Racket Programming Assignment #3: Lambda and Basic Lisp

Brandon LaPointe

Learning Abstract

This assignment features a mix of several interactions within the Racket programming language. The first task involves using small anonymous functions called lambda functions. These include displaying a list of three ascending integers starting at the given single integer parameter, reversing list order of given lists, and the use of a random number generator that produces a result between two given parameters. The second two task is simply a demo of the sampler preliminary code given to further extend. The third and fourth tasks are considerably more involved and required further extensions of an existing program to perform the demos provided. Task three is a color interpreter that takes in a list containing a command, either random, all, or an integer value between 1 and the length of the list of colors, and a second parameter being a list of colors. The fourth task involved creating hands of two cards and then classifying those hands into two card poker hands through the fabrication of the pick-two-cards, higher-rank, and classify-two-cards-ur functions as well as helper functions to identify a straight, flush, and pair. The final demo shows the classify-two-cards function which shows the same information as the classify-two-cards-ur function, but in a more visually appealing manner.

Task 1a: Lambda – Three Ascending Integers

```
Welcome to DrRacket, version 8.6 [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)
>
```

Task 1b: Lambda – Reversing List Order

```
Welcome to DrRacket, version 8.6 [cs].
Language: racket, with debugging; memory limit: 128 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: Lambda – Reversing List Order

```
Welcome to DrRacket, version 8.6 [cs].
Language: racket, with debugging; memory limit: 128 MB.
> ( ( lambda ( x y ) ( cons ( + ( random ( + ( - y x ) 1 ) ) x ) '() ) ) 3 5 )
'(3)
> ( ( lambda ( x y ) ( cons ( + ( random ( + ( - y x ) 1 ) ) x ) '() ) ) 3 5 )
'(3)
> ( ( lambda ( x y ) ( cons ( + ( random ( + ( - y x ) 1 ) ) x ) '() ) ) 3 5 )
'(5)
> ( ( lambda ( x y ) ( cons ( + ( random ( + ( - y x ) 1 ) ) x ) '() ) ) 3 5 )
'(4)
> ( ( lambda ( x y ) ( cons ( + ( random ( + ( - y x ) 1 ) ) x ) '() ) ) 3 5 )
'(4)
> ( ( lambda ( x y ) ( cons ( + ( random ( + ( - y x ) 1 ) ) x ) '() ) ) 3 5 )
'(4)
> ( ( lambda ( x y ) ( cons ( + ( random ( + ( - y x ) 1 ) ) x ) '() ) ) 3 5 )
'(5)
> ( ( lambda ( x y ) ( cons ( + ( random ( + ( - y x ) 1 ) ) x ) '() ) ) 3 5 )
'(4)
> ( ( lambda ( x y ) ( cons ( + ( random ( + ( - y x ) 1 ) ) x ) '() ) ) 3 5 )
'(5)
> ( ( lambda ( x y ) ( cons ( + ( random ( + ( - y x ) 1 ) ) x ) '() ) ) 3 5 )
'(4)
> ( ( lambda ( x y ) ( cons ( + ( random ( + ( - y x ) 1 ) ) x ) '() ) ) 11 17 )
'(11)
> ( ( lambda ( x y ) ( cons ( + ( random ( + ( - y x ) 1 ) ) x ) '() ) ) 11 17 )
'(15)
> ( ( lambda ( x y ) ( cons ( + ( random ( + ( - y x ) 1 ) ) x ) '() ) ) 11 17 )
'(17)
> ( ( lambda ( x y ) ( cons ( + ( random ( + ( - y x ) 1 ) ) x ) '() ) ) 11 17 )
'(17)
> ( ( lambda ( x y ) ( cons ( + ( random ( + ( - y x ) 1 ) ) x ) '() ) ) 11 17 )
'(12)
> ( ( lambda ( x y ) ( cons ( + ( random ( + ( - y x ) 1 ) ) x ) '() ) ) 11 17 )
'(13)
> ( ( lambda ( x y ) ( cons ( + ( random ( + ( - y x ) 1 ) ) x ) '() ) ) 11 17 )
'(12)
> ( ( lambda ( x y ) ( cons ( + ( random ( + ( - y x ) 1 ) ) x ) '() ) ) 11 17 )
'(13)
> ( ( lambda ( x y ) ( cons ( + ( random ( + ( - y x ) 1 ) ) x ) '() ) ) 11 17 )
'(13)
> ( ( lambda ( x y ) ( cons ( + ( random ( + ( - y x ) 1 ) ) x ) '() ) ) 11 17 )
'(17)
>
```

Task 2: List Processing Referencers and Constructors

```
Welcome to DrRacket, version 8.6 [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 animals ) ) ) )

animals: undefined;
 cannot reference an identifier before its definition
> ( 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 3a: Little Color Interpreter – Sampler

