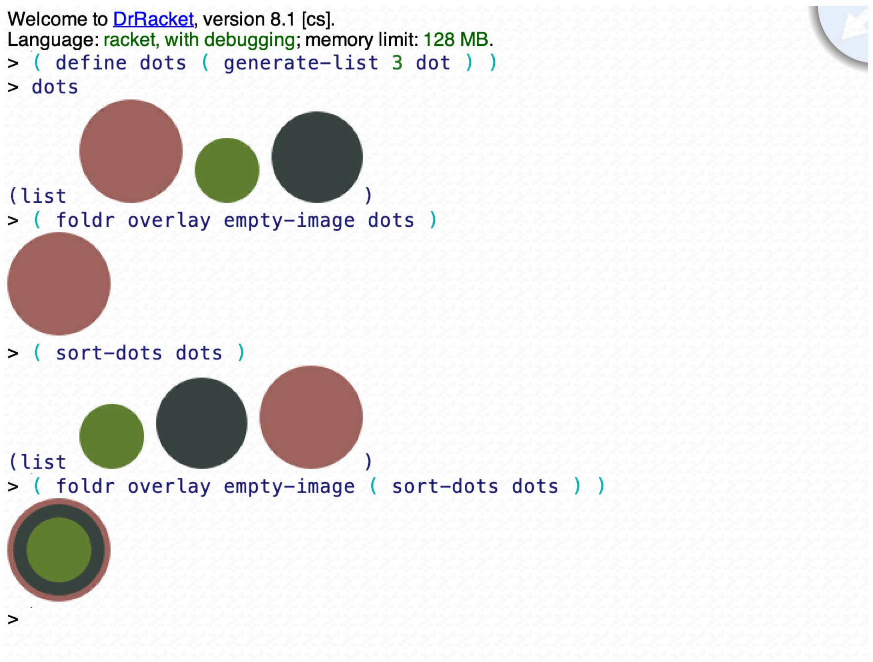
Racket Programming Assignment #4: RLP and HoFs



```
Welcome to DrRacket, version 8.1 [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 ) )
>
```

Working within the DrRacket PDE, please do each of the following tasks. The first five tasks pertain to straightforward recursive list processing. The remaining tasks pertain to list processing with higher order functions. The final task, which you might like to work on from the start, is the document compilation/posting task, which as usual amounts to crafting a single structured document that reflects your work on each programming task, and saving it as a pdf file.

Task 1 - Generate Uniform List

Specification

Define a **recursive** function called `generate-uniform-list` according to the following specification:

1. The first parameter is presumed to be a nonnegative integer.
2. The second parameter is presumed to be a Lisp object.
3. The value of the function will be a list of length equal to the value of the first parameter containing just that many instances of the value of the second parameter.

Demo

```
Welcome to DrRacket, version 8.1 [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

Write the recursive function definition, mimic the demo, and build the source and the demo into your presentation document.

Task 2 - Association List Generator

Specification

Define a **recursive** function called `a-list` according to the following specification:

1. The first parameter is presumed to be a list of objects.
2. The second parameter is presumed to be a list of objects of the same length as the value of the first parameter.
3. The value of the function will be a list of pairs obtained by "consing" successive elements of the two lists.

Demo

```
Welcome to DrRacket, version 8.1 [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

Write the recursive function definition, mimic the demo, and build the source and the demo into your presentation document.

Task 3 - Assoc

Specification

Define a **recursive** function called `assoc` according to the following specification:

1. The first parameter is presumed to be a lisp object.
2. The third parameter is presumed to be an association list.
3. The value of the function will be the first pair in the given association list for which the car of the pair equals the value of the first parameter, or '() if there is no such element.

Demo

```
Welcome to DrRacket, version 8.1 [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

Write the recursive function definition, mimic the demo, and build the source and the demo into your presentation document.

Task 4 - Rassoc

Specification

Define a **recursive** function called `rassoc` according to the following specification:

1. The first parameter is presumed to be a lisp object.
2. The third parameter is presumed to be an association list.
3. The value of the function will be the first pair in the given association list for which the cdr of the pair equals the value of the first parameter, or '() if there is no such element.

Demo

```
Welcome to DrRacket, version 8.1 [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

Write the recursive function definition, mimic the demo, and build the source and the demo into your presentation document.

Task 5 - Los->s

Specification

Define a **recursive** function called `los->s` according to the following specification:

1. The first and only parameter is presumed to be a list of character strings.
2. The value of the function will a string containing the strings found in the value of the parameter separated by spaces.

