Racket Programming Assignment #3: Lambda and Basic Lisp

Learning Abstract:

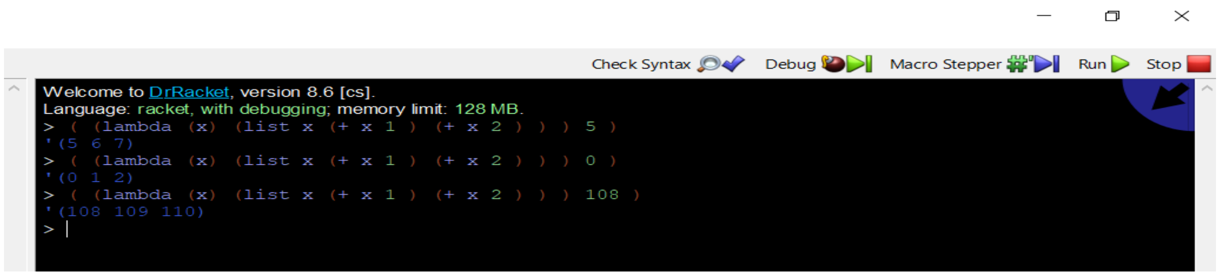
The first assignment required using small undefined anonymous lambda functions and mimicking the demos that were provided

The second assignment required us to produce the output for the functions we were given in lesson 6

The third assignment required us to recreate a program called sampler and use that code as a base for a program called color thing. The sampler program would take input and then output a random element from the list that was just inputted. The color program used our sampler code for its base, but this program would select a color from the given list and then output a rectangle of a specified size with that color.

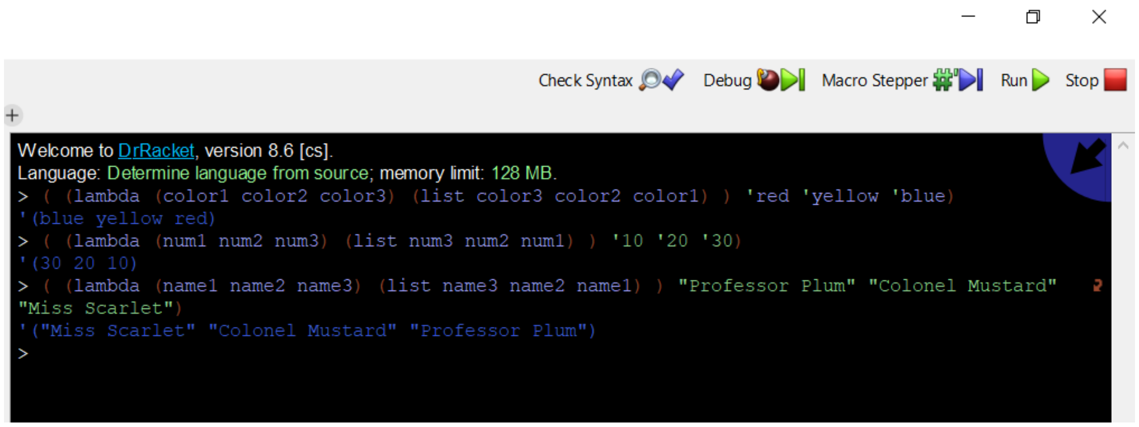
The fourth assignment required us to create several programs for poker. We were given a initial piece of code and was asked to modify it as we proceeded through the tasks. The functions that would be added were to pick two cards, determine a higher rank, classify two cards, and classify two cards while determining which is higher.

Task 1 Lambda:



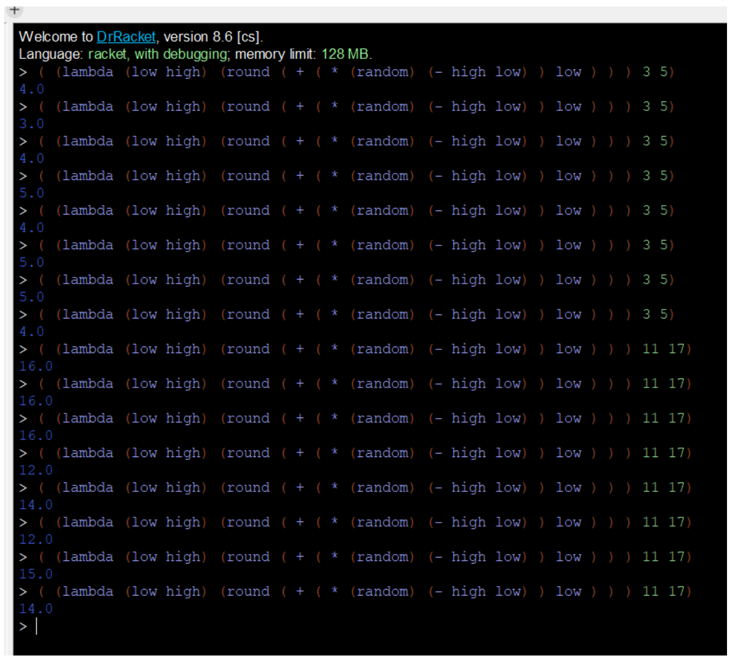
```
Welcome to DrRacket, version 8.6 [cs].
Language: racket, with debugging; memory limit: 128 MB.
> ( (lambda (x) (list x (+ x 1 ) (+ x 2 ) ) ) 5 )
'(5 6 7)
> ( (lambda (x) (list x (+ x 1 ) (+ x 2 ) ) ) 0 )
'(0 1 2)
> ( (lambda (x) (list x (+ x 1 ) (+ x 2 ) ) ) 108 )
'(108 109 110)
> |
```

Task 1a code



```
Welcome to DrRacket, version 8.6 [cs].
Language: Determine language from source; memory limit: 128 MB.
> ( (lambda (color1 color2 color3) (list color3 color2 color1) ) 'red 'yellow 'blue)
'(blue yellow red)
> ( (lambda (num1 num2 num3) (list num3 num2 num1) ) '10 '20 '30)
'(30 20 10)
> ( (lambda (name1 name2 name3) (list name3 name2 name1) ) "Professor Plum" "Colonel Mustard"
"Miss Scarlet")
'("Miss Scarlet" "Colonel Mustard" "Professor Plum")
>
```

