Racket Programming Assignment #3: Lambda and Basic Lisp

Learning Abstract

For this assignment, there are four tasks that are based on Lesson 6 “Basic Lisp Programming”. The first task involve using lambda functions which is highly constrained. The second task is to create a demo that is presented in the section titled “Redacted Racket Session Featuring Referencers and Constructors” in Lesson 6. The third task is to not only creating and generating a demo based on the “sampler” program in the penultimate section of Lesson 6 but also changing and extending the program. The final task is to not only establishing and generating a demo based on the playing card code presented in Lesson 6 but also creating more card-based functions such as classifying two cards, picking two cards and higher rank.

Task 1 - Lambda

Task 1a - Three ascending integers

```
Welcome to DrRacket, version 8.6 [cs].
Language: racket, with debugging; memory limit: 256 MB.
> ((lambda (n) ( cons n ( cons ( + n 1 ) ( cons ( + n 2) '()))))5)
'(5 6 7)
> ((lambda (n) ( cons n ( cons ( + n 1 ) ( cons ( + n 2) '()))))0)
'(0 1 2)
> ((lambda (n) ( cons n ( cons ( + n 1 ) ( cons ( + n 2) '()))))108)
'(108 109 110)
```

Task 1b - Make list in reverse order

```
Welcome to DrRacket, version 8.6 [cs].
Language: racket, with debugging; memory limit: 256 MB.
> (( 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

```
Welcome to DrRacket, version 8.6 [cs].
Language: racket, with debugging; memory limit: 256 MB.
> ( (lambda ( x y) ( random x y) (random x (+ y 1)) ) 3 5)
5
> ( (lambda ( x y) ( random x y) (random x (+ y 1)) ) 3 5)
4
> ( (lambda ( x y) ( random x y) (random x (+ y 1)) ) 3 5)
4
> ( (lambda ( x y) ( random x y) (random x (+ y 1)) ) 3 5)
3
> ( (lambda ( x y) ( random x y) (random x (+ y 1)) ) 3 5)
3
> ( (lambda ( x y) ( random x y) (random x (+ y 1)) ) 3 5)
5
> ( (lambda ( x y) ( random x y) (random x (+ y 1)) ) 3 5)
4
> ( (lambda ( x y) ( random x y) (random x (+ y 1)) ) 3 5)
4
> ( (lambda ( x y) ( random x y) (random x (+ y 1)) ) 3 5)
5
> ( (lambda ( x y) ( random x y) (random x (+ y 1)) ) 3 5)
3
```

```
Welcome to DrRacket, version 8.6 [cs].
Language: racket, with debugging; memory limit: 256 MB.
> ( (lambda ( x y) ( random x y) (random x (+ y 1)) ) 11 17)
15
> ( (lambda ( x y) ( random x y) (random x (+ y 1)) ) 11 17)
17
> ( (lambda ( x y) ( random x y) (random x (+ y 1)) ) 11 17)
12
> ( (lambda ( x y) ( random x y) (random x (+ y 1)) ) 11 17)
16
> ( (lambda ( x y) ( random x y) (random x (+ y 1)) ) 11 17)
16
> ( (lambda ( x y) ( random x y) (random x (+ y 1)) ) 11 17)
14
> ( (lambda ( x y) ( random x y) (random x (+ y 1)) ) 11 17)
15
> ( (lambda ( x y) ( random x y) (random x (+ y 1)) ) 11 17)
15
> ( (lambda ( x y) ( random x y) (random x (+ y 1)) ) 11 17)
12
> ( (lambda ( x y) ( random x y) (random x (+ y 1)) ) 11 17)
14
```

Task 2 - List Processing Referencers and Constructors

```
Welcome to DrRacket, version 8.6 [cs].
Language: racket, with debugging; memory limit: 256 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))
> (define animals '(lion zebra hyena elephant))
> (car (cdr (cdr (cdr animals))))
'elephant
> (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 - Little Color Interpreter

