COMP1811 – Scheme Project Report

Name	**MD MOSTAFIZUR RAHMAN BADHON**	SID	**001134460**
Partner's name	**VLAD DRAGULELE**	SID	**001173034**

1. Software Design

Briefly describe the design of your coursework software and the strategy you took for solving it. – e.g., did you choose either recursion or high-order programming and why…

In this solitaire game, we used both recursive function and high order function what we learnt from our lectures and labs. Such as an example any of function with lambda is high order program and in function 2.6, we used value of which is recursive program. In one function (valueOf), recursion was the most appropriate because we had to add the values of each card on the deck. Recursion meaning, we had to keep calling the function with the rest of the deck, returning 1 value. We also used High order programming (probability) because it let us change the behaviour of a function by passing it some new behaviour to add to its own. High-order programming lets us pass some values to a function and get back a more complex function that will do something specific for us. So, here for the coursework we used both strategy for solving it when we need to solve the problem easier.

2. Code Listing

Provide a complete listing of the entire Scheme code you developed. Make sure your code is well commented, specially stressing informally the contracts for parameters on every symbol you may define. The code listed here must match that uploaded to Moodle. Please copy and paste the actual code – no screenshots please! Make it easy for the tutor to read. Marks WILL BE DEDUCTED if screenshots are included. Add explanatory narrative if necessary – though you are in-code comments should be clear enough.

Here is all our code which we used to develop the solitaire game project.

2.1 Function 1 – card?

```
;; Returns true if val is a pair and contains numbers 1-7, J, Q, K and the suit
(define (card? val) ;; val is the card (pair)
  (and
   (pair? val)
   (list? (member (car val) '( 1 2 3 4 5 6 7 #\J #\Q #\K)))
   (list? (member (cdr val) '( #\S #\C #\H #\D)))
   )
  )
```

2.2 Function 2 - suit

```
;; Returns the suit of the card (S, C, H, D)
(define (suit card)
  (cdr card)
```

2.3 Function 3 - numeral

```
;; Returns the number or face of the card
(define (numeral card)
  (car card)
  )
```

2.4 Function 4 – face?

```
;; Checks if the card is a Jack, Queen or King
(define (face? card)
  (list? (member (car card) '( #\J #\Q #\K)))
```

```
 )
```

2.5 Function 5 – card->string

```
;; Turns the card formatted from a dotted pair to a human readable string
(define (card->string val)
  (format "~a~a" (car val)(cdr val))
  )
```

2.6 Function 6 - value

```
;; Returns the value of the card
(define (value card)
  (cond
    [(face? card) 0.5] ;; If it is a face, the value is 0.5
    [else (car card)] ;; Otherwise, the value is the number on the card
  ))
```

2.7 Function 7 – deck?

```
;; Check if it is a list of cards (deck)
(define (deck? deck)
  (cond
    [(empty? deck) #f] ;; Without this, an empty deck would return true.
    [else
     (andmap card? deck)] ;; Checks every member of the deck if it is a card
   )
  )
```

2.8 Function 8 - valueOf

```
;; Value of the deck of cards given
(define (valueOf deck)
  (cond
    [(empty? deck) 0] ;; If deck is empty, value is 0
    [else ;; Otherwise, recursively adds the value of the first card of the deck
     (+
      (value (first deck))
      (valueOf (rest deck)))])
  )
```

2.9 Function 9 – do-suite

```
(define (do-suite val) ;; val is the suite
  (map
   (lambda (m) (cons m val))
   '( 1 2 3 4 5 6 7 #\J #\Q #\K)))
```

2.10 Function 10 - deck

```
;; Generate a deck of cards
(define deck
  (append (do-suite #\S) (do-suite #\C) (do-suite #\H) (do-suite #\D)))
```

2.11 Function 11 – deck->strings

```
;; Turns the deck into  human readable string
(define (deck->strings deck) ;; deck = human readable deck
  (map card->string deck))
```

2.12 Function 12 - count

```
; ;(define (playS deck hand strategy)
;;F3 - Probabilities.
;; Count function to count how many cards fit the criteria for the probability function
(define (count comp num card)
  (if (comp (value card) num)
      1
      0))
```

2.13 Function 13 - probability

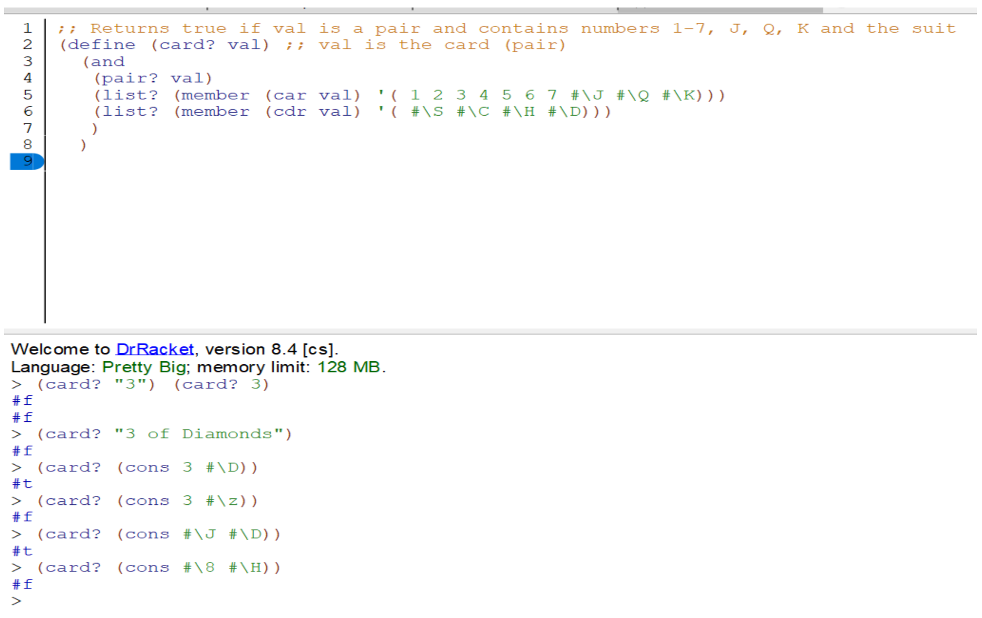
```
;; Returns number of cards that fit the criteria (comp is a comparative operator)
(define (probability comp num deck)
  (apply +
        (map
         (lambda (x) (count comp num x))
         deck)))
```

3. Results – Output Obtained

Provide screenshots that demonstrate the results generated by running your code. That is show the output obtained in the REPL when calling your functions. Alternatively, you may simply cut and paste from the REPL.