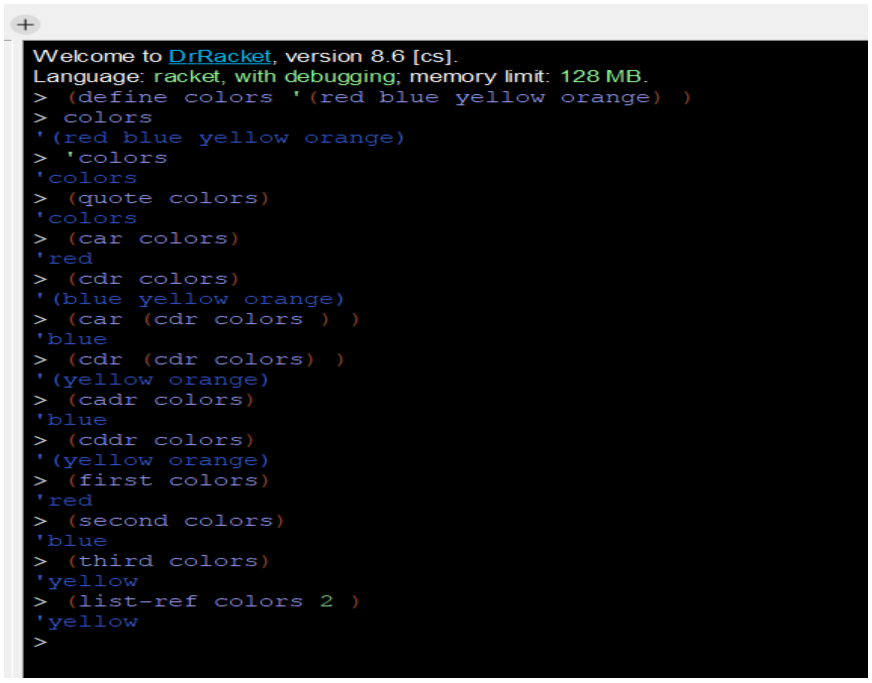
Task 1b code



```
Welcome to DrRacket, version 8.6 [cs].
Language: racket, with debugging; memory limit: 128 MB.
> ( (lambda (low high) (round ( + ( * (random) (- high low) ) low ) ) ) 3 5)
4.0
> ( (lambda (low high) (round ( + ( * (random) (- high low) ) low ) ) ) 3 5)
3.0
> ( (lambda (low high) (round ( + ( * (random) (- high low) ) low ) ) ) 3 5)
4.0
> ( (lambda (low high) (round ( + ( * (random) (- high low) ) low ) ) ) 3 5)
5.0
> ( (lambda (low high) (round ( + ( * (random) (- high low) ) low ) ) ) 3 5)
4.0
> ( (lambda (low high) (round ( + ( * (random) (- high low) ) low ) ) ) 3 5)
5.0
> ( (lambda (low high) (round ( + ( * (random) (- high low) ) low ) ) ) 3 5)
5.0
> ( (lambda (low high) (round ( + ( * (random) (- high low) ) low ) ) ) 3 5)
4.0
> ( (lambda (low high) (round ( + ( * (random) (- high low) ) low ) ) ) 11 17)
16.0
> ( (lambda (low high) (round ( + ( * (random) (- high low) ) low ) ) ) 11 17)
16.0
> ( (lambda (low high) (round ( + ( * (random) (- high low) ) low ) ) ) 11 17)
16.0
> ( (lambda (low high) (round ( + ( * (random) (- high low) ) low ) ) ) 11 17)
12.0
> ( (lambda (low high) (round ( + ( * (random) (- high low) ) low ) ) ) 11 17)
14.0
> ( (lambda (low high) (round ( + ( * (random) (- high low) ) low ) ) ) 11 17)
12.0
> ( (lambda (low high) (round ( + ( * (random) (- high low) ) low ) ) ) 11 17)
15.0
> ( (lambda (low high) (round ( + ( * (random) (- high low) ) low ) ) ) 11 17)
14.0
> |
```

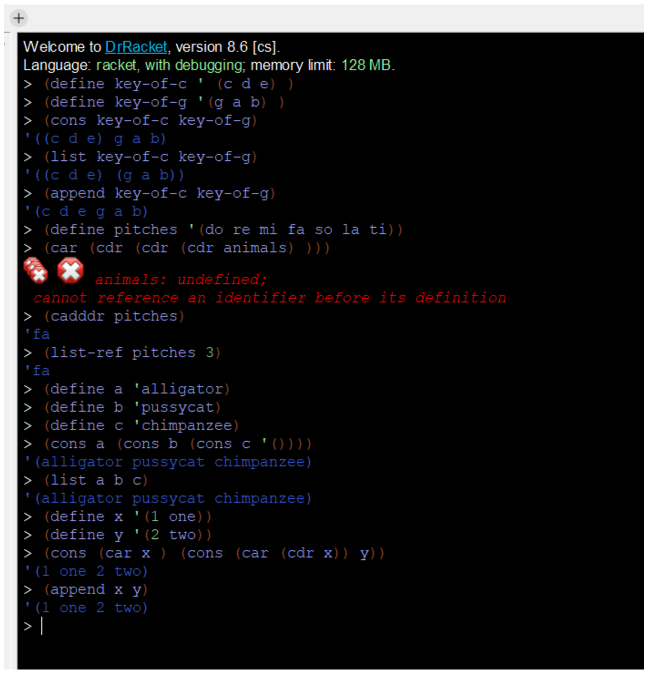
Task 1c code

Task 2 List Processing References 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
>
```

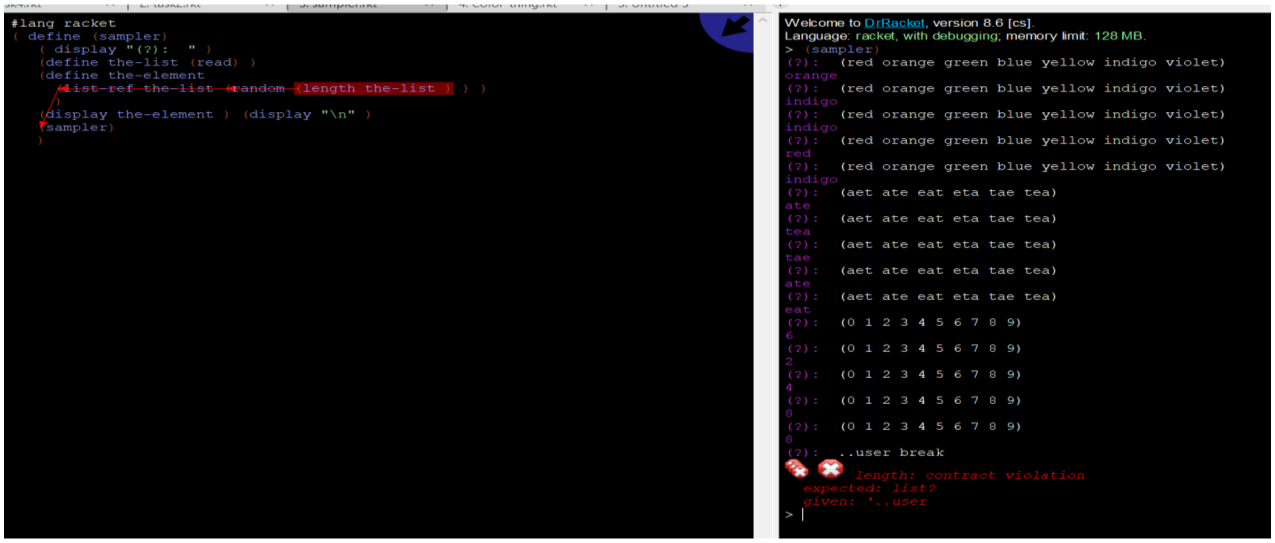
Task 2 part 1



```
Welcome to DrRacket, version 8.6 [cs].
Language: racket, with debugging; memory limit: 128 MB.
> (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 2 Part 2

