Abstract:

The Racket programming assignment tested our problem-solving skills and taught us about the significance of RLP and HoFs. We gained an appreciation for their usefulness and importance in programming.

Task 1 - Simple List Generators

Task 1a - iota

Code:

```
(define (iota n)
  (if (= n 1)
      '(1)
      (append (iota (- n 1)) (list n))))
```

Demo:

```
> ( iota 10)
'(1 2 3 4 5 6 7 8 9 10)
> (iota 1)
'(1)
> (iota 12)
'(1 2 3 4 5 6 7 8 9 10 11 12)
```

Task 1b - Same

Code:

```
(define (same n obj)
  (if (= n 0)
      '()
      (cons obj (same (- n 1) obj))))
```

Demo:

```
> ( same 5 'five)
'(five five five five five)
> ( same 10 2 )
'(2 2 2 2 2 2 2 2 2 2)
> (same 0 'whatever)
'()
> ( same 2 '(racket prolog haskell rust) )
'((racket prolog haskell rust) (racket prolog haskell rust))
```

Task 1c - Alternator

Code:

```
(define (alternator n lst)
  (if (= n 0)
      '()
      (cons (car lst)
            (alternator (- n 1) (append (cdr lst) (list (car 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 9 '(1 2 3 4 ) )

'(1 2 3 4 1 2 3 4 1)
> ( alternator 15 '( x y ) )
'(x y x y x y x y x y x y x y x)
```

Task 1d - Sequence

Code:

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

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

Code:

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

Demo:

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

Task 2b - Repetition Counting

Code:

```
(define (r-count lst)
  (cond ((null? 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 - Mixing Counting 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) ) )
```

Task 3 : Association Lists

Task 3a:

Code:

```
(define (zip lst1 lst2)
  (cond ((or (null? lst1) (null? lst2)) '())
        (else (cons (cons (car lst1) (car lst2))
                    (zip (cdr lst1) (cdr lst2))))))
```

Demo:

```
> ( zip '(one two three four five) '(un deuz trois quatre cinq) )
'((one . un) (two . deuz) (three . trois) (four . quatre) (five . cinq))
>  ( zip '(one two three four five) '(un deux trois quatre cinq) )
'((one . un) (two . deux) (three . trois) (four . quatre) (five . cinq))
> ( zip '() '() )
'()
> (zip '(this) '(that) )
'((this . that))
> ( zip '( one two three) '( (1) (2 2) (3 3 3) ) )
'((one 1) (two 2 2) (three 3 3 3))
```

Task 3b -Assoc

Code:

```
(define (assoc key alist)
  (cond ((null? alist) '())
        ((eq? key (caar alist)) (car alist))
        (else (assoc key (cdr alist)))))
```

Demo:

```
> (define al1
    (zip '( one two three four) '( un deux trois quatre)
    )
> (define al2
    (zip '(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 3c - Establishing some Association Lists

Code:

```
(define (assoc key alist)
  (cond ((null? alist) '())
        ((eq? key (caar alist)) (car alist))
        (else (assoc key (cdr alist)))))

( 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 - Number to Notes to ABC

Task 4a - nr->note

Code:

```
(define (nr->note nr alist)
  (cdr (assoc nr alist)))
```

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 )
nr->: undefined;
 cannot reference an identifier before its defi
> ( nr->note 4 scale-zip-wholetone-C )
"^F"
```

Task 4b - nrs->notes

Code:

```
(define (nrs->notes nrs alist)
  (map (lambda (nr) (cdr (assoc nr alist))) nrs))
```

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-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 )
'("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 (r-count '(1 2 3) ) ) scale-zip-CM )
'("C" "D" "D" "D" "D" "E" "E" "E" "E" "E" "E" "E" "E" "E")
```

Task 4c - nrs->abc

Code:

```
(define (nrs->abc n ass1)
  (string-join (nrs->notes n ass1 )
               )
  )
```

Demo:

```
> (nrs->abc ( iota 7) scale-zip-CM )
"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 (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 )
nrs->abc: arity mismatch;
 the expected number of arguments does not match the given number
  expected: 2
  given: 0
> (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

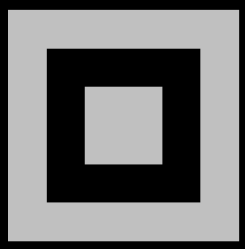
Code:

```
(require 2htdp/image)

(define (stella ass1)
  (foldr overlay empty-image
         (map (lambda (pair) (square (car pair) "solid" (cdr pair) ) )
              ass1 ) ) )
```

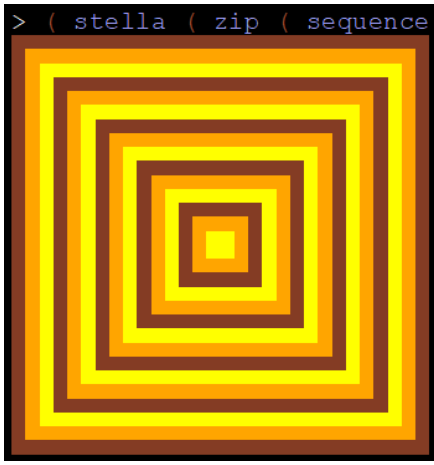
Demo:

```
> (stella '( ( 70 . silver ) (140 . black) (210 . silver) (280 . black) ) )
```



