

---

---

## Racket Programming Assignment #3: Lambda and Basic Lisp

---

---

---

---

### What's It All About?

---

---

The first two tasks are highly constrained. One of these pertains to lambda functions, and the other to basic list processing operations in Lisp. The second two tasks, which are considerably more involved, ask you to extend programs that are presented in Lesson 6 “Basic Lisp Programming”. These two tasks will afford you enough freedom to make some mistakes along the way, and to see opportunities to revise your code to make it better, even after you get your programs to do what they are supposed to do.

---

---

### Overall Charge

---

---

Working within the DrRacket PDE, please do each of the numbered programming tasks. Then please do the document compilation/posting task.

---

---

### Task 1 - Lambda

---

---

---

---

#### Task 1a - Three ascending integers

---

---

Consider the following demo:

```
> ( asc 5 )
'(5 6 7)
> ( asc 0 )
'(0 1 2)
> ( asc 108 )
'(108 109 110)
>
```

Your job is to generate this exact demo, except that you must replace the call to the `asc` function in each of the three applications with a **lambda** function. Note that the Definitions area will not come into play in this exercise, nor will you create any named functions. Your demo will simply feature three anonymous function applications, the first with argument 5, the second with argument 0, and the third with argument 108.

---

## Task 1b - Make list in reverse order

---

Consider the following demo:

```
> ( mlr 'red 'yellow 'blue )
'(blue yellow red)
> ( mlr 10 20 30 )
'(30 20 10)
> ( mlr "Professor Plum" "Colonel Mustard" "Miss Scarlet" )
'("Miss Scarlet" "Colonel Mustard" "Professor Plum")
>
```

Your job is to generate this exact demo, except that you must replace the call to the `mlr` function in each of the three applications with a **lambda** function. Note that the Definitions area will not come into play in this exercise, nor will you create any named functions. Your demo will simply feature three anonymous function applications, each using the same three arguments that I used in the given demo.

---

## Task 1c - Random number generator

---

Consider the following demo:

```
> ( rn 3 5 )
5
> ( rn 3 5 )
3
> ( rn 3 5 )
5
> ( rn 3 5 )
5
> ( rn 3 5 )
3
> ( rn 3 5 )
4
> ( rn 3 5 )
3
> ( rn 3 5 )
3
> ( rn 3 5 )
5
> ( rn 3 5 )
3
> ( rn 11 17 )
17
> ( rn 11 17 )
12
> ( rn 11 17 )
14
> ( rn 11 17 )
12
```

```
> ( rn 11 17 )
12
> ( rn 11 17 )
14
> ( rn 11 17 )
16
> ( rn 11 17 )
14
> ( rn 11 17 )
16
> ( rn 11 17 )
13
>
```

Your job is to generate demo like this one, except that you must replace the call to the `rn` function in each of the three applications with a **lambda** function. Note that the Definitions area will not come into play in this exercise, nor will you create any named functions. Your demo will simply feature ten anonymous function applications with arguments 3 and 5, and ten anonymous function applications with arguments 11 and 17.

---

## Contribution to the solution document

---

For the section of your solution document that corresponds to this part of your assignment, please include:

1. The first demo.
2. The second demo.
3. The third demo.

---

## Task 2 - List Processing Referencers and Constructors

---

Simply create the demo, for real, that is presented in redacted form in Lesson 6 “Basic Lisp Programming”, in the section titled “Redacted Racket Session Featuring Referencers and Constructors”.

---

## Contribution to the solution document

---

For the section of your solution document that corresponds to this part of your assignment, please simply include the required demo.

---

## Task 3 - Little Color Interpreter

---

---

This task invites you to extend a very simple interpreter into a slightly less simple interpreter.

---

### Task 3a - Establishing the Sampler code from Lesson 6

---

1. In a file called `sampler.rkt`, establish the “sampler” program that can be found in the penultimate section of Lesson 6.
2. Generate a demo by mimicking the demo which accompanies the “sampler” program in Lesson 6.

---

### Task 3b - Color Thing Interpreter

---

1. In a file called `color_thing.rkt`, copy and paste the code from the `sampler.rkt` file. In the two obvious places, change `sampler` to `color_thing`. Then run the program, just to make sure that everything still works.
2. Change/extend the program so that it will interpret three commands, each of which takes the form of a list of length 2, as suggested in the accompanying demo. You needn't do error checking for this program. Simply assume that the user capably types in a list of length two when required, the first element of which is either `random`, or `all`, or an integer between 1 and the length of the second parameter, which will always be a list of color names that the Racket `2htdp/image` library recognizes. **Constraint: No iterative constructs are allowed.**
3. Create a demo that is just like the accompanying demo, except for the randomness bits.
4. Create another demo, a similar demo, but with different colors than in the accompanying demo. You can find a list of colors known to Racket at:  
[https://docs.racket-lang.org/draw/color-database\\_\\_\\_.html](https://docs.racket-lang.org/draw/color-database___.html)

### The accompanying demo:

```
> ( 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
```

---

## Contribution to the solution document

---

For the section of your solution document that corresponds to this part of your assignment, please include:

1. The “sampler” code associated with Task 3a.
2. The demo associated with Task 3a.
3. The “color thing” code associated with task 3b.
4. The two demos associated with Task 3b.