Here you will find some explanation about the output of this code after every image.

F1(I):

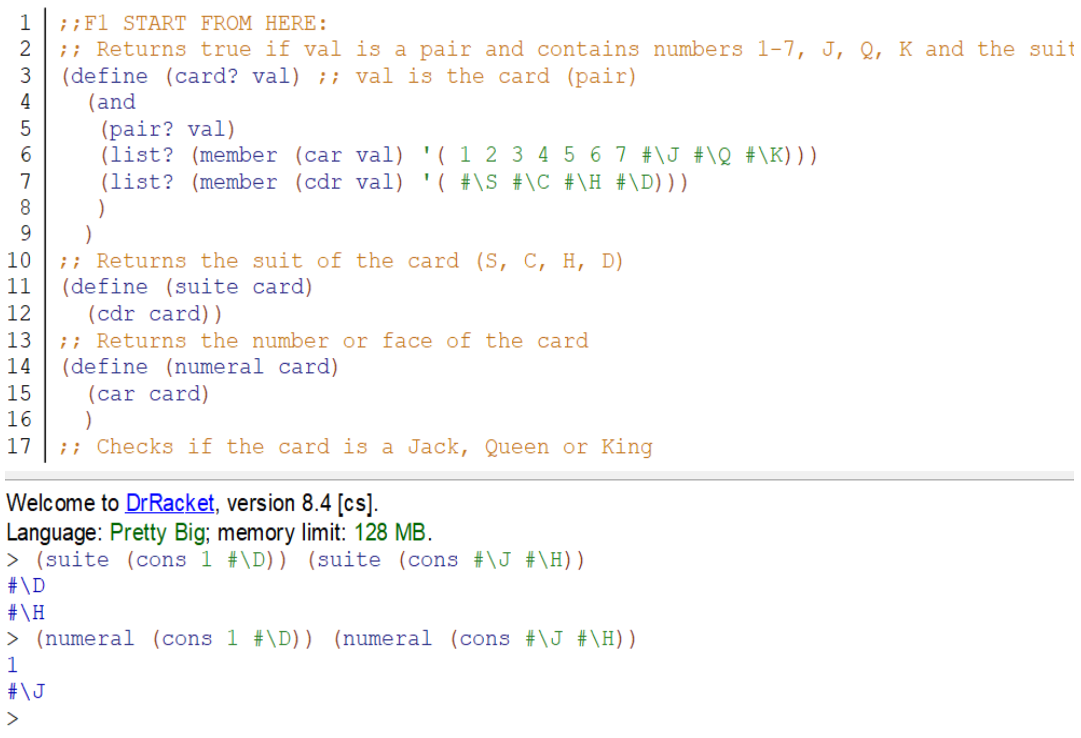


```
;; Returns true if val is a pair and contains numbers 1-7, J, Q, K and the suit
(define (card? val) ;; val is the card (pair)
  (and
   (pair? val)
   (list? (member (car val) '( 1 2 3 4 5 6 7 #\J #\Q #\K)))
   (list? (member (cdr val) '( #\S #\C #\H #\D)))
   )
  )
```

```
Welcome to DrRacket, version 8.4 [cs].
Language: Pretty Big; memory limit: 128 MB.
> (card? "3") (card? 3)
#f
#f
> (card? "3 of Diamonds")
#f
> (card? (cons 3 #\D))
#t
> (card? (cons 3 #\z))
#f
> (card? (cons #\J #\D))
#t
> (card? (cons #\8 #\H))
#f
>
```

>> card? yields #t when it corresponds to the following agreement, and #f otherwise. Definition of all the attributes of a card. Returns true if val is a pair and contains numbers 1-7, J, Q, K and the suite.

F1(II):

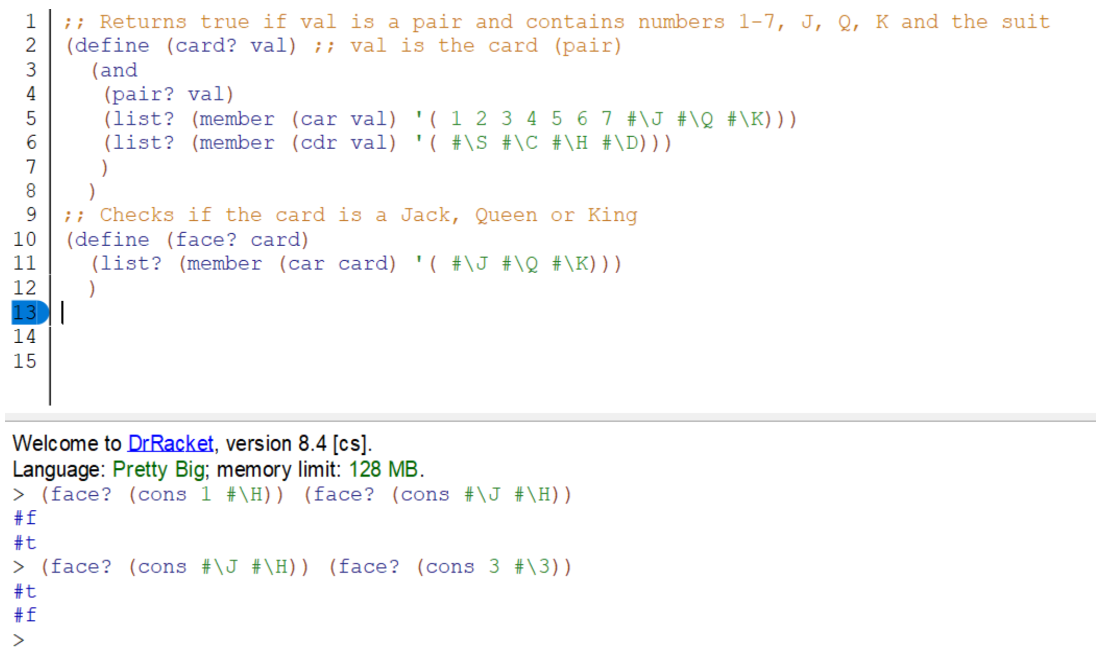


```scheme
;;F1 START FROM HERE:
;; Returns true if val is a pair and contains numbers 1-7, J, Q, K and the suit
(define (card? val) ;; val is the card (pair)
  (and
   (pair? val)
   (list? (member (car val) '( 1 2 3 4 5 6 7 #\J #\Q #\K)))
   (list? (member (cdr val) '( #\S #\C #\H #\D)))
   )
  )
;; Returns the suit of the card (S, C, H, D)
(define (suite card)
  (cdr card))
;; Returns the number or face of the card
(define (numeral card)
  (car card)
  )
;; Checks if the card is a Jack, Queen or King
```

```
Welcome to DrRacket, version 8.4 [cs].
Language: Pretty Big; memory limit: 128 MB.
> (suite (cons 1 #\D)) (suite (cons #\J #\H))
#\D
#\H
> (numeral (cons 1 #\D)) (numeral (cons #\J #\H))
1
#\J
>
```

>> Suite returns the suite of a card. numeral returns the number/face of a card.