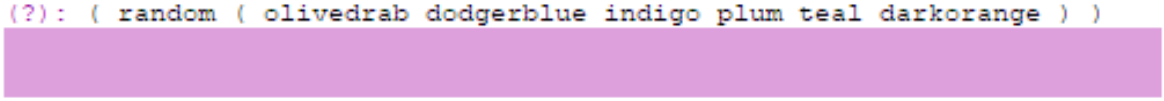
```
Welcome to DrRacket, version 8.6 [cs].
Language: racket, with debugging; memory limit: 128 MB.
> ( sampler )
(?): ( red orange yellow green blue indigo violet )
indigo
(?):  ( red orange yellow green blue indigo violet )
orange
(?):  ( red orange yellow green blue indigo violet )
violet
(?):  ( red orange yellow green blue indigo violet )
yellow
(?):  ( red orange yellow green blue indigo violet )
blue
(?):  ( red orange yellow green blue indigo violet )
green
(?): ( aet ate eat eta tae tea )
tae
(?): ( aet ate eat eta tae tea )
ate
(?): ( aet ate eat eta tae tea )
eta
(?): ( aet ate eat eta tae tea )
aet
(?): ( aet ate eat eta tae tea )
tae
(?): ( aet ate eat eta tae tea )
aet
(?): ( 0 1 2 3 4 5 6 7 8 9 )
4
(?): ( 0 1 2 3 4 5 6 7 8 9 )
8
(?): ( 0 1 2 3 4 5 6 7 8 9 )
2
(?): ( 0 1 2 3 4 5 6 7 8 9 )
3
(?): ( 0 1 2 3 4 5 6 7 8 9 )
3
(?): ( 0 1 2 3 4 5 6 7 8 9 )
3
(?): user break
>
```

Task 3a: Code

```
1  #lang racket
2
3  ( define ( sampler )
4     ( display "(?): " )
5     ( define the-list ( read ) )
6     ( define the-element
7        ( list-ref the-list ( random ( length the-list ) ) )
8     )
9     ( display the-element ) ( display "\n" )
10    ( sampler )
11 )
```

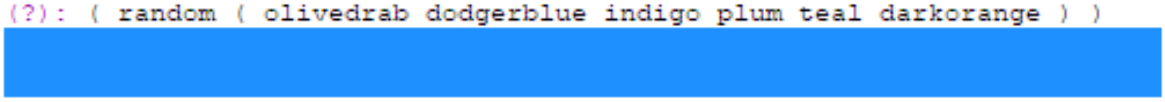
Task 3b: Color Thing Interpreter – Color Thing Interpreter Demo 1

```
Welcome to DrRacket, version 8.6 [cs].
Language: racket, with debugging; memory limit: 128 MB.
> ( color-thing )
(?): ( random ( olivedrab dodgerblue indigo plum teal darkorange ) )
```

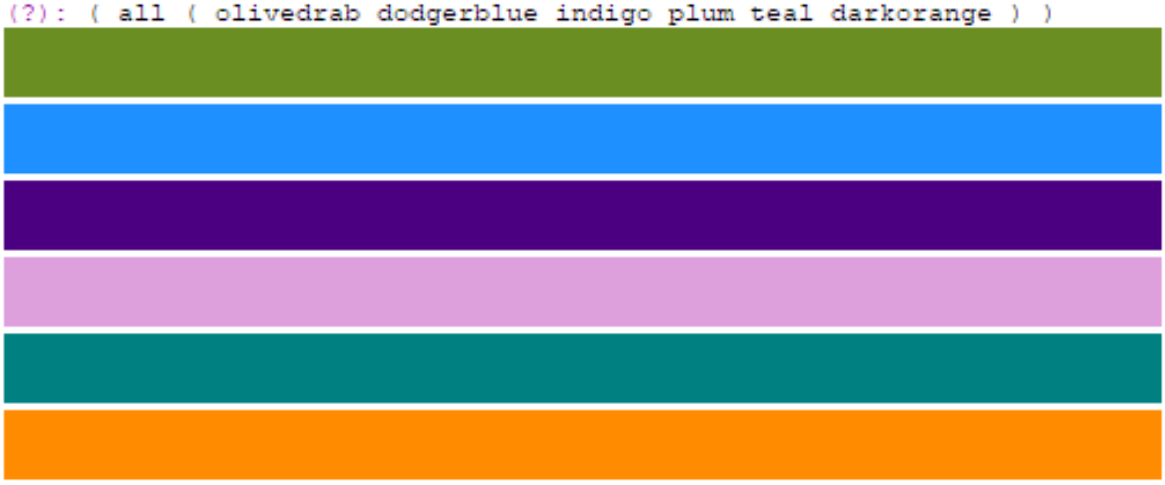


```
(?): ( random ( olivedrab dodgerblue indigo plum teal darkorange ) )
```