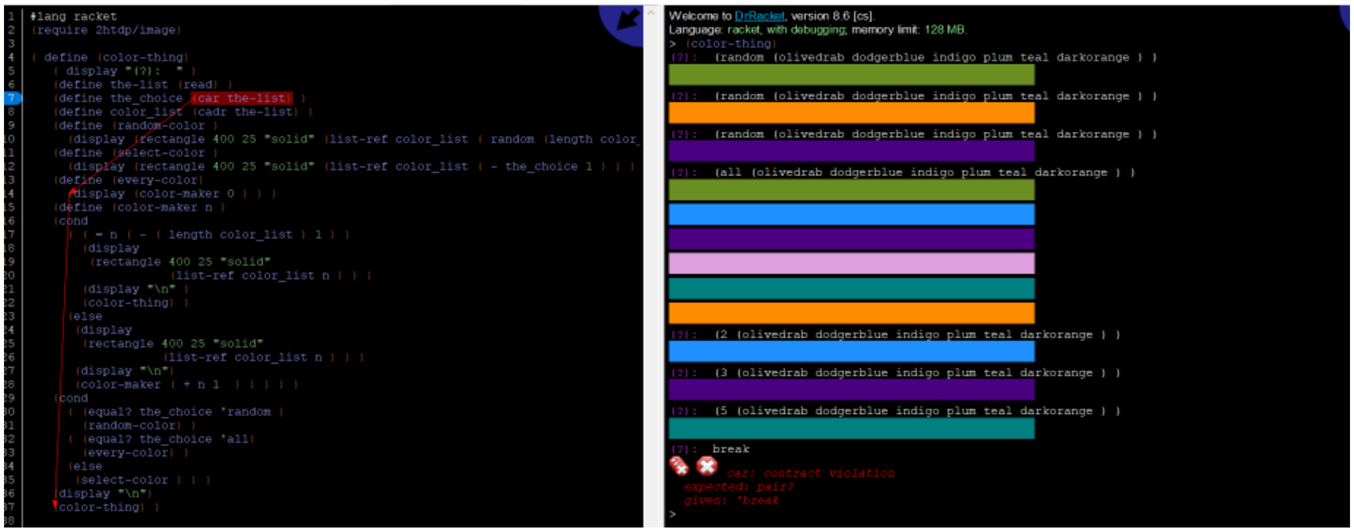
Task 3 Little Color Interpreter



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

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

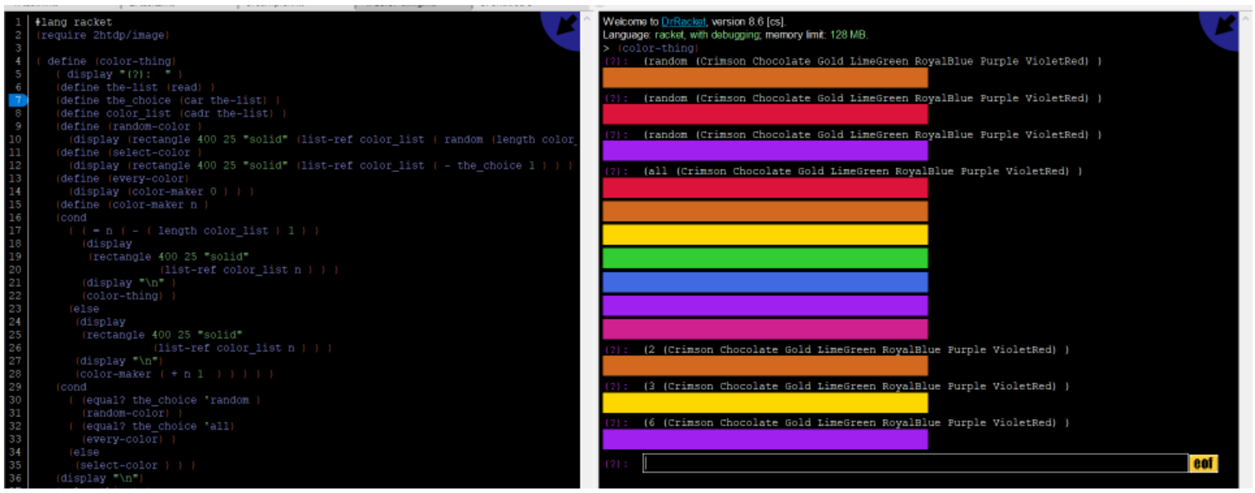
Task 3A code and demo



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

( define (color-thing)
   ( display "(?):  " )
   (define the-list (read) )
   (define the_choice (car the-list) )
   (define color_list (cadr the-list) )
   (define (random-color )
     (display (rectangle 400 25 "solid" (list-ref color_list ( random (length color_
   (define (select-color )
     (display (rectangle 400 25 "solid" (list-ref color_list ( - the_choice 1 ) ) ) )
   (define (every-color)
     (display (color-maker 0 ) ) ) )
   (define (color-maker n )
   (cond
     ( ( = n ( - ( length color_list ) 1 ) ) )
       (display
        (rectangle 400 25 "solid"
                   (list-ref color_list n ) ) ) )
       (display "\n" )
       (color-thing) )
     (else
      (display
       (rectangle 400 25 "solid"
                  (list-ref color_list n ) ) ) )
      (display "\n")
      (color-maker ( + n 1  ) ) ) ) ) )
   (cond
     ( (equal? the_choice 'random )
       (random-color) )
     ( (equal? the_choice 'all)
       (every-color) )
     (else
      (select-color ) ) ) )
   (display "\n")
   (color-thing) )
```

```
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 ) )
(?):  break
    car: contract violation
  expected: pair?
  given: 'break
>
```

Task 3B code and demo 1

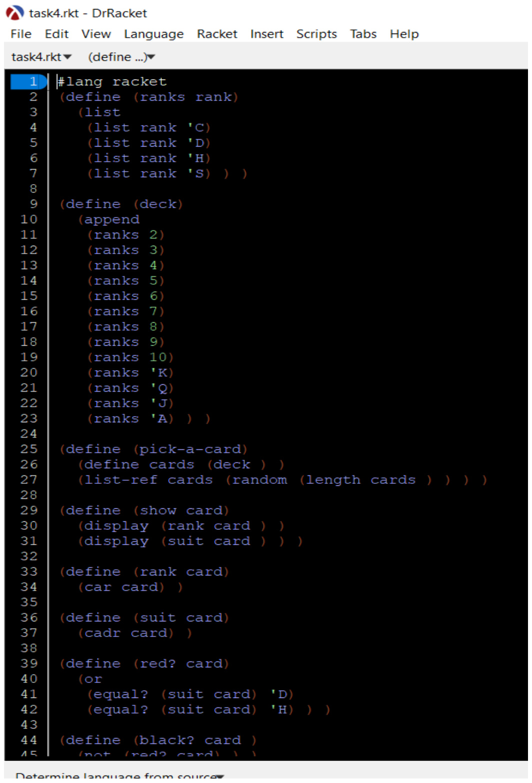


```
#lang racket
(require 2htdp/image)

