
L^AT_EX Racket Assignment 4

Matilda Knapp

March 2022

Learning Abstract

Assignment 4 is focused on using recursion and higher order functions to transform data in lists. In problem one the count function counts the number of a specified object in a list. In problem two the set function returns a list of unique objects from a list of objects with some of the same objects. In problem 3 I created an association list, this can be used to create a relationship between two lists. Following this is the Assoc function in problem 4, this function is like an index, it is given the first value in an a-list and find that a-list within a list of a-lists. In the problems after this I use instances of the functions I made to help them function. The frequency table code was provided to me in the outline, but it uses the count, a-list, and set functions. After problem 5 the following problems tend to use map, or foldr functions. In problem 6 I made the generate-list function which generates a list of objects. This function is used to help me make a list of circles and diamonds in problem 6 and 7. The foldr function is used to overlay these shapes. Problem 8 uses foldr and map, which references a list of simple notes, and links them with a colored block. Foldr is this time used with the beside function to help put the string of blocks together. In problem 9 I am given recursive code which I have to make non-recursive using higher-order functions to substitute what the recursion normally does. I thought that problem 9 was the most difficult problem in this assignment. The function in problem 9 flips for offset of 100 coin flips. Lastly we have problem 10, these functions all focus on blood pressure trend. For part of the assignment I made the code to visualize the blood pressure trends. Red is a very bad blood pressure, orange is a pretty bad blood pressure, yellow is a somewhat good blood pressure, and blue is a very good blood pressure. I could definitely see how a program like this could be used in the medical field.

Problem 1 - Count DEMO

```
> ( count 'b '(a a b a b c a b c d ) )
3
> ( count 5 '(1 5 2 5 3 5 4 5 ) )
4
> ( count 'cherry '(apple peach blueberry ) )
0
```

Problem 1 - Count CODE

```
#lang racket
( define (count lis-obj lst )
  ( cond
    ((empty? lst )
     + 0)
    ((equal? (car lst) lis-obj)
     (+ 1 (count lis-obj (cdr lst))))
    )
  ( else
    (count lis-obj (cdr lst))
    )
  )
)
```

Problem 2 - list->set DEMO

```
> ( list->set '(a b c b c d c d e) )
'(a b c d e)
> ( list->set '(1 2 3 2 3 4 3 4 5 6) )
'(1 2 3 4 5 6)
> ( list->set '(apple banana apple banana cherry) )
'(apple banana cherry)
```

Problem 2 - list->set CODE

```
#lang racket
(define (list->set lst)
  (cond
    ((empty? lst)
     '())
    ((member (car lst) (cdr lst))
     (list->set (cdr lst)))
    (else
     (cons (car lst) (list->set (cdr lst)))))
  )
)
```

Problem 3 - Association List Generator DEMO

```
> ( 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))
```

Problem 3 - Association List Generator CODE

```

#lang racket
( define (a-list lst1 lst2)
  ( cond
    ((empty? lst1)
      '()
    )
    (else
      (cons (cons (car lst1)(car lst2)) (a-list (cdr lst1) (cdr lst2)))
    )
  )
)

```

Problem 4 - Assoc DEMO

```

> ( 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 )
'()

```

Problem 4 - Assoc CODE

```

( define (assoc obj al)
  ( cond
    ((empty? al)
      '()
    )
  )
)

```

```

((equal? obj (car (car al)))
 (cons (car al) (assoc obj (cdr al) )
      )
 )
( else
  (assoc obj (cdr al))
 )
)
)

```

Problem 5 - Frequency Table QUESTIONS

1. List the names of the functions used within the *ft* function that you were asked to write in this programming assignment.

I was asked to write the *a – list* function, the *list – > set* function and the *count* function, for the *ft* function to work.

2. Within the *ft* function, what function is provided to the higher order function *map*? Since you cannot name this function, please write down the complete definition of this function.

In the *ft* function a lambda function is being used by the higher order *map* function. In this lambda function there is one parameter *x*, this references *the-set* function which references the *list->set* function, and uses the given list to make a list of only one of each type of object in a list, then this list goes back into lambda, and the count function is used. In the count function the list values are counted and returned.

3. How many parameters must the functional argument to the application of *map* in the *ft* function take?

It would take one parameter which is in lambda represented by *x*, lambda expects *x* to be a list.

