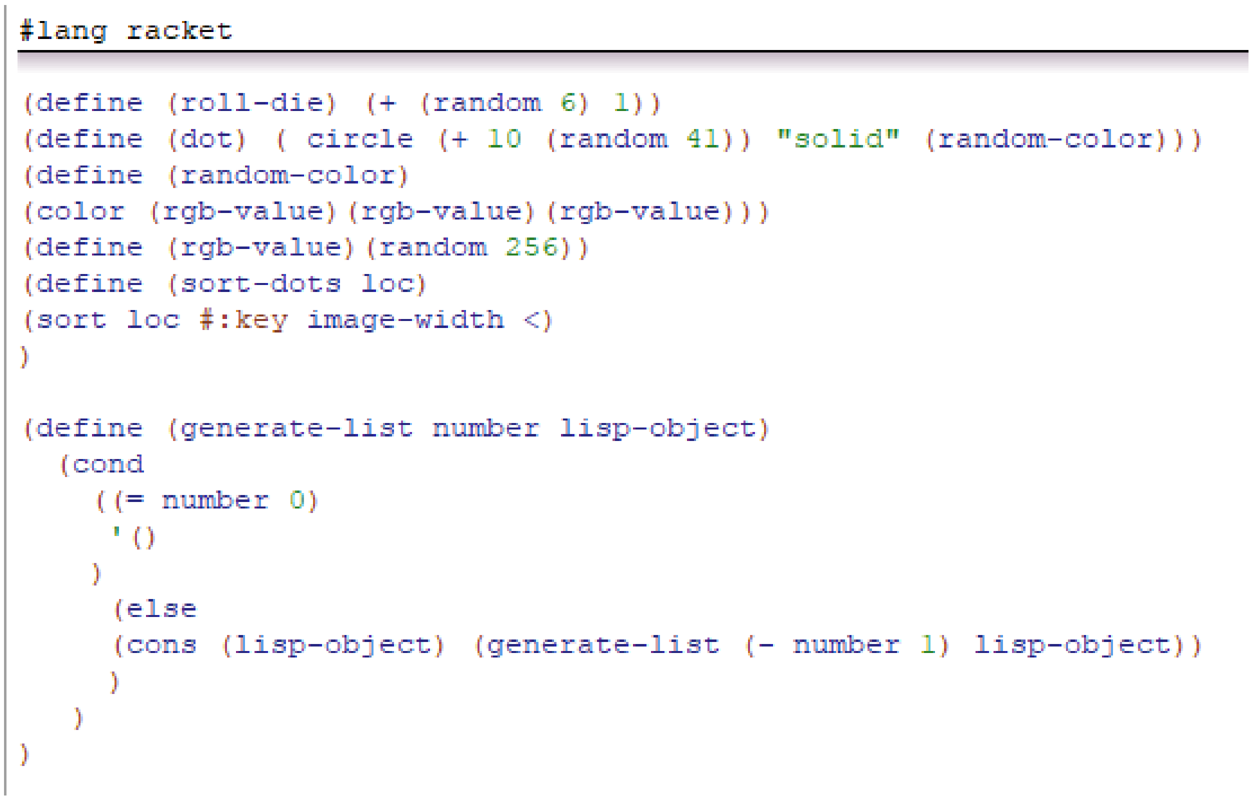
Task 5 – Los ->s



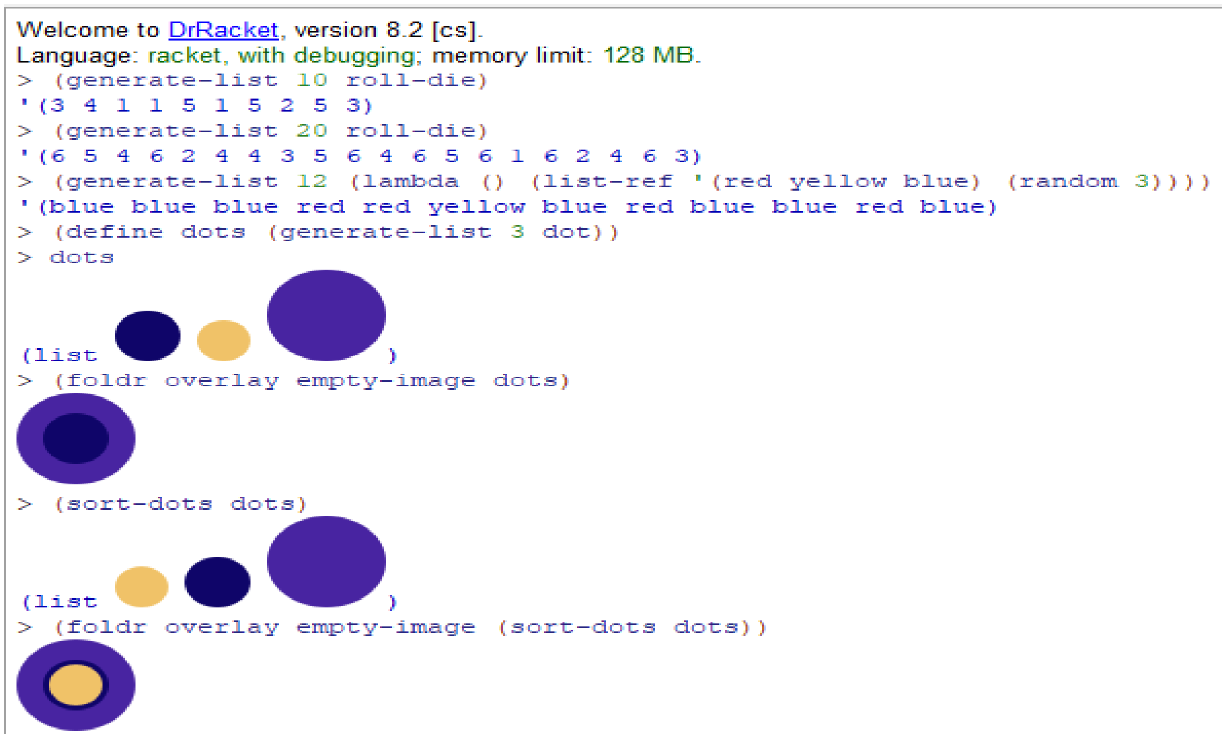
```
#lang racket
)

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