Demo

```
Welcome to DrRacket, version 8.1 [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

Write the recursive function definition, mimic the demo, and build the source and the demo into your presentation document.

Task 6 - Generate list

Specification

Define a **recursive** function called `generate-list` according to the following specification:

1. The first parameter is a nonnegative integer.
2. The second parameter is a parameterless function that returns a lisp object.
3. The function returns a list of length equal to the value of the first parameter containing objects created by calls to the function represented by the second parameter.

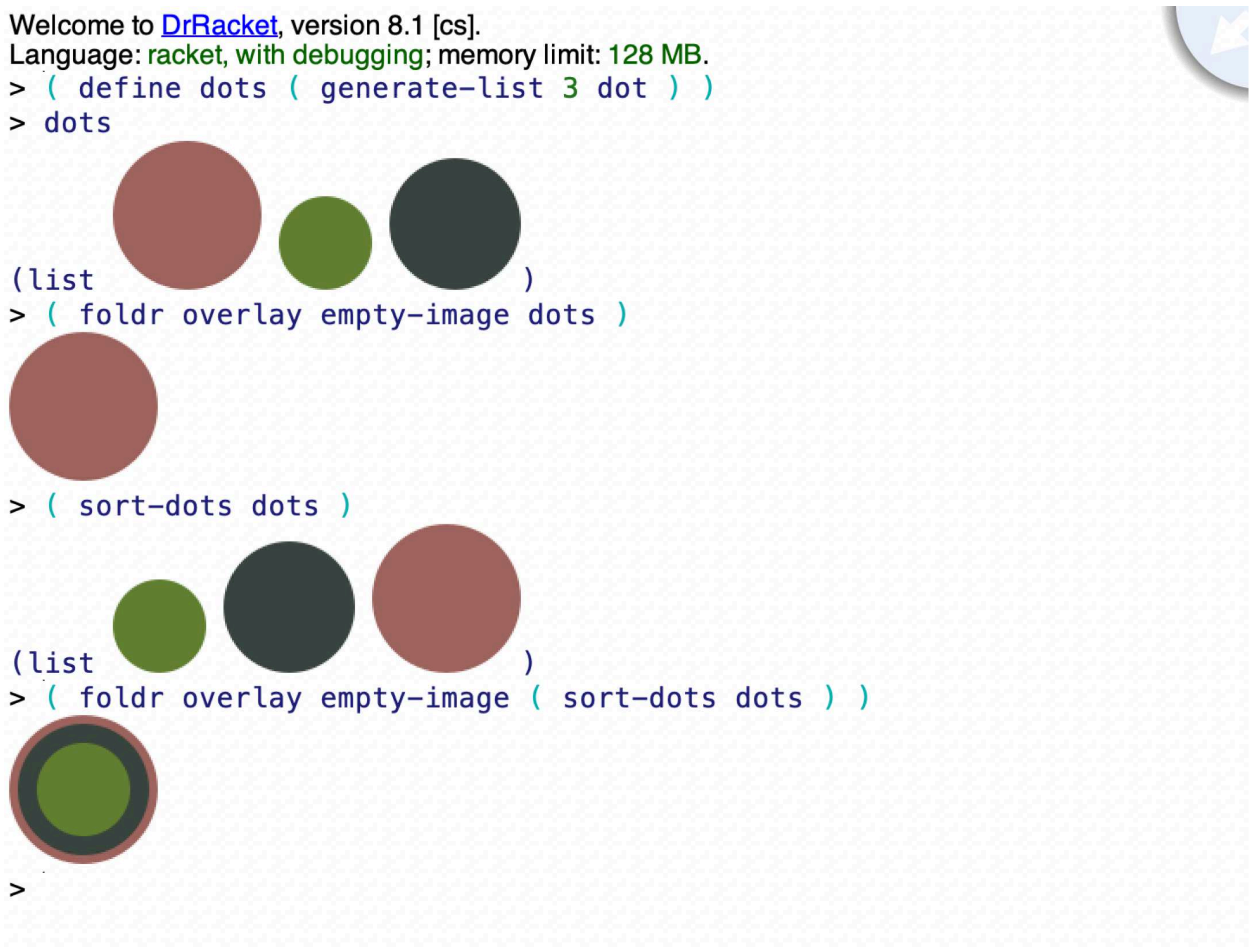
Some auxiliary code to support the demo

