Third Racket Assignment

Justin Cesarini

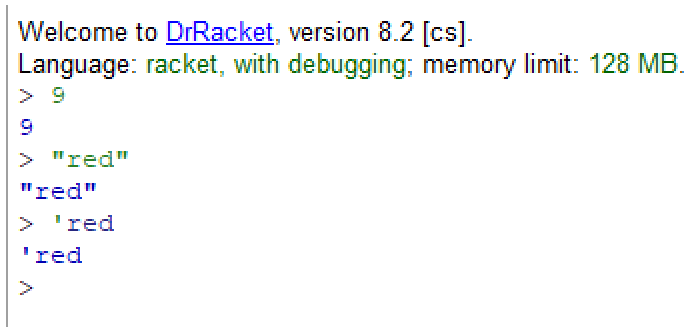
Learning Abstract

There are many things that I learned with this assignment. I got very familiar with lists, cdr, car, and cons functions. I learned how to define several objects as well as using lambda. The practice and repetition of this assignment helped me to grasp how to code in these areas. It was built up to the point where I could manipulate my own deck of cards. This assignment was extremely helpful.

Task 1 - Historical Lisp

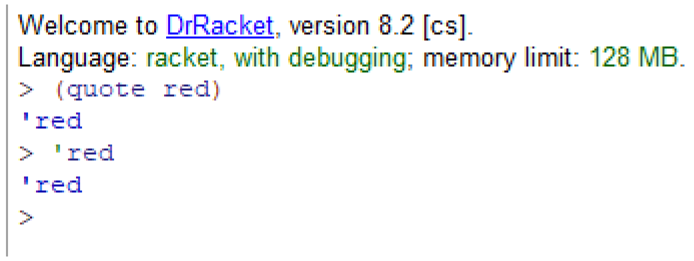
Quote and Eval

Interactions - Constants 9 and "red" and 'red



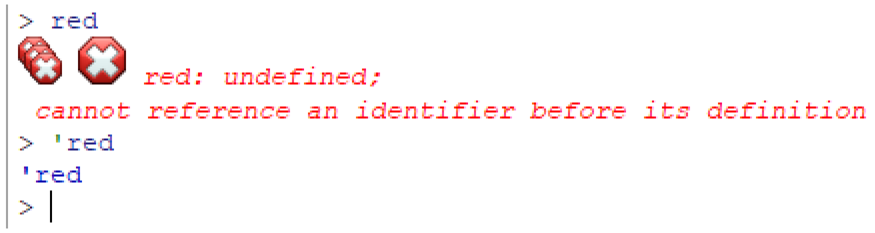
```
Welcome to DrRacket, version 8.2 [cs].
Language: racket, with debugging; memory limit: 128 MB.
> 9
9
> "red"
"red"
> 'red
'red
>
```

Interactions - Variants of the quote special form



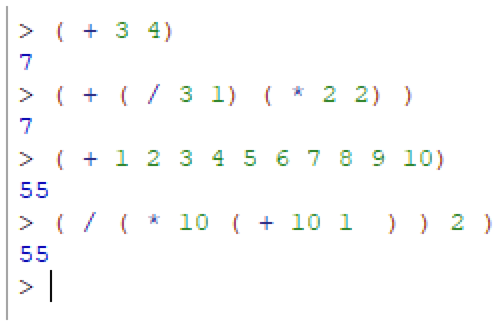
```
Welcome to DrRacket, version 8.2 [cs].
Language: racket, with debugging; memory limit: 128 MB.
> (quote red)
'red
> 'red
'red
>
```

Interactions - Illustrating the "unbound variable" error



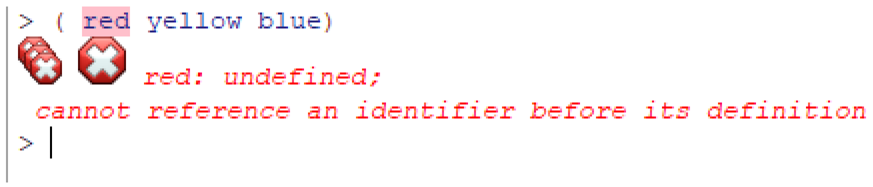
```
> red
red: undefined;
 cannot reference an identifier before its definition
> 'red
'red
> 
```

Interactions - Examples of standard form evaluation



```
> ( + 3 4)
7
> ( + ( / 3 1) ( * 2 2) )
7
> ( + 1 2 3 4 5 6 7 8 9 10)
55
> ( / ( * 10 ( + 10 1  ) ) 2 )
55
> |
```