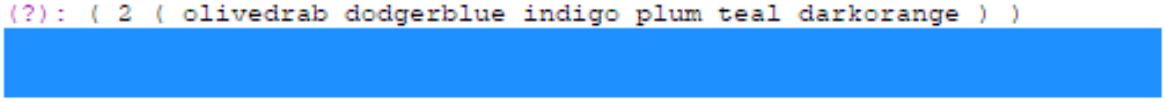
```
(?): ( random ( olivedrab dodgerblue indigo plum teal darkorange ) )
```



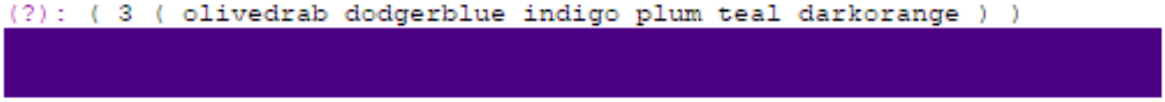
```
(?): ( all ( olivedrab dodgerblue indigo plum teal darkorange ) )
```



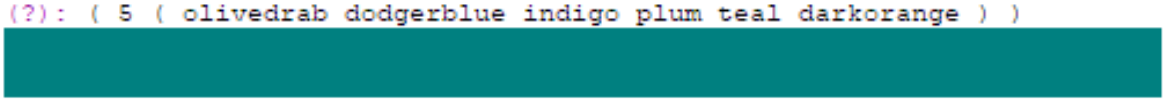
```
(?): ( 2 ( olivedrab dodgerblue indigo plum teal darkorange ) )
```



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



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



```
(?):
```

eof

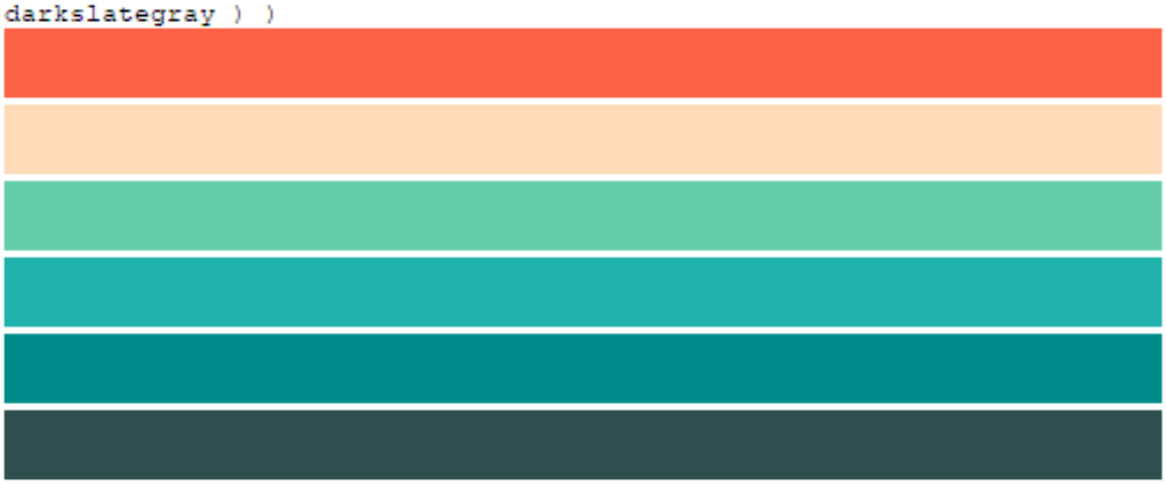
Task 3b: Color Thing Interpreter – Color Thing Interpreter Demo 2

```
Welcome to DrRacket, version 8.6 [cs].
Language: racket, with debugging; memory limit: 128 MB.
> ( color-thing )
(?): ( random ( tomato peachpuff mediumaquamarine lightseagreen darkcyan
darkslategray ) )
```

```
(?): ( random ( tomato peachpuff mediumaquamarine lightseagreen darkcyan
darkslategray ) )
```

```
(?): ( random ( tomato peachpuff mediumaquamarine lightseagreen darkcyan
darkslategray ) )
```

```
(?): ( all ( tomato peachpuff mediumaquamarine lightseagreen darkcyan
darkslategray ) )
```



```
(?): ( 2 ( tomato peachpuff mediumaquamarine lightseagreen darkcyan
darkslategray ) )
```

```
(?): ( 3 ( tomato peachpuff mediumaquamarine lightseagreen darkcyan
darkslategray ) )
```

```
(?): ( 5 ( tomato peachpuff mediumaquamarine lightseagreen darkcyan
darkslategray ) )
```

```
(?):
```

