Sam Ghent

CSC344

Assignment 5: RLP and HoFs

Learning abstract: Task 1 is dedicated to defining four simple list generators. Three of these require the use of recursion. One requires a classic application of higher order functions. Task 2 features programs generate number sequences by performing some interesting sorts of "counting." These programs serve to channel one of Tom Johnson's many "automatic composition" techniques. Task 3 affords you an opportunity to get acquainted with "association lists," which are a classic data structure introduced in McCarthy's original Lisp. This task also serves as a segue into Task 4, which pertains to the transformation of number sequences to musical notes represented in ABC notation. Task 5 channels Frank Stella, famous for (among other things) his nested squares. Task 6 simulates a cognitive phenomenon known as chromesthesia, the mapping of musical pitches to colors. Task 7 simulations grapheme to color synesthesia, in which letters are mapped to colors.

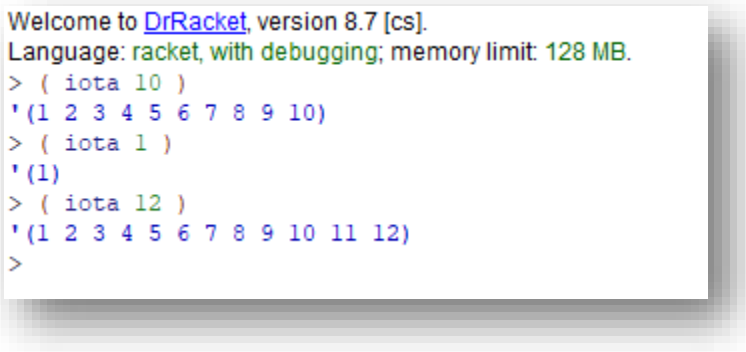
Task 1: Simple List Generator

Task 1a – iota

Function Definition:

```
#lang racket
( define ( iota n )
  ( cond
    ( ( = n 0) '() )
    ( else
      (snoc n (iota (- n 1 ) ) )
      )
    )
  )
```

Demo:

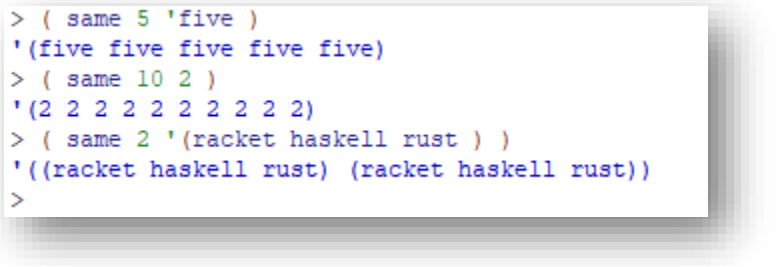


Task 1b – Same:

Function Definition:

```
( define ( same num obj )
  ( cond
    ( ( zero? num ) '() )
    ( else
      ( cons obj ( same ( - num 1 ) obj ) )
      )
    )
  )
```

Demo:



Task 1c – Alternator

Function Definition:

```
( define ( alternator n lst )
  ( cond
    ( ( = n 0 ) '() )
    ( else
      ( cons ( car lst) (alternator ( - n 1 ) (snoc(car lst) (cdr lst))))
      )
    )
  )
```

Demo:

```
> ( alternator 7 '(black white) )
'(black white black white black white black)
> ( alternator 12 '(red yellow blue) )
'(red yellow blue red yellow blue red yellow blue red yellow blue)
> ( alternator 15 '(x y) )
'(x y x y x y x y x y x y x y x)
```

Task 1d – Sequence

Function Definition:

```
( define ( sequence n num )
  ( map ( lambda (x) (* x num ) ) (iota n))
  )

( define ( a-count lst)
  ( cond
    ( ( empty? lst ) '() )
    ( else
      ( append ( iota (car lst)) (a-count (cdr lst)))
            )
     )
    )
```

Demo:

```
> ( sequence 5 20 )
'(20 40 60 80 100)
> ( sequence 10 7 )
'(7 14 21 28 35 42 49 56 63 70)
> ( sequence 8 50 )
'(50 100 150 200 250 300 350 400)
>
```