```
( 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 < )
)
```

Demo 1

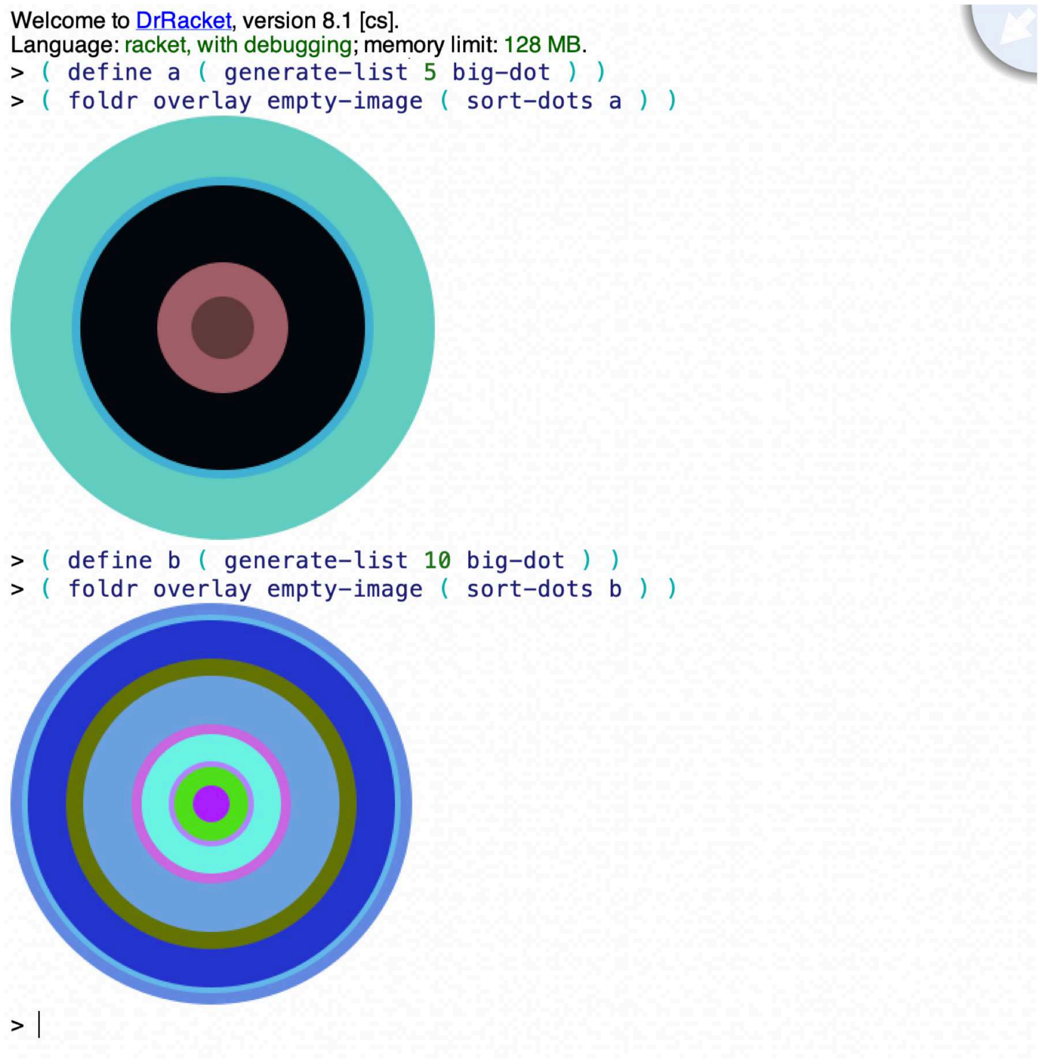
```
Welcome to DrRacket, version 8.1 [cs].
Language: racket, with debugging; memory limit: 128 MB.
> ( generate-list 10 roll-die )
'(6 2 5 6 3 3 2 3 1 6)
> ( generate-list 20 roll-die )
'(3 2 5 4 3 3 5 1 4 6 1 6 6 2 5 6 1 1 2 6)
> ( generate-list 12
    ( lambda () ( list-ref '( red yellow blue ) ( random 3 ) ) )
  )
'(yellow yellow blue yellow blue blue blue blue blue red blue yellow)
>
```

Demo 2



```
Welcome to DrRacket, version 8.1 [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 ) )

>
```

Demo 3



```
Welcome to DrRacket, version 8.1 [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

Write the recursive function definition, mimic the **three** demos, and build the source code and the **three** demos into your presentation document.

