
Racket Programming Assignment #3: Lambda and Basic Lisp

Learning Abstract

This assignment focuses on Lisp list processing by way of Racket and exploring and implementing Lambda functions. We also continue our practice in implementing and defining recursive functions throughout this exercise.

Task 1: Lambda

Task 1a - Three ascending integers

```
> ( ( 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)  
>
```

Task 1b - Make list in reverse order

```
> ( ( lambda (x y z) ( cons z ( cons y ( cons x '() ) ) ) ) 'red 'yellow 'blue )  
'(blue yellow red)  
> ( ( lambda (x y z) ( cons z ( cons y ( cons x '() ) ) ) ) 10 20 30 )  
'(30 20 10)  
> ( ( lambda (x y z) ( cons z ( cons y ( cons x '() ) ) ) ) "Professor Plum" "Colonel Mustard" "Miss Scarlet" )  
'("Miss Scarlet" "Colonel Mustard" "Professor Plum")  
> |
```

Task 1c – Random number generator

```
> ( ( 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 )
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 )
5
> ( ( lambda (x y) ( random x ( + y 1 ) ) ) 3 5 )
5
> ( ( lambda (x y) ( random x ( + y 1 ) ) ) 3 5 )
5
> ( ( 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 )
11
> ( ( lambda (x y) ( random x ( + y 1 ) ) ) 11 17 )
14
> ( ( lambda (x y) ( random x ( + y 1 ) ) ) 11 17 )
17
> ( ( lambda (x y) ( random x ( + y 1 ) ) ) 11 17 )
17
> ( ( lambda (x y) ( random x ( + y 1 ) ) ) 11 17 )
11
> ( ( lambda (x y) ( random x ( + y 1 ) ) ) 11 17 )
14
>
```



Task 2: List Processing Referencers and Constructors

```
> ( 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
> ( caddr 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 animals ) ) ) )
  animals: undefined;
cannot reference an identifier before its definition
> ( caddr 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: Little Color Interpreter

Task 3a – Establishing the Sampler code from Lesson 6

```
sampler.rkt (define ...)
1 | #lang racket
2 | ( define ( sampler )
3 |   ( display "(?): " )
4 |   ( define the-list ( read ) )
5 |   ( define the-element
6 |     ( list-ref the-list ( random ( length the-list ) ) )
7 |   )
8 |   ( display the-element ) ( display "\n" )
9 |   ( sampler )
10 | )
```

Welcome to [DrRacket](#), version 8.6 [cs].
Language: racket, with debugging; memory limit: 128 MB.

```
> ( sampler)
(?) : ( red orange yellow green blue indigo violet )
green
(?) : ( red orange yellow green blue indigo violet )
green
(?) : ( red orange yellow green blue indigo violet )
indigo
(?) : ( red orange yellow green blue indigo violet )
red
(?) : ( red orange yellow green blue indigo violet )
red
(?) : ( red orange yellow green blue indigo violet )
green
(?) : ( aet ate eat eta tae tea )
tae
(?) : ( aet ate eat eta tae tea )
tae
(?) : ( aet ate eat eta tae tea )
eat
(?) : ( aet ate eat eta tae tea )
tea
(?) : ( aet ate eat eta tae tea )
ate
(?) : ( aet ate eat eta tae tea )
ate
(?) : ( 0 1 2 3 4 5 6 7 8 9 )
9
(?) : ( 0 1 2 3 4 5 6 7 8 9 )
7
(?) : ( 0 1 2 3 4 5 6 7 8 9 )
9
(?) : ( 0 1 2 3 4 5 6 7 8 9 )
5
(?) : ( 0 1 2 3 4 5 6 7 8 9 )
6
(?) : ( 0 1 2 3 4 5 6 7 8 9 )
3
...
```

Task 3b – Color Thing Interpreter

Code ...