F1(iii):

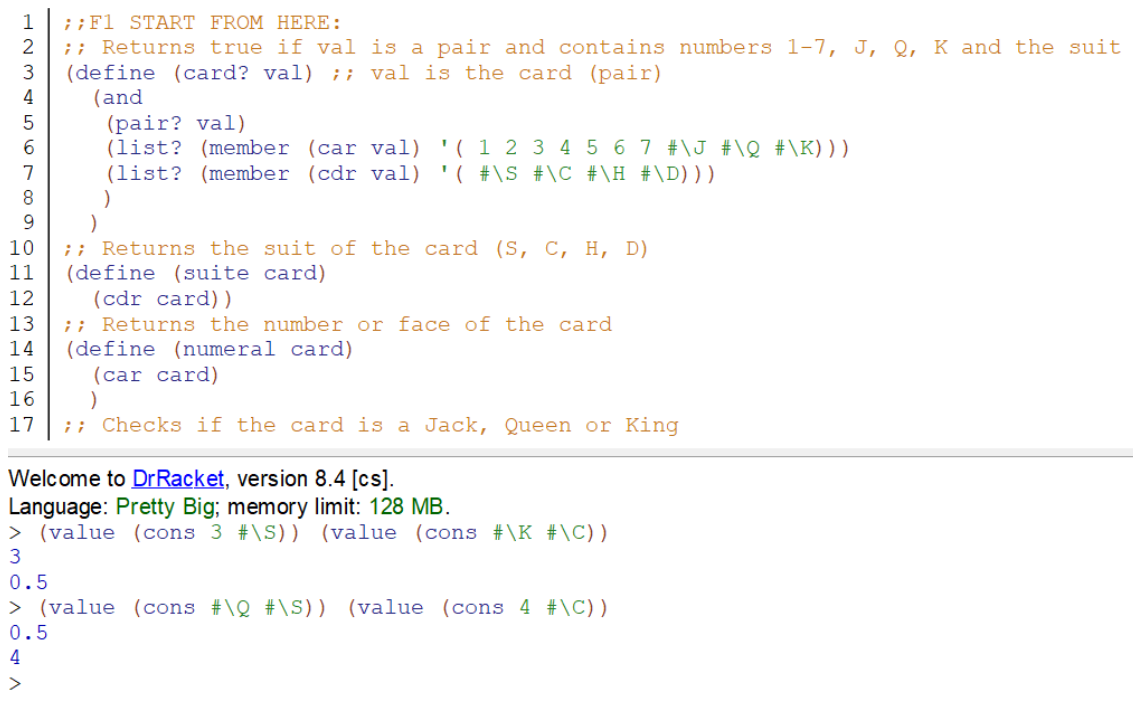


```scheme
;; Returns true if val is a pair and contains numbers 1-7, J, Q, K and the suit
(define (card? val) ;; val is the card (pair)
  (and
   (pair? val)
   (list? (member (car val) '( 1 2 3 4 5 6 7 #\J #\Q #\K)))
   (list? (member (cdr val) '( #\S #\C #\H #\D)))
   )
  )
;; Checks if the card is a Jack, Queen or King
(define (face? card)
  (list? (member (car card) '( #\J #\Q #\K)))
  )
```

```
Welcome to DrRacket, version 8.4 [cs].
Language: Pretty Big; memory limit: 128 MB.
> (face? (cons 1 #\H)) (face? (cons #\J #\H))
#f
#t
> (face? (cons #\J #\H)) (face? (cons 3 #\3))
#t
#f
>
```

- face? returns either #t if a given card is a face card (Jack, Queen, or King) or #f if it is not

F1(iv):

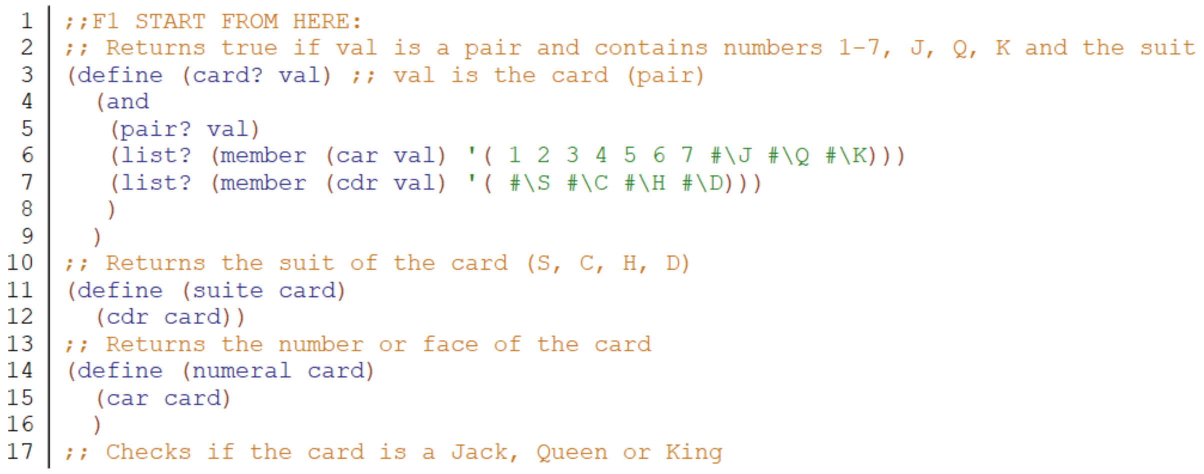


```
;;F1 START FROM HERE:
;; Returns true if val is a pair and contains numbers 1-7, J, Q, K and the suit
(define (card? val) ;; val is the card (pair)
  (and
   (pair? val)
   (list? (member (car val) '( 1 2 3 4 5 6 7 #\J #\Q #\K)))
   (list? (member (cdr val) '( #\S #\C #\H #\D)))
   )
  )
;; Returns the suit of the card (S, C, H, D)
(define (suite card)
  (cdr card))
;; Returns the number or face of the card
(define (numeral card)
  (car card)
  )
;; Checks if the card is a Jack, Queen or King
```