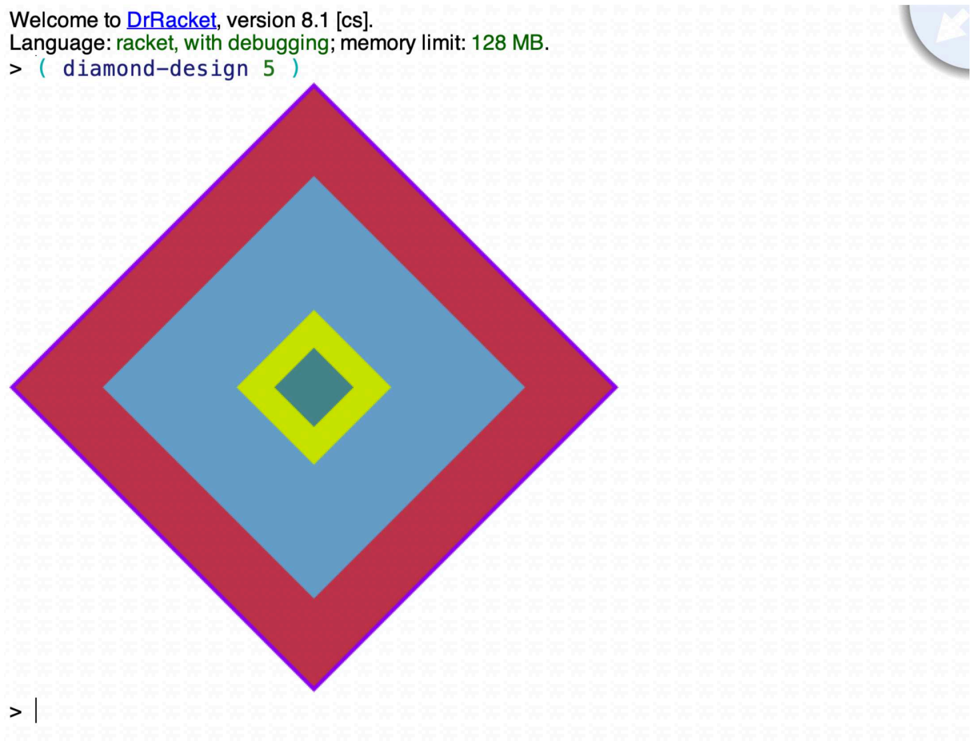
Task 7 - The Diamond

Specification

Using what you learned from Task 6 as a hint, define a function called `diamond` that is consistent with the following specification:

1. The sole parameter is a number indicating how many diamonds will be featured in the design.
2. The function returns an image which consists of the number of diamonds specified by the parameter, where each diamond is randomly colored and has a side length between 20 and 400.

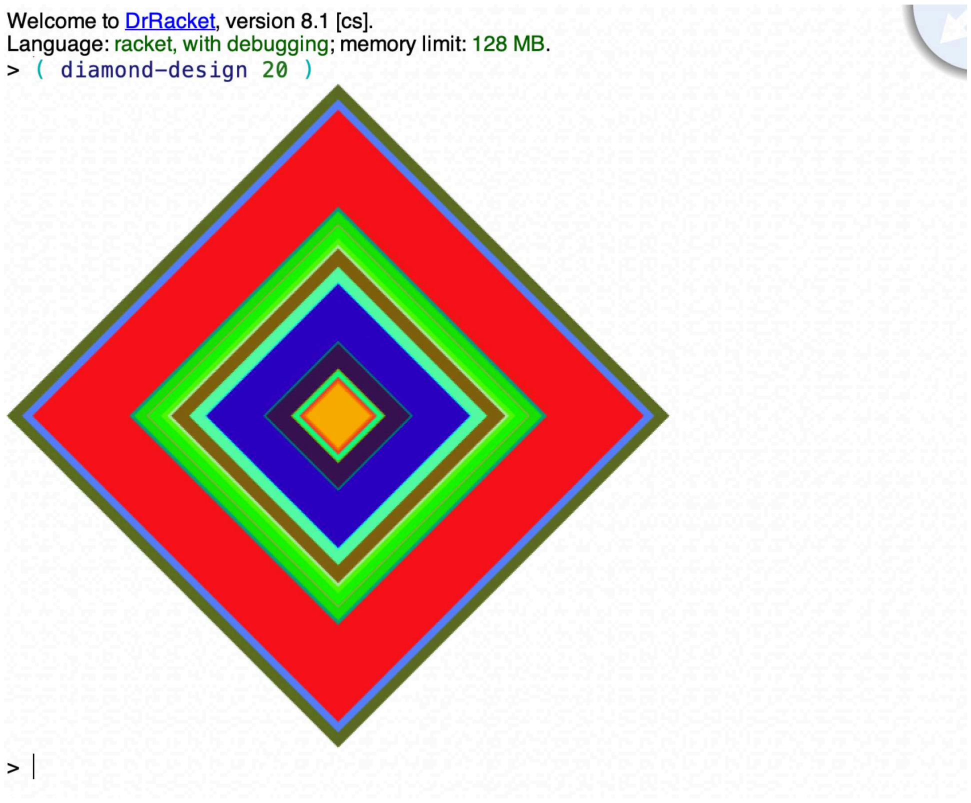
Demo 1



```
Welcome to DrRacket, version 8.1 [cs].
Language: racket, with debugging; memory limit: 128 MB.
> ( diamond-design 5 )
```

```
> |
```