```

#lang racket
( require 2htdp/image )
( define ( color-thing )
  ( display "(?): " )
  ( define userIn ( read ) )
  ( define command ( list-ref userIn 0 ) )
  ( define color-list ( list-ref userIn 1 ) )
  ( define ( print-rect color )
    ( display ( rectangle 500 30 'solid color ) )
    ( display "\n" )
  )
  ( define ( print-list color-list n )
    ( cond
      [( < n ( length color-list ) )
        ( print-rect ( list-ref color-list n ) )
        ( print-list color-list ( + n 1 ) ) ]
    )
  )
  ( cond
    [( eq? command 'random )
      ( print-rect ( list-ref color-list ( random ( length color-list ) ) ) ) ]
    [( eq? command 'all ) ( print-list color-list 0 ) ]
    [ else
      ( print-rect ( list-ref color-list ( - command 1 ) ) )
    ]
  )
  ( color-thing )
)

```

Demo 1...

Welcome to [DrRacket](#), version 8.6 [cs].

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

> (color-thing)

(?): (random (olivedrab dodgerblue indigo plum teal darkorange))



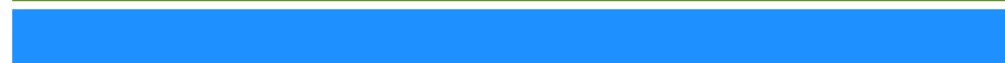
(?): (random (olivedrab dodgerblue indigo plum teal darkorange))



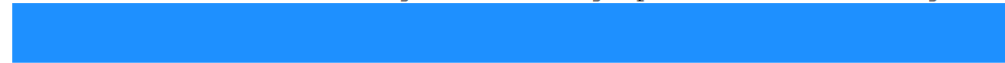
(?): (random (olivedrab dodgerblue indigo plum teal darkorange))



(?): (all (olivedrab dodgerblue indigo plum teal darkorange))



(?): (2 (olivedrab dodgerblue indigo plum teal darkorange))



(?): (3 (olivedrab dodgerblue indigo plum teal darkorange))



(?): (5 (olivedrab dodgerblue indigo plum teal darkorange))



(?):

Welcome to [DrRacket](#), version 8.6 [cs].

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

> (color-thing)

(?): (random (orchid firebrick peru palegreen darkcyan navy))



(?): (random (orchid firebrick peru palegreen darkcyan navy))



(?): (random (orchid firebrick peru palegreen darkcyan navy))



(?): (all (orchid firebrick peru palegreen darkcyan navy))



(?): (2 (orchid firebrick peru palegreen darkcyan navy))



(?): (3 (orchid firebrick peru palegreen darkcyan navy))



(?): (5 (orchid firebrick peru palegreen darkcyan navy))




(?):

Task 4: Two Card Poker

Task 4a – Establishing the Card code from Lesson 6

Demo ...

```
> ( define c1 '( 7 C ))
> ( define c2 '( Q H ))
> c1
'(7 C)
> c2
'(Q H)
> ( rank c1 )
7
> ( suit c1 )
'C
> ( rank c2 )
'Q
> ( rank c2 )
'Q
> ( suit c2 )
'H
> ( red? c1 )
#f
> ( red? c2 )
#t
> ( black? c1 )
#t
> ( black? c2 )
#f
> ( aces '( A C ) '( A S ))
 aces: undefined;
cannot reference an identifier before its definition
> ( aces? '( A C ) '( A S ))
#t
> ( aces? '( K S ) '( A C ))
#f
> ( ranks 4 )
'((4 C) (4 D) (4 H) (4 S))
> ( ranks 'K )
'((K C) (K D) (K H) (K S))
> ( length ( deck ) )
52
> ( display ( deck ) )
( (2 C) (2 D) (2 H) (2 S) (3 C) (3 D) (3 H) (3 S) (4 C) (4 D) (4 H) (4 S) (5 C) (5 D) (5 H) (5 S) (6
C) (6 D) (6 H) (6 S) (7 C) (7 D) (7 H) (7 S) (8 C) (8 D) (8 H) (8 S) (9 C) (9 D) (9 H) (9 S) (X C)
(X D) (X H) (X S) (J C) (J D) (J H) (J S) (Q C) (Q D) (Q H) (Q S) (K C) (K D) (K H) (K S) (A C) (A
D) (A H) (A S))
> ( pick-a-card )

'(9 S)
> ( pick-a-card )
'(A C)
> ( pick-a-card )
'(9 C)
> ( pick-a-card )
'(J D)
> ( pick-a-card )
'(A C)
> ( pick-a-card )
'(5 D)
>
```

Task 4b – Two Card Poker Classifier, IR Version

Pick two cards Demo ...

```
Welcome to DrRacket, version 8.6 [cs].
Language: racket, with debugging; memory limit: 128 MB.
> ( pick-two-cards )
'((K H) (3 C))
> ( pick-two-cards )
'((7 S) (Q D))
> ( pick-two-cards )
'((K S) (4 H))
> ( pick-two-cards )
'((X D) (9 H))
> ( pick-two-cards )
'((4 S) (X C))
```

Higher rank Demo ...