Task 3a - Sampler Code from Lesson 6

```
#lang racket
(require 2htdp/image)

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

Task 3a - Demo of Sampler Code from Lesson 6

```
Welcome to DrRacket, version 8.6 [cs].
Language: racket, with debugging; memory limit: 256 MB.
> (sampler)
(?): (red orange yellow green blue indigo violet)
indigo
(?): (red orange yellow green blue indigo violet)
indigo
(?): (red orange yellow green blue indigo violet)
indigo
(?): (red orange yellow green blue indigo violet)
yellow
(?): (red orange yellow green blue indigo violet)
orange
(?): (red orange yellow green blue indigo violet)
green
(?): (aet ate eat eta tae tea)
eta
(?): (aet ate eat eta tae tea)
ate
(?): (aet ate eat eta tae tea)
aet
(?): (aet ate eat eta tae tea)
tea
(?): (aet ate eat eta tae tea)
aet
(?): (aet ate eat eta tae tea)
aet
(?): (0 1 2 3 4 5 6 7 8 9)
2
(?): (0 1 2 3 4 5 6 7 8 9)
1
(?): (0 1 2 3 4 5 6 7 8 9)
6
(?): (0 1 2 3 4 5 6 7 8 9)
2
(?): (0 1 2 3 4 5 6 7 8 9)
2
(?): (0 1 2 3 4 5 6 7 8 9)
2
```

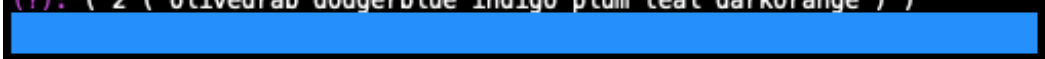
Task 3b - Color-thing Code

```
#lang racket
(require 2htdp/image)

(define (color-thing )
  (display "(?): " )
  (define the-list ( read ))
  (define shade (list-ref the-list 1))
  (define rainbows (list-ref (list-ref the-list 1) (random (length (list-ref the-list 1)))))
  (define (color-shape color) (rectangle 500 20 'solid color))
  (define space (square 5 "solid" "transparent"))
  (define (color-shapes  shade x )
    (display (color-shape (list-ref shade x)))
    (display "\n")
    (cond
      ((< x (-(length shade)1))
      (color-shapes shade ( + x 1) )
      )
    )
    )
  (define the-element
   (list-ref the-list (random (length the-list)))
    )
  (cond
    (( eq? (list-ref the-list 0) 'random)
    (display (color-shape))
     )
    ((eq? (list-ref the-list 0) 'all)
     (color-shapes shade 0)

    )
    (else
     (display (color-shape(list-ref shade ( - (list-ref the-list 0) 1))))
     )
    )
  (display "\n")
  (color-thing)
  )
```

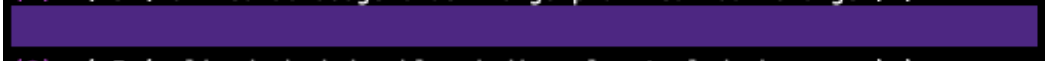
Task 3b - Two Demos of Color-thing Code

```
Welcome to DrRacket, version 8.6 [cs].
Language: racket, with debugging; memory limit: 256 MB.
> (color-thing)
(?): ( all ( olivedrab dodgerblue indigo plum teal darkorange ) )
```

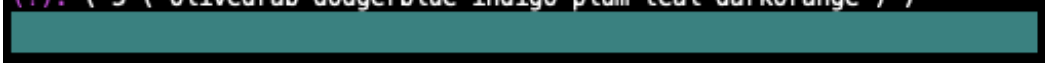
```
(?): ( 2 ( olivedrab dodgerblue indigo plum teal darkorange ) )
```



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



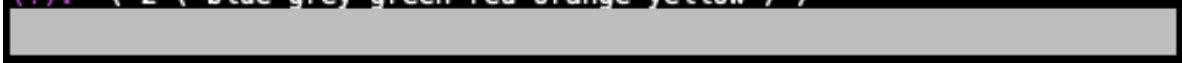
```
(?): ( 5 ( olivedrab dodgerblue indigo plum teal darkorange ) )
```



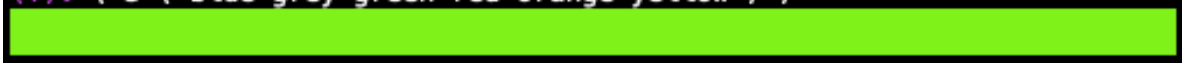
Welcome to DrRacket, version 8.6 [cs].
Language: racket, with debugging; memory limit: 256 MB.