eof

Task 3b: Code

```
#lang racket
; -----------------------------------------------------------------------
; Program that mimics the "Preliminary Code" that consists of three functions,
; each of which takes the form of a list of length 2:
; (1) a function to display a random color of the given color list,
; (2) a function to display all the colors of the given color list, and
; (3) a function to display a select color of the given color list in the
;     form of an integer between 1 and the length of the color list.
;
;  EXAMPLE COMMANDS
;  ( random ( olivedrab dodgerblue indigo plum teal darkorange ) )
;  ( all ( olivedrab dodgerblue indigo plum teal darkorange ) )
;  ( 2 ( olivedrab dodgerblue indigo plum teal darkorange ) )
;  ( random ( tomato peachpuff mediumaquamarine lightseagreen darkcyan darkslategray ) )
;  ( all ( tomato peachpuff mediumaquamarine lightseagreen darkcyan darkslategray ) )
;  ( 2 ( tomato peachpuff mediumaquamarine lightseagreen darkcyan darkslategray ) )
;
; -----------------------------------------------------------------------
; Import the image library from Version 2 of "How to Design Programs"
;
(require 2htdp/image)


; -----------------------------------------------------------------------
; Color-thing Definitions
;
( define ( color-thing )
  ( display "(?): " )
  ( define list ( read ) )
  ( define command ( car list ) )
  ( define color-list ( cadr list ) )
; -----------------------------------------------------------------------
; Command Definitions
;
  ( define ( random-color )
    ( display ( rectangle 500 30 "solid" ( list-ref color-list ( random ( length color-list ) ) ) ) )
  )

  ( define ( select-color )
    ( display ( rectangle 500 30 "solid" ( list-ref color-list ( - command 1 ) ) ) )
  )

  ( define ( all-colors )
    ( display ( paint-colors 0 ) )
  )
  ( define ( paint-colors n )
    ( cond
      ( ( = n ( - ( length color-list ) 1 ) )
        ( display ( rectangle 500 30 "solid" ( list-ref color-list n ) ) )
        ( display "\n" )
        ( color-thing )
      )
      ( else
        ( display ( rectangle 500 30 "solid" ( list-ref color-list n ) ) )
        ( display "\n" )
        ( paint-colors ( + n 1 ) )
      )
    )
  )
; -----------------------------------------------------------------------
; Command Conditions
;
  ( cond
    ( ( equal? command 'random )
      ( random-color )
    )
    ( ( equal? command 'all )
      ( all-colors )
    )
    ( else
      ( select-color )
    )
  )
  ( display "\n" )
; -----------------------------------------------------------------------
; Recursive Call to color-thing
;
  ( color-thing )
)
```

Task 4a: Two Card Poker – Cards

```
Welcome to DrRacket, version 8.6 [cs].
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 )
'(6 C)
> ( pick-a-card )
'(8 C)
> ( pick-a-card )
'(3 H)
> ( pick-a-card )
'(9 H)
> ( pick-a-card )
'(7 S)
> ( pick-a-card )
'(6 C)
>
```

Task 4a: 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 )
  )
)
```

Task 4b: Two Card Poker – UR Classifier

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

Task 4b: Code

```
#lang racket
( require racket/trace )
; ---------------------------------------------------------------------------
; EXAMPLE COMMANDS
; ( pick-two-cards )
; ( higher-rank ( pick-a-card ) ( pick-a-card ) )
; ( classify-two-cards-ur ( pick-two-cards ) )

; ---------------------------------------------------------------------------
; Preliminary Code
;
( 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 )
  )
)
; ---------------------------------------------------------------------------
; Pick-two-cards Function
;
( define ( pick-two-cards )
  ( define card1 ( pick-a-card ) )
  ( define card2 ( pick-a-card ) )
  ( cond
    ( ( equal? card1 card2 )
      ( pick-two-cards )
    )
    ( else
      ( list card1 card2 )
    )
  )
)
; ---------------------------------------------------------------------------
; Higher-rank Function
```