```
Welcome to DrRacket, version 8.6 [cs].
Language: racket, with debugging; memory limit: 128 MB.
> ( higher-rank ( pick-a-card ) ( pick-a-card ) )
>(higher-rank '(3 C) '(2 H))
<3
3
> ( higher-rank ( pick-a-card ) ( pick-a-card ) )
>(higher-rank '(X H) '(3 S))
<'X
'X
> ( higher-rank ( pick-a-card ) ( pick-a-card ) )
>(higher-rank '(K H) '(Q H))
<'K
'K
> ( higher-rank ( pick-a-card ) ( pick-a-card ) )
>(higher-rank '(2 C) '(J C))
<'J
'J
> ( higher-rank ( pick-a-card ) ( pick-a-card ) )
>(higher-rank '(A S) '(K C))
<'A
'A
```

UR Classifier Demo ...

```
Welcome to DrRacket, version 8.6 [cs].
Language: racket, with debugging; memory limit: 128 MB.
> ( classify-two-cards-ur ( pick-two-cards ) )
((K D) (2 S)): K high
> ( classify-two-cards-ur ( pick-two-cards ) )
((5 H) (4 C)): 5 high straight
> ( classify-two-cards-ur ( pick-two-cards ) )
((A H) (8 D)): A high
> ( classify-two-cards-ur ( pick-two-cards ) )
((A H) (3 D)): A high
> ( classify-two-cards-ur ( pick-two-cards ) )
((2 H) (5 H)): 5 high flush
> ( classify-two-cards-ur ( pick-two-cards ) )
((K S) (9 C)): K high
> ( classify-two-cards-ur ( pick-two-cards ) )
((A D) (Q C)): A high
> ( classify-two-cards-ur ( pick-two-cards ) )
((7 D) (X S)): X high
> ( classify-two-cards-ur ( pick-two-cards ) )
((A C) (7 D)): A high
> ( classify-two-cards-ur ( pick-two-cards ) )
((X S) (6 S)): X high flush
> ( classify-two-cards-ur ( pick-two-cards ) )
((7 S) (8 C)): 8 high straight
> ( classify-two-cards-ur ( pick-two-cards ) )
((6 C) (6 S)): Pair of 6's
> ( classify-two-cards-ur ( pick-two-cards ) )
((8 C) (6 D)): 8 high
> ( classify-two-cards-ur ( pick-two-cards ) )
((3 D) (3 H)): Pair of 3's
> ( classify-two-cards-ur ( pick-two-cards ) )
((4 S) (A D)): A high
> ( classify-two-cards-ur ( pick-two-cards ) )
((5 C) (4 D)): 5 high straight
> ( classify-two-cards-ur ( pick-two-cards ) )
((7 C) (6 D)): 7 high straight
> ( classify-two-cards-ur ( pick-two-cards ) )
((9 S) (7 C)): 9 high
> ( classify-two-cards-ur ( pick-two-cards ) )
((Q D) (A S)): A high
> ( classify-two-cards-ur ( pick-two-cards ) )
((X H) (Q C)): Q high
>
```

Code ...

```

58 ( define ( pick-two-cards )
59   (define c1 ( pick-a-card ) )
60   (define c2 ( pick-a-card ) )
61   ( cond
62     [( and ( equal? (rank c1)(rank c2) ) ( equal? (suit c1)(suit c2) ) ) ( pick-two-cards ) ]
63     [ else ( list c1 c2 ) ]
64   )
65 )
66 ( define ( card-value c )
67   ( define r ( rank c ) )
68   ( cond
69     [( equal? r 'X )10]
70     [( equal? r 'J )11]
71     [( equal? r 'Q )12]
72     [( equal? r 'K )13]
73     [( equal? r 'A )14]
74     [else ( + r 0 )]
75   )
76 )
77 ( define ( higher-rank c1 c2 )
78   ( define v1 ( card-value c1 ) )
79   ( define v2 ( card-value c2 ) )
80   ( cond
81     [( > v1 v2 ) ( rank c1 ) ]
82     [( < v1 v2 ) ( rank c2 ) ]
83     ;[ else ( rank c1 ) ]
84   )
85 )
86 ;( trace higher-rank )
87 ( define ( pair? c1 c2 )
88   ( equal?
89     (rank c1)
90     (rank c2)
91   )
92 )
93 ( define ( straight? c1 c2 )
94   ( or
95     ( = ( + (card-value c1) 1)(card-value c2))
96     ( = ( - (card-value c1) 1)(card-value c2))
97   )
98 )
99 ( define ( flush? c1 c2 )
100   ( equal?
101     (suit c1)
102     (suit c2)
103   )
104 )

