Racket Programming Assignment #3: Lambda and Basic Lisp

Learning Abstract

This Assignment will focus on Lambda functions, Referencers, Constructors, Fun with colors, and a Poker Hand Classifier. First we will make use of lambda functions that will create a list of consecutive numbers, reverse a list, and generate a random number. I will then recreate a demo in Task 2 that uses different list references in constructors. In Task 3 I will be creating an interpreter that will select a specific, random, or all colors from a list and display them. Task 4 will take in a list of two cards and classify what type of hand you are holding.

Task 1: Lambda

1a: Three ascending integers

```
Welcome to DrRacket, version 8.3 [cs].
Language: racket, with debugging; memory limit: 128 MB.
> (lambda (x) (cons x (cons (+ x 1) (cons (+ x 2) '()))))
#<procedure>
> (lambda (x) (cons x (cons (+ x 1) (cons (+ x 2) '())))) 5)
#<procedure>
> ((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)
>
```

1b: Make list in reverse order

```
Welcome to DrRacket, version 8.3 [cs].
Language: racket, with debugging; memory limit: 128 MB.
> ((lambda (list) (cons (caddr list) (cons (cadr list) (cons (car list) '()))))
   '(red yellow blue))
'(blue yellow red)
> ((lambda (list) (cons (caddr list) (cons (cadr list) (cons (car list) '()))))
   '(10 20 30))
'(30 20 10)
> ((lambda (list) (cons (caddr list) (cons (cadr list) (cons (car list) '()))))
   '("Professor Plum" "Colonel Mustard" "Miss Scarlet"))
'("Miss Scarlet" "Colonel Mustard" "Professor Plum")
>
```

1c: Random number generator

```
Welcome to DrRacket, version 8.3 [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)
4
> ((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)
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)
3
> ((lambda (x y) (random x (+ y 1))) 3 5)
5
>
> ((lambda (x y) (random x (+ y 1))) 3 5)
4
```

```
Welcome to DrRacket, version 8.3 [cs].
Language: racket, with debugging; memory limit: 128 MB.
> ((lambda (x y) (random x (+ y 1))) 11 17)
15
> ((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)
16
> ((lambda (x y) (random x (+ y 1))) 11 17)
16
> ((lambda (x y) (random x (+ y 1))) 11 17)
16
> ((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)
14
> ((lambda (x y) (random x (+ y 1))) 11 17)
11
```

Task 2: List Processing References and Constructors

```
Welcome to DrRacket, version 8.3 [cs].
Language: racket, with debugging; memory limit: 128 MB.
> ( define languages '(racket prolog haskell rust) )
> languages
'(racket prolog haskell rust)
> 'languages
'languages
> (quote languages)
'languages
> (car languages)
'racket
> (cdr languages)
'(prolog haskell rust)
> (car (cdr languages))
'prolog
> (cdr (cdr languages))
'(haskell rust)
> (cadr languages)
'prolog
> (cddr languages)
'(haskell rust)
> (first languages)
'racket
> (second languages)
'prolog
> (third languages)
'haskell
> (list-ref languages 2)
'haskell
> (define numbers '(1 2 3))
> (define letters '(a b c))
> (cons numbers letters)
'((1 2 3) a b c)
> (list numbers letters)
'((1 2 3) (a b c))
> (append numbers letters)
'(1 2 3 a b c)
> (define animals '(ant bat cat dot eel))
> (car (cdr (cdr animals)))
'dot
> (cadddr animals)
'dot
> (list-ref animals 3)
'dot
> (define a 'apple)
> (define b 'peach)
> (define c 'cherry)
> (cons a ( cons b (cons c '())))
'(apple peach cherry)
> (list a b c)
'(apple peach cherry)
> (define x '(one fish))
> (define y '(two fish))
> (cons (car x)(cons (car (cdr x))y))
'(one fish two fish)
> (append x y)
'(one fish two fish)
>
```