```
;
( define ( higher-rank card1 card2 )
  ( display ( list card1 card2 ) ) ( display "\n" )
  ( cond
    ( ( equal? ( rank card1 ) 'A )
      ( display ( rank card1 ) )
    )
    ( ( equal? ( rank card2 ) 'A )
      ( display ( rank card2 ) )
    )
    ( ( equal? ( rank card1 ) 'K )
      ( display ( rank card1 ) )
    )
    ( ( equal? ( rank card2 ) 'K )
      ( display ( rank card2 ) )
    )
    ( ( equal? ( rank card1 ) 'Q )
      ( display ( rank card1 ) )
    )
    ( ( equal? ( rank card2 ) 'Q )
      ( display ( rank card2 ) )
    )
    ( ( equal? ( rank card1 ) 'J )
      ( display ( rank card1 ) )
    )
    ( ( equal? ( rank card2 ) 'J )
      ( display ( rank card2 ) )
    )
    ( ( equal? ( rank card1 ) 'X )
      ( display ( rank card1 ) )
    )
    ( ( equal? ( rank card2 ) 'X )
      ( display ( rank card2 ) )
    )
    ( else
      ( cond
        ( ( equal? ( rank card1 ) ( rank card2 ) )
          (  rank card1 )
        )
        ( ( > ( rank card1 ) ( rank card2 ) )
          ( display ( rank card1 ) )
        )
        ( ( < ( rank card1 ) ( rank card2 ) )
          ( display ( rank card2 ) )
        )
       )
    )
  )
)
; ---------------------------------------------------------------------------
; Classify-two-cards-ur Function
;
( define ( classify-two-cards-ur cards )
  ( display cards ) ( display ": " )
  ( define card1 ( first cards ) )
  ( define card2 ( second cards ) )
  ( cond
    ( ( equal? ( rank card1 ) ( rank card2 ) )
      ( display "Pair of " ) ( display ( rank card1 ) ) ( display "'s" )
    )
    ( ( equal? ( rank card1 ) 'A )
      ( display ( rank card1 ) ) ( display " High" )
    )
    ( ( equal? ( rank card2 ) 'A )
      ( display ( rank card2 ) ) ( display " High" )
    )
    ( ( equal? ( rank card1 ) 'K )
      ( display ( rank card1 ) ) ( display " High" )
    )
    ( ( equal? ( rank card2 ) 'K )
      ( display ( rank card2 ) ) ( display " High" )
    )
    ( ( equal? ( rank card1 ) 'Q )
      ( display ( rank card1 ) ) ( display " High" )
    )
    ( ( equal? ( rank card2 ) 'Q )
      ( display ( rank card2 ) ) ( display " High" )
    )
    ( ( equal? ( rank card1 ) 'J )
      ( display ( rank card1 ) ) ( display " High" )
    )
    ( ( equal? ( rank card2 ) 'J )
      ( display ( rank card2 ) ) ( display " High" )
    )
    ( ( equal? ( rank card1 ) 'X )
      ( display ( rank card1 ) ) ( display " High" )
    )
    ( ( equal? ( rank card2 ) 'X )
```

```
      ( display ( rank card2 ) ) ( display " High" )
    )
    ( else
      ( cond
        ( ( > ( rank card1 ) ( rank card2 ) )
          ( display ( rank card1 ) ) ( display " High" )
        )
        ( ( < ( rank card1 ) ( rank card2 ) )
          ( display ( rank card2 ) ) ( display " High" )
        )
      )
    )
  )
  ( straight? card1 card2 )
  ( flush? card1 card2 )
)
; ---------------------------------------------------------------------------
; Straight Identifier
;
( define ( straight? card1 card2 )
  ( cond
    ( ( equal? ( rank card1 ) 'A )
      ( cond
        ( ( equal? ( rank card2 ) 'K )
          ( display " Straight" )
        )
      )
    )
    ( ( equal? ( rank card1 ) 'K )
      ( cond
        ( ( equal? ( rank card2 ) 'A )
          ( display " Straight" )
        )
        ( ( equal? ( rank card2 ) 'Q )
          ( display " Straight" )
        )
      )
    )
    ( ( equal? ( rank card1 ) 'Q )
      ( cond
        ( ( equal? ( rank card2 ) 'K )
          ( display " Straight" )
        )
        ( ( equal? ( rank card2 ) 'J )
          ( display " Straight" )
        )
      )
    )
    ( ( equal? ( rank card1 ) 'J )
      ( cond
        ( ( equal? ( rank card2 ) 'Q )
          ( display " Straight" )
        )
        ( ( equal? ( rank card2 ) 'X )
          ( display " Straight" )
        )
      )
    )
    ( ( equal? ( rank card1 ) 'X )
      ( cond
        ( ( equal? ( rank card2 ) 'J )
          ( display " Straight" )
        )
        ( ( equal? ( rank card2 ) "9" )
          ( display " Straight" )
        )
      )
    )
    ( ( equal? ( rank card1 ) '9 )
      ( cond
        ( ( equal? ( rank card2 ) 'X )
          ( display " Straight" )
        )
        ( ( equal? ( rank card2 ) '8 )
          ( display " Straight" )
        )
      )
    )
    ( ( equal? ( rank card1 ) '8 )
      ( cond
        ( ( equal? ( rank card2 ) '9 )
          ( display " Straight" )
        )
        ( ( equal? ( rank card2 ) '7 )
          ( display " Straight" )
        )
      )
    )
```

