Racket Assignment #4: Lambda and Basic Lisp

Learning Abstract

In this assignment, I will learn the fundamentals of Lisp processing and Lambda functions using the Racket programming language. Through practical exercises, I will gain a foundational understanding of these concepts and how to implement them effectively in Racket.

Task 1: Lambda

Demo for Task 1a – Three ascending integers

```
Welcome to DrRacket, version 8.7 [cs].
Language: racket, with debugging; memory limit: 128 MB.
> ( ( lambda ( x ) ( cons x ( cons ( + x 1 ) ( cons ( + x 2 ) '() ) ) ) ) 5 )
'(5 6 7)
> ( ( lambda ( x ) ( cons x ( cons ( + x 1 ) ( cons ( + x 2 ) '() ) ) ) ) 0 )
'(0 1 2)
> ( ( lambda ( x ) ( cons x ( cons ( + x 1 ) ( cons ( + x 2 ) '() ) ) ) ) 108 )
'(108 109 110)
>
```

Demo for Task 1b – Make list in reverse order

```
Welcome to DrRacket, version 8.7 [cs].
Language: racket, with debugging; memory limit: 128 MB.
>  ( ( lambda ( data1 data2 data3 ) ( list data3 data2 data1 ) ) 'red 'yellow 'blue )
'(blue yellow red)
>  ( ( lambda ( data1 data2 data3 ) ( list data3 data2 data1 ) ) 10 20 30 )
'(30 20 10)
>  ( ( lambda ( data1 data2 data3 ) ( list data3 data2 data1 ) ) "Professor Plum" "Colonel Mustard" "Miss Mustard" )
'("Miss Mustard" "Colonel Mustard" "Professor Plum")
>
```

Demo for Task 1c – Random number generator

```
Welcome to DrRacket, version 8.7 [cs].
Language: racket, with debugging; memory limit: 128 MB.
> ( ( lambda ( x y ) ( random x ( + y 1 ) ) ) 3 5 )
5
> ( ( lambda ( x y ) ( random x ( + y 1 ) ) ) 3 5 )
3
> ( ( lambda ( x y ) ( random x ( + y 1 ) ) ) 3 5 )
3
> ( ( lambda ( x y ) ( random x ( + y 1 ) ) ) 3 5 )
4
> ( ( lambda ( x y ) ( random x ( + y 1 ) ) ) 3 5 )
3
> ( ( lambda ( x y ) ( random x ( + y 1 ) ) ) 3 5 )
5
> ( ( lambda ( x y ) ( random x ( + y 1 ) ) ) 3 5 )
4
> ( ( lambda ( x y ) ( random x ( + y 1 ) ) ) 3 5 )
4
> ( ( lambda ( x y ) ( random x ( + y 1 ) ) ) 3 5 )
5
> ( ( lambda ( x y ) ( random x ( + y 1 ) ) ) 3 5 )
4
> ( ( lambda ( x y ) ( random x ( + y 1 ) ) ) 11 17 )
15
> ( ( lambda ( x y ) ( random x ( + y 1 ) ) ) 11 17 )
15
> ( ( lambda ( x y ) ( random x ( + y 1 ) ) ) 11 17 )
12
> ( ( lambda ( x y ) ( random x ( + y 1 ) ) ) 11 17 )
13
> ( ( lambda ( x y ) ( random x ( + y 1 ) ) ) 11 17 )
14
> ( ( lambda ( x y ) ( random x ( + y 1 ) ) ) 11 17 )
14
> ( ( lambda ( x y ) ( random x ( + y 1 ) ) ) 11 17 )
11
> ( ( lambda ( x y ) ( random x ( + y 1 ) ) ) 11 17 )
17
> ( ( lambda ( x y ) ( random x ( + y 1 ) ) ) 11 17 )
12
> ( ( lambda ( x y ) ( random x ( + y 1 ) ) ) 11 17 )
14
>
```