```
> (color-thing)
(?): ( all ( blue grey green red orange yellow ) )
```

```
(?):  ( 2 ( blue grey green red orange yellow ) )
```



```
(?): ( 3 ( blue grey green red orange yellow ) )
```



```
(?): ( 5 ( blue grey green red orange yellow ) )
```

Task 4 - Two Card Poker

Task 4a - Playing Cards Code from Lesson 6

```
#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 )
   )
)
```

Task 4a - Demo of Playing Cards Code from Lesson 6

```
Welcome to DrRacket, version 8.6 [cs].
Language: racket, with debugging; memory limit: 256 MB.
> ( define c1 '( 7 C ) )
> ( define c2 '( Q H ) )
> c1
'(7 C)
> c2
'(Q H)
> ( rank c1 )
7
> ( suit c1 )
'C
> ( rank c2 )
'Q
> ( suit c2 )
'H
> ( red? c1 )
#f
> ( red? c2 )
#t
> ( black? c1 )
#t
> ( black? c2 )
#f
> ( 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 )
'(Q C)
> ( pick-a-card )
'(6 S)
> ( pick-a-card )
'(J C)
> ( pick-a-card )
'(5 C)
> ( pick-a-card )
'(3 H)
> ( pick-a-card )
'(5 H)
```

Task 4b - Two Cards Poker Classifier Code

```
#lang racket

(require racket/trace)

(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 )
   )
)

( define ( pick-two-cards )
   (define  cards (deck))
   (define c1 (pick-a-card))
   (define c2 (pick-a-card))
   (cond
     (( eq? c1 c2)
      (pick-two-cards)
      )
     )
   (list c1 c2)
   )
```

```
(define ( position rank)
 (cond
  ( ( equal? rank 2 ) 1 )
  ( ( equal? rank 3 ) 2 )
  ( ( equal? rank 4 ) 3 )
  ( ( equal? rank 5 ) 4 )
  ( ( equal? rank 6 ) 5 )
  ( ( equal? rank 7 ) 6 )
  ( ( equal? rank 8 ) 7 )
  ( ( equal? rank 9 ) 8 )
  ( ( equal? rank 'X ) 9 )
  ( ( equal? rank 'J ) 10 )
  ( ( equal? rank 'Q ) 11 )
  ( ( equal? rank 'K ) 12 )
  ( ( equal? rank 'A ) 13 )
 )
)



(define ( higher-rank c1 c2)
  ( define rank-c1 ( position ( list-ref c1 0 )))
  ( define rank-c2 ( position ( list-ref c2 0 )))
    ( cond
     ( ( > rank-c1 rank-c2 ) ( list-ref c1 0 ))
   ( else
    ( list-ref c2 0 )
    )
  )
)

; ( trace higher-rank )


( define ( classify-two-cards-ur card )
   ( display card ) ( display ": " )
   ( define c1 ( first card ) )
   ( define c2 ( second card ) )

     ( cond
        ( ( equal? ( rank c1 ) ( rank c2 ) )
          ( display ( rank c1 ) ) ( display " pair" )
         )
        (  ( or ( equal? ( rank c1 ) 'A ) ( equal? ( rank c2 ) 'A ))
          ( display ( rank c1 ) ) ( display " high" )
         )