105 ( define ( classify-two-cards-ur hand )
106   ( define c1 ( list-ref hand 0 ) )
107   ( define c2 ( list-ref hand 1 ) )
108   ( display hand ) ( display ": " )
109   ( cond
110     [(pair? c1 c2)(display "Pair of ")(display (rank c1))(display "'s")]
111     [else
112       (display (higher-rank c1 c2))(display " high ")
113       ( cond [(straight? c1 c2)(display "straight ")] )
114       ( cond [(flush? c1 c2)(display "flush ")] )
115     ]
116   )
117 )

```

Task 4c – Two Card Poker Classifier

Demo ...

```
> ( classify-two-cards ( pick-two-cards ) )
((4 D) (J D)): Jack high flush
> ( classify-two-cards ( pick-two-cards ) )
((9 D) (7 S)): Nine high
> ( classify-two-cards ( pick-two-cards ) )
((2 S) (6 D)): Six high
> ( classify-two-cards ( pick-two-cards ) )
((J S) (6 S)): Jack high flush
> ( classify-two-cards ( pick-two-cards ) )
((2 D) (J D)): Jack high flush
> ( classify-two-cards ( pick-two-cards ) )
((5 D) (6 C)): Six high straight
> ( classify-two-cards ( pick-two-cards ) )
((7 S) (2 C)): Seven high
> ( classify-two-cards ( pick-two-cards ) )
((Q C) (J D)): Queen high straight
> ( classify-two-cards ( pick-two-cards ) )
((K S) (2 S)): King high flush
> ( classify-two-cards ( pick-two-cards ) )
((2 H) (4 H)): Four high flush
> ( classify-two-cards ( pick-two-cards ) )
((A H) (2 D)): Ace high
> ( classify-two-cards ( pick-two-cards ) )
((6 S) (9 H)): Nine high
> ( classify-two-cards ( pick-two-cards ) )
((7 D) (K C)): King high
> ( classify-two-cards ( pick-two-cards ) )
((5 H) (2 H)): Five high flush
> ( classify-two-cards ( pick-two-cards ) )
((3 D) (2 C)): Three high straight
> ( classify-two-cards ( pick-two-cards ) )
((2 D) (2 H)): Pair of Twos
> ( classify-two-cards ( pick-two-cards ) )
((2 C) (Q H)): Queen high
> ( classify-two-cards ( pick-two-cards ) )
((5 D) (6 H)): Six high straight
> ( classify-two-cards ( pick-two-cards ) )
((A S) (2 D)): Ace high
> ( classify-two-cards ( pick-two-cards ) )
((9 D) (3 C)): Nine high
```

Code ...

```

119 ( define (translate n)
120   (cond [(equal? n 2) 'Two]
121         [(equal? n 3) 'Three]
122         [(equal? n 4) 'Four]
123         [(equal? n 5) 'Five]
124         [(equal? n 6) 'Six]
125         [(equal? n 7) 'Seven]
126         [(equal? n 8) 'Eight]
127         [(equal? n 9) 'Nine]
128         [(equal? n 'X) 'Ten]
129         [(equal? n 'J) 'Jack]
130         [(equal? n 'Q) 'Queen]
131         [(equal? n 'K) 'King]
132         [(equal? n 'A) 'Ace]
133   )
134 )
135 ( define ( classify-two-cards hand )
136   ( define c1 ( list-ref hand 0 ) )
137   ( define c2 ( list-ref hand 1 ) )
138   ( display hand ) ( display ": " )
139   ( cond
140     [(pair? c1 c2)(display "Pair of ")(display ( translate (rank c1)))(display "s")]
141     [else
142       (display ( translate (higher-rank c1 c2)))(display " high ")
143       ( cond [(straight? c1 c2)(display "straight ")] )
144       ( cond [(flush? c1 c2)(display "flush ")] )
145     ]
146   )
147 )

```