---

## Task 4 - Two Card Poker

---

---

This task invites you to programmably explore the hands that are dealt in a game of two card poker. You will start by merely entering and demoing the card playing code presented in Lesson 6 “Basics Lisp Programming”. You will then write a classifier for the hands in this game, assuming a uniform representational (`ur`) scheme (output of cards will be represented in the primitive internal representation of cards). As a final step, you will arrange for the output to appear in a more natural fashion.

---

### Task 4a - Establishing the Card code from Lesson 6

---

---

1. Please establish a file called `cards.rkt` and place the code given within the section titled “Basic List Processing Example - Playing Cards” of Lesson 6 into the file.
2. Recreate the demo provided in that same part of the lesson.

---

### Task 4b - Two Card Poker Classifier, IR Version

---

---

In this part of the assignment you will be lead in a step by step manner to program a two card poker hand classifier.

1. Please closely consider the the accompanying demo called “UR classifier demo”, from which you should be able to infer the big picture of what you are expected to do for this part of your assignment.
2. Please establish a file called `classifier_ur.rkt`, copy and paste the code from your “`cards.rkt`” file into the newly established file, save it, and run it, by which mean just make sure that the code still works. Note: You will make use of some, but not all, of the functionality that you copied into this file as you proceed to develop the classifier program.
3. Within the classifier file, define a parameterless function called `pick-two-cards` which returns a list of two cards from a virgin deck of cards, assuring that the two cards are distinct. One way to do this would be to call `pick-a-card` twice, compare the resulting cards, return them in the form of a list if they are different, or recursively call this function if the should happen to be the same. Please see the accompanying demo called “Pick two cards demo” for assurance that you understand what this function is all about.
4. Recreate the pick two cards demo (except for the fact that your two card results will differ due the to randomness).
5. Within the classifier file, define a function called `higher-rank` taking two parameters, each one being a card, which returns the “highest rank” of the cards. There are lots of ways to do this. Pick one! Please see the accompanying demo called “Higher rank demo” for assurance that you understand what this function is all about.
6. Recreate the accompanying higher rank demo. Note: In order to do this you will need add a couple of extra lines of code to your file. Please add ( `require racket/trace` ) just after the language specification statement. Please add ( `trace higher-rank` ) at the end of the file. Do the demo. After you copy the demo, “comment out” the trace form at the end of the file.
7. Within the classifier file, define a function called `classify-two-cards-ur` taking one parameter, a list of two lists, each representing one card, which displays the two cards, followed by a colon, followed by the appropriate classification of the two cards. Note that there are five basic ways to classify two cards: as a pair, as a straight flush, as a straight, as a flush, and as a mere high card. The syntax of the classification expressions can be found in the demo. Please adhere to it, noting that the maximal rank of the pair leads off each classification expression. Please, once again, refer to the accompanying demo called “Higher rank demo” for details and clarification.

8. Please “recreate” the accompanying uniform representation classifier demo, subject to the fact that randomness will cause different pairs of cards to appear throughout the demo. That is, generate a demo which calls this function 20 times.

---

## UR classifier demo

---

```
> ( classify-two-cards-ur ( pick-two-cards ) )
((J D) (4 C)): J high
> ( classify-two-cards-ur ( pick-two-cards ) )
((Q C) (X S)): Q high
> ( classify-two-cards-ur ( pick-two-cards ) )
((Q H) (J S)): Q high straight
> ( classify-two-cards-ur ( pick-two-cards ) )
((A H) (8 D)): A high
> ( classify-two-cards-ur ( pick-two-cards ) )
((9 H) (5 H)): 9 high flush
> ( classify-two-cards-ur ( pick-two-cards ) )
((8 S) (7 S)): 8 high straight flush
> ( classify-two-cards-ur ( pick-two-cards ) )
((4 H) (X H)): X high flush
> ( classify-two-cards-ur ( pick-two-cards ) )
((A H) (9 S)): A high
> ( classify-two-cards-ur ( pick-two-cards ) )
((J S) (9 D)): J high
> ( classify-two-cards-ur ( pick-two-cards ) )
((5 C) (3 H)): 5 high
> ( classify-two-cards-ur ( pick-two-cards ) )
((K H) (X S)): K high
> ( classify-two-cards-ur ( pick-two-cards ) )
((9 H) (8 D)): 9 high straight
> ( classify-two-cards-ur ( pick-two-cards ) )
((2 S) (5 H)): 5 high
> ( classify-two-cards-ur ( pick-two-cards ) )
((K C) (A H)): A high straight
> ( classify-two-cards-ur ( pick-two-cards ) )
((6 H) (A H)): A high flush
> ( classify-two-cards-ur ( pick-two-cards ) )
((3 H) (2 H)): 3 high straight flush
> ( classify-two-cards-ur ( pick-two-cards ) )
((9 S) (5 D)): 9 high
> ( classify-two-cards-ur ( pick-two-cards ) )
((7 D) (J H)): J high
> ( classify-two-cards-ur ( pick-two-cards ) )
((4 D) (3 D)): 4 high straight flush
>
```