        ( ( or ( equal? ( rank c1 ) 'K ) ( equal? ( rank c2 ) 'K ))
          ( display ( rank c1 ) ) ( display " high" )
         )

        ( ( or ( equal? ( rank c1 ) 'Q ) ( equal? ( rank c2 ) 'Q ))
          ( display ( rank c1 ) ) ( display " high" )
         )
```

```
          ( ( or ( equal? ( rank c1 ) 'J ) ( equal? ( rank c2 ) 'J ))
           ( display ( rank c1 ) ) ( display " high" )
          )
         ( ( or ( equal? ( rank c1 ) 'X ) ( equal? ( rank c2 ) 'X ))
           ( display ( rank c1 ) ) ( display " high" )
          )
        ( else
          ( cond
            ( ( > ( rank c1 ) ( rank c2 ) )
              ( display ( rank c1 ) ) ( display " high" )
             )
            ( ( < ( rank c1 ) ( rank c2 ) )
             ( display ( rank c2 ) ) ( display " high" )
            )
           )
          )
         )

 ( straight? c1 c2 )
 ( flush? c1 c2 )

)

( define ( straight? c1 c2 )
 ( cond
   ( ( equal? ( rank c1 ) 'A )
     ( cond
       ( ( equal? ( rank c2 ) 'K )
       ( display " straight" )
      )
     )
     )
    ( ( equal? ( rank c1 ) 'K )
      ( cond
       ( ( or ( equal? ( rank c2 ) 'A ) ( equal? ( rank c2 ) 'Q ))
         ( display " straight" )
         )
    )
  )
    ( ( equal? ( rank c1 ) 'Q )
         ( cond
           ( ( or ( equal? ( rank c2 ) 'K ) ( equal? ( rank c2 ) 'J ))
         ( display " straight" )
         )
         )
      )
        ( ( equal? ( rank c1 ) 'J )
         ( cond
          ( ( or ( equal? ( rank c2 ) 'Q ) ( equal? ( rank c2 ) 'X ))
         ( display " straight" )
         )
       )
      )
```

```
  ( ( equal? ( rank c1 ) 'X )
    ( cond
      ( ( or ( equal? ( rank c2 ) 'j ) ( equal? ( rank c2 ) 9 ))
     ( display " straight" )
     )
  )
 )
  ( ( equal? ( rank c1 ) 9 )
    ( cond
       ( ( or ( equal? ( rank c2 ) 'X ) ( equal? ( rank c2 ) 8 ))
     ( display " straight" )
     )
     )
  )
  ( ( equal? ( rank c1 ) 8 )
     ( cond
       ( ( or ( equal? ( rank c2 ) 9 ) ( equal? ( rank c2 ) 7 ))
     ( display " straight" )
     )
     )
 )
 ( ( equal? ( rank c1 ) 7 )
    ( cond
      ( ( or ( equal? ( rank c2 ) 8 ) ( equal? ( rank c2 ) 6 ))
     ( display " straight" )
     )
    )
 )
( ( equal? ( rank c1 ) 6 )
  ( cond
     ( ( or ( equal? ( rank c2 ) 7 ) ( equal? ( rank c2 ) 5 ))
     ( display " straight" )
     )
    )
 )
  ( ( equal? ( rank c1 ) 5 )
     ( cond
       ( ( or ( equal? ( rank c2 ) 6 ) ( equal? ( rank c2 ) 4 ))
     ( display " straight" )
     )
     )
  )
 ( ( equal? ( rank c1 ) 4 )
    ( cond
       ( ( or ( equal? ( rank c2 ) 5 ) ( equal? ( rank c2 ) 3 ))
     ( display " straight" )
     )
    )
 )
 ( ( equal? ( rank c1 ) 3 )
    ( cond
     ( ( or ( equal? ( rank c2 ) 4 ) ( equal? ( rank c2 ) 2 ))
     ( display " straight" )
     )
```