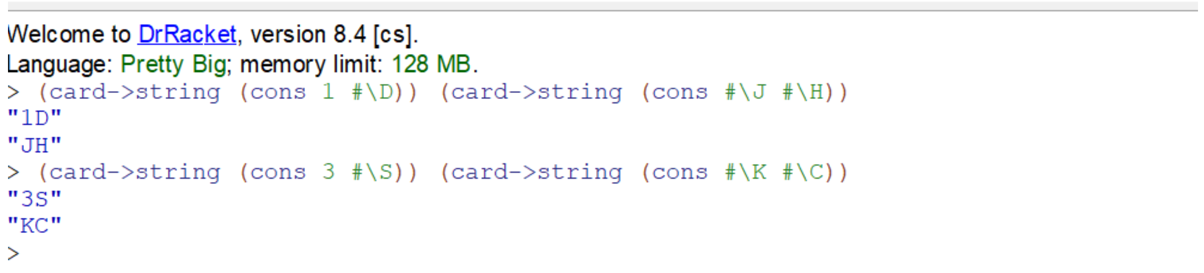
```
Welcome to DrRacket, version 8.4 [cs].
Language: Pretty Big; memory limit: 128 MB.
> (value (cons 3 #\S)) (value (cons #\K #\C))
3
0.5
> (value (cons #\Q #\S)) (value (cons 4 #\C))
0.5
4
>
```

>> value yields the value for a card. If it is a face, the value is 0.5. Otherwise, the value is the number on the card.

F1(v):



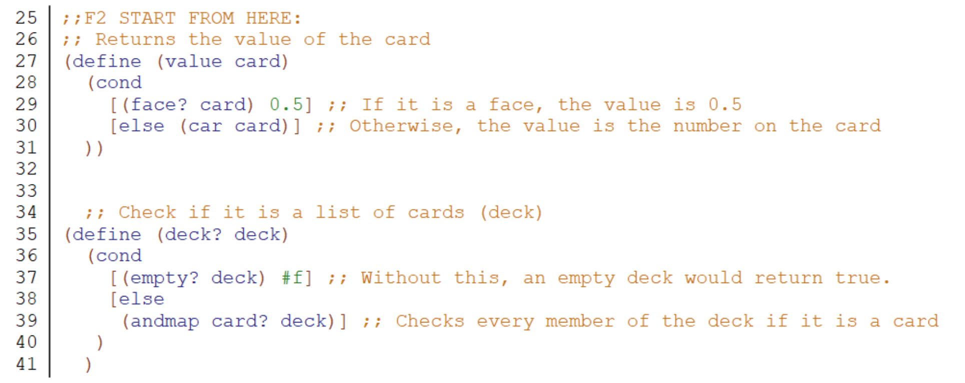
```scheme
;;F1 START FROM HERE:
;; Returns true if val is a pair and contains numbers 1-7, J, Q, K and the suit
(define (card? val) ;; val is the card (pair)
  (and
   (pair? val)
   (list? (member (car val) '( 1 2 3 4 5 6 7 #\J #\Q #\K)))
   (list? (member (cdr val) '( #\S #\C #\H #\D)))
   )
  )
;; Returns the suit of the card (S, C, H, D)
(define (suite card)
  (cdr card))
;; Returns the number or face of the card
(define (numeral card)
  (car card)
  )
;; Checks if the card is a Jack, Queen or King
```



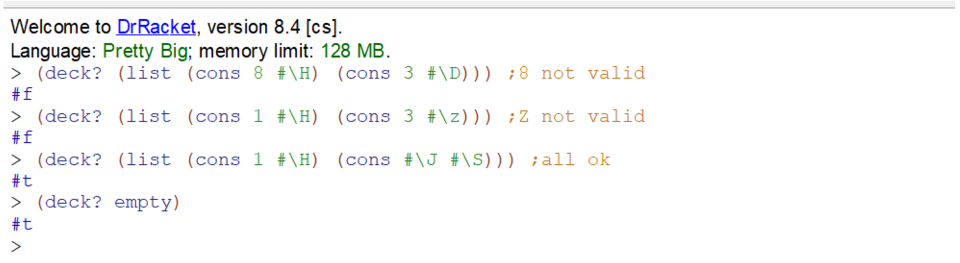
```
Welcome to DrRacket, version 8.4 [cs].
Language: Pretty Big; memory limit: 128 MB.
> (card->string (cons 1 #\D)) (card->string (cons #\J #\H))
"1D"
"JH"
> (card->string (cons 3 #\S)) (card->string (cons #\K #\C))
"3S"
"KC"
>
```

>> card->string returns a human readable string for it, and we used the format function here.

F2(i):