```
Welcome to DrRacket, version 8.2 [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"
>
```

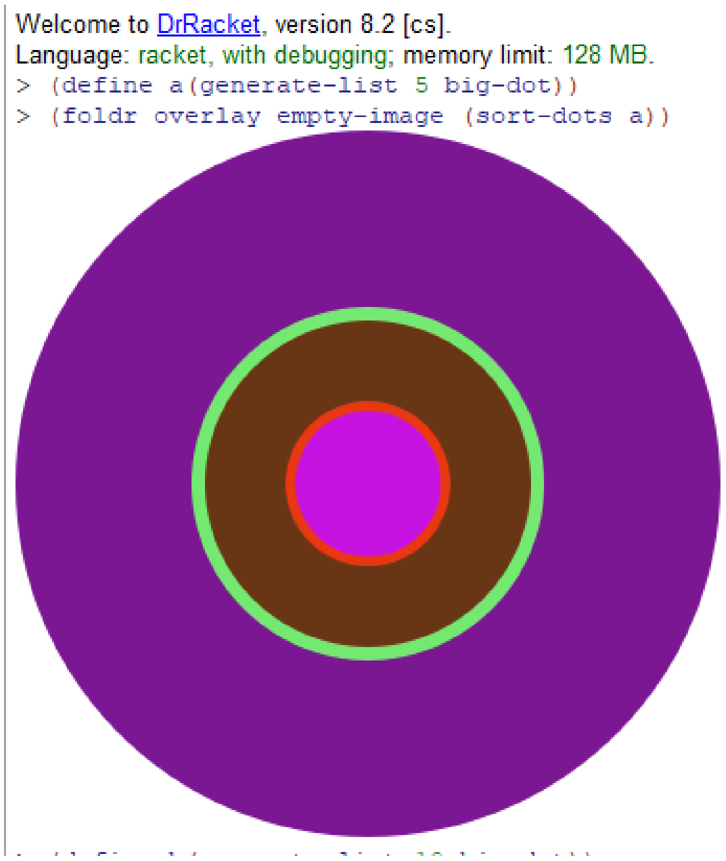
Task 6 – Generate List



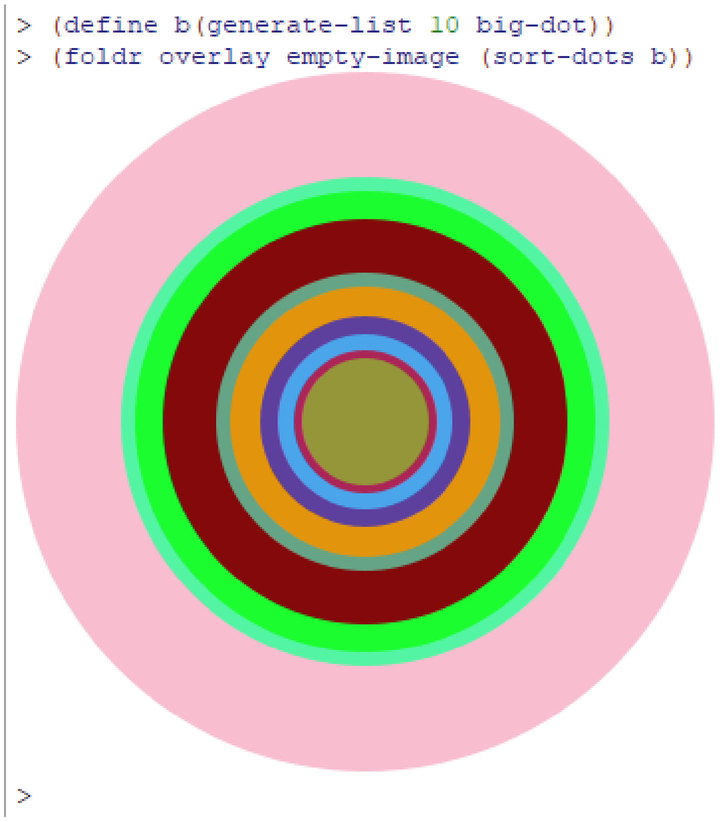
```
#lang racket

(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 number lisp-object)
  (cond
    ((= number 0)
     '()
    )
     (else
     (cons (lisp-object) (generate-list (- number 1) lisp-object))
     )
   )
)
```



```
Welcome to DrRacket, version 8.2 [cs].
Language: racket, with debugging; memory limit: 128 MB.
> (generate-list 10 roll-die)
'(3 4 1 1 5 1 5 2 5 3)
> (generate-list 20 roll-die)
'(6 5 4 6 2 4 4 3 5 6 4 6 5 6 1 6 2 4 6 3)
> (generate-list 12 (lambda () (list-ref '(red yellow blue) (random 3))))
'(blue blue blue red red yellow blue red blue blue red blue)
> (define dots (generate-list 3 dot))
> dots
(list                   )
> (foldr overlay empty-image dots)

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



```
Welcome to DrRacket, version 8.2 [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