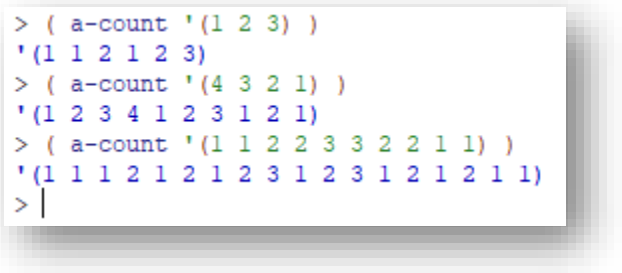
Task 2 – Counting

Task 2a – Accumulation Counting

Function Definition:

```
( define ( a-count lst)
  ( cond
    ( ( empty? lst ) '() )
    ( else
      ( append ( iota (car lst)) (a-count (cdr lst)))
            )
     )
    )
```

Demo:



Task 2b – Repetition Counting

Function Definition:

```
( define ( r-count lst )
  ( cond
    ( ( empty? lst ) '() )
    ( else
      ( append ( same ( car lst ) ( car lst) ) (r-count ( cdr lst ) ) )
      )
    )
```

)

Demo:

```
> ( r-count '(1 2 3) )
'(1 2 2 3 3 3)
> ( r-count '(4 3 2 1) )
'(4 4 4 4 3 3 3 2 2 1)
> ( r-count '(1 1 2 2 3 3 2 2 1 1) )
'(1 1 2 2 2 2 3 3 3 3 3 3 2 2 2 2 1 1)
>
```

Task 2c – Mixed Counting Demo

Demo:

```
> ( a-count '(1 2 3) )
'(1 1 2 1 2 3)
> ( r-count '(1 2 3) )
'(1 2 2 3 3 3)
> ( r-count ( a-count '(1 2 3) ) )
'(1 1 2 2 1 2 2 3 3 3)
> (a-count ( r-count '(1 2 3) ) )
'(1 1 2 1 2 1 2 3 1 2 3 1 2 3)
> ( a-count '(2 2 5 3) )
'(1 2 1 2 1 2 3 4 5 1 2 3)
> ( r-count '(2 2 5 3) )
'(2 2 2 2 5 5 5 5 5 3 3 3)
> ( r-count ( a-count '(2 2 5 3) ) )
'(1 2 2 1 2 2 1 2 2 3 3 3 4 4 4 4 5 5 5 5 5 1 2 2 3 3 3)
> ( a-count ( r-count '(2 2 5 3) ) )
'(1 2 1 2 1 2 1 2 1 2 3 4 5 1 2 3 4 5 1 2 3 4 5 1 2 3 4 5 1 2 3 4 5 1 2 3 1 2 3 1 2 3)
>
```

Task 3 – Association Lists

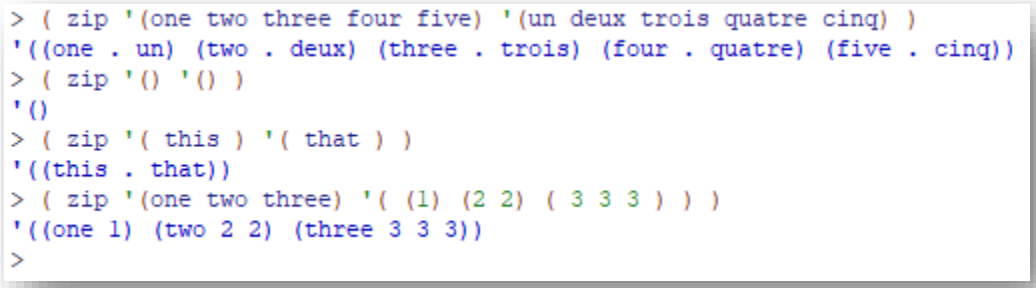
Task 3a – Zip

Function Definition:

```
( define ( zip lst lst2 )
  ( cond
    ( ( empty? lst ) '() )
    ( else
```

```
    ( cons ( list* ( car lst )  ( car lst2 )) ( zip ( cdr lst ) ( cdr lst2 ) ) )
    )
   )
  )
```

Demo:

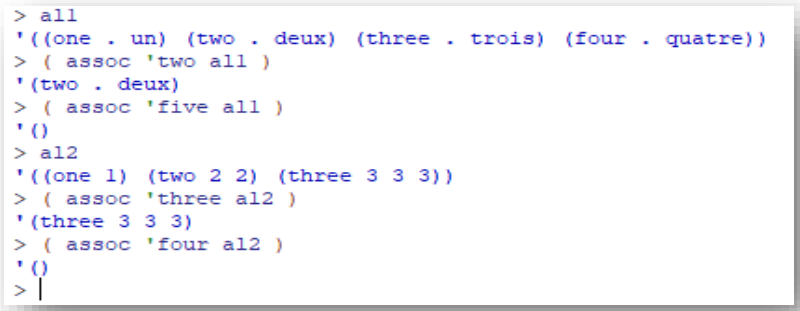


Task 3b – Assoc

Function definition:

```
( define ( assoc obj lst )
  ( cond
    ( ( empty? lst ) '() )
    ( ( eq? obj ( caar lst )) ( car lst ) )
    ( else ( assoc obj ( cdr lst ) ) )
    )
  )
```

Demo:



Task 3c – Establishing some Association Lists

Code:

```
( define al1 ( zip '(one two three four) '(un deux trois quatre ) ) )
( define al2 ( zip '(one two three) '( (1) ( 2 2 ) ( 3 3 3 ) ) ) )


( define scale-zip-CM
  ( zip ( iota 7 ) '("C" "D" "E" "F" "G" "A" "B") )
  )
( define scale-zip-short-Am
  ( zip ( iota 7 ) '("A/2" "B/2" "C/2" "D/2" "E/2" "F/2" "G/2") )
  )
( define scale-zip-short-low-Am
  ( zip ( iota 7 ) '("A,/2" "B,/2" "C,/2" "D,/2" "E,/2" "F,/2" "G,/2") )
  )
( define scale-zip-short-low-blues-Dm
  ( zip ( iota 7 ) '("D,/2" "F,/2" "G,/2" "_A,/2" "A,/2" "c,/2" "d,/2") )
  )
( define scale-zip-wholetone-C
  ( zip ( iota 7 ) '("C" "D" "E" "^F" "^G" "^A" "c") )
  )
```

Demo:

```
> scale-zip-CM
'((1 . "C") (2 . "D") (3 . "E") (4 . "F") (5 . "G") (6 . "A") (7 . "B"))
> scale-zip-short-Am
'((1 . "A/2") (2 . "B/2") (3 . "C/2") (4 . "D/2") (5 . "E/2") (6 . "F/2") (7 . "G/2"))
> scale-zip-short-low-Am
'((1 . "A,/2") (2 . "B,/2") (3 . "C,/2") (4 . "D,/2") (5 . "E,/2") (6 . "F,/2") (7 . "G,/2"))
> scale-zip-short-low-blues-Dm
'((1 . "D,/2") (2 . "F,/2") (3 . "G,/2") (4 . "_A,/2") (5 . "A,/2") (6 . "c,/2") (7 . "d,/2"))
> scale-zip-wholetone-C
'((1 . "C") (2 . "D") (3 . "E") (4 . "^F") (5 . "^G") (6 . "^A") (7 . "c"))
>
```

Task 4 – Numbers to Notes to ABC

Task 4a- nr->note

Function Definition:

```
( define ( nr->note n lst )
  ( define Note ( append ( list-ref lst (- n 1 ) ) ) )
  (cdr Note )
  )
```

Demo:

```
> ( nr->note 1 scale-zip-CM )
"C"
> ( nr->note 1 scale-zip-short-Am )
"A/2"
> ( nr->note 1 scale-zip-short-low-Am )
"A,/2"
> ( nr->note 3 scale-zip-CM )
"E"
> ( nr->note 4 scale-zip-short-Am )
"D/2"
> ( nr->note 5 scale-zip-short-low-Am )
"E,/2"
> ( nr->note 4 scale-zip-short-low-blues-Dm )
"_A,/2"
> ( nr->note 4 scale-zip-wholetone-C )
"^F"
>
```

Task 4b – nrs->notes

Function Definition:

```
( define ( nrs->notes lst lst2 )
( map ( lambda ( n ) ( cdr ( list-ref lst2 ( - n 1 ) ) ) ) lst )
)
```