4. What would be the challenge involved in writing a named function to take the place of the lambda function within the *ft* function. Do your best to articulate this challenge in just one sentence.

Map takes only two arguments, so using a lambda function I'm able to meet that requirement, and still add the necessary code.

5. Within the *ft* function, what function is provided to the higher order function *sort*? Since you can name this function, please simply write down its name.
This would be the *#:key* car function, or *key* function.
6. What is a “keyword argument”?
A keyword argument is an argument made up of a keyword and expression. This argument can act as a keyword often in a map function. The keyword is started by adding a hash, and the word after is the expression.
7. Within the *ft*-visualizer function, what function is provided to the higher order function *map*? Since you can name this function, please simply write down its name.
It is the *pair*-visualizer function.
8. Why was the challenge involved in using a named function in the application of map in the *ft* function absent in the application of map in the *ft*-visualization function?
The *pair*-visualizer function only takes one parameter so map would only be getting two arguments and there won't be an issue.
9. Within the *pair*-visualizer function, what function is provided to the higher order function *foldr*? Since you can name this function, please simply write down its name
This function is *string*-append.
10. Does the *add*-blanks function make use of any higher order functions?
No instead it uses recursion.
11. Why do you think the *display* function, with the empty string as its argument, was called in the *ft*-visualizer function?
It is added to help the program display the output, similar to adding an empty list at the end of a recursive function.
12. What data structure is being used to represent a frequency table in this implementation? Please be as precise as you can be in articulating your answer, preferring abstraction to detail in your precision of expression.
When I output the data within a *ft*(frequency table), I get a list of dotted pairs back, this is later used by the visualizer to output the stars.
13. Is the *make*-list function used in the *pair*-visualizer function a primitive function in Racket?
Yes this is a primitive function in racket.
14. What do you think is the most interesting aspect of the given frequency table generating code?

I find the use of the *make-list* function the most interesting, I think it's cool how we use the pair number to generate the specified number of stars.

15. Please ask a meaningful question about some aspect of the accompanying code. Do your best to make it a question that you think a reasonable number of your classmates will find interesting.

Could you use the *pair-visualizer* by itself to get an output, and if so how would you do this?

Problem 5 - Frequency Table DEMO

```
> ( define ft1 ( ft '(10 10 10 10 1 1 1 1 9 9 9 2 2 2 8 8 3 3 4 5 6 7) ) )
> ft1
'((1 . 4) (2 . 3) (3 . 2) (4 . 1) (5 . 1) (6 . 1) (7 . 1) (8 . 2) (9 . 3) (10 . 4))
> ( ft-visualizer ft1)
```

```
1:    ****
2:    ***
3:    **
4:    *
5:    *
6:    *
7:    *
8:    **
9:    ***
10:   ****
```

```
> ( define ft2 ( ft '( 1 10 2 9 3 8 4 4 7 7 6 6 6 5 5 5 ) ) )
> ft2
'((1 . 1) (2 . 1) (3 . 1) (4 . 2) (5 . 3) (6 . 3) (7 . 2) (8 . 1) (9 . 1) (10 . 1))
> ( ft-visualizer ft2)
```

```
1:    *
2:    *
3:    *
4:    **
5:    ***
6:    ***
7:    **
8:    *
```

9: *
10: *

Problem 5 - Frequency Table CODE

```
( define ( ft the-list )
  ( define the-set ( list->set the-list ) )
  ( define the-counts
    ( map ( lambda (x) ( count x the-list ) ) the-set )
  )
  ( define association-list (a-list the-set the-counts ) )
  ( sort association-list < #:key car )
)

( define ( ft-visualizer ft )
  ( map pair-visualizer ft )
  ( display "" )
)

( define ( pair-visualizer pair )
  ( define label
    ( string-append ( number->string ( car pair ) ) ":" )
  )
  ( define fixed-size-label ( add-blanks label ( - 5 ( string-length label ) ) )
  ( display fixed-size-label )
  ( display
    ( foldr
      string-append
      ""
      ( make-list ( cdr pair ) "*" )
    )
  )
  ( display "\\n" )
)

( define ( add-blanks s n )
  ( cond
    ( ( = n 0 ) s )
```



```

    ( else ( add-blanks ( string-append s " " ) ( - n 1 ) ) )
  )
)

```

Important Note *a – list*, *count*, and *list– > set* functions are all required to run this program.