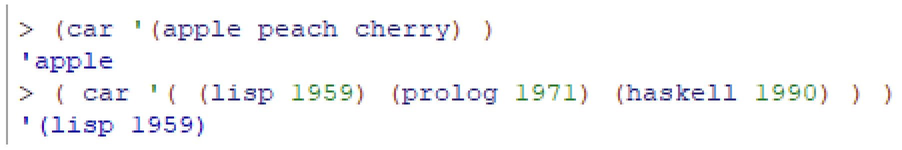
Interactions - Illustrating the “unbound function” error



```
> ( red yellow blue)
        red: undefined;
 cannot reference an identifier before its definition
> |
```

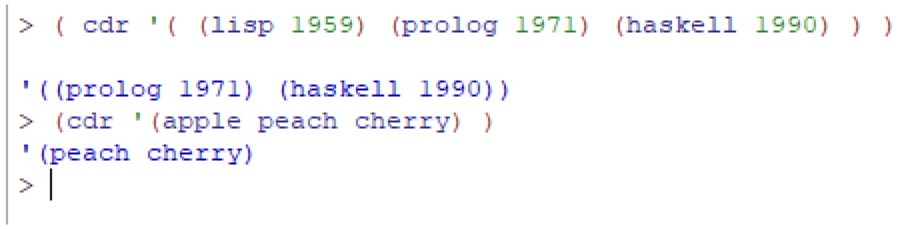
CAR CDR and CONS

Interactions – Examples of the car function



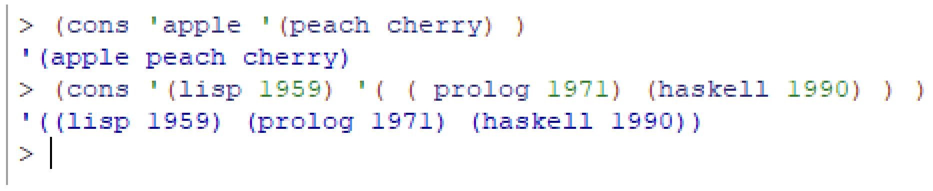
```
> (car '(apple peach cherry) )
'apple
> ( car '( (lisp 1959) (prolog 1971) (haskell 1990) ) )
'(lisp 1959)
```

Interactions – Examples of the cdr function



```
> ( cdr '( (lisp 1959) (prolog 1971) (haskell 1990) ) )

'((prolog 1971) (haskell 1990))
> (cdr '(apple peach cherry) )
'(peach cherry)
> |
```

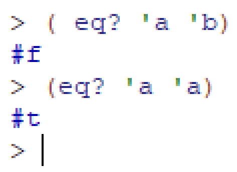
Interactions - Examples of the cons function



```
> (cons 'apple '(peach cherry) )
'(apple peach cherry)
> (cons '(lisp 1959) '( ( prolog 1971) (haskell 1990) ) )
'((lisp 1959) (prolog 1971) (haskell 1990))
> |
```

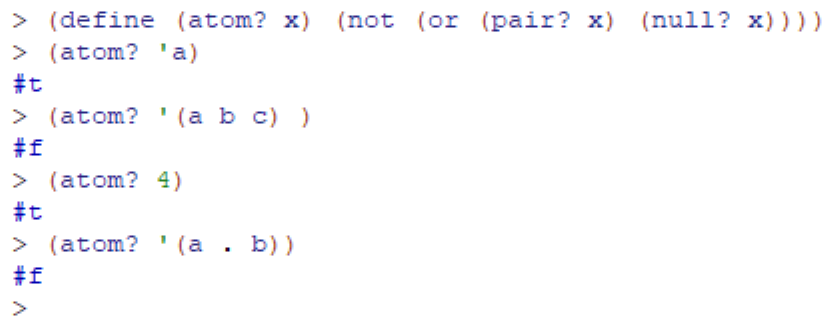
EQ and ATOM

Interactions - Examples of the eq? function



```
> ( eq? 'a 'b)
#f
> (eq? 'a 'a)
#t
> |
```