Task 3: Little Color Interpreter

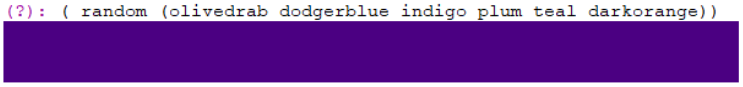
3a: Establishing the Sampler code from lesson 6

```
> ( define ( sampler )
( display "(?): " )
( define the-list ( read ) )
( define the-element
( list-ref the-list ( random ( length the-list ) ) ) )
)
( display the-element ) ( display "\n" )
( sampler )
)
> (sampler)
(?): ( red orange yellow green blue indigo violet )
blue
(?): ( red orange yellow green blue indigo violet )
orange
(?): ( red orange yellow green blue indigo violet )
red
(?): ( red orange yellow green blue indigo violet )
green
(?): ( red orange yellow green blue indigo violet )
orange
(?): ( red orange yellow green blue indigo violet )
orange
(?): ( aet ate eat eta tae tea )
tae
(?): ( aet ate eat eta tae tea )
tea
(?): ( aet ate eat eta tae tea )
eat
(?): ( aet ate eat eta tae tea )
aet
(?): ( aet ate eat eta tae tea )
ate
(?): ( aet ate eat eta tae tea )
ate
(?):  ( 0 1 2 3 4 5 6 7 8 9 )
5
(?):  ( 0 1 2 3 4 5 6 7 8 9 )
5
(?):  ( 0 1 2 3 4 5 6 7 8 9 )
0
(?):  ( 0 1 2 3 4 5 6 7 8 9 )
8
(?):  ( 0 1 2 3 4 5 6 7 8 9 )
1
(?):  ( 0 1 2 3 4 5 6 7 8 9 )
7
```

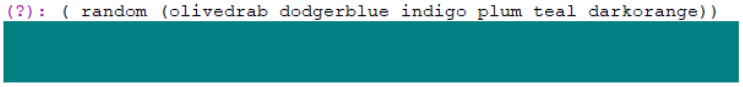
3b: Color Thing Interpreter

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

( define ( color-thing )
   ( display "(?): " )
   ( define the-list ( read ) )
   (cond
     ((equal? (car the-list) 'random)
      (random-color (cadr the-list)))
     (else
      (cond
        ((equal? (car the-list) 'all)
        (all-colors (cadr the-list)))
       (else
        (select-color (car the-list) (cadr the-list))))))
   (color-thing)
 )

(define (stripe color)
  (display (rectangle 600 50 "solid" color))
  (display "\n")
  )

( define (random-color the-list)
  (stripe (list-ref the-list (random (length the-list)))))

(define (select-color number the-list)
  (stripe ( list-ref the-list (- number 1) ) ) )