Problem 6 - Generate list DEM01

```

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

```

Generate list DEM02

```

Welcome to DrRacket, version 8.3 [cs].
Language: racket, with debugging; memory limit: 128 MB.
> ( define dots ( generate-list 3 dot ) )
> dots
(list (image of a large red circle) (image of a small blue circle) (image of a large green circle) )
> ( foldr overlay empty-image dots )
(image of a large red circle)
> ( sort-dots dots )
(list (image of a small blue circle) (image of a large red circle) (image of a large green circle) )
> ( foldr overlay empty-image ( sort-dots dots ) )
(image of a large red circle with a small blue circle inside)
>

```

Generate list DEMO3

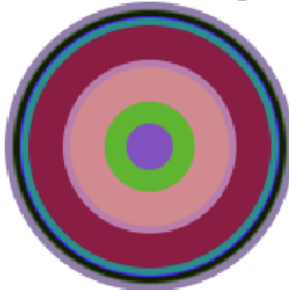
Welcome to [DrRacket](#), version 8.3 [cs].

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 ) )
```



```
>
```

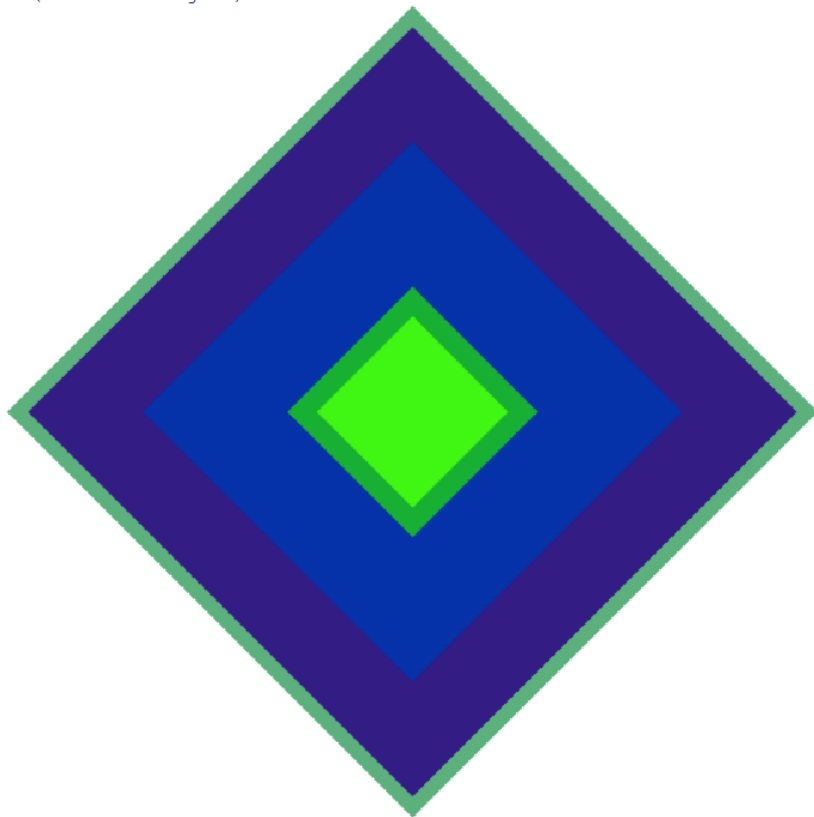
Problem 6 - Generate list CODE

```
#lang racket  
(require 2htdp/image)  
( define (generate-list num obj)  
  ( cond ((eq? num 0)  
          '()  
        )  
        ( else  
          (cons (obj) (generate-list (- num 1) obj))  
        )  
      )  
)  
  