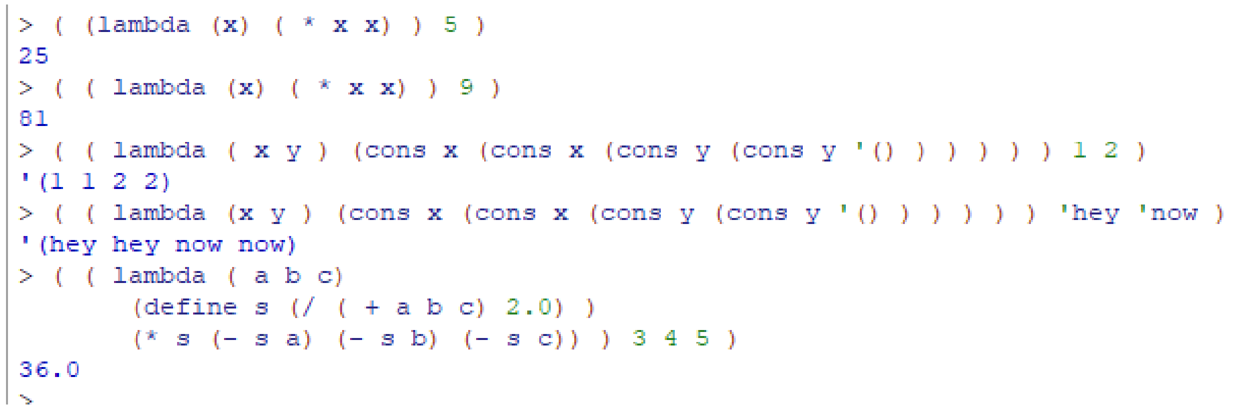
Interactions - Examples of the atom? Function



```
> (define (atom? x) (not (or (pair? x) (null? x))))
> (atom? 'a)
#t
> (atom? '(a b c) )
#f
> (atom? 4)
#t
> (atom? '(a . b))
#f
>
```

Lambda

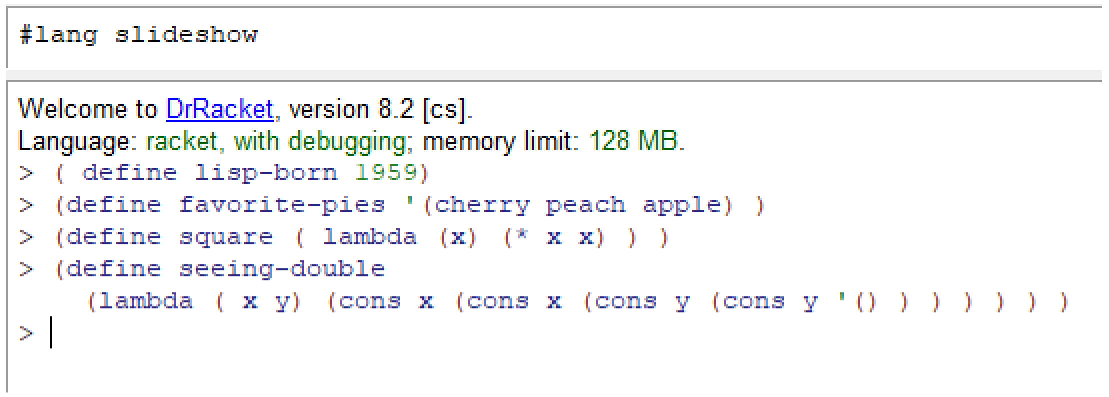
Interactions – Interactions featuring lambda function application



```
> ( (lambda (x) ( * x x) ) 5 )
25
> ( ( lambda (x) ( * x x) ) 9 )
81
> ( ( lambda ( x y ) (cons x (cons x (cons y (cons y '() ) ) ) ) ) 1 2 )
'(1 1 2 2)
> ( ( lambda (x y ) (cons x (cons x (cons y (cons y '() ) ) ) ) ) 'hey 'now )
'(hey hey now now)
> ( ( lambda ( a b c)
       (define s (/ ( + a b c) 2.0) )
       (* s (- s a) (- s b) (- s c)) ) 3 4 5 )
36.0
>
```

Define

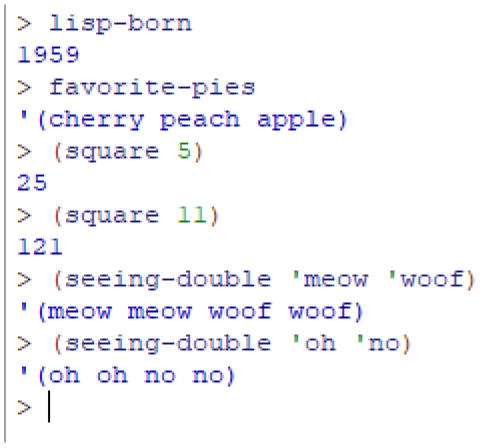
Definitions – Defining four items, two variables and two functions



```
#lang slideshow
```

