Racket Assignment Number 4

By: Justin Cesarini

Abstract

There were many different things I learned with this assignment. I learned how to create lists and how to use them. I learned how to sort, as well as manipulate lists. Later in the assignment you will see how it developed into skills of creating circles, diamonds and even menus. Overall this assignment was more challenging but I was definitely able to learn a lot.

Task 1

Source:

```
> (require 2htdp/image)
> (define (generate-uniform-list number word)
    (cond ((eq? number 0) empty) (else (cons word (generate-uniform-list (- number 1) word) ) ) ) )
```

Demo:

```
> (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

Source:

```
Welcome to DrRacket, version 8.2 [cs].
Language: slideshow, with debugging; memory limit: 128 MB.
> (define (a-list numl num2) (cond ((eq? (length numl) 0) empty) (else (map cons numl num2))))
>
```

Demo:

```
> (define all (a-list '(one two three four) '(un deux trois quatre) ) )
> (a-list '(one two three four five) '(un deux trois quatre cinq)) '((one . un) (two . deux) (three . trois) (four . quatre) (five . cinq))
'((one . un) (two . deux) (three . trois) (four . quatre) (five . cinq))
'((one . un) (two . deux) (three . trois) (four . quatre) (five . cinq))
> (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

Source:

```
> (define (assoc object assoc-list) (cond ((= (length assoc-list) 0) '() ((eq? (car (car assoc-lsit)) object) (car assoc-list)) ((assoc object (cdr assoc-list))))))
>
```

Demo:

```
> (define (a-list list1 list2) (cond ((eq? (length list1) 0) empty) (else (map cons list1 list2))))
> (define (assoc object assoc-list) (cond ((= (length assoc-list) 0) '() ((eq? (car (car assoc-lsit)) object) (car assoc-list)) ((assoc object (cdr assoc-list))))))
> (define mylist1 (a-list '(one two three four) '(un deux trois quatre)))
#<procedure:>>
> (define mylist2 (a-list '(one two three) '( ( 1) (2 2 ) (3 3 3))))
> mylist1
'((one . un) (two . deux) (three . trois) (four . quatre))
> (assoc 'five mylist1)
> mylist2
'((one 1) (two 2 2) (three 3 3 3))
> (assoc 'three mylist2)
> (assoc 'four mylist2)
>
```

Task 4

Source:

```
(define (rassoc word assocList) (cond ((empty? assocList) '()) ((equal? word (cdr (car assocList))) (car assocList)) (else (rassoc word (cdr assocList)))))
```

Demo:

```
> (define all (a-list '(one two three four) '(un deux trois quatre)))
> (define number2 (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)
> number2
'((one 1) (two 2 2) (three 3 3 3))
> (rassoc '(1) number2)
'(one 1)
> (rassoc '(3 3 3) number2)
'(three 3 3 3)
> (rassoc 1 number2)
'()
>
```

Task 5

Source:

```
> (define (los->s TheList)
    (cond
      ( ( = (length TheList ) 0 ) "")
      ( ( = (length TheList ) 1 )
        (car TheList) )
      ( ( > (length TheList 1) (string-append (car TheList) " " (los->s (cdr TheList) ) )))))
```

Demo:

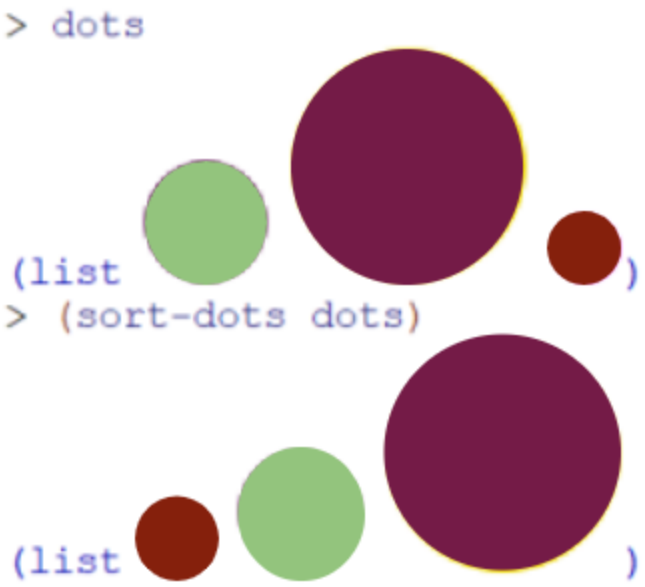
```
> (los->s '("red" "yellow" "blue" "purple"))
"red yellow blue purple"
> (los->s (generate-uniform-list 20 "-"))
"- - - - - - - - - - - - - - - - - - - -"
> (los->s '())
""
> (los->s '("whatever"))
"whatever"
>
```