```
      )
       ( ( equal? ( rank c1 ) 2 )
         ( cond
           ( ( or ( equal? ( rank c2 ) 3 ) ( equal? ( rank c2 ) 1 ))
            ( display " straight" )
             )

       )
        )
       (( equal? ( rank c1 ) 1 )
       (cond
           ( ( equal? ( rank c2 ) 2 )
                ( display " straight" )
            )
           )
         )
      )
   )

( define ( flush? c1 c2 )
 ( cond
 ( ( equal? ( suit c1 ) ( suit c2 ) )
 ( display " flush" )
 )
 )
)
```

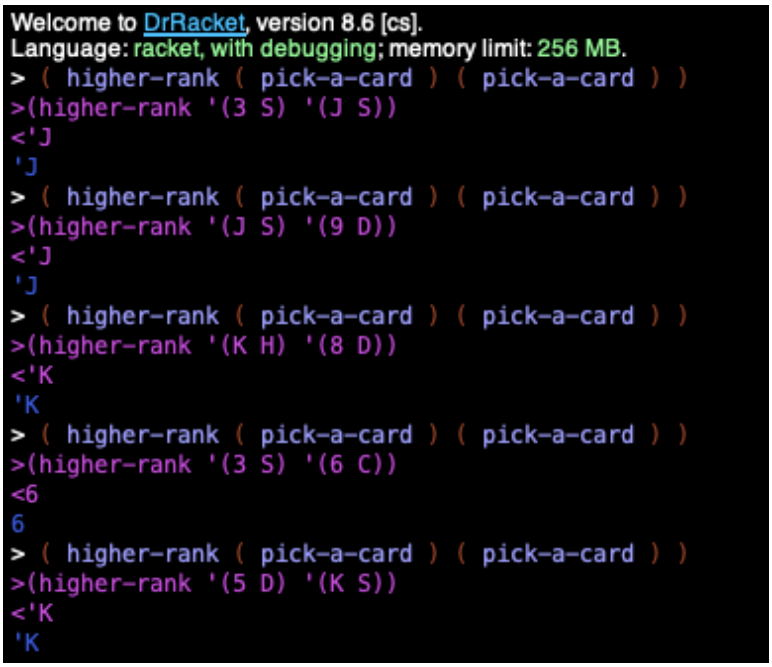
Task 4b - Demo of Two Cards Poker Classifier Code

```
Welcome to DrRacket, version 8.6 [cs].
Language: racket, with debugging; memory limit: 256 MB.
> ( classify-two-cards-ur ( pick-two-cards ) )
((6 H) (2 S)): 6 high
> ( classify-two-cards-ur ( pick-two-cards ) )
((7 D) (3 H)): 7 high
> ( classify-two-cards-ur ( pick-two-cards ) )
((4 H) (6 D)): 6 high
> ( classify-two-cards-ur ( pick-two-cards ) )
((2 C) (7 D)): 7 high
> ( classify-two-cards-ur ( pick-two-cards ) )
((5 C) (8 D)): 8 high
> ( classify-two-cards-ur ( pick-two-cards ) )
((A H) (2 S)): A high
> ( classify-two-cards-ur ( pick-two-cards ) )
((5 H) (Q H)): 5 high flush
> ( classify-two-cards-ur ( pick-two-cards ) )
((J S) (A D)): J high
> ( classify-two-cards-ur ( pick-two-cards ) )
((5 S) (8 C)): 8 high
> ( classify-two-cards-ur ( pick-two-cards ) )
((5 D) (9 S)): 9 high
> ( classify-two-cards-ur ( pick-two-cards ) )
((Q D) (9 C)): Q high
> ( classify-two-cards-ur ( pick-two-cards ) )
((7 C) (7 C)): 7 pair flush
> ( classify-two-cards-ur ( pick-two-cards ) )
((J D) (2 H)): J high
> ( classify-two-cards-ur ( pick-two-cards ) )
((X D) (A D)): X high flush
> ( classify-two-cards-ur ( pick-two-cards ) )
((9 D) (X S)): 9 high straight
> ( classify-two-cards-ur ( pick-two-cards ) )
((8 S) (A S)): 8 high flush
> ( classify-two-cards-ur ( pick-two-cards ) )
((3 D) (7 C)): 7 high
> ( classify-two-cards-ur ( pick-two-cards ) )
((9 D) (9 S)): 9 pair
> ( classify-two-cards-ur ( pick-two-cards ) )
((5 C) (6 D)): 6 high straight
> ( classify-two-cards-ur ( pick-two-cards ) )
((8 C) (9 S)): 9 high straight
```

Task 4b - Pick Two Cards Demo of Two Cards Poker Classifier Code