Demo:

```
> ( nrs->notes '(3 2 3 2 1 1) scale-zip-CM )
'("E" "D" "E" "D" "C" "C")
> ( nrs->notes '(3 2 3 2 1 1) scale-zip-short-Am )
'("C/2" "B/2" "C/2" "B/2" "A/2" "A/2")
> ( nrs->notes ( iota 7 ) scale-zip-CM )
'("C" "D" "E" "F" "G" "A" "B")
> ( nrs->notes ( iota 7 ) scale-zip-short-low-Am )
'("A,/2" "B,/2" "C,/2" "D,/2" "E,/2" "F,/2" "G,/2")
> ( nrs->notes ( a-count '(4 3 2 1) ) scale-zip-CM )
'("C" "D" "E" "F" "C" "D" "E" "C" "D" "C")
> > ( nrs->notes ( r-count '(4 3 2 1) ) scale-zip-CM )
#<procedure:>>
'("F" "F" "F" "F" "E" "E" "E" "D" "D" "C")
> ( nrs->notes ( a-count ( r-count '(1 2 3) ) ) scale-zip-CM )
'("C" "C" "D" "C" "D" "C" "D" "E" "C" "D" "E" "C" "D" "E")
> ( nrs->notes ( r-count ( a-count '(1 2 3) ) ) scale-zip-CM )
'("C" "C" "D" "D" "C" "D" "D" "E" "E" "E")
>
```

Task 4c – nrs->abc

Function Definition:

(define (nrs->abc lst lst2)

(string-join (nrs->notes lst lst2))

)

Demo:

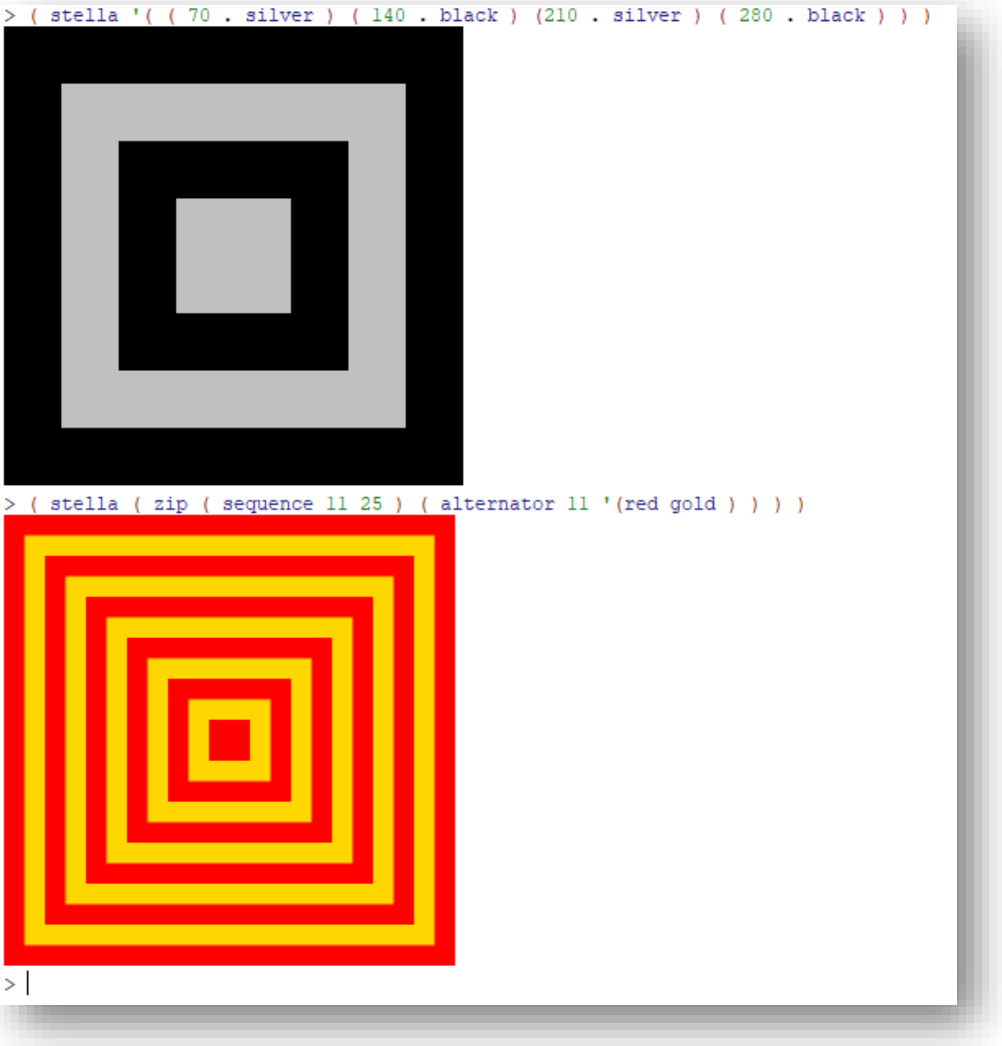
```
> > ( nrs->abc ( iota 7 ) scale-zip-CM )
#<procedure:>>
"C D E F G A B"
> ( nrs->abc ( iota 7 ) scale-zip-short-Am )
"A/2 B/2 C/2 D/2 E/2 F/2 G/2"
> ( nrs->abc ( a-count '( 3 2 1 3 2 1 ) ) scale-zip-CM )
"C D E C D C C D E C D C"
> ( nrs->abc ( r-count '( 3 2 1 3 2 1 ) ) scale-zip-CM )
"E E E D D C E E E D D C"
> ( nrs->abc ( r-count ( a-count '(4 3 2 1) ) ) scale-zip-CM )
"C D D E E E F F F F C D D E E E C D D C"
> ( nrs->abc ( a-count ( r-count '(4 3 2 1) ) ) scale-zip-CM )
"C D E F C D E F C D E F C D E F C D E C D E C D E C D C D C"
>
```