( define (color-thing)
   ( display "(?):  " )
   (define the-list (read) )
   (define the_choice (car the-list) )
   (define color_list (cadr the-list) )
   (define (random-color )
     (display (rectangle 400 25 "solid" (list-ref color_list ( random (length color_
   (define (select-color )
     (display (rectangle 400 25 "solid" (list-ref color_list ( - the_choice 1 ) ) ) )
   (define (every-color)
     (display (color-maker 0 ) ) )
   (define (color-maker n )
   (cond
     ( ( = n ( - ( length color_list ) 1 ) )
       (display
        (rectangle 400 25 "solid"
                   (list-ref color_list n ) ) )
       (display "\n" )
       (color-thing) )
     (else
      (display
       (rectangle 400 25 "solid"
                  (list-ref color_list n ) ) )
      (display "\n")
      (color-maker ( + n 1  ) ) ) ) ) )
   (cond
     ( (equal? the_choice 'random )
       (random-color) )
     ( (equal? the_choice 'all)
       (every-color) )
     (else
      (select-color ) ) )
   (display "\n")
```

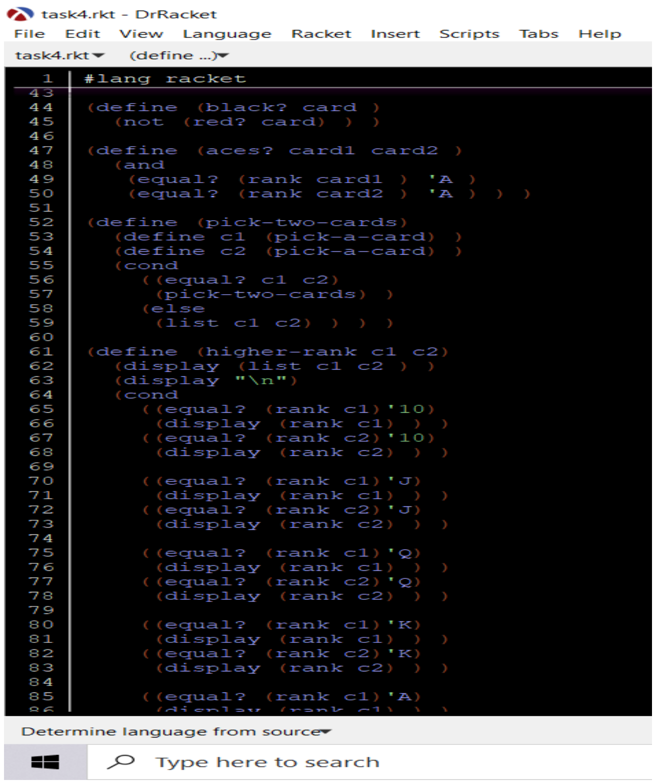
```
Welcome to DrRacket, version 8.6 [cs].
Language: racket, with debugging; memory limit: 128 MB.
> (color-thing)
(?):  (random (Crimson Chocolate Gold LimeGreen RoyalBlue Purple VioletRed) )

(?):  (random (Crimson Chocolate Gold LimeGreen RoyalBlue Purple VioletRed) )

(?):  (random (Crimson Chocolate Gold LimeGreen RoyalBlue Purple VioletRed) )

(?):  (all (Crimson Chocolate Gold LimeGreen RoyalBlue Purple VioletRed) )







(?):  (2 (Crimson Chocolate Gold LimeGreen RoyalBlue Purple VioletRed) )

(?):  (3 (Crimson Chocolate Gold LimeGreen RoyalBlue Purple VioletRed) )

(?):  (6 (Crimson Chocolate Gold LimeGreen RoyalBlue Purple VioletRed) )

(?):
```

Task 3B code and Demo 2

Task 4 Two Card Poker

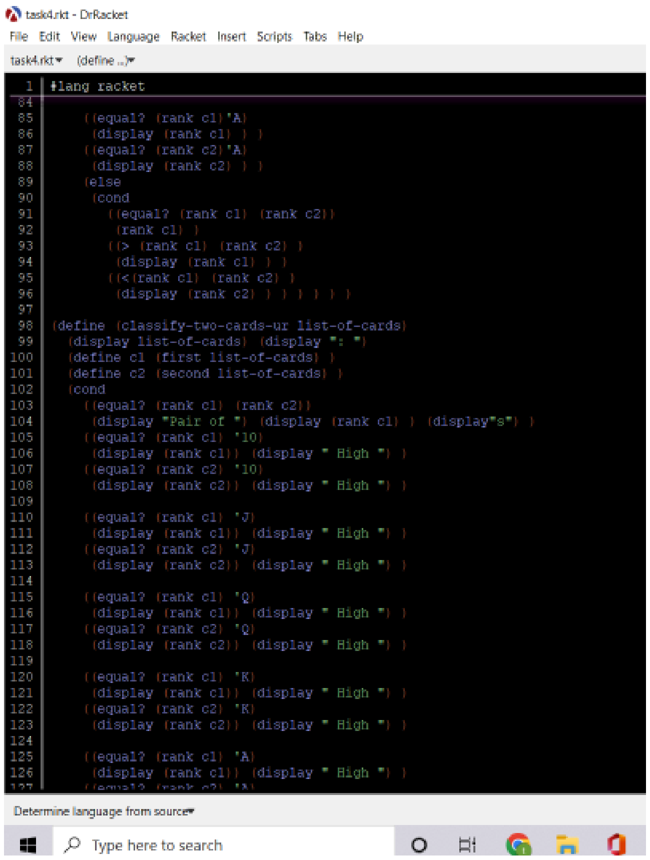


```
#lang racket
(define (ranks rank)
  (list
   (list rank 'C)
   (list rank 'D)
   (list rank 'H)
   (list rank 'S) ) )

(define (deck)
  (append
   (ranks 2)
   (ranks 3)
   (ranks 4)
   (ranks 5)
   (ranks 6)
   (ranks 7)
   (ranks 8)
   (ranks 9)
   (ranks 10)
   (ranks 'K)
   (ranks 'Q)
   (ranks 'J)
   (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) ) )
```

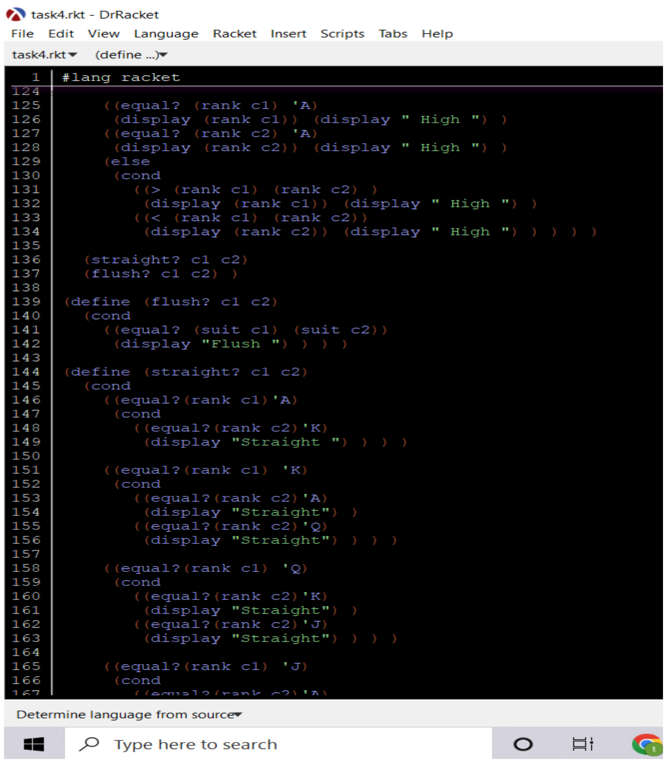