```
Welcome to DrRacket, version 8.6 [cs].
Language: racket, with debugging; memory limit: 256 MB.
> ( pick-two-cards )
'((5 C) (J H))
> ( pick-two-cards )
'((9 H) (8 S))
> ( pick-two-cards )
'((9 H) (A H))
> ( pick-two-cards )
'((5 S) (Q D))
> ( pick-two-cards )
'((A D) (8 S))
```

Task 4b - Higher Rank Demo of Two Cards Poker Classifier Code



Task 4c - Two Cards Poker Classifier Code (2)

```
#lang racket

(require racket/trace)

(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 )
   )
)

( define ( pick-two-cards )
   (define  cards (deck))
   (define c1 (pick-a-card))
   (define c2 (pick-a-card))
   (cond
     (( eq? c1 c2)
      (pick-two-cards)
      )
     )
   (list c1 c2)
   )
```

```
(define ( position rank)
 (cond
  ( ( equal? rank 2 ) 1 )
  ( ( equal? rank 3 ) 2 )
  ( ( equal? rank 4 ) 3 )
  ( ( equal? rank 5 ) 4 )
  ( ( equal? rank 6 ) 5 )
  ( ( equal? rank 7 ) 6 )
  ( ( equal? rank 8 ) 7 )
  ( ( equal? rank 9 ) 8 )
  ( ( equal? rank 'X ) 9 )
  ( ( equal? rank 'J ) 10 )
  ( ( equal? rank 'Q ) 11 )
  ( ( equal? rank 'K ) 12 )
  ( ( equal? rank 'A ) 13 )
 )
)



(define ( higher-rank c1 c2)
  ( define rank-c1 ( position ( list-ref c1 0 )))
  ( define rank-c2 ( position ( list-ref c2 0 )))
    ( cond
     ( ( > rank-c1 rank-c2 ) ( list-ref c1 0 ))
   ( else
    ( list-ref c2 0 )
    )
  )
)

; ( trace higher-rank )


( define ( classify-two-cards-ur card )
   ( display card ) ( display ": " )
   ( define c1 ( first card ) )
   ( define c2 ( second card ) )

     ( cond
        ( ( equal? ( rank c1 ) ( rank c2 ) )
          ( display ( rank c1 ) ) ( display " pair" )
         )
        (  ( or ( equal? ( rank c1 ) 'A ) ( equal? ( rank c2 ) 'A ))
          ( display ( rank c1 ) ) ( display " high" )
         )

        ( ( or ( equal? ( rank c1 ) 'K ) ( equal? ( rank c2 ) 'K ))
          ( display ( rank c1 ) ) ( display " high" )
         )
```

```
        ( ( or ( equal? ( rank c1 ) 'Q ) ( equal? ( rank c2 ) 'Q ))
          ( display ( rank c1 ) ) ( display " high" )
         )
         ( ( or ( equal? ( rank c1 ) 'J ) ( equal? ( rank c2 ) 'J ))
          ( display ( rank c1 ) ) ( display " high" )
         )
        ( ( or ( equal? ( rank c1 ) 'X ) ( equal? ( rank c2 ) 'X ))
          ( display ( rank c1 ) ) ( display " high" )
         )
       ( else
          ( cond
            ( ( > ( rank c1 ) ( rank c2 ) )
              ( display ( rank c1 ) ) ( display " high" )
             )
            ( ( < ( rank c1 ) ( rank c2 ) )
             ( display ( rank c2 ) ) ( display " high" )
            )
          )
         )
        )