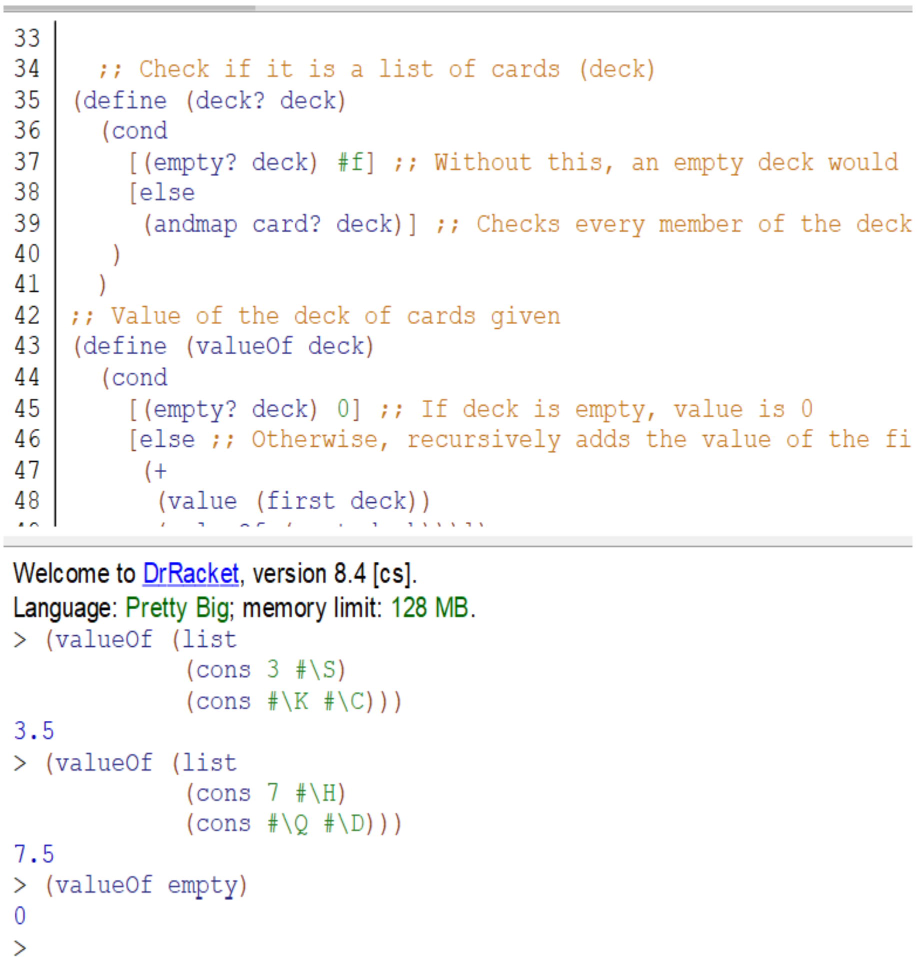
```scheme
;;F2 START FROM HERE:
;; Returns the value of the card
(define (value card)
  (cond
    [(face? card) 0.5] ;; If it is a face, the value is 0.5
    [else (car card)] ;; Otherwise, the value is the number on the card
  ))


  ;; Check if it is a list of cards (deck)
(define (deck? deck)
  (cond
    [(empty? deck) #f] ;; Without this, an empty deck would return true.
    [else
     (andmap card? deck)] ;; Checks every member of the deck if it is a card
    )
  )
```



```
Welcome to DrRacket, version 8.4 [cs].
Language: Pretty Big; memory limit: 128 MB.
> (deck? (list (cons 8 #\H) (cons 3 #\D))) ;8 not valid
#f
> (deck? (list (cons 1 #\H) (cons 3 #\z))) ;Z not valid
#f
> (deck? (list (cons 1 #\H) (cons #\J #\S))) ;all ok
#t
> (deck? empty)
#t
>
```

>> deck? returns #t if it is a valid deck or #f otherwise. Without empty? function, an empty deck would return true.

F2(ii):



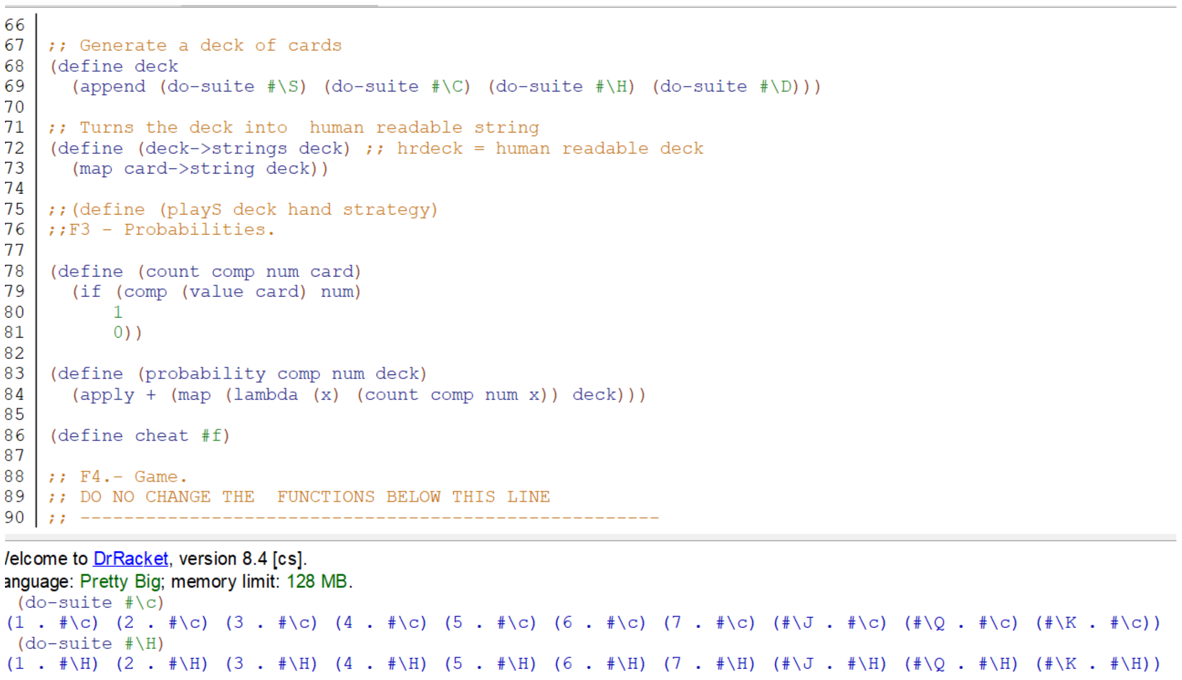
```
  ;; Check if it is a list of cards (deck)
(define (deck? deck)
  (cond
    [(empty? deck) #f] ;; Without this, an empty deck would
    [else
     (andmap card? deck)] ;; Checks every member of the deck
    )
  )
;; Value of the deck of cards given
(define (valueOf deck)
  (cond
    [(empty? deck) 0] ;; If deck is empty, value is 0
    [else ;; Otherwise, recursively adds the value of the fi
     (+
      (value (first deck))
```

```
Welcome to DrRacket, version 8.4 [cs].
Language: Pretty Big; memory limit: 128 MB.
> (valueOf (list
            (cons 3 #\S)
            (cons #\K #\C)))
3.5
> (valueOf (list
            (cons 7 #\H)
            (cons #\Q #\D)))
7.5
> (valueOf empty)
0
>
```

>>value Of calculates the sum of values of each of the cards in a deck. If the deck is empty, the value is 0.

Otherwise, recursively adds the value of the first card of the deck.