Task 6

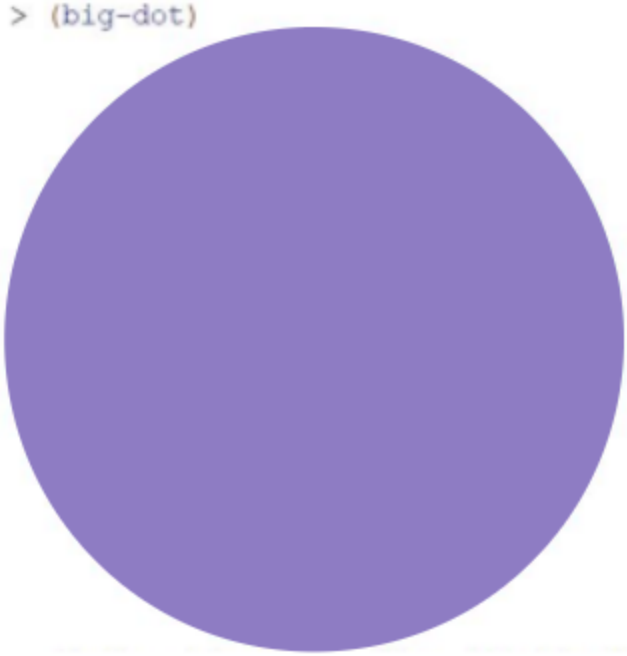
Source:

```
> ( generate-list 10 roll-die )
'(6 1 1 3 1 2 2 2 4 2)
> ( generate-list 20 roll-die )
'(4 2 3 4 1 5 6 5 2 2 6 4 3 3 4 5 6 5 1 6)
> ( generate-list 12 ( lambda () (list-ref '(red yellow blue) (random 3) ) ) )
'(blue yellow yellow red blue red red red red blue yellow red)
> ( define dots ( generate-list 3 dot ) )
```

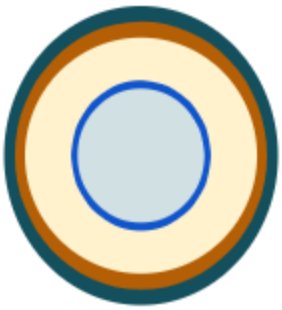
Demo:



```
> (big-dot)
```



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



Task 7

Source:

```
> (define (diamond-design n)
  (define (diamonda)(rotate 45 (square (+ 100 (random 100)) "solid" (random-color))))
  (foldr overlay empty-image (sort-dots (generate-list n diamonda)))
  )
```

Demo:

```
> (diamond-design 5)
```

Task 8

Source:

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

Demo:

```
> (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

Source:

```
> (define item '(Burger Pizza Pepsi CottonCandy ChickenTenders ))
>(define price '(3.5 2.5 1 2 5.5))
>(define menu ( a-list item price))
>(define (cdr-menu n)(assoc n menu))
>(define (random-selection n)
 (cond
 ((= n 0)
 '())
 ((> n 0)
 (cons (list-ref item (random 5))(random-selection (- n 1))))
 )
 )
>(define sales (random-selection 50))
>(define (total list menu)
(foldr + 0 (map cdr-menu(filter (lambda (n)(equal? n menu)) sales)
 ) )
)
```

Demo:

```
> menu
'((Burger . 3.5) (Pizza . 2.5) (Pepsi . 1) (CottonCandy . 2) (ChickenTenders . 5.5))
> sales
'(Pepsi
  Pepsi
  ChickenTenders
  Pepsi
  ChickenTenders
  CottonCandy
  ChickenTenders
  ChickenTenders
  Pepsi
  ChickenTenders
  Pizza
  Pizza
  ChickenTenders
  Pepsi
  CottonCandy
  Pepsi
  Burger
  Pizza
  Pepsi
  Pizza
  Pizza
  Pepsi
  Burger
  Pepsi
  Pepsi
  CottonCandy
  Burger
  Burger
  CottonCandy
  CottonCandy
  CottonCandy
  ChickenTenders
  Pizza
  Pizza
  ChickenTenders
  Pizza
  CottonCandy
  ChickenTenders
  ChickenTenders
  ChickenTenders
  Burger
  Pepsi
  Pepsi
  CottonCandy
  CottonCandy
  ChickenTenders
  CottonCandy
  ChickenTenders
  Pepsi
  Pepsi)
>
```