```
    ( ( equal? ( rank card1 ) '7 )
      ( cond
        ( ( equal? ( rank card2 ) '8 )
          ( display " Straight" )
        )
        ( ( equal? ( rank card2 ) '6 )
          ( display " Straight" )
        )
      )
    )
    ( ( equal? ( rank card1 ) '6 )
      ( cond
        ( ( equal? ( rank card2 ) '7 )
          ( display " Straight" )
        )
        ( ( equal? ( rank card2 ) '5 )
          ( display " Straight" )
        )
      )
    )
    ( ( equal? ( rank card1 ) '5 )
      ( cond
        ( ( equal? ( rank card2 ) '6 )
          ( display " Straight" )
        )
        ( ( equal? ( rank card2 ) '4 )
          ( display " Straight" )
        )
      )
    )
    ( ( equal? ( rank card1 ) '4 )
      ( cond
        ( ( equal? ( rank card2 ) '5 )
          ( display " Straight" )
        )
        ( ( equal? ( rank card2 ) '3 )
          ( display " Straight" )
        )
      )
    )
    ( ( equal? ( rank card1 ) '3 )
      ( cond
        ( ( equal? ( rank card2 ) '4 )
          ( display " Straight" )
        )
        ( ( equal? ( rank card2 ) '2 )
          ( display " Straight" )
        )
      )
    )
    ( ( equal? ( rank card1 ) '2 )
      ( cond
        ( ( equal? ( rank card2 ) '3 )
          ( display " Straight" )
        )
        ( ( equal? ( rank card2 ) '1 )
          ( display " Straight" )
        )
      )
    )
    ( ( equal? ( rank card1 ) '1 )
      ( cond
        ( ( equal? ( rank card2 ) '2 )
          ( display " Straight" )
        )
      )
    )
  )
)
; ------------------------------------------------------------------------
; Flush Identifier
;
( define ( flush? card1 card2 )
  ( cond
    ( ( equal? ( suit card1 ) ( suit card2 ) )
      ( display " Flush" )
    )
  )
)
;( trace higher-rank )
```

Task 4c: Two Card Poker – Classifier

```
Welcome to DrRacket, version 8.6 [cs].
Language: racket, with debugging; memory limit: 128 MB.
> ( classify-two-cards ( pick-two-cards ) )
((J H) (6 C)): Jack High
> ( classify-two-cards ( pick-two-cards ) )
((K C) (3 S)): King High
> ( classify-two-cards ( pick-two-cards ) )
((3 S) (K S)): King High Flush
> ( classify-two-cards ( pick-two-cards ) )
((J D) (X D)): Jack High Straight Flush
> ( classify-two-cards ( pick-two-cards ) )
((2 D) (Q S)): Queen High
> ( classify-two-cards ( pick-two-cards ) )
((4 D) (5 C)): Five High Straight
> ( classify-two-cards ( pick-two-cards ) )
((8 C) (5 S)): Eight High
> ( classify-two-cards ( pick-two-cards ) )
((J S) (9 C)): Jack High
> ( classify-two-cards ( pick-two-cards ) )
((8 D) (2 H)): Eight High
> ( classify-two-cards ( pick-two-cards ) )
((A C) (Q C)): Ace High Flush
> ( classify-two-cards ( pick-two-cards ) )
((4 S) (9 C)): Nine High
> ( classify-two-cards ( pick-two-cards ) )
((9 C) (X S)): Ten High Straight
> ( classify-two-cards ( pick-two-cards ) )
((4 H) (7 H)): Seven High Flush
> ( classify-two-cards ( pick-two-cards ) )
((6 D) (A C)): Ace High
> ( classify-two-cards ( pick-two-cards ) )
((8 C) (A D)): Ace High
> ( classify-two-cards ( pick-two-cards ) )
((7 H) (4 D)): Seven High
> ( classify-two-cards ( pick-two-cards ) )
((6 C) (6 S)): Pair of 6's
> ( classify-two-cards ( pick-two-cards ) )
((5 D) (9 C)): Nine High
> ( classify-two-cards ( pick-two-cards ) )
((3 S) (X S)): Ten High Flush
> ( classify-two-cards ( pick-two-cards ) )
((2 S) (8 C)): Eight High
>
```

Task 4c: Code

```
#lang racket
( require racket/trace )
; ---------------------------------------------------------------------
; EXAMPLE COMMANDS
;
; ( pick-two-cards )
; ( higher-rank ( pick-a-card ) ( pick-a-card ) )
; ( classify-two-cards-ur ( pick-two-cards ) )
; ( classify-two-cards ( pick-two-cards ) )

; ---------------------------------------------------------------------
; Preliminary Code
;
( 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 )
  )
)
; ---------------------------------------------------------------------
; Pick-two-cards Function
;
```