Task 5 – Stella

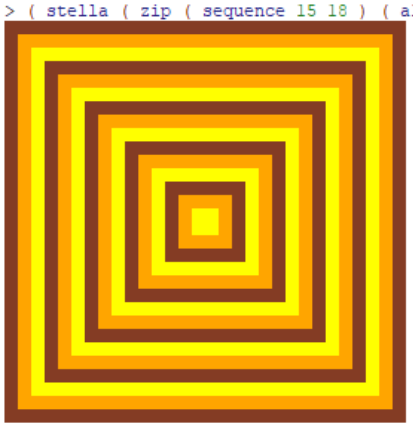
Function Definition:

```
( define ( stella lst )
  ( foldr overlay empty-image
        ( map ( lambda ( x ) ( square ( car x ) "solid" ( cdr x ) ) ) lst )
        )
  )
```

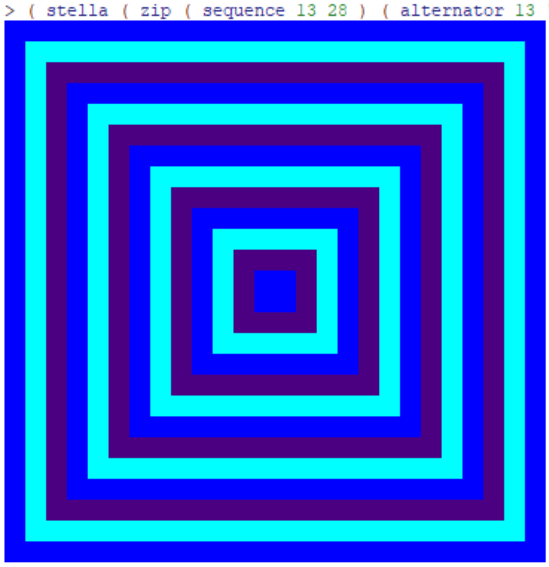
The Five Demos:



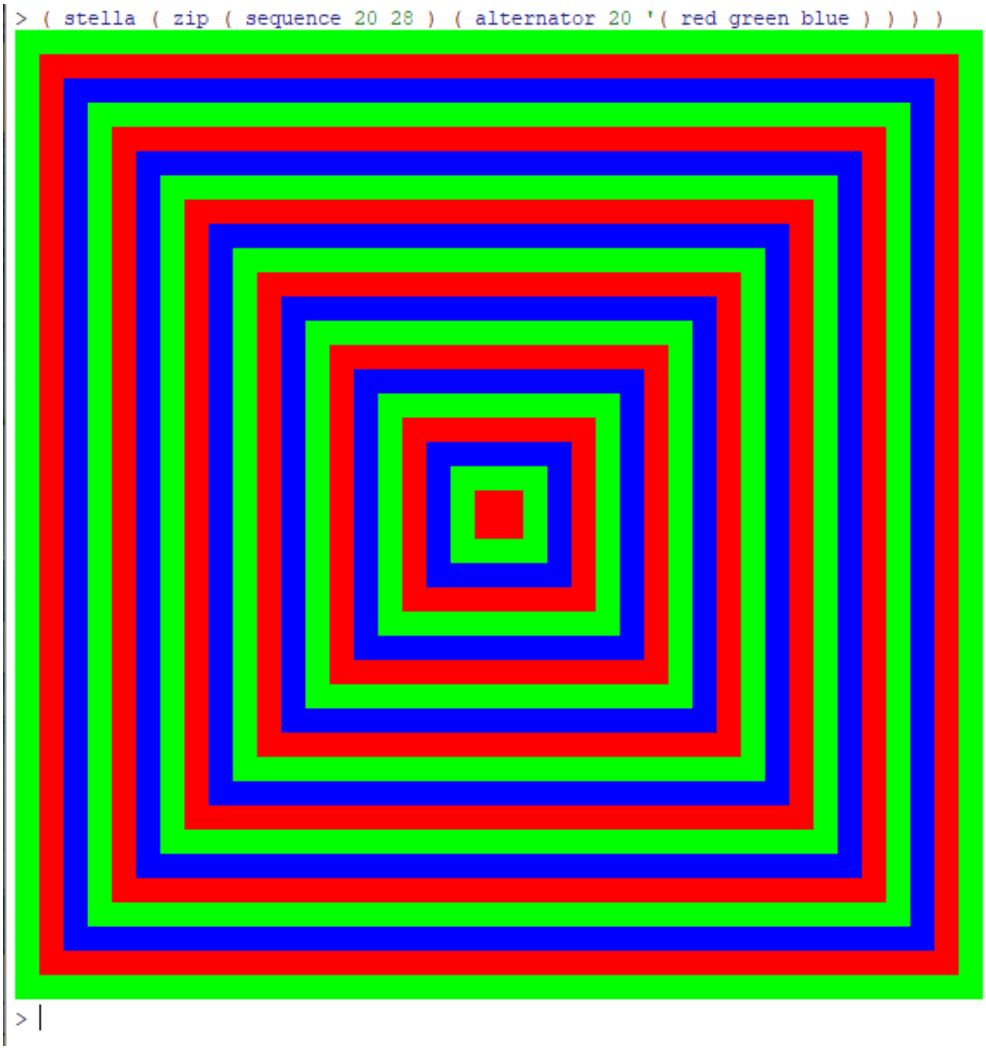
```
> ( stella ( zip ( sequence 15 18 ) ( alternator 15 '( yellow orange brown ) ) ) )
```



```
> ( stella ( zip ( sequence 13 28 ) ( alternator 13 '( blue indigo aqua ) ) ) )
```



```
>
```



Task 6 – Chromesthetic Renderings

Code:

```
( define (play pitchList )
  ( define colorList ( map pc->color pitchList ) )
  ( define boxList ( map color->box colorList ) )
  ( foldr beside empty-image boxList )
  )
```

Demo:

Task 7 – Grapheme to Color Synesthesia

Code:

```
( define AI (text "A" 36 "orange") )
( define BI (text "B" 36 "red") )
( define CI (text "C" 36 "blue") )
( define DI (text "D" 36 "cornsilk") )
( define EI (text "E" 36 "cyan") )
( define FI (text "F" 36 "coral") )
( define GI (text "G" 36 "orangered") )
( define HI (text "H" 36 "orchid") )
( define II (text "I" 36 "palegreen") )
( define JI (text "J" 36 "navy") )
( define KI (text "K" 36 "mistyrose") )
( define LI (text "L" 36 "mintcream") )
( define MI (text "M" 36 "gray") )
( define NI (text "N" 36 "gainsboro") )
( define OI (text "O" 36 "forestgreen") )
( define PI (text "P" 36 "fuchsia") )
( define QI (text "Q" 36 "honeydew") )
( define RI (text "R" 36 "indigo") )
```

(define SI (text "S" 36 "tomato"))

(define TI (text "T" 36 "violet"))

(define UI (text "U" 36 "thistle"))

(define VI (text "V" 36 "yellow"))

(define WI (text "W" 36 "teal"))

(define XI (text "X" 36 "tan"))

(define YI (text "Y" 36 "steelblue"))