```
#lang racket

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

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

(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 (higher-rank c1 c2)
  (display (list c1 c2 ) )
  (display "\n")
  (cond
    ((equal? (rank c1)'10)
     (display (rank c1) ) )
    ((equal? (rank c2)'10)
     (display (rank c2) ) )

    ((equal? (rank c1)'J)
     (display (rank c1) ) )
    ((equal? (rank c2)'J)
     (display (rank c2) ) )

    ((equal? (rank c1)'Q)
     (display (rank c1) ) )
    ((equal? (rank c2)'Q)
     (display (rank c2) ) )

    ((equal? (rank c1)'K)
     (display (rank c1) ) )
    ((equal? (rank c2)'K)
     (display (rank c2) ) )

    ((equal? (rank c1)'A)
```

Determine language from source
Type here to search

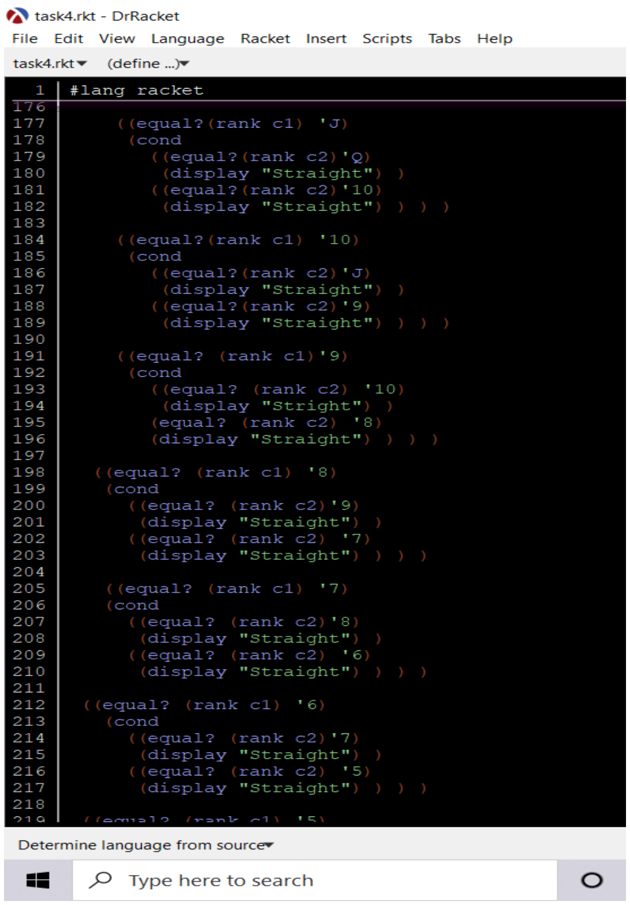

task4.rkt - DrRacket
File Edit View Language Racket Insert Scripts Tabs Help
task4.rkt (define ...)

```
#lang racket

    ((equal? (rank c1)'A)
     (display (rank c1) ) )
    ((equal? (rank c2)'A)
     (display (rank c2) ) )
    (else
     (cond
       ((equal? (rank c1) (rank c2))
        (rank c1) )
       ((> (rank c1) (rank c2) )
        (display (rank c1) ) )
       ((<(rank c1) (rank c2) )
        (display (rank c2) ) ) ) ) ) )

(define (classify-two-cards-ur list-of-cards)
  (display list-of-cards) (display ": ")
  (define c1 (first list-of-cards) )
  (define c2 (second list-of-cards) )
  (cond
    ((equal? (rank c1) (rank c2))
     (display "Pair of ") (display (rank c1) ) (display"s") )
    ((equal? (rank c1) '10)
     (display (rank c1)) (display " High ") )
    ((equal? (rank c2) '10)
     (display (rank c2)) (display " High ") )

    ((equal? (rank c1) 'J)
     (display (rank c1)) (display " High ") )
    ((equal? (rank c2) 'J)
     (display (rank c2)) (display " High ") )

    ((equal? (rank c1) 'Q)
     (display (rank c1)) (display " High ") )
    ((equal? (rank c2) 'Q)
     (display (rank c2)) (display " High ") )

    ((equal? (rank c1) 'K)
     (display (rank c1)) (display " High ") )
    ((equal? (rank c2) 'K)
     (display (rank c2)) (display " High ") )

    ((equal? (rank c1) 'A)
     (display (rank c1)) (display " High ") )
```

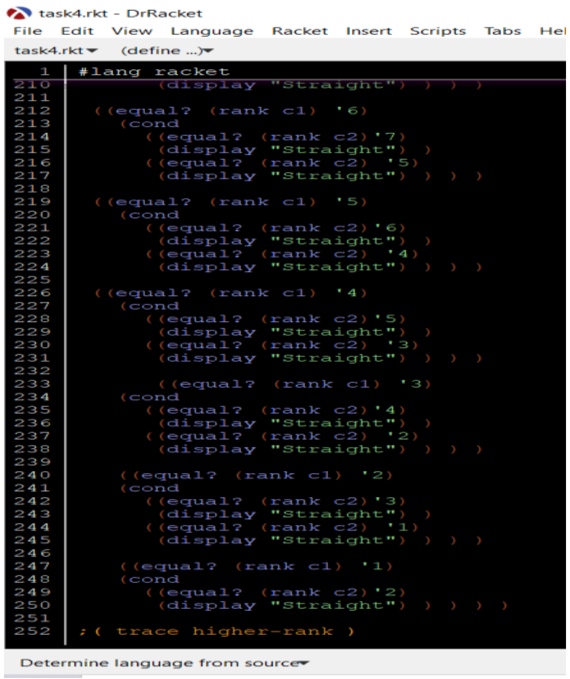


```racket
#lang racket

     ((equal? (rank c1) 'A)
      (display (rank c1)) (display " High ") )
     ((equal? (rank c2) 'A)
      (display (rank c2)) (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 (flush? c1 c2)
   (cond
     ((equal? (suit c1) (suit c2))
      (display "Flush ") ) ) )

(define (straight? c1 c2)
   (cond
     ((equal?(rank c1)'A)
      (cond
        ((equal?(rank c2)'K)
         (display "Straight ") ) ) )

     ((equal?(rank c1) 'K)
      (cond
        ((equal?(rank c2)'A)
         (display "Straight") )
        ((equal?(rank c2)'Q)
         (display "Straight") ) ) )

     ((equal?(rank c1) 'Q)
      (cond
        ((equal?(rank c2)'K)
         (display "Straight") )
        ((equal?(rank c2)'J)
         (display "Straight") ) ) )