Demo 2



```
Welcome to DrRacket, version 8.1 [cs].
Language: racket, with debugging; memory limit: 128 MB.
> ( diamond-design 20 )
```

```
> |
```

Task

Write the function recursive function definition, mimic the **two** demos, and build the source code and the **two** demos into your presentation document.

Task 8 - Chromesthetic renderings

Specification

Define a function called `play` according to the following specification:

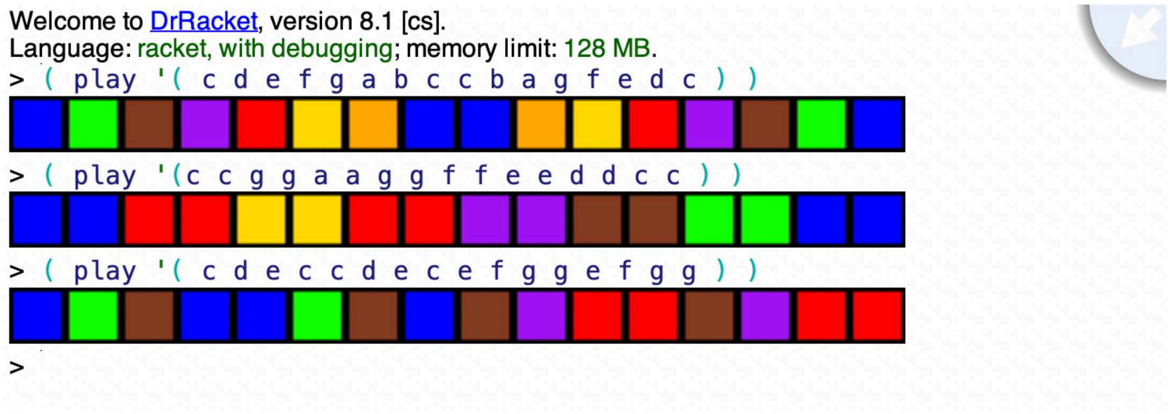
1. The sole parameter is a list of pitch names drawn from the set {c, d, e, f, g, a, b}.
2. The result is an image consisting of a sequence of colored squares with black frames, with the colors determined by the following mapping: c→blue; d→green; e→brown; f→purple; g→red; a→gold; b→orange.

Constraint: Your function definition must use `map` twice and `foldr` one time.

Some auxilliary for you to use

```
( 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 ) )
)
```