(define (all-colors the-list)
  (cond
    ((empty? the-list)
     (display "\n"))
    (else
     (stripe (car the-list))
     (all-colors (cdr the-list) ) ) ) )
```

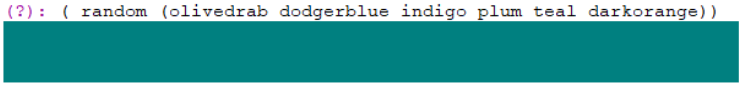
```
Welcome to DrRacket, version 8.3 [cs].
Language: racket, with debugging; memory limit: 128 MB.
> (color-thing)
(?): ( random (olivedrab dodgerblue indigo plum teal darkorange))
```



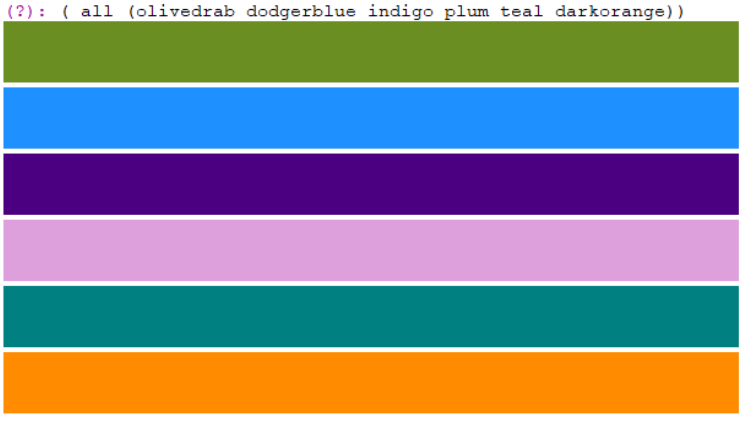
```
(?): ( random (olivedrab dodgerblue indigo plum teal darkorange))
```



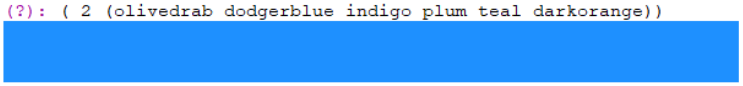
```
(?): ( random (olivedrab dodgerblue indigo plum teal darkorange))
```



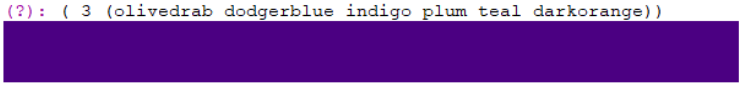
```
(?): ( all (olivedrab dodgerblue indigo plum teal darkorange))
```



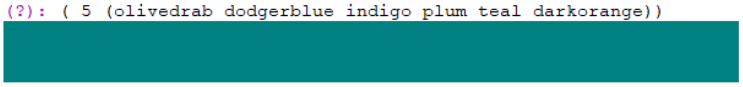
```
(?): ( 2 (olivedrab dodgerblue indigo plum teal darkorange))
```



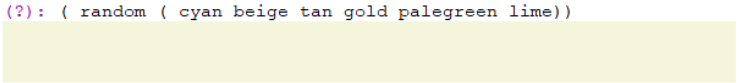
```
(?): ( 3 (olivedrab dodgerblue indigo plum teal darkorange))
```



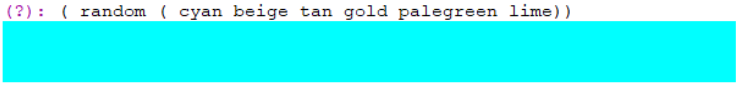
```
(?): ( 5 (olivedrab dodgerblue indigo plum teal darkorange))
```



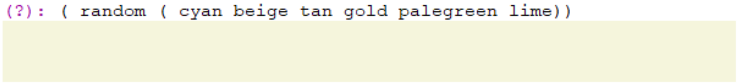
```
Welcome to DrRacket, version 8.3 [cs].
Language: racket, with debugging; memory limit: 128 MB.
> (color-thing)
(?): ( random ( cyan beige tan gold palegreen lime))
```



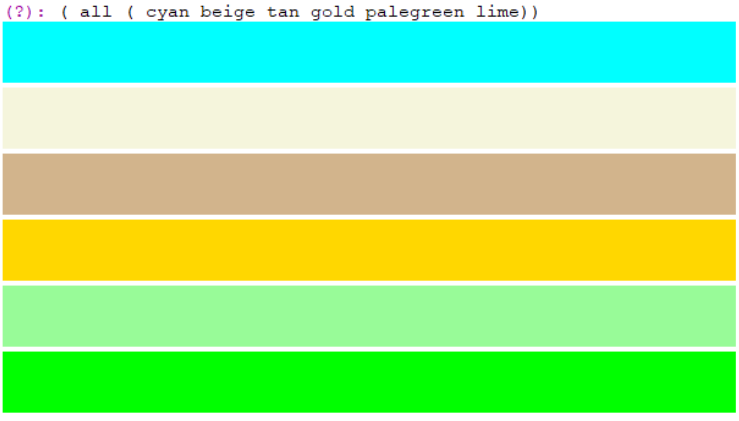
```
(?): ( random ( cyan beige tan gold palegreen lime))
```



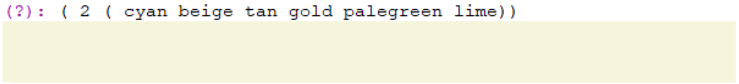
```
(?): ( random ( cyan beige tan gold palegreen lime))
```



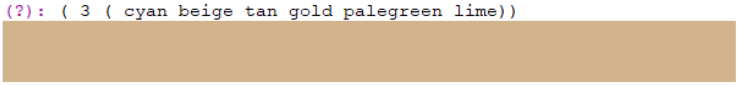
```
(?): ( all ( cyan beige tan gold palegreen lime))
```



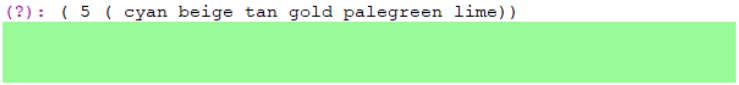
```
(?): ( 2 ( cyan beige tan gold palegreen lime))
```



```
(?): ( 3 ( cyan beige tan gold palegreen lime))
```



```
(?): ( 5 ( cyan beige tan gold palegreen lime))
```



Task 4: Two Card Poker

4a: Establishing the Card code from lesson 6

```
#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 )
)
)
```

```
Language: racket, with debugging; memory limit: 128 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 S)
> (pick-a-card)
'(8 D)
> (pick-a-card)
'(A H)
> (pick-a-card)
'(7 H)
> (pick-a-card)
'(A D)
> (pick-a-card)
'(4 C)
```

4b: Establishing the Card code from lesson 6

```
#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 ( suit card )
( cadr card )
)