     ((equal?(rank c1) 'J)
      (cond
```

Determine language from source
Type here to search

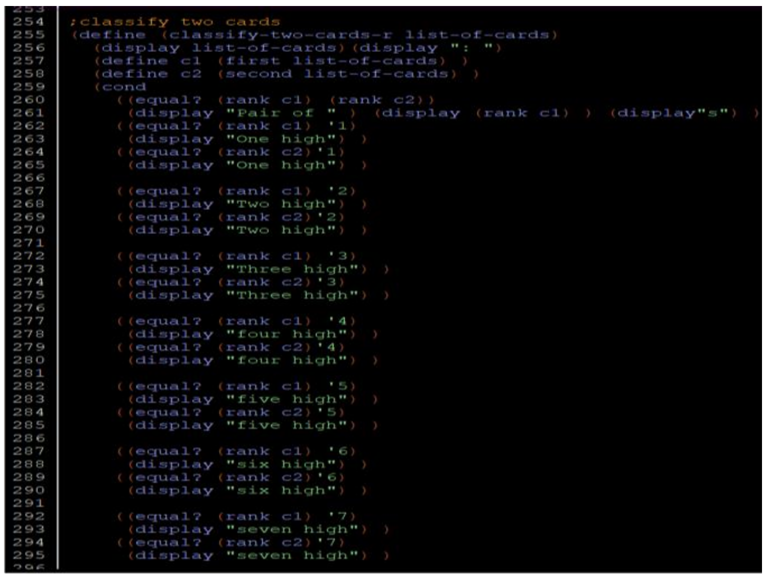

task4.rkt - DrRacket
File Edit View Language Racket Insert Scripts Tabs Help
task4.rkt (define ...)

```racket
#lang racket

     ((equal?(rank c1) 'J)
      (cond
        ((equal?(rank c2)'Q)
         (display "Straight") )
        ((equal?(rank c2)'10)
         (display "Straight") ) ) )

     ((equal?(rank c1) '10)
      (cond
        ((equal?(rank c2)'J)
         (display "Straight") )
        ((equal?(rank c2)'9)
         (display "Straight") ) ) )

     ((equal? (rank c1)'9)
      (cond
        ((equal? (rank c2) '10)
         (display "Stright") )
        (equal? (rank c2) '8)
        (display "Straight") ) ) )

   ((equal? (rank c1) '8)
    (cond
      ((equal? (rank c2)'9)
       (display "Straight") )
      ((equal? (rank c2) '7)
       (display "Straight") ) ) )

    ((equal? (rank c1) '7)
    (cond
      ((equal? (rank c2)'8)
       (display "Straight") )
      ((equal? (rank c2) '6)
       (display "Straight") ) ) )

  ((equal? (rank c1) '6)
    (cond
      ((equal? (rank c2)'7)
       (display "Straight") )
      ((equal? (rank c2) '5)
       (display "Straight") ) ) )
```

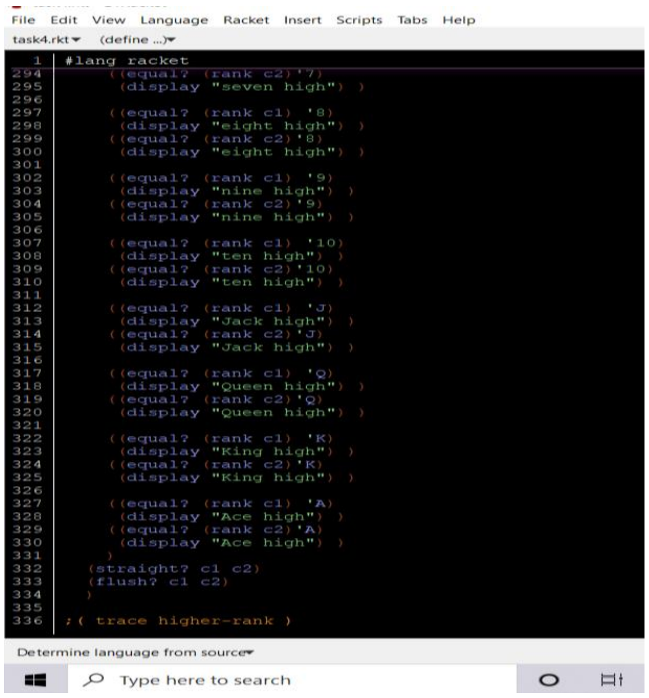

task4.rkt - DrRacket

```
#lang racket
          (display "Straight") ) ) )

  ((equal? (rank c1) '6)
    (cond
      ((equal? (rank c2)'7)
       (display "Straight") )
      ((equal? (rank c2) '5)
       (display "Straight") ) ) )

  ((equal? (rank c1) '5)
    (cond
      ((equal? (rank c2)'6)
       (display "Straight") )
      ((equal? (rank c2) '4)
       (display "Straight") ) ) )

  ((equal? (rank c1) '4)
    (cond
      ((equal? (rank c2)'5)
       (display "Straight") )
      ((equal? (rank c2) '3)
       (display "Straight") ) ) )

       ((equal? (rank c1) '3)
    (cond
      ((equal? (rank c2)'4)
       (display "Straight") )
      ((equal? (rank c2) '2)
       (display "Straight") ) ) )

    ((equal? (rank c1) '2)
    (cond
      ((equal? (rank c2)'3)
       (display "Straight") )
      ((equal? (rank c2) '1)
       (display "Straight") ) ) )

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

;( trace higher-rank )
```

Determine language from source

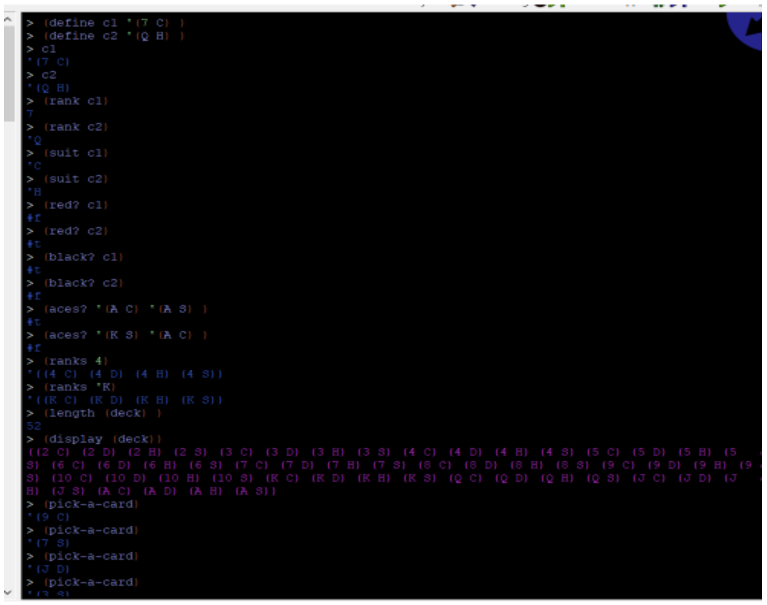


```
;classify two cards
(define (classify-two-cards-r list-of-cards)
  (display list-of-cards)(display ": ")
  (define c1 (first list-of-cards) )
  (define c2 (second list-of-cards) )
  (cond
    ((equal? (rank c1) (rank c2))
     (display "Pair of " ) (display (rank c1) ) (display"s") )
    ((equal? (rank c1) '1)
     (display "One high") )
    ((equal? (rank c2)'1)
     (display "One high") )