( define ( roll-die ) ( + ( random 6 ) 1 ) )  
( define ( dot )
```

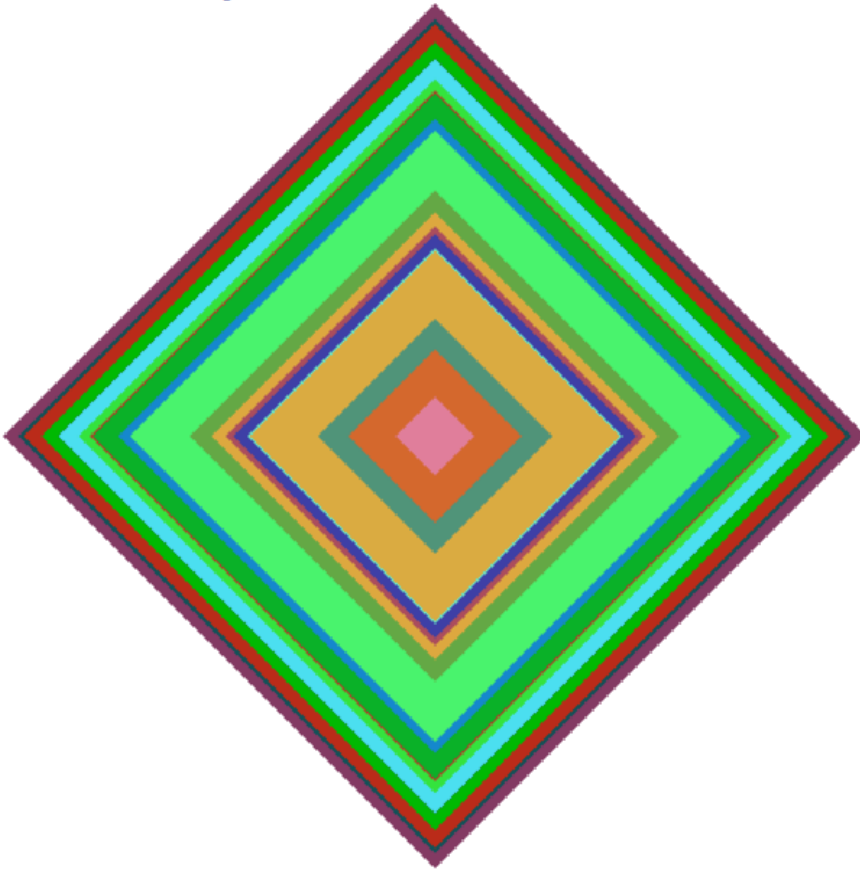
```
( circle ( + 10 ( random 41 ) ) "solid" ( random-color ) )  
)  
( define ( big-dot )  
  ( circle ( + 10 ( random 100 ) ) "solid" ( random-color ) )  
)  
( define ( random-color )  
  ( color ( random 256 ) ( random 256 ) ( random 256 ) )  
)  
( define ( sort-dots loc )  
  ( sort loc #:key image-width < )  
)
```

Problem 7 - The Diamond DEMO

Welcome to [DrRacket](#), version 8.3 [cs].
Language: racket, with debugging; memory limit: 128 MB.
> (diamond-design 5)



```
> (diamond-design 20)
```



Problem 7 - The Diamond CODE

```
( define (diamond)
  (rotate 45 (square (+ 20 (random 381)) "solid" ( random-color ) ) )
)
( define (diamond-design num)
  ( define diamonds ( generate-list num diamond ) )
  ( foldr overlay empty-image ( sort-dots diamonds ) )
)
```

*Uses code from problem 6

Problem 8 - Chromesthetic renderings DEMO

Welcome to [DrRacket](#), version 8.3 [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) )
```



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



*I added an extra song to the demo, because I thought this was especially cool. I hope that's okay.

Problem 8 - Chromesthetic renderings CODE

```
#lang racket
(require 2htdp/image)
(define (a-list lst1 lst2)
  (cond
    ((empty? lst1)
     '())
    (else
     (cons (cons (car lst1)(car lst2)) (a-list (cdr lst1) (cdr lst2)))))
  )
)
(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 ) )
)
; Your function definition must use map twice and foldr one time
( define (play pitch-list )
  ;map letters to color-name
  (define cl (map pc->color pitch-list) )
  ;map color-name to color-block
  (define bl (map color->box cl ) )
  ;foldr read from left to right
  (define fold (foldr beside empty-image bl))
  (display fold)
)

```

Problem 9 - Transformation of a Recursive Sampler DEMO

```

> ( flip-for-offset 100 )
0
> ( flip-for-offset 100 )
16
> ( flip-for-offset 100 )
4

```

```

> ( flip-for-offset 100 )
2
> ( flip-for-offset 100 )
6
> ( demo-for-flip-for-offset )
-18: *
-14: **
-10: *****
-8:  *****
-6:  *****
-4:  *****
-2:  *****
0:   *****
2:   *****
4:   *****
6:   ****
8:   *****
10:  *
12:  ***
14:  ***
18:  *
> ( demo-for-flip-for-offset )
-18: *
-16: *
-14: *
-12: ***
-10: *****
-8:  *****
-6:  *****
-4:  *****
-2:  *****
0:   *****
2:   *****
4:   *****
6:   *****
8:   ***
10:  ***
12:  ***
14:  **
18:  *