F2(iii):

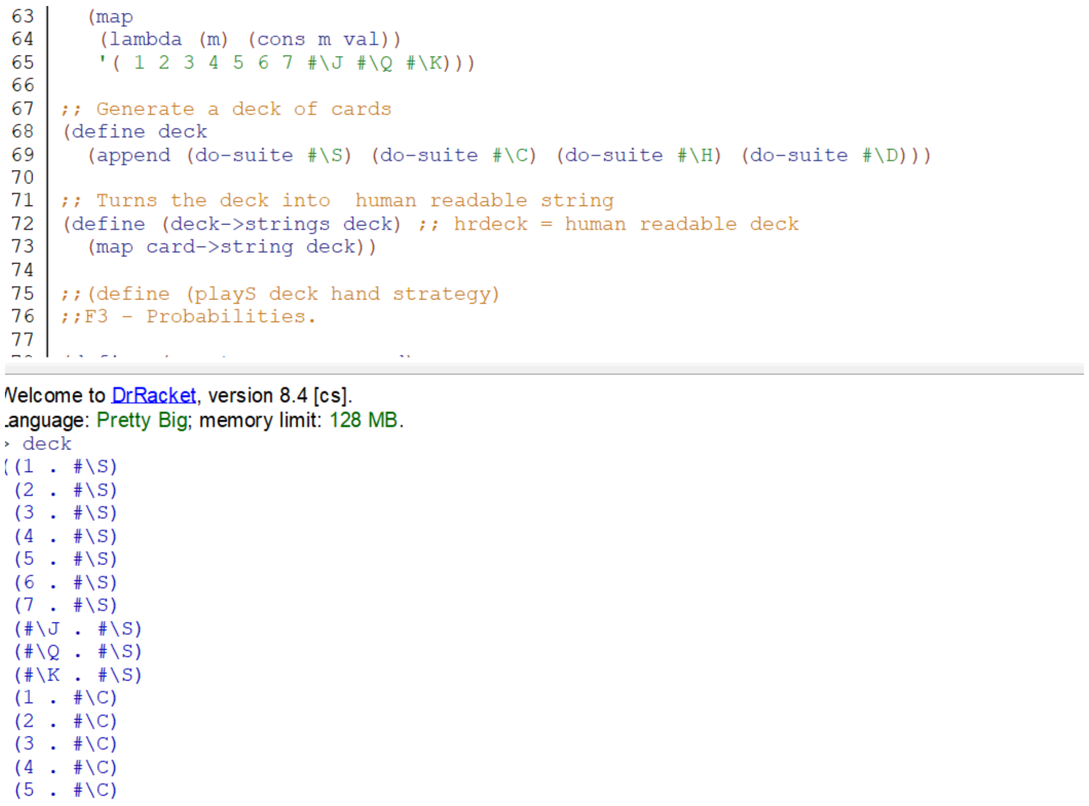


```
;; Generate a deck of cards
(define deck
  (append (do-suite #\S) (do-suite #\C) (do-suite #\H) (do-suite #\D)))

;; Turns the deck into  human readable string
(define (deck->strings deck) ;; hrdeck = human readable deck
  (map card->string deck))

;;(define (playS deck hand strategy)
;;F3 - Probabilities.

(define (count comp num card)
  (if (comp (value card) num)
      1
      0))

(define (probability comp num deck)
  (apply + (map (lambda (x) (count comp num x)) deck)))

(define cheat #f)

;; F4.- Game.
;; DO NO CHANGE THE  FUNCTIONS BELOW THIS LINE
;; ----------------------------------------------------
```

```
elcome to DrRacket, version 8.4 [cs].
anguage: Pretty Big; memory limit: 128 MB.
 (do-suite #\c)
(1 . #\c) (2 . #\c) (3 . #\c) (4 . #\c) (5 . #\c) (6 . #\c) (7 . #\c) (#\J . #\c) (#\Q . #\c) (#\K . #\c))
 (do-suite #\H)
(1 . #\H) (2 . #\H) (3 . #\H) (4 . #\H) (5 . #\H) (6 . #\H) (7 . #\H) (#\J . #\H) (#\Q . #\H) (#\K . #\H))
```

>>do-suite returns the whole series for a suite, including the numbers from 1 to 7 and the faces #\J, #\Q and #\K.

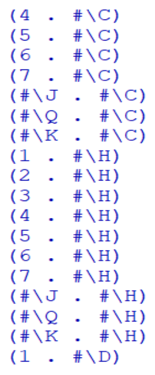
F2(iv):



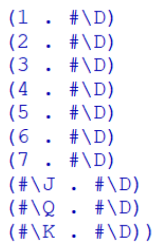
```
  (map
   (lambda (m) (cons m val))
   '( 1 2 3 4 5 6 7 #\J #\Q #\K)))

;; Generate a deck of cards
(define deck
  (append (do-suite #\S) (do-suite #\C) (do-suite #\H) (do-suite #\D)))

;; Turns the deck into  human readable string
(define (deck->strings deck) ;; hrdeck = human readable deck
  (map card->string deck))

;;(define (playS deck hand strategy)
;;F3 - Probabilities.
```

```
elcome to DrRacket, version 8.4 [cs].
anguage: Pretty Big; memory limit: 128 MB.
 deck
((1 . #\S)
 (2 . #\S)
 (3 . #\S)
 (4 . #\S)
 (5 . #\S)
 (6 . #\S)
 (7 . #\S)
 (#\J . #\S)
 (#\Q . #\S)
 (#\K . #\S)
 (1 . #\C)
 (2 . #\C)
 (3 . #\C)
 (4 . #\C)
 (5 . #\C)
```

>>deck includes the entire list of the 40 valid cards.

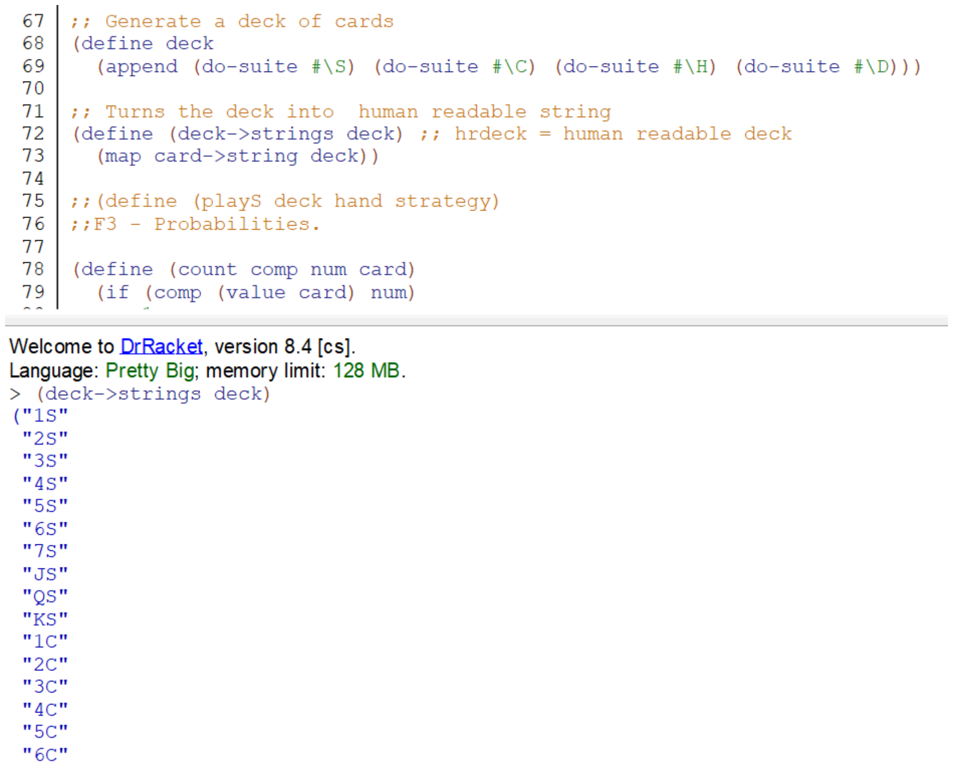


```
 (4 . #\C)
 (5 . #\C)
 (6 . #\C)
 (7 . #\C)
 (#\J . #\C)
 (#\Q . #\C)
 (#\K . #\C)
 (1 . #\H)
 (2 . #\H)
 (3 . #\H)
 (4 . #\H)
 (5 . #\H)
 (6 . #\H)
 (7 . #\H)
 (#\J . #\H)
 (#\Q . #\H)
 (#\K . #\H)
 (1 . #\D)
```