---

## Pick two cards demo

---

```
> ( pick-two-cards )
```

```

>((9 D) (X S))
> ( pick-two-cards )
>((J S) (A H))
> ( pick-two-cards )
>((X D) (K H))
> ( pick-two-cards )
>((X C) (3 D))
> ( pick-two-cards )
>((9 D) (J S))
>

```

---

## Higher rank demo

---

```

> ( higher-rank ( pick-a-card ) ( pick-a-card ) )
>(higher-rank '(5 C) '(A D))
<'A
'A
> ( higher-rank ( pick-a-card ) ( pick-a-card ) )
>(higher-rank '(6 D) '(8 H))
<8
8
> ( higher-rank ( pick-a-card ) ( pick-a-card ) )
>(higher-rank '(4 S) '(5 D))
<5
5
> ( higher-rank ( pick-a-card ) ( pick-a-card ) )
>(higher-rank '(8 H) '(A H))
<'A
'A
> ( higher-rank ( pick-a-card ) ( pick-a-card ) )
>(higher-rank '(J H) '(X S))
<'J
'J
>

```

---

## Task 4c - Two Card Poker Classifier

---

In this part of the assignment you are asked to change the classifier so that the output is a bit more natural.

1. Please closely consider the the accompanying demo called “Classifier demo”, from which you should be able to infer the big picture of what you are expected to do for this part of your assignment.
2. Please establish a file called `classifier.rkt`, copy and paste the code from your `classifier_ur.rkt` file into the newly established file, save it, and run it, by which I mean just make sure that the code still works.
3. Within the newly created classifier file, define a function called `classify-two-cards` taking one parameter, a list of two lists, each representing one card, which displays the two cards, followed by a colon, followed by the more natural classification of the two cards. The method of classification is the same as before. The only difference is that the output will appear a bit more natural. The syntax of the more natural classification expressions can be found in the demo. Please adhere to it. Note: You can make a mess of this, or you can do



it in a fairly clean manner. Please strive for the latter!

4. Please “recreate” the accompanying classifier demo, with the more expressive representation of the classifications. That is, generate a demo which calls this function 20 times.

---

## Classifier demo

---

```
> ( classify-two-cards ( pick-two-cards ) )
((4 D) (J S)): jack high
> ( classify-two-cards ( pick-two-cards ) )
((Q H) (8 C)): queen high
> ( classify-two-cards ( pick-two-cards ) )
((2 H) (8 C)): eight high
> ( classify-two-cards ( pick-two-cards ) )
((8 H) (2 C)): eight high
> ( classify-two-cards ( pick-two-cards ) )
((2 C) (6 C)): six high flush
> ( classify-two-cards ( pick-two-cards ) )
((3 C) (K C)): king high flush
> ( classify-two-cards ( pick-two-cards ) )
((5 S) (3 S)): five high flush
> ( classify-two-cards ( pick-two-cards ) )
((X D) (5 S)): ten high
> ( classify-two-cards ( pick-two-cards ) )
((6 C) (8 C)): eight high flush
> ( classify-two-cards ( pick-two-cards ) )
((6 H) (2 D)): six high
> ( classify-two-cards ( pick-two-cards ) )
((Q D) (3 S)): queen high
> ( classify-two-cards ( pick-two-cards ) )
((5 S) (9 H)): nine high
> ( classify-two-cards ( pick-two-cards ) )
((J C) (9 S)): jack high
> ( classify-two-cards ( pick-two-cards ) )
((K S) (6 C)): king high
> ( classify-two-cards ( pick-two-cards ) )
((J C) (Q H)): queen high straight
> ( classify-two-cards ( pick-two-cards ) )
((5 S) (X S)): ten high flush
> ( classify-two-cards ( pick-two-cards ) )
((7 C) (X S)): ten high
> ( classify-two-cards ( pick-two-cards ) )
((7 S) (2 D)): seven high
> ( classify-two-cards ( pick-two-cards ) )
((6 S) (7 S)): seven high straight flush
>
```

---

## Contribution to the solution document

---

For the section of your solution document that corresponds to this part of your assignment, please include the code and demo(s) associated with each of the four tasks.

---

## Due Date

---

Friday, October 7, 2022. **But Note: Although this assignment is not officially due until a bit after your first exam, doing the bulk of the assignment (perhaps everything but creating and posting your solution document) prior to the exam will be a great way for you to study for the exam!**