Demo



```
Welcome to DrRacket, version 8.1 [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

Write the function definition subject to the various constraints, mimic the demo, and build the source code and the demo into your presentation document.

Task 9 - Diner

Specification

Imagine a diner which has a menu of exactly 6 items. Furthermore, assume the menu is maintained as an association list of item/price pairs. Also, assume that the items sold for a day are maintained as a linear list. With this in mind, define a function called `total` according to the following specification:

1. The first parameter is a linear list of the items sold over some period of time.
2. The second parameter is an item that appears on the menu.
3. The result is the total amount of money collected on the sale of the given item over the period of time.

Constraint: Your function definition must use `map` one time and `filter` one time and `foldr` one time.

Hints: (1) You might want to write a function which takes the name of an item as its sole parameter and returns the price of the item. (2) Lambda functions can be very useful.

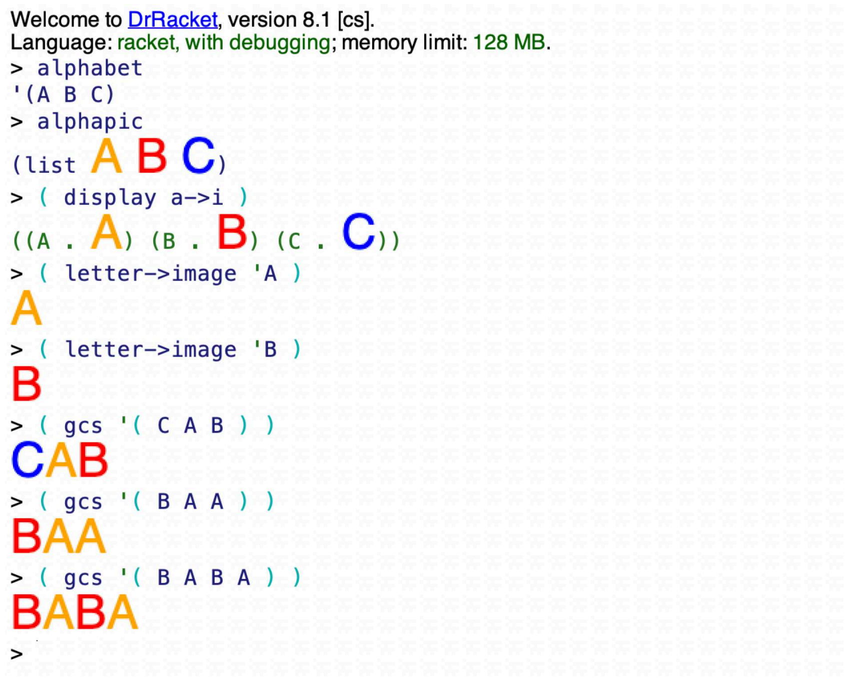
Demo

```
Welcome to DrRacket, version 8.1 [cs].
Language: racket, with debugging; memory limit: 128 MB.
> menu
'((hamburger . 5.5) (grilledcheese . 4.5) (malt . 3) (coke . 1) (coffee . 1) (pie . 3.5))
> sales
'(hamburger
  coke
  grilledcheese
  malt
  grilledcheese
  coke
  pie
  coffee
  hamburger
  hamburger
  coke
  hamburger
  malt
  hamburger
  malt
  pie
  coffee
  pie
  coffee
  grilledcheese
  malt
  hamburger
  hamburger
  coke
  pie
  coffee
  pie
  coffee
  hamburger
  malt
  hamburger
  malt)
> ( total sales 'hamburger )
49.5
> ( total sales 'hotdog )
0
> ( total sales 'grilledcheese )
13.5
> ( total sales 'malt )
18
> ( total sales 'coke )
4
> ( total sales 'coffee )
5
>
```

Task

Write the function definition subject to the various constraints, create a demo of your own with a **different menu** of exactly six items and an appropriate sales list of at least 30 items, and build your source and the demo into your presentation document.

Task 10 - Grapheme Color Synesthesia



```
Welcome to DrRacket, version 8.1 [cs].
Language: racket, with debugging; memory limit: 128 MB.
> 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
>
```

Task 10a - ABC

1. Add the following code to your source file, after the code which is associated with the previous tasks:

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

2. Making good use of the association list to which the variable `a->i` is bound, and making good use of the `assoc` function, write the `letter->image` function, and test it.
3. Making good use of both the `map` function and the `foldr` function, write the `gcs` function, and test it.

Task 10b - ABC Demo

Recreate the demo that opens this grapheme to color synesthesia task.

Task 10c - The full alphabet

Extend the program so that it will map the full alphabet, not just the first three letters, to colored images.

Task 10d - Alphabet Demo

Create a demo that illustrates grapheme to color synesthesia on at least 10 words, including alphabet and dandelion.

Task 11 - Document Compilation/Posting

Craft a nicely structured document that contains:

1. A nice title, indicating that this is your fourth Racket assignment.
2. A nice learning abstract, which artfully says something about recursive list processing and list processing with higher order functions.
3. A section that provides the code and the demo for Task 1
4. A section that provides the code and the demo for Task 2
5. A section that provides the code and the demo for Task 3
6. A section that provides the code and the demo for Task 4
7. A section that provides the code and the demo for Task 5
8. A section that provides the code and the demos for Task 6
9. A section that provides the code and the demos for Task 7
10. A section that provides the code and the demo for Task 8
11. A section that provides the code and the demo for Task 9
12. A section that provides the (extended) code and the two demos for Task 10

Post your document to you web work site.

Due Date

Please complete your work on this assignment, and post your work to your web work site, by the end of the day on Wednesday, October 26, 2021.