    ((equal? (rank c1) '2)
     (display "Two high") )
    ((equal? (rank c2)'2)
     (display "Two high") )

    ((equal? (rank c1) '3)
     (display "Three high") )
    ((equal? (rank c2)'3)
     (display "Three high") )

    ((equal? (rank c1) '4)
     (display "four high") )
    ((equal? (rank c2)'4)
     (display "four high") )

    ((equal? (rank c1) '5)
     (display "five high") )
    ((equal? (rank c2)'5)
     (display "five high") )

    ((equal? (rank c1) '6)
     (display "six high") )
    ((equal? (rank c2)'6)
     (display "six high") )

    ((equal? (rank c1) '7)
     (display "seven high") )
    ((equal? (rank c2)'7)
     (display "seven high") )
```


File Edit View Language Racket Insert Scripts Tabs Help
task4.rkt (define ...)
Determine language from source
Type here to search

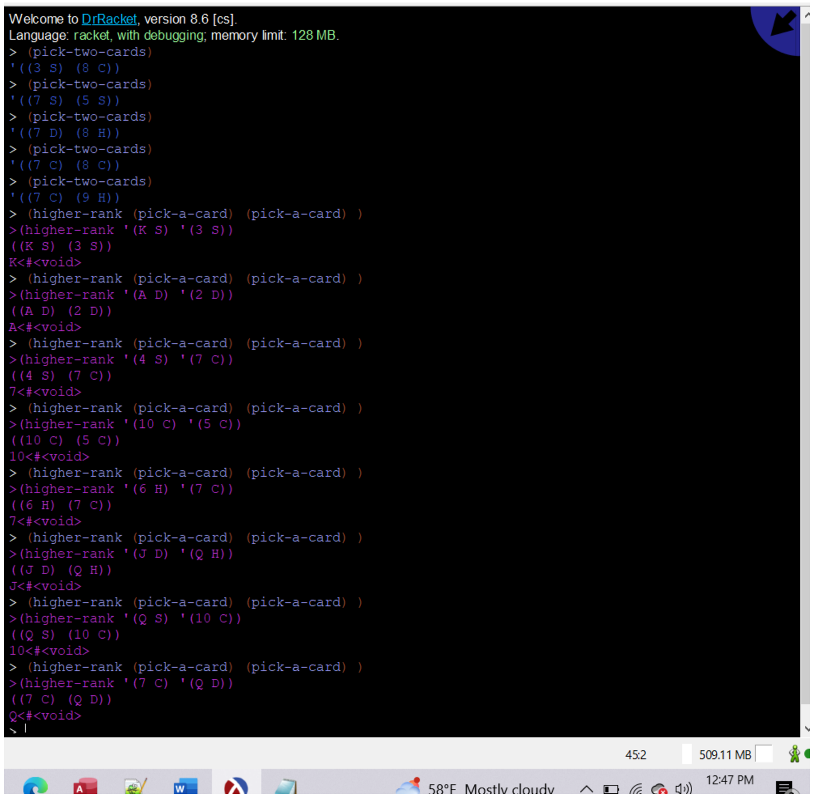
```
#lang racket
      ((equal? (rank c2) '7)
       (display "seven high") )

      ((equal? (rank c1)  '8)
       (display "eight high") )
      ((equal? (rank c2) '8)
       (display "eight high") )

      ((equal? (rank c1)  '9)
       (display "nine high") )
      ((equal? (rank c2) '9)
       (display "nine high") )

      ((equal? (rank c1)  '10)
       (display "ten high") )
      ((equal? (rank c2) '10)
       (display "ten high") )

      ((equal? (rank c1)  'J)
       (display "Jack high") )
      ((equal? (rank c2) 'J)
       (display "Jack high") )

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

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

      ((equal? (rank c1)  'A)
       (display "Ace high") )
      ((equal? (rank c2) 'A)
       (display "Ace high") )
      )
    (straight? c1 c2)
    (flush? c1 c2)
    )

;( trace higher-rank )
```

These pictures contain the code for task 4A, 4B and 4C



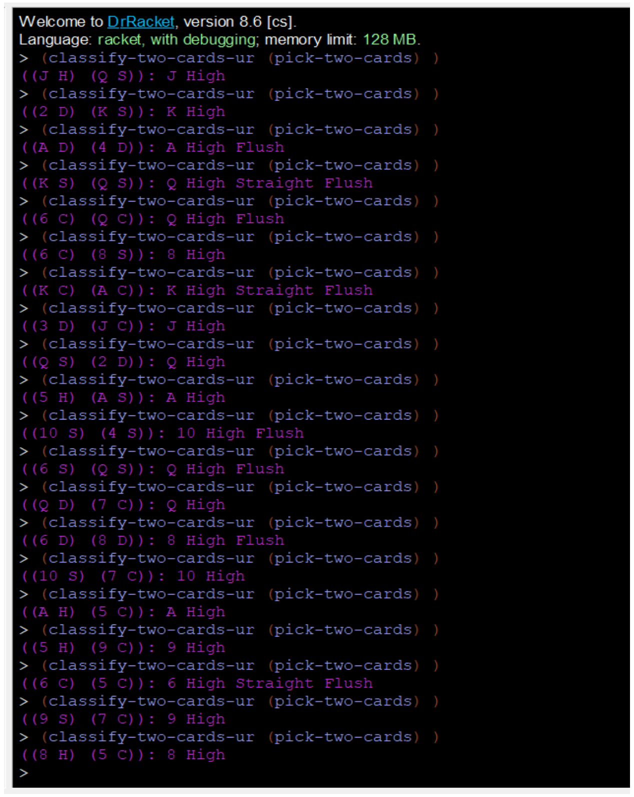
```
> (define c1 '(7 C) )
> (define c2 '(Q H) )
> c1
'(7 C)
> c2
'(Q H)
> (rank c1)
7
> (rank c2)
'Q
> (suit c1)
'C
> (suit c2)
'H
> (red? c1)
#f
> (red? c2)
#t
> (black? c1)
#t
> (black? c2)
#f
> (aces? '(A C) '(A S) )
#t
> (aces? '(K S) '(A C) )
#f
> (ranks 4)
'((4 C) (4 D) (4 H) (4 S))
> (ranks 'K)
'((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) (10 C) (10 D) (10 H) (10 S) (K C) (K D) (K H) (K S) (Q C) (Q D) (Q H) (Q S) (J C) (J D) (J
H) (J S) (A C) (A D) (A H) (A S))
> (pick-a-card)
'(9 C)
> (pick-a-card)
'(7 S)
> (pick-a-card)
'(J D)
> (pick-a-card)
'(3 S)
```

Task 4A Demo



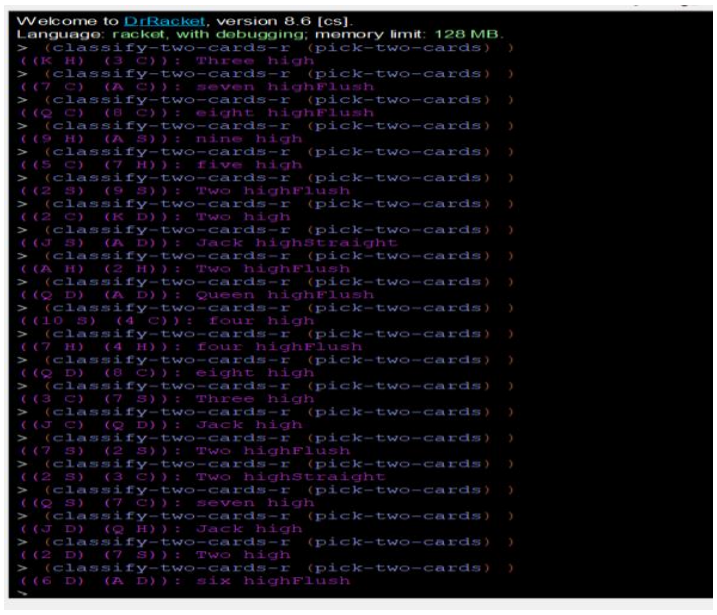
```
Welcome to DrRacket, version 8.6 [cs].
Language: racket, with debugging; memory limit: 128 MB.
> (pick-two-cards)
'((3 S) (8 C))
> (pick-two-cards)
'((7 S) (5 S))
> (pick-two-cards)
'((7 D) (8 H))
> (pick-two-cards)
'((7 C) (8 C))
> (pick-two-cards)
'((7 C) (9 H))
> (higher-rank (pick-a-card) (pick-a-card) )
>(higher-rank '(K S) '(3 S))
((K S) (3 S))
K<#<void>
> (higher-rank (pick-a-card) (pick-a-card) )
>(higher-rank '(A D) '(2 D))
((A D) (2 D))
A<#<void>
> (higher-rank (pick-a-card) (pick-a-card) )
>(higher-rank '(4 S) '(7 C))
((4 S) (7 C))
7<#<void>
> (higher-rank (pick-a-card) (pick-a-card) )
>(higher-rank '(10 C) '(5 C))
((10 C) (5 C))
10<#<void>
> (higher-rank (pick-a-card) (pick-a-card) )
>(higher-rank '(6 H) '(7 C))
((6 H) (7 C))
7<#<void>
> (higher-rank (pick-a-card) (pick-a-card) )
>(higher-rank '(J D) '(Q H))
((J D) (Q H))
J<#<void>
> (higher-rank (pick-a-card) (pick-a-card) )
>(higher-rank '(Q S) '(10 C))
((Q S) (10 C))
10<#<void>
> (higher-rank (pick-a-card) (pick-a-card) )
>(higher-rank '(7 C) '(Q D))
((7 C) (Q D))
Q<#<void>
>
```

Task 4B Pick two cards and Higher Rank demo