```
( define ( pick-two-cards )
  ( define card1 ( pick-a-card ) )
  ( define card2 ( pick-a-card ) )
  ( cond
    ( ( equal? card1 card2 )
      ( pick-two-cards )
    )
    ( else
      ( list card1 card2 )
    )
  )
)
; ---------------------------------------------------------------------
; Higher-rank Function
;
( define ( higher-rank card1 card2 )
  ( display ( list card1 card2 ) ) ( display "\n" )
  ( cond
    ( ( equal? ( rank card1 ) 'A )
      ( display ( rank card1 ) )
    )
    ( ( equal? ( rank card2 ) 'A )
      ( display ( rank card2 ) )
    )
    ( ( equal? ( rank card1 ) 'K )
      ( display ( rank card1 ) )
    )
    ( ( equal? ( rank card2 ) 'K )
      ( display ( rank card2 ) )
    )
    ( ( equal? ( rank card1 ) 'Q )
      ( display ( rank card1 ) )
    )
    ( ( equal? ( rank card2 ) 'Q )
      ( display ( rank card2 ) )
    )
    ( ( equal? ( rank card1 ) 'J )
      ( display ( rank card1 ) )
    )
    ( ( equal? ( rank card2 ) 'J )
      ( display ( rank card2 ) )
    )
    ( ( equal? ( rank card1 ) 'X )
      ( display ( rank card1 ) )
    )
    ( ( equal? ( rank card2 ) 'X )
      ( display ( rank card2 ) )
    )
    ( else
      ( cond
        ( ( equal? ( rank card1 ) ( rank card2 ) )
          (  rank card1 )
        )
        ( ( > ( rank card1 ) ( rank card2 ) )
          ( display ( rank card1 ) )
        )
        ( ( < ( rank card1 ) ( rank card2 ) )
          ( display ( rank card2 ) )
        )
      )
    )
  )
  ( straight? card1 card2 )
  ( flush? card1 card2 )
)
; ---------------------------------------------------------------------
; Classify-two-cards-ur Function
;
( define ( classify-two-cards-ur cards )
  ( display cards ) ( display ": " )
  ( define card1 ( first cards ) )
  ( define card2 ( second cards ) )
  ( cond
    ( ( equal? ( rank card1 ) ( rank card2 ) )
      ( display "Pair of " ) ( display ( rank card1 ) ) ( display "s" )
```

```
    )
    ( ( equal? ( rank card1 ) 'A )
      ( display ( rank card1 ) ) ( display " High" )
    )
    ( ( equal? ( rank card2 ) 'A )
      ( display ( rank card2 ) ) ( display " High" )
    )
    ( ( equal? ( rank card1 ) 'K )
      ( display ( rank card1 ) ) ( display " High" )
    )
    ( ( equal? ( rank card2 ) 'K )
      ( display ( rank card2 ) ) ( display " High" )
    )
    ( ( equal? ( rank card1 ) 'Q )
      ( display ( rank card1 ) ) ( display " High" )
    )
    ( ( equal? ( rank card2 ) 'Q )
      ( display ( rank card2 ) ) ( display " High" )
    )
    ( ( equal? ( rank card1 ) 'J )
      ( display ( rank card1 ) ) ( display " High" )
    )
    ( ( equal? ( rank card2 ) 'J )
      ( display ( rank card2 ) ) ( display " High" )
    )
    ( ( equal? ( rank card1 ) 'X )
      ( display ( rank card1 ) ) ( display " High" )
    )
    ( ( equal? ( rank card2 ) 'X )
      ( display ( rank card2 ) ) ( display " High" )
    )
    ( else
      ( cond
        ( ( > ( rank card1 ) ( rank card2 ) )
          ( display ( rank card1 ) ) ( display " High" )
        )
        ( ( < ( rank card1 ) ( rank card2 ) )
          ( display ( rank card2 ) ) ( display " High" )
        )
      )
    )
  )
  ( straight? card1 card2 )
  ( flush? card1 card2 )
)
; ---------------------------------------------------------------------
; Straight Identifier
;
( define ( straight? card1 card2 )
  ( cond
    ( ( equal? ( rank card1 ) 'A )
      ( cond
        ( ( equal? ( rank card2 ) 'K )
          ( display " Straight" )
        )
      )
    )
    ( ( equal? ( rank card1 ) 'K )
      ( cond
        ( ( equal? ( rank card2 ) 'A )
          ( display " Straight" )
        )
        ( ( equal? ( rank card2 ) 'Q )
          ( display " Straight" )
        )
      )
    )
    ( ( equal? ( rank card1 ) 'Q )
      ( cond
        ( ( equal? ( rank card2 ) 'K )
          ( display " Straight" )
        )
        ( ( equal? ( rank card2 ) 'J )
          ( display " Straight" )
        )
```

```
 )
)
( ( equal? ( rank card1 ) 'J )
 ( cond
   ( ( equal? ( rank card2 ) 'Q )
     ( display " Straight" )
   )
   ( ( equal? ( rank card2 ) 'X )
     ( display " Straight" )
   )
 )
)
( ( equal? ( rank card1 ) 'X )
 ( cond
   ( ( equal? ( rank card2 ) 'J )
     ( display " Straight" )
   )
   ( ( equal? ( rank card2 ) "9" )
     ( display " Straight" )
   )
 )
)
( ( equal? ( rank card1 ) '9 )
 ( cond
   ( ( equal? ( rank card2 ) 'X )
     ( display " Straight" )
   )
   ( ( equal? ( rank card2 ) '8 )
     ( display " Straight" )
   )
 )
)
( ( equal? ( rank card1 ) '8 )
 ( cond
   ( ( equal? ( rank card2 ) '9 )
     ( display " Straight" )
   )
   ( ( equal? ( rank card2 ) '7 )
     ( display " Straight" )
   )
 )
)
( ( equal? ( rank card1 ) '7 )
 ( cond
   ( ( equal? ( rank card2 ) '8 )
     ( display " Straight" )
   )
   ( ( equal? ( rank card2 ) '6 )
     ( display " Straight" )
   )
 )
)
( ( equal? ( rank card1 ) '6 )
 ( cond
   ( ( equal? ( rank card2 ) '7 )
     ( display " Straight" )
   )
   ( ( equal? ( rank card2 ) '5 )
     ( display " Straight" )
   )
 )
)
( ( equal? ( rank card1 ) '5 )
 ( cond
   ( ( equal? ( rank card2 ) '6 )
     ( display " Straight" )
   )
   ( ( equal? ( rank card2 ) '4 )
     ( display " Straight" )
   )
 )
)
( ( equal? ( rank card1 ) '4 )
 ( cond
   ( ( equal? ( rank card2 ) '5 )
```