```
> ( stella ( zip ( sequence 11 25 ) (alternator 11 '( red gold  ) ) ) )
```

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



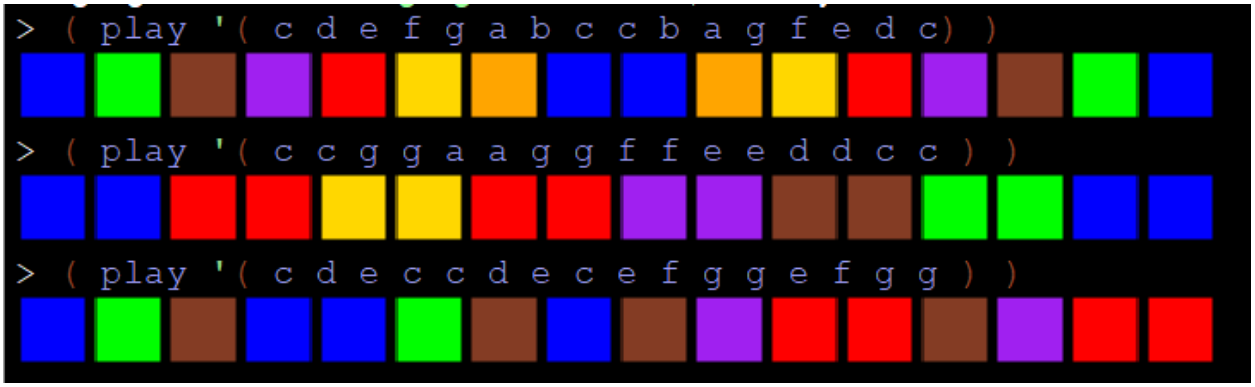
Task 6 - Chromesthetic Renderings

Code:

```
( require 2htdp/image )
( 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 ( a-list list1 list2 )
 ( define x ( length list2 ) )
 ( cond
 ( ( = x 0 )
 '()
 )
 ( else
 ( cons ( cons ( car list1 ) ( car list2 ) ) ( a-list ( cdr
list1 ) ( cdr list2 ) ) )
 )
 )
 )
```

```
( 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 ) )
 )
( define ( play notes )
 ( define colors
 ( map ( lambda (a) ( pc->color a ) ) notes )
 )
 ( define rainbow-squares
 ( map ( lambda (a) ( color->box a ) ) colors )
 )
 ( foldr beside empty-image rainbow-squares )
 )
```

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 7: Grapheme to Color Synesthesia

Code:

```
( require 2htdp/image )
( define AI ( text "A" 36 "orange" ) )
( define BI ( text "B" 36 "red" ) )
( define CI ( text "C" 36 "blue" ) )
( define DI ( text "D" 36 "chocolate" ) )
( define EI ( text "E" 36 "green" ) )
( define FI ( text "F" 36 "indigo" ) )
( define GI ( text "G" 36 "dark gray" ) )
( define HI ( text "H" 36 "yellow" ) )
( define II ( text "I" 36 "violet" ) )
( define JI ( text "J" 36 "steel blue" ) )
( define KI ( text "K" 36 "yellow green" ) )
( define LI ( text "L" 36 "tan" ) )
( define MI ( text "M" 36 "khaki" ) )
( define NI ( text "N" 36 "olive" ) )
( define OI ( text "O" 36 "maroon" ) )
( define PI ( text "P" 36 "sandy brown" ) )
( define QI ( text "Q" 36 "forest green" ) )
( define RI ( text "R" 36 "light blue" ) )
( define SI ( text "S" 36 "dodger blue" ) )
( define TI ( text "T" 36 "cadetblue" ) )
( define UI ( text "U" 36 "goldenrod" ) )
( define VI ( text "V" 36 "orchid" ) )
( define WI ( text "W" 36 "plum" ) )
( define XI ( text "X" 36 "indian red" ) )
( define YI ( text "Y" 36 "aqua" ) )
( define ZI ( text "Z" 36 "sienna" ) )
( 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-list list1 list2 )
 ( define x ( length list2 ) )
 ( cond
 ( ( = x 0 )
```

```
 ( ( = x 0 )
 '()
 )
 ( else
 ( cons ( cons ( car list1 ) ( car list2 ) ) ( a-list ( cdr
list1 ) ( cdr list2 ) ) )
 )
 )
 )
( define ( assoc text any-list )
 ( cond
 ( ( eq? any-list '() )
 '()
 )
 ( ( equal? ( car (car any-list ) ) text )
 ( car any-list )
 )
 ( else
 ( assoc text ( cdr any-list ) )
 )
 )
 )
( define a->i ( a-list alphabet alphapic ) )
( define ( letter->image alphabet) ( cdr ( assoc alphabet a->i ) ) )
( define ( gcs letters )
 ( cond
 ( ( empty? letters ) ( empty-image ) )
 (else
 ( foldr beside empty-image ( map letter->image letters ) ) )
 )
 )
```

Demo:

```
> 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
```

Demo2:

```
> ( gcs '( A L P H A B E T) )
ALPHABET
> (gcs '(D A N D E L I O N ) )
DANDELION
> (gcs '( O S W E G O ) )
OSWEGO
> (gcs '( W A T E R ) )
WATER
> (gcs '( A P P L E ) )
APPLE
> (gcs '( L A P T O P ) )
LAPTOP
> ( gcs '( C O F F E E) )
COFFEE
> ( gcs '( C O D E ) )
CODE
> (gcs '( I S L A N D ) )
ISLAND
> ( gcs '( L O N G ) )
LONG
>
```