```
Welcome to DrRacket, version 8.6 [cs].
Language: racket, with debugging; memory limit: 128 MB.
> (classify-two-cards-ur (pick-two-cards) )
((J H) (Q S)): J High
> (classify-two-cards-ur (pick-two-cards) )
((2 D) (K S)): K High
> (classify-two-cards-ur (pick-two-cards) )
((A D) (4 D)): A High Flush
> (classify-two-cards-ur (pick-two-cards) )
((K S) (Q S)): Q High Straight Flush
> (classify-two-cards-ur (pick-two-cards) )
((6 C) (Q C)): Q High Flush
> (classify-two-cards-ur (pick-two-cards) )
((6 C) (8 S)): 8 High
> (classify-two-cards-ur (pick-two-cards) )
((K C) (A C)): K High Straight Flush
> (classify-two-cards-ur (pick-two-cards) )
((3 D) (J C)): J High
> (classify-two-cards-ur (pick-two-cards) )
((Q S) (2 D)): Q High
> (classify-two-cards-ur (pick-two-cards) )
((5 H) (A S)): A High
> (classify-two-cards-ur (pick-two-cards) )
((10 S) (4 S)): 10 High Flush
> (classify-two-cards-ur (pick-two-cards) )
((6 S) (Q S)): Q High Flush
> (classify-two-cards-ur (pick-two-cards) )
((Q D) (7 C)): Q High
> (classify-two-cards-ur (pick-two-cards) )
((6 D) (8 D)): 8 High Flush
> (classify-two-cards-ur (pick-two-cards) )
((10 S) (7 C)): 10 High
> (classify-two-cards-ur (pick-two-cards) )
((A H) (5 C)): A High
> (classify-two-cards-ur (pick-two-cards) )
((5 H) (9 C)): 9 High
> (classify-two-cards-ur (pick-two-cards) )
((6 C) (5 C)): 6 High Straight Flush
> (classify-two-cards-ur (pick-two-cards) )
((9 S) (7 C)): 9 High
> (classify-two-cards-ur (pick-two-cards) )
((8 H) (5 C)): 8 High
>
```

Task 4B Classify-two-cards-ur



```
Welcome to DrRacket, version 8.6 [cs].
Language: racket, with debugging; memory limit: 128 MB.
>  (classify-two-cards-r (pick-two-cards) )
((K H) (3 C)): Three high
>  (classify-two-cards-r (pick-two-cards) )
((7 C) (A C)): seven highFlush
>  (classify-two-cards-r (pick-two-cards) )
((Q C) (8 C)): eight highFlush
>  (classify-two-cards-r (pick-two-cards) )
((9 H) (A S)): nine high
>  (classify-two-cards-r (pick-two-cards) )
((5 C) (7 H)): five high
>  (classify-two-cards-r (pick-two-cards) )
((2 S) (9 S)): Two highFlush
>  (classify-two-cards-r (pick-two-cards) )
((2 C) (K D)): Two high
>  (classify-two-cards-r (pick-two-cards) )
((J S) (A D)): Jack highStraight
>  (classify-two-cards-r (pick-two-cards) )
((A H) (2 H)): Two highFlush
>  (classify-two-cards-r (pick-two-cards) )
((Q D) (A D)): Queen highFlush
>  (classify-two-cards-r (pick-two-cards) )
((10 S) (4 C)): four high
>  (classify-two-cards-r (pick-two-cards) )
((7 H) (4 H)): four highFlush
>  (classify-two-cards-r (pick-two-cards) )
((Q D) (8 C)): eight high
>  (classify-two-cards-r (pick-two-cards) )
((3 C) (7 S)): Three high
>  (classify-two-cards-r (pick-two-cards) )
((J C) (Q D)): Jack high
>  (classify-two-cards-r (pick-two-cards) )
((7 S) (2 S)): Two highFlush
>  (classify-two-cards-r (pick-two-cards) )
((2 S) (3 C)): Two highStraight
>  (classify-two-cards-r (pick-two-cards) )
((Q S) (7 C)): seven high
>  (classify-two-cards-r (pick-two-cards) )
((J D) (Q H)): Jack high
>  (classify-two-cards-r (pick-two-cards) )
((2 D) (7 S)): Two high
>  (classify-two-cards-r (pick-two-cards) )
((6 D) (A D)): six highFlush
>
```

Task 4C Classifier demo