 ( straight? c1 c2 )
 ( flush? c1 c2 )

)

( define ( straight? c1 c2 )
 ( cond
   ( ( equal? ( rank c1 ) 'A )
     ( cond
       ( ( equal? ( rank c2 ) 'K )
       ( display " straight" )
      )
     )
     )
    ( ( equal? ( rank c1 ) 'K )
      ( cond
       ( ( or ( equal? ( rank c2 ) 'A ) ( equal? ( rank c2 ) 'Q ))
         ( display " straight" )
         )
   )
  )
    ( ( equal? ( rank c1 ) 'Q )
         ( cond
           ( ( or ( equal? ( rank c2 ) 'K ) ( equal? ( rank c2 ) 'J ))
         ( display " straight" )
         )
         )
     )
        ( ( equal? ( rank c1 ) 'J )
         ( cond
          ( ( or ( equal? ( rank c2 ) 'Q ) ( equal? ( rank c2 ) 'X ))
         ( display " straight" )
         )
```

```
 )
)
 ( ( equal? ( rank c1 ) 'X )
  ( cond
    ( ( or ( equal? ( rank c2 ) 'j ) ( equal? ( rank c2 ) 9 ))
   ( display " straight" )
   )
 )
)
 ( ( equal? ( rank c1 ) 9 )
  ( cond
     ( ( or ( equal? ( rank c2 ) 'X ) ( equal? ( rank c2 ) 8 ))
   ( display " straight" )
   )
   )
 )
 ( ( equal? ( rank c1 ) 8 )
   ( cond
     ( ( or ( equal? ( rank c2 ) 9 ) ( equal? ( rank c2 ) 7 ))
   ( display " straight" )
   )
   )
)
( ( equal? ( rank c1 ) 7 )
  ( cond
    ( ( or ( equal? ( rank c2 ) 8 ) ( equal? ( rank c2 ) 6 ))
   ( display " straight" )
   )
  )
)
( ( equal? ( rank c1 ) 6 )
  ( cond
     ( ( or ( equal? ( rank c2 ) 7 ) ( equal? ( rank c2 ) 5 ))
   ( display " straight" )
   )
  )
)
 ( ( equal? ( rank c1 ) 5 )
   ( cond
     ( ( or ( equal? ( rank c2 ) 6 ) ( equal? ( rank c2 ) 4 ))
   ( display " straight" )
   )
   )
 )
( ( equal? ( rank c1 ) 4 )
  ( cond
     ( ( or ( equal? ( rank c2 ) 5 ) ( equal? ( rank c2 ) 3 ))
   ( display " straight" )
   )
  )
)
( ( equal? ( rank c1 ) 3 )
  ( cond
   ( ( or ( equal? ( rank c2 ) 4 ) ( equal? ( rank c2 ) 2 ))
```

```
          ( display " straight" )
          )
          )
        )
        ( ( equal? ( rank c1 ) 2 )
          ( cond
            ( ( or ( equal? ( rank c2 ) 3 ) ( equal? ( rank c2 ) 1 ))
             ( display " straight" )
              )

        )
         )
        (( equal? ( rank c1 ) 1 )
        (cond
            ( ( equal? ( rank c2 ) 2 )
                 ( display " straight" )
              )
            )
          )
       )
   )

( define ( flush? c1 c2 )
 ( cond
 ( ( equal? ( suit c1 ) ( suit c2 ) )
 ( display " flush" )
 )
 )
)

( define ( classify-two-cards card )
 ( display card ) ( display ": " )
 ( define c1 ( first card ) )
 ( define c2 ( second card ) )