Task 2: List Processing References and Constructors

Demo

```
Welcome to DrRacket, version 8.7 [cs].
Language: racket, with debugging; memory limit: 128 MB.
> ( define colors '(red blue yellow orange ) )
> colors
'(red blue yellow orange)
> 'colors
'colors
> ( quote colors )
'colors
> ( car colors )
'red
> ( cdr colors )
'(blue yellow orange)
> ( car ( cdr colors ) )
'blue
> ( cdr ( cdr colors ) )
'(yellow orange)
> ( cadr colors )
'blue
> ( cddr colors )
'(yellow orange)
> ( first colors )
'red
> ( second colors )
'blue
> ( third colors )
'yellow
> ( list-ref colors 2 )
'yellow
> ( define key-of-c '(c d e) )
> ( define key-of-g '(g a b) )
> ( cons key-of-c key-of-g )
'((c d e) g a b)
```

```
> ( list key-of-c key-of-g )
'((c d e) (g a b))
> ( append key-of-c key-of-g )
'(c d e g a b)
> ( define pitches '(do re mi fa so la ti) )
> ( car ( cdr ( cdr ( cdr pitches ) ) ) )
'fa
> ( cadddr pitches )
'fa
> ( list-ref pitches 3 )
'fa
> ( define a 'alligator )
> ( define b 'pussycat )
> ( define c 'chimpanzee )
> ( cons a ( cons b ( cons c '() ) ) )
'(alligator pussycat chimpanzee)
> ( list a b c )
'(alligator pussycat chimpanzee)
> ( define x '(1 one) )
> ( define y '(2 two) )
> ( cons ( car x ) ( cons ( car ( cdr x ) ) y
'(1 one 2 two)
> ( append x y )
'(1 one 2 two)
>
```

Task 3: The Sampler Program

Code

```
#lang racket
( define ( sampler )
   ( display "(?): " )
   ( define the-list ( read ) )
   ( define the-element
      ( list-ref the-list ( random ( length the-list ) ) )
   )
   ( display the-element ) ( display "\n" )
   ( sampler )
)
```

Demo

```
Welcome to DrRacket, version 8.7 [cs].
Language: racket, with debugging; memory limit: 128 MB.
> ( sampler )
(?): ( red orange yellow green blue indigo violet )
orange
(?): ( red orange yellow green blue indigo violet )
orange
(?): ( red orange yellow green blue indigo violet )
blue
(?): ( red orange yellow green blue indigo violet )
green
(?): ( red orange yellow green blue indigo violet )
red
(?): ( red orange yellow green blue indigo violet )
red
(?): ( aet ate eat eta tae tea )
tae
(?): ( aet ate eat eta tae tea )
eat
(?): ( aet ate eat eta tae tea )
eta
(?): ( aet ate eat eta tae tea )
aet
(?): ( aet ate eat eta tae tea )
eta
(?): ( aet ate eat eta tae tea )
tae
(?): ( 0 1 2 3 4 5 6 7 8 9 )
9
(?): ( 0 1 2 3 4 5 6 7 8 9 )
0
(?): ( 0 1 2 3 4 5 6 7 8 9 )
6
(?): ( 0 1 2 3 4 5 6 7 8 9 )
5
(?): ( 0 1 2 3 4 5 6 7 8 9 )
8
(?): ( 0 1 2 3 4 5 6 7 8 9 )
5
```

Task 4: Playing Cards

Code

```
#lang racket

( define ( ranks rank )
   ( list
     ( list rank 'C )
     ( list rank 'D )
     ( list rank 'H )
     ( list rank 'S )
   )
)

( define ( deck )
   ( append
     ( ranks 2 )
     ( ranks 3 )
     ( ranks 4 )
     ( ranks 5 )
     ( ranks 6 )
     ( ranks 7 )
     ( ranks 8 )
     ( ranks 9 )
     ( ranks 'X )
     ( ranks 'J )
     ( ranks 'Q )
     ( ranks 'K )
     ( ranks 'A )
   )
)

( define ( pick-a-card )
   ( define cards ( deck ) )
   ( list-ref cards ( random ( length cards ) ) )
)

( define ( show card )
   ( display ( rank card ) )
   ( display ( suit card ) )
)

( define ( rank card )
   ( car card )
)

( define ( suit card )
   ( cadr card )
)
```

```
( define ( red? card )
    ( or
      ( equal? ( suit card ) 'D )
      ( equal? ( suit card ) 'H )
    )
)

( define ( black? card )
    ( not ( red? card ) )
)

( define ( aces? card1 card2 )
    ( and
      ( equal? ( rank card1 ) 'A )
      ( equal? ( rank card2 ) 'A )
    )
)
```

Demo