```

Problem 9 - Transformation of a Recursive Sampler CODE

```
#lang racket
;Other functions referenced
(define (generate-list num obj)
  (cond ((eq? num 0)
        '())
        (else
         (cons (obj) (generate-list (- num 1) obj))))
  )
)

(define (list->set lst)
  (cond
    ((empty? lst)
     '())
    ((member (car lst) (cdr lst))
     (list->set (cdr lst)))
    (else
     (cons (car lst) (list->set (cdr lst))))
  )
)

(define (count lis-obj lst )
  (cond
    ((empty? lst )
     + 0)
    ((equal? (car lst) lis-obj)
     (+ 1 (count lis-obj (cdr lst))))
    (else
     (count lis-obj (cdr lst)))
  )
)
```



```

    )
  )
( define (a-list lst1 lst2)
  ( cond
    ((empty? lst1)
      '()
    )
    (else
      (cons (cons (car lst1)(car lst2)) (a-list (cdr lst1) (cdr lst2)))
    )
  )
)

( define ( ft the-list )
  ( define the-set ( list->set the-list) )
  ( define the-counts
    ( map ( lambda (x) ( count x the-list) ) the-set )
  )
  ( define association-list (a-list the-set the-counts ) )
  ( sort association-list < #:key car )
)

( define ( ft-visualizer ft )
  ( map pair-visualizer ft )
  ( display "" )
)

( define ( pair-visualizer pair )
  ( define label
    ( string-append ( number->string ( car pair ) ) ":" )
  )
  ( define fixed-size-label ( add-blanks label ( - 5 ( string-length label ) ) )
  ( display fixed-size-label )
  ( display
    ( foldr
      string-append
      ""
      ( make-list ( cdr pair ) "*" )
    )
  )
)

```

```

    ( display "\n" )
  )

( define ( add-blanks s n )
  ( cond
    ( ( = n 0 ) s )
    ( else ( add-blanks ( string-append s " " ) ( - n 1 ) ) ) )
  )

( define ( flip-coin )
  ( define outcome ( random 2 ) )
  ( cond
    ( ( = outcome 1 )
      'h
    )
    ( ( = outcome 0 )
      't
    )
  )
)

;-----
;(Flip for offset function)
( define ( flip-for-offset n )
  (define x (generate-list n flip-coin))
  (define ht '(h t))
  (define tonum (map (lambda (y) (count y x)) ht))
  (foldr - (car tonum) (cdr tonum) )
)

;(Flip for offset demo function)
( define ( demo-for-flip-for-offset )
  ( define offsets
    ( generate-list
      100
      ( lambda () (flip-for-offset 50 ) )
    )
  )
  ( ft-visualizer (ft offsets ) )
)

```

Problem 10 - Blood Pressure Trend Visualizer [Part One] DEMO

ONE

```
> ( sample 120 )
134
> ( sample 80 )
84
```

TWO

```
> ( data-for-one-day 110 )
'(119 95)
> ( data-for-one-day 110 )
'(121 77)
> ( data-for-one-day 110 )
'(129 91)
> ( data-for-one-day 90 )
'(111 61)
> ( data-for-one-day 90 )
'(105 73)
> ( data-for-one-day 90 )
'(103 71)
```

THREE

```
> ( data-for-one-week 110 )
'((115 99) (133 91) (131 89) (147 99) (135 79) (137 101) (139 99))
> ( data-for-one-week 100 )
'((130 74) (116 88) (128 78) (118 76) (118 84) (116 74) (122 78))
> ( data-for-one-week 90 )
'((115 63) (109 79) (105 61) (121 67) (107 69) (109 63) (123 73))
```