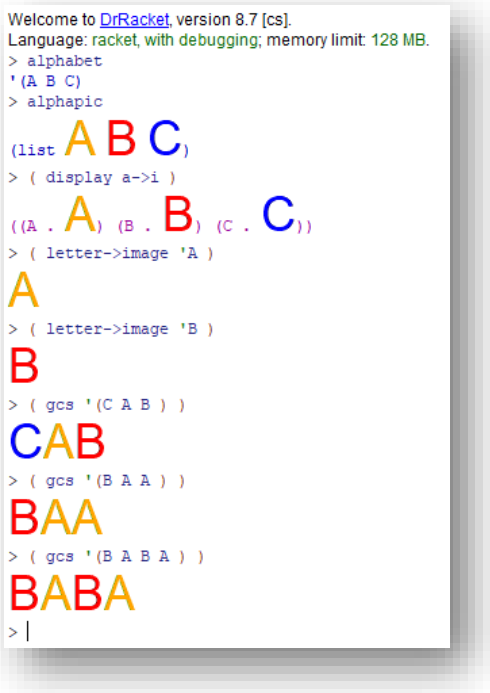
(define ZI (text "Z" 36 "springgreen"))

(define alphabet '(A B C D E F G H I J K L M N O P Q R S T U V W X Y Z))

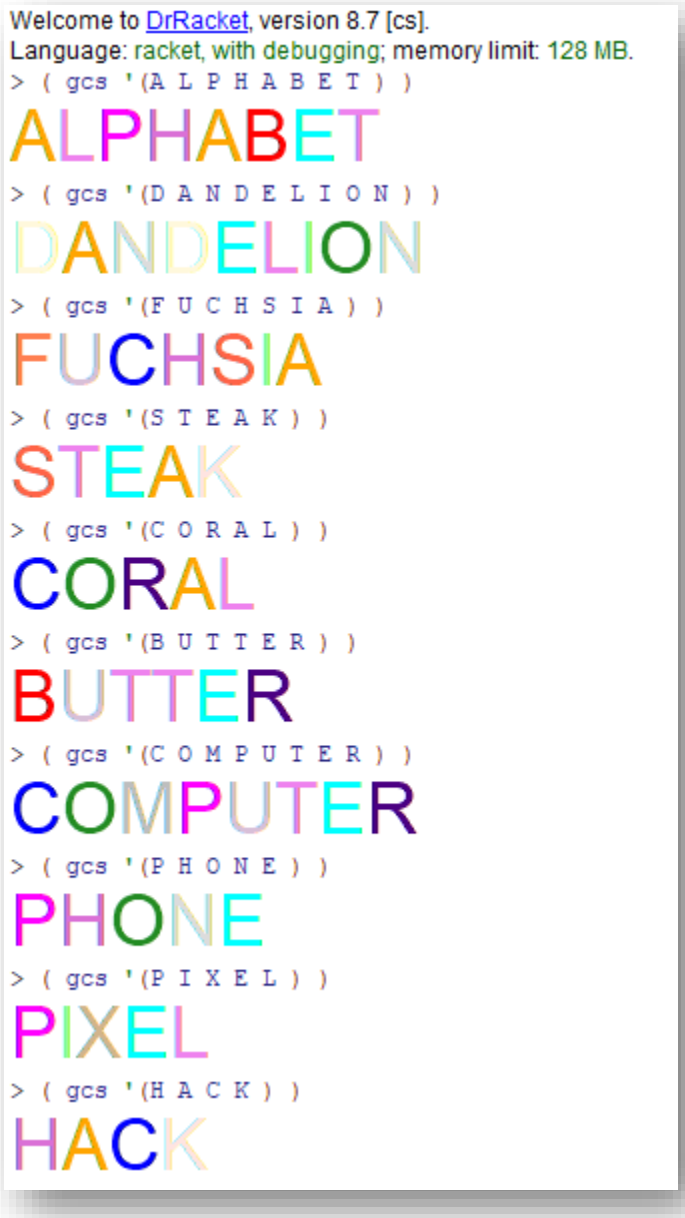
(define alphapic (list AI BI CI DI EI FI GI HI II JI KI LI MI NI OI PI QI RI SI TI UI VI WI XI YI ZI))

(define a->i (zip alphabet alphapic))'

Demo 1:



Demo 2:


Welcome to DrRacket, version 8.7 [cs].
Language: racket, with debugging; memory limit: 128 MB.
> (gcs '(A L P H A B E T))
ALPHABET
> (gcs '(D A N D E L I O N))
DANDELION
> (gcs '(F U C H S I A))
FUCHSIA
> (gcs '(S T E A K))
STEAK
> (gcs '(C O R A L))
CORAL
> (gcs '(B U T T E R))
BUTTER
> (gcs '(C O M P U T E R))
COMPUTER
> (gcs '(P H O N E))
PHONE
> (gcs '(P I X E L))
PIXEL
> (gcs '(H A C K))
HACK