(define (pick-two-cards)
  (define c1 (pick-a-card))
  (define c2 (pick-a-card))
  (cond
    ((equal? c1 c2)
     (pick-two-cards))
    (else
     (list c1 c2))))
(define (rank card)
    (define cn (car card))
    (cond
    ((number? cn)
     (car card))
    (else
     (cond
       ((equal? cn 'X)10)
       (else
        (cond
          ((equal? cn 'J)11)
          (else
           (cond
             ((equal? cn 'Q)12)
             (else
              (cond
                ((equal? cn 'K) 13)
                (else
                 (cond
                   ((equal? cn 'A)14)))))))))))))

(define (higher-rank c1 c2)
  (cond
    ((> (rank c1) (rank c2))
     (car c1))
    (else
     (car c2))))


(define (classify-two-cards-ur cards)
  (define c1 (caar cards))
  (define c2 (caadr cards))
  (define s1 (suit (car cards)))
  (define s2 (suit (cadr cards)))
  (define hi (higher-rank (car cards) (cadr cards)))
  (display cards) (display ": ")
  (cond
    ((equal? c1 c2)
        (display "Pair of ")(display c1)(display "'s"))
    (else
     (cond
       ((equal? s1 s2)
        (cond
        ((or
          (= 1 (- (rank (car cards)) (rank (cadr cards))))
          (= 1 (- (rank (cadr cards)) (rank (car cards)))))
         (display hi) (display " high straight flush"))
        (else
        (display hi)(display "  high flush"))))
       (else
        (cond
           ((or
           (= 1 (- (rank (car cards)) (rank (cadr cards))))
           (= 1 (- (rank (cadr cards)) (rank (car cards)))))
            (display hi) (display " high straight"))
           (else
            (display hi) (display " high")))))))))

(trace higher-rank)
```

```
Welcome to DrRacket, version 8.3 [cs].
Language: racket, with debugging; memory limit: 128 MB.
> ( pick-two-cards )
'((X C) (A H))
> ( pick-two-cards )
'((3 S) (Q D))
> ( pick-two-cards )
'((9 D) (J S))
> ( pick-two-cards )
'((7 C) (4 D))
> ( pick-two-cards )
'((6 C) (8 H))
>
```

```
Welcome to DrRacket, version 8.3 [cs].
Language: racket, with debugging; memory limit: 128 MB.
> ( higher-rank ( pick-a-card ) ( pick-a-card ) )
>(higher-rank '(X C) '(2 D))
<'X
'X
> ( higher-rank ( pick-a-card ) ( pick-a-card ) )
>(higher-rank '(J D) '(4 D))
<'J
'J
> ( higher-rank ( pick-a-card ) ( pick-a-card ) )
>(higher-rank '(X C) '(5 D))
<'X
'X
> ( higher-rank ( pick-a-card ) ( pick-a-card ) )
>(higher-rank '(5 C) '(K H))
<'K
'K
> ( higher-rank ( pick-a-card ) ( pick-a-card ) )
>(higher-rank '(3 H) '(9 C))
<9
9
>
```

```
Welcome to DrRacket, version 8.3 [cs].
Language: racket, with debugging; memory limit: 128 MB.
> ( classify-two-cards-ur ( pick-two-cards ) )
((6 D) (K D)): K  high flush
> ( classify-two-cards-ur ( pick-two-cards ) )
((J H) (8 C)): J high
> ( classify-two-cards-ur ( pick-two-cards ) )
((6 D) (8 D)): 8  high flush
> ( classify-two-cards-ur ( pick-two-cards ) )
((3 C) (8 C)): 8  high flush
> ( classify-two-cards-ur ( pick-two-cards ) )
((3 H) (9 C)): 9 high
> ( classify-two-cards-ur ( pick-two-cards ) )
((K D) (8 D)): K  high flush
> ( classify-two-cards-ur ( pick-two-cards ) )
((4 S) (X D)): X high
> ( classify-two-cards-ur ( pick-two-cards ) )
((A C) (J H)): A high
> ( classify-two-cards-ur ( pick-two-cards ) )
((K D) (Q H)): K high straight
> ( classify-two-cards-ur ( pick-two-cards ) )
((Q C) (X H)): Q high
> ( classify-two-cards-ur ( pick-two-cards ) )
((9 S) (7 S)): 9  high flush
> ( classify-two-cards-ur ( pick-two-cards ) )
((J C) (5 D)): J high
> ( classify-two-cards-ur ( pick-two-cards ) )
((2 C) (5 S)): 5 high
> ( classify-two-cards-ur ( pick-two-cards ) )
((4 H) (5 H)): 5 high straight flush
> ( classify-two-cards-ur ( pick-two-cards ) )
((2 S) (J H)): J high
> ( classify-two-cards-ur ( pick-two-cards ) )
((2 H) (4 D)): 4 high
> ( classify-two-cards-ur ( pick-two-cards ) )
((6 S) (2 S)): 6  high flush
> ( classify-two-cards-ur ( pick-two-cards ) )
((9 D) (A S)): A high
> ( classify-two-cards-ur ( pick-two-cards ) )
((6 C) (5 D)): 6 high straight
> ( classify-two-cards-ur ( pick-two-cards ) )
((A D) (5 D)): A  high flush
>
```

4c: Two Card Poker Classifier