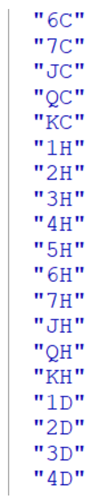
```
(1 . #\D)
(2 . #\D)
(3 . #\D)
(4 . #\D)
(5 . #\D)
(6 . #\D)
(7 . #\D)
(#\J . #\D)
(#\Q . #\D)
(#\K . #\D))
```

F2(V):

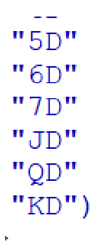


```
;; Generate a deck of cards
(define deck
  (append (do-suite #\S) (do-suite #\C) (do-suite #\H) (do-suite #\D)))

;; Turns the deck into  human readable string
(define (deck->strings deck) ;; hrdeck = human readable deck
  (map card->string deck))

;;(define (playS deck hand strategy)
;;F3 - Probabilities.

(define (count comp num card)
  (if (comp (value card) num)
```

```
Welcome to DrRacket, version 8.4 [cs].
Language: Pretty Big; memory limit: 128 MB.
> (deck->strings deck)
("1S"
 "2S"
 "3S"
 "4S"
 "5S"
 "6S"
 "7S"
 "JS"
 "QS"
 "KS"
 "1C"
 "2C"
 "3C"
 "4C"
 "5C"
 "6C"
```



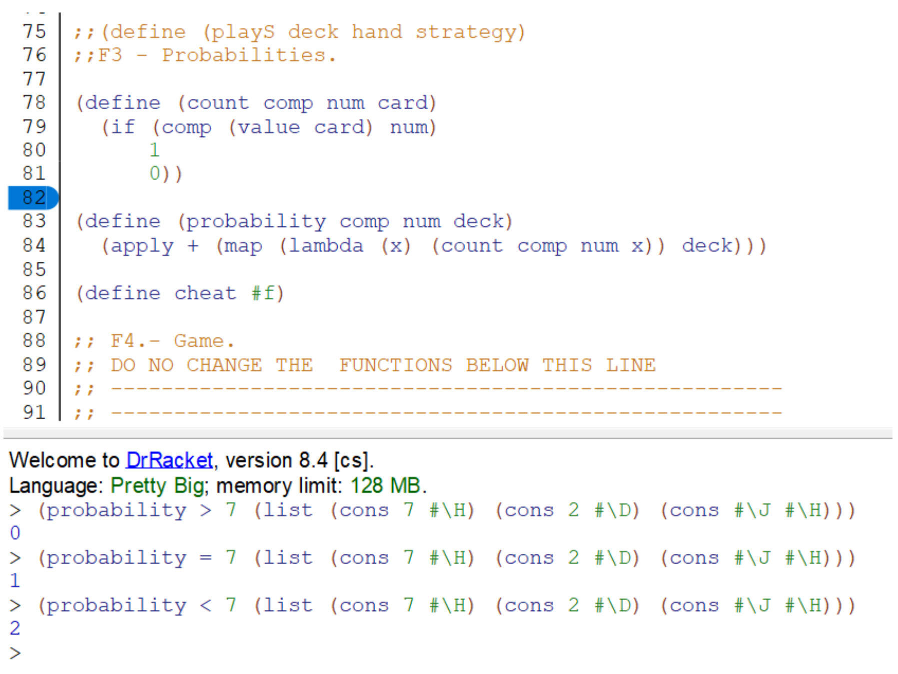
```
"6C"
"7C"
"JC"
"QC"
"KC"
"1H"
"2H"
"3H"
"4H"
"5H"
"6H"
"7H"
"JH"
"QH"
"KH"
"1D"
"2D"
"3D"
"4D"
```



```
"5D"
"6D"
"7D"
"JD"
"QD"
"KD")
```

>> returns a human representation of a deck.

F3(i):



```
;;(define (playS deck hand strategy)
;;F3 - Probabilities.

(define (count comp num card)
  (if (comp (value card) num)
      1
      0))

(define (probability comp num deck)
  (apply + (map (lambda (x) (count comp num x)) deck)))

(define cheat #f)

;; F4.- Game.
;; DO NO CHANGE THE  FUNCTIONS BELOW THIS LINE
;; ----------------------------------------------------
;; ----------------------------------------------------
```

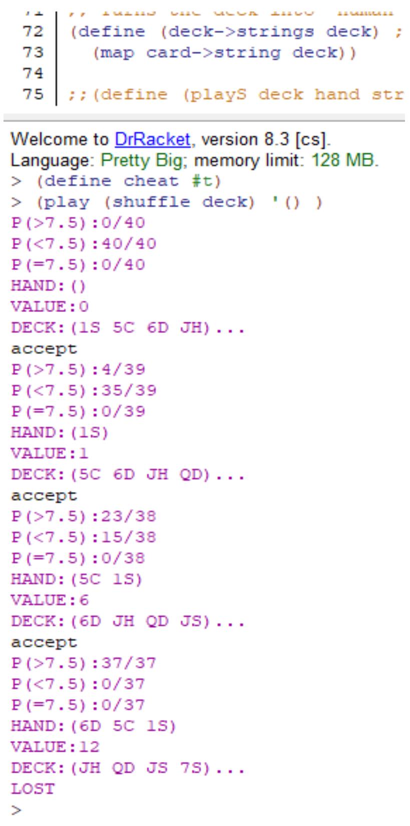
```
Welcome to DrRacket, version 8.4 [cs].
Language: Pretty Big; memory limit: 128 MB.
> (probability > 7 (list (cons 7 #\H) (cons 2 #\D) (cons #\J #\H)))
0
> (probability = 7 (list (cons 7 #\H) (cons 2 #\D) (cons #\J #\H)))
1
> (probability < 7 (list (cons 7 #\H) (cons 2 #\D) (cons #\J #\H)))
2
>
```

>> probability works out the number of cards in the list having values less, greater, or equal than the provided number, respectively.