```
          ( display " Straight" )
        )
        ( ( equal? ( rank card2 ) '3 )
          ( display " Straight" )
        )
      )
    )
    ( ( equal? ( rank card1 ) '3 )
      ( cond
        ( ( equal? ( rank card2 ) '4 )
          ( display " Straight" )
        )
        ( ( equal? ( rank card2 ) '2 )
          ( display " Straight" )
        )
      )
    )
    ( ( equal? ( rank card1 ) '2 )
      ( cond
        ( ( equal? ( rank card2 ) '3 )
          ( display " Straight" )
        )
        ( ( equal? ( rank card2 ) '1 )
          ( display " Straight" )
        )
      )
    )
    ( ( equal? ( rank card1 ) '1 )
      ( cond
        ( ( equal? ( rank card2 ) '2 )
          ( display " Straight" )
        )
      )
    )
  )
)
; ---------------------------------------------------------------------------
; Flush Identifier
;
( define ( flush? card1 card2 )
  ( cond
    ( ( equal? ( suit card1 ) ( suit card2 ) )
      ( display " Flush" )
    )
  )
)

( define ( pair? card1 card2 )
  ( equal? ( rank card1 ) ( rank card2 ) )
)
; ---------------------------------------------------------------------------
; Classify-two-cards Function
;
( define ( classify-two-cards cards )
  ( display cards ) ( display ": " )
  ( define card1 ( first cards ) )
  ( define card2 ( second cards ) )
  ( cond
    ( ( equal? ( rank card1 ) ( rank card2 ) )
      ( display "Pair of " ) ( display ( rank card1 ) ) ( display "'s" )
    )
    ( ( equal? ( rank card1 ) 'A )
      ( display "Ace High" )
    )
    ( ( equal? ( rank card2 ) 'A )
      ( display "Ace High" )
    )
    ( ( equal? ( rank card1 ) 'K )
      ( display "King High" )
    )
    ( ( equal? ( rank card2 ) 'K )
      ( display "King High" )
    )
    ( ( equal? ( rank card1 ) 'Q )
      ( display "Queen High" )
```

```
    )
    ( ( equal? ( rank card2 ) 'Q )
      ( display "Queen High" )
    )
    ( ( equal? ( rank card1 ) 'J )
      ( display "Jack High" )
    )
    ( ( equal? ( rank card2 ) 'J )
      ( display "Jack High" )
    )
    ( ( equal? ( rank card1 ) 'X )
      ( display "Ten High" )
    )
    ( ( equal? ( rank card2 ) 'X )
      ( display "Ten High" )
    )
    ( ( equal? ( rank card1 ) '9 )
      ( display "Nine High" )
    )
    ( ( equal? ( rank card2 ) '9 )
      ( display "Nine High" )
    )
    ( ( equal? ( rank card1 ) '8 )
      ( display "Eight High" )
    )
    ( ( equal? ( rank card2 ) '8 )
      ( display "Eight High" )
    )
    ( ( equal? ( rank card1 ) '7 )
      ( display "Seven High" )
    )
    ( ( equal? ( rank card2 ) '7 )
      ( display "Seven High" )
    )
    ( ( equal? ( rank card1 ) '6 )
      ( display "Six High" )
    )
    ( ( equal? ( rank card2 ) '6 )
      ( display "Six High" )
    )
    ( ( equal? ( rank card1 ) '5 )
      ( display "Five High" )
    )
    ( ( equal? ( rank card2 ) '5 )
      ( display "Five High" )
    )
    ( ( equal? ( rank card1 ) '4 )
      ( display "Four High" )
    )
    ( ( equal? ( rank card2 ) '4 )
      ( display "Four High" )
    )
    ( ( equal? ( rank card1 ) '3 )
      ( display "Three High" )
    )
    ( ( equal? ( rank card2 ) '3 )
      ( display "Three High" )
    )
    ( ( equal? ( rank card1 ) '2 )
      ( display "Two High" )
    )
    ( ( equal? ( rank card2 ) '2 )
      ( display "Two High" )
    )
    ( ( equal? ( rank card1 ) '1 )
      ( display "One High" )
    )
    ( ( equal? ( rank card2 ) '1 )
      ( display "One High" )
    )
  )
  ( straight? card1 card2 )
  ( flush? card1 card2 )
)
; ( trace higher-rank )
```