```
#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 ( suit card )
( cadr card )
)

(define (pick-two-cards)
  (define c1 (pick-a-card))
  (define c2 (pick-a-card))
  (cond
    ((equal? c1 c2)
     (pick-two-cards))
    (else
     (list c1 c2))))
(define (rank cn)
    (cond
    ((number? cn)
     cn)
    (else
     (cond
       ((equal? cn 'X)10)
       (else
        (cond
          ((equal? cn 'J)11)
          (else
           (cond
             ((equal? cn 'Q)12)
             (else
              (cond
                ((equal? cn 'K) 13)
               (else
                (cond
                  ((equal? cn 'A)14)))))))))))))

(define (higher-rank c1 c2)
  (cond
    ((> (rank c1) (rank c2))
     c1)
    (else
     c2)))


(define (classify-two-cards cards)
  (define c1 (caar cards))
  (define c2 (caadr cards))
  (define s1 (suit (car cards)))
  (define s2 (suit (cadr cards)))
  (define hi (higher-rank (caar cards) (caadr cards)))
  (display cards) (display ": ")
  (cond
    ((equal? c1 c2)
        (display "Pair of ")(display (card-name hi))(display "'s"))
    (else
     (cond
       ((equal? s1 s2)
        (cond
        ((or
          (= 1 (- (rank c1) (rank c2)))
          (= 1 (- (rank c2) (rank c1))))
         (display (card-name (rank hi))) (display " high straight flush"))
        (else
        (display (card-name (rank hi)))(display " high flush"))))
       (else
        (cond
          ((or
          (= 1 (- (rank c1) (rank c2)))
          (= 1 (- (rank c2) (rank c1))))
           (display (card-name (rank hi))) (display " high straight"))
          (else
           (display (card-name (rank hi))) (display " high"))))))))

(define rank-names (list 'one 'two 'three 'four 'five 'six 'seven
'eight 'nine 'ten 'jack 'queen 'king 'ace))

(define (card-name card)
  (list-ref rank-names (- card 1)))
```

```
Welcome to DrRacket, version 8.3 [cs].
Language: racket, with debugging; memory limit: 128 MB.
> ( classify-two-cards ( pick-two-cards ) )
((9 D) (9 H)): Pair of nine's
> ( classify-two-cards ( pick-two-cards ) )
((6 D) (9 S)): nine high
> ( classify-two-cards ( pick-two-cards ) )
((6 C) (2 C)): six high flush
> ( classify-two-cards ( pick-two-cards ) )
((6 D) (J C)): jack high
> ( classify-two-cards ( pick-two-cards ) )
((2 H) (A H)): ace high flush
> ( classify-two-cards ( pick-two-cards ) )
((2 D) (5 D)): five high flush
> ( classify-two-cards ( pick-two-cards ) )
((J C) (X C)): jack high straight flush
> ( classify-two-cards ( pick-two-cards ) )
((2 S) (X H)): ace high
> ( classify-two-cards ( pick-two-cards ) )
((3 C) (5 C)): five high flush
> ( classify-two-cards ( pick-two-cards ) )
((9 H) (8 D)): nine high straight
> ( classify-two-cards ( pick-two-cards ) )
((Q D) (5 D)): queen high flush
> ( classify-two-cards ( pick-two-cards ) )
((4 C) (4 H)): Pair of four's
> ( classify-two-cards ( pick-two-cards ) )
((8 C) (K C)): king high flush
> ( classify-two-cards ( pick-two-cards ) )
((2 C) (K D)): king high
> ( classify-two-cards ( pick-two-cards ) )
((J H) (6 S)): jack high
> ( classify-two-cards ( pick-two-cards ) )
((4 H) (Q S)): queen high
> ( classify-two-cards ( pick-two-cards ) )
((6 D) (4 H)): six high
> ( classify-two-cards ( pick-two-cards ) )
((7 H) (A D)): ace high
> ( classify-two-cards ( pick-two-cards ) )
((K D) (J C)): king high
> ( classify-two-cards ( pick-two-cards ) )
((J D) (6 H)): jack high
>
```