4. TESTING

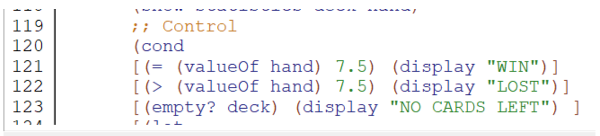
Provide a test plan covering all your functions and the results of applying the tests.

After providing all our code Here we tried to play the game and we can successfully played it.

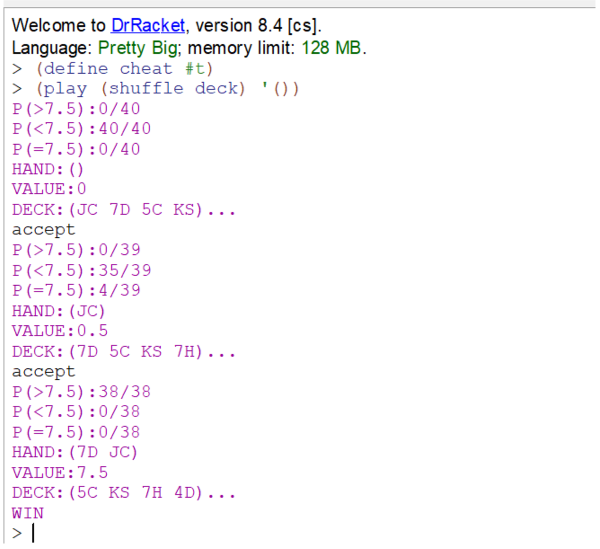


```
;; turns the deck into human
(define (deck->strings deck) ;
  (map card->string deck))

;;(define (playS deck hand str
```

```
Welcome to DrRacket, version 8.3 [cs].
Language: Pretty Big; memory limit: 128 MB.
> (define cheat #t)
> (play (shuffle deck) '() )
P(>7.5):0/40
P(<7.5):40/40
P(=7.5):0/40
HAND:()
VALUE:0
DECK:(1S 5C 6D JH)...
accept
P(>7.5):4/39
P(<7.5):35/39
P(=7.5):0/39
HAND:(1S)
VALUE:1
DECK:(5C 6D JH QD)...
accept
P(>7.5):23/38
P(<7.5):15/38
P(=7.5):0/38
HAND:(5C 1S)
VALUE:6
DECK:(6D JH QD JS)...
accept
P(>7.5):37/37
P(<7.5):0/37
P(=7.5):0/37
HAND:(6D 5C 1S)
VALUE:12
DECK:(JH QD JS 7S)...
LOST
>
```



```
;; Control
(cond
[(= (valueOf hand) 7.5) (display "WIN")]
[(> (valueOf hand) 7.5) (display "LOST")]
[(empty? deck) (display "NO CARDS LEFT") ]
```



```
Welcome to DrRacket, version 8.4 [cs].
Language: Pretty Big; memory limit: 128 MB.
> (define cheat #t)
> (play (shuffle deck) '())
P(>7.5):0/40
P(<7.5):40/40
P(=7.5):0/40
HAND:()
VALUE:0
DECK:(JC 7D 5C KS)...
accept
P(>7.5):0/39
P(<7.5):35/39
P(=7.5):4/39
HAND:(JC)
VALUE:0.5
DECK:(7D 5C KS 7H)...
accept
P(>7.5):38/38
P(<7.5):0/38
P(=7.5):0/38
HAND:(7D JC)
VALUE:7.5
DECK:(5C KS 7H 4D)...
WIN
> |
```

5. Evaluation

*Evaluate your implementation and discuss what you would do if you had more time to work on the code. Critically reflect on the following point and write **300-400 words overall**.*

Points for reflection:

- *What went well and what went less well?*

>> Here we used everything that we learnt from all our lectures and tutorials. One of the things that did not go well and that was we faced many bugs in our developed code. An example could be that the string's function was not working, and we asked the teacher why we faced a problem like this. After that the teacher approached us and told us that we used the wrong formula. Then he gave us an idea that how to overcome this kind of problem. And lastly, we solved everything. In this project, both of us work hard to gain the best results.

- *What did you learn from your experience? (Not just about Scheme, but about programming in general, project development and time management, etc.)*

>> ***This project increased our knowledge about functional programming (Scheme programming) and by applying this knowledge we can develop a program. It was a bit tough but with the help of our teacher, we felt confident and solved the problem. This project also gives us experience that how groups projects mean to us and how to communicate with another person. We learnt how to develop a project in timely and when you faced problems how you discuss with your group members.***

- *How would a similar task be completed differently?*

>> ***In this case, we used everything that we learned from our lectures. There were two different functions, one is recursion, and the other is high order. So, we do not have another option that we can use. Hope we will learn extra how to solve this kind of problem differently.***

- *What can you carry forward for future development projects?*

>>***in this coursework we learnt the functional language, which is a different type of computer programming language compared to python. We can learn more about this by reading our lectures slide more and apply these on many different projects. I think we could carry forward recursion and high order programming (like lambda) because it's a different way of solving problems like loops.***

6. Group Pro forma

Describe the division of work and agree percentage contributions. The pro forma must be signed by all group members and an identical copy provided in each report. If you cannot agree percentage contributions, please indicate so in the notes column and provide your reasoning.

(THIS SECTION SHOULD BE THE SAME FOR BOTH PARTNERS)

Partner ID	Tasks/Features Completed	%Contribution	Signature	Notes

Partner ID	Tasks/Features Completed	%Contribution	Signature	Notes
001134460	F1, F2(iii, iv)and F3	50%	MOSTAFIZ	
001173034	All of F1, F2 (i. ii. v.)	50%	VLAD	
	Total	100%		