FOUR

```
> ( define getting-worse '(95 98 100 102 105) )
> ( define getting-better '(105 102 100 98 95) )
> ( generate-data getting-worse )
'(((108 84) (110 70) (120 86) (112 80) (108 74) (128 80) (114 74))
  ((123 73) (121 87) (107 81) (121 79) (121 75) (113 69) (103 75))
  ((120 86) (140 86) (116 78) (122 80) (126 76) (126 84) (104 90))
  ((117 75) (111 77) (117 67) (119 83) (109 77) (113 81) (119 79)))
```

```

((117 93) (115 89) (111 87) (119 85) (129 81) (115 69) (125 85)))
> ( generate-data getting-better )
'(((133 77) (127 83) (123 83) (141 91) (125 85) (125 93) (131 87))
  ((123 79) (121 81) (133 65) (125 91) (129 87) (131 71) (135 83))
  ((118 76) (130 72) (108 74) (110 80) (114 78) (130 82) (116 76))
  ((115 81) (127 81) (117 81) (125 69) (117 79) (113 77) (121 79))
  ((116 80) (100 72) (110 74) (120 76) (112 74) (120 78) (112 78)))

```

Problem 10 - Blood Pressure Trend Visualizer [Part Two] DEMO

Welcome to [DrRacket](#), version 8.3 [cs].

Language: **racket**, with **debugging**; memory limit: 128 MB.

```
> ( one-day-visualization '(125 83) )
```



```
> ( one-day-visualization '(125 78) )
```



```
> ( one-day-visualization '(116 87) )
```



```
> ( one-day visualization '(114 75) )
```



one-day: undefined;

cannot reference an identifier before its definition

```
> ( one-day-visualization '(114 75) )
```



```
> ( define bad-week (data-for-one-week 110) )
```

```
> ( define good-week (data-for-one-week 90) )
```

```
> bad-week
```

```
'((139 83) (115 93) (131 87) (121 89) (141 91) (119 97) (131 91))
```

```
> good-week
```

```
'((119 73) (115 59) (119 69) (115 73) (105 71) (97 85) (109 73))
```

```
> ( one-week-visualization bad-week )
```



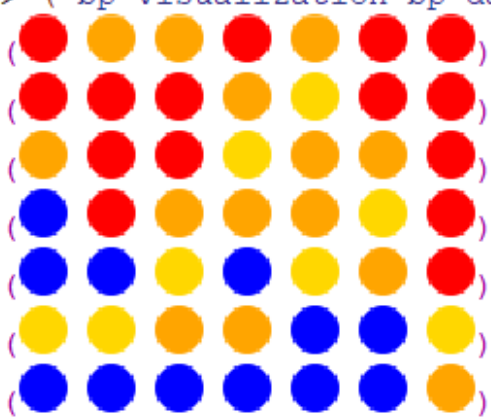
```
> ( one-week-visualization good-week )
```



```

> ( define bp-data ( generate-data '(110 105 102 100 98 95 90) ) )
> ( bp-visualization bp-data )

```



Problem 10 - Blood Pressure Trend Visualizer CODE

```

(require 2htdp/image)
;(Cardio Code)
( define ( sample cardio-index )
  ( + cardio-index ( flip-for-offset ( quotient cardio-index 2 ) ) )
)
;(one Blood Pressure Reading)
( define ( data-for-one-day middle-base)
  ( list
    ( sample ( + middle-base 20 ) )
    ( sample ( - middle-base 20 ) )
  )
)
;One Day BP Data *My Code*
( define ( one-day-visualization bp )
  ( define red (circle 10 "solid" "red"))
  ( define gold (circle 10 "solid" "gold"))
  ( define orange (circle 10 "solid" "orange"))
  ( define blue (circle 10 "solid" "blue"))
  ( cond ((< (car bp) 120)
    ( cond ((< (car(cdr bp)) 80)
      blue
    )
    ( else
      orange
    )
  )
)

```

```

        )
      )
    )
  ( else
    ( cond ((< (car(cdr bp)) 80)
            gold
            )
      ( else
        red
        )
      )
    )
  )
)

;(one week of Blood Pressure Readings)
( define ( data-for-one-week middle-base )
  ( generate-list
    7
    ( lambda () ( data-for-one-day middle-base ) )
  )
)

;One week visualization MINE
( define (one-week-visualization lst )
  (display (map one-day-visualization lst))
  (display "\n")
  )

;(Several Weeks of BPR)
( define ( generate-data base-sequence )
  ( map data-for-one-week base-sequence )
  )

;BP Several Weeks Visualizer
( define (bp-visualization endvis )
  ( map one-week-visualization endvis)
  (display "")
  )

```

*Note: Code used in problem 9 was also referenced