   ( cond
    ( ( and ( equal? ( rank c1 ) 'A ) ( equal? ( rank c2 ) 'A ) )
       ( display "ace pair" )
    )
    ( ( and ( equal? ( rank c1 ) 'K ) ( equal? ( rank c2 ) 'K ) )
       ( display "king pair" )
    )
    ( ( and ( equal? ( rank c1 ) 'Q ) ( equal? ( rank c2 ) 'Q ) )
       ( display "queen pair" )
   )
    ( ( and ( equal? ( rank c1 ) 'J ) ( equal? ( rank c2 ) 'J ) )
       ( display "jack pair" )
   )
    ( ( and ( equal? ( rank c1 ) 'X ) ( equal? ( rank c2 ) 'X ) )
       ( display "ten pair" )
   )
    ( ( and ( equal? ( rank c1 ) 9 ) ( equal? ( rank c2 ) 9 ) )
       ( display "nine pair" )
   )
```

```
  ( ( and ( equal? ( rank c1 ) 8 ) ( equal? ( rank c2 ) 8 ) )
     ( display "eight pair" )
 )
  ( ( and ( equal? ( rank c1 ) 7 ) ( equal? ( rank c2 ) 7 ) )
     ( display "seven pair" )
 )
  ( ( and ( equal? ( rank c1 ) 6 ) ( equal? ( rank c2 ) 9 ) )
     ( display "six pair" )
 )
  ( ( and ( equal? ( rank c1 ) 5 ) ( equal? ( rank c2 ) 5 ) )
     ( display "five pair" )
 )
  ( ( and ( equal? ( rank c1 ) 4 ) ( equal? ( rank c2 ) 4 ) )
     ( display "four pair" )
 )
  ( ( and ( equal? ( rank c1 ) 3 ) ( equal? ( rank c2 ) 3 ) )
     ( display "three pair" )
 )
  ( ( and ( equal? ( rank c1 ) 2 ) ( equal? ( rank c2 ) 2 ) )
     ( display "two pair" )
 )
  ( ( or ( equal? ( rank c1 ) 'A ) ( equal? ( rank c2 ) 'A ) )
     ( display "ace high" )
 )
 ( ( or ( equal? ( rank c1 ) 'K ) ( equal? ( rank c2 ) 'K ) )
     ( display "king high" )
 )
 ( ( or ( equal? ( rank c1 ) 'Q ) ( equal? ( rank c2 ) 'Q ) )
     ( display "queen high" )
 )
 ( ( or ( equal? ( rank c1 ) 'J ) ( equal? ( rank c2 ) 'K ) )
     ( display "jack high" )
 )
( ( or ( equal? ( rank c1 ) 'X ) ( equal? ( rank c2 ) 'X ) )
     ( display "ten high" )
 )
( ( or ( equal? ( rank c1 ) 9 ) ( equal? ( rank c2 ) 9 ) )
     ( display "nine high" )
 )
( ( or ( equal? ( rank c1 ) 8 ) ( equal? ( rank c2 ) 8 ) )
     ( display "eight high" )
 )
 ( ( or ( equal? ( rank c1 ) 7 ) ( equal? ( rank c2 ) 7 ) )
     ( display "seven high" )
 )
( ( or ( equal? ( rank c1 ) 6 ) ( equal? ( rank c2 ) 6 ) )
     ( display "six high" )
 )
 ( ( or ( equal? ( rank c1 ) 5 ) ( equal? ( rank c2 ) 5 ) )
     ( display "five high" )
 )
  ( ( or ( equal? ( rank c1 ) 4 ) ( equal? ( rank c2 ) 4 ) )
     ( display "four high" )
 )
```

```
    ( ( or ( equal? ( rank c1 ) 3 ) ( equal? ( rank c2 ) 3 ) )
        ( display "three high" )
    )
    ( ( or ( equal? ( rank c1 ) 2 ) ( equal? ( rank c2 ) 2 ) )
        ( display "two high" )
    )


 )

 ( straight? c1 c2 )
 ( flush? c1 c2 )

)
```

Task 4c - Demo of Two Cards Poker Classifier Code (2)