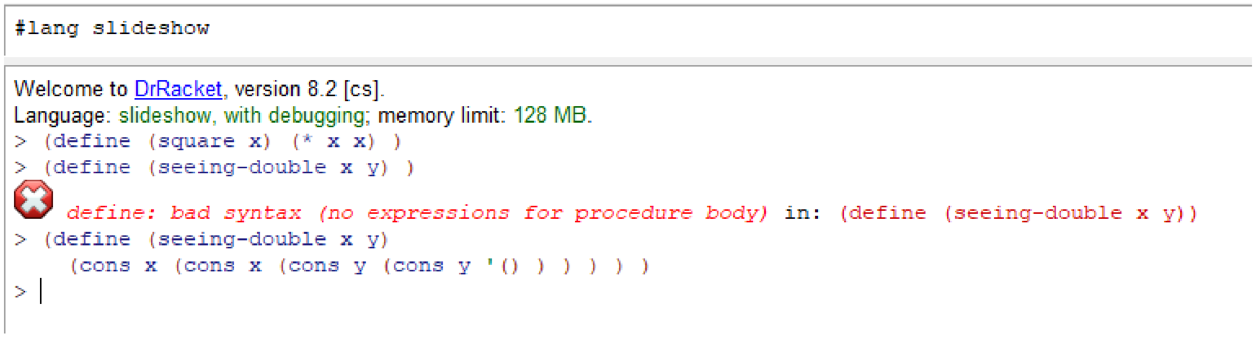
```
Welcome to DrRacket, version 8.2 [cs].
Language: racket, with debugging; memory limit: 128 MB.
> ( define lisp-born 1959)
> (define favorite-pies '(cherry peach apple) )
> (define square ( lambda (x) (* x x) ) )
> (define seeing-double
    (lambda ( x y) (cons x (cons x (cons y (cons y '() ) ) ) ) ) )
> |
```

Interactions – Referencing the two variables and applying the two functions



```
> lisp-born
1959
> favorite-pies
'(cherry peach apple)
> (square 5)
25
> (square 11)
121
> (seeing-double 'meow 'woof)
'(meow meow woof woof)
> (seeing-double 'oh 'no)
'(oh oh no no)
> |
```

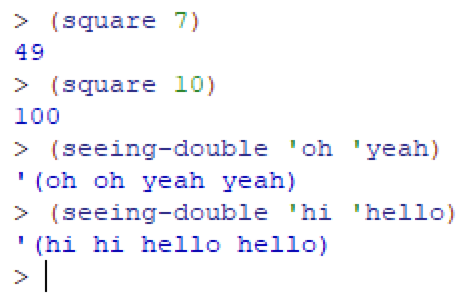
Definitions – Redefining the two functions (do it in a fresh pane)



```
#lang slideshow
```

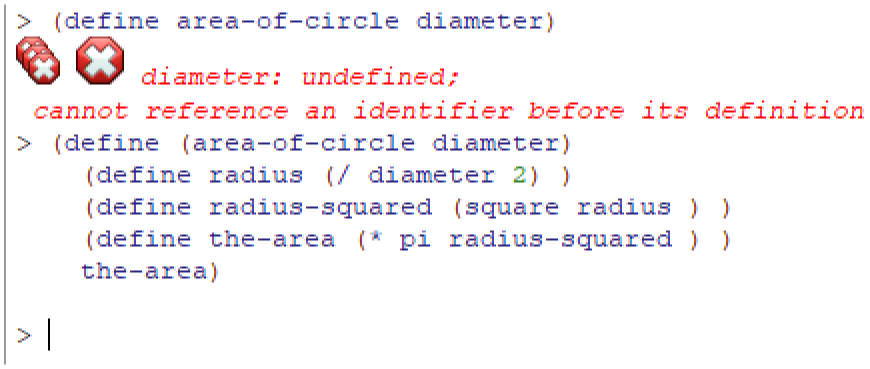
```
Welcome to DrRacket, version 8.2 [cs].
Language: slideshow, with debugging; memory limit: 128 MB.
> (define (square x) (* x x) )
> (define (seeing-double x y) )
  define: bad syntax (no expressions for procedure body) in: (define (seeing-double x y))
> (define (seeing-double x y)
    (cons x (cons x (cons y (cons y '() ) ) ) ) )
> |
```

Interactions – Illustrating the application of these functions (even though this was not explicitly indicated in the lesson)



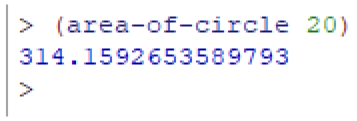
```
> (square 7)
49
> (square 10)
100
> (seeing-double 'oh 'yeah)
'(oh oh yeah yeah)
> (seeing-double 'hi 'hello)
'(hi hi hello hello)
> |
```

Definitions – Defining the area-of-circle function



```
> (define area-of-circle diameter)
      diameter: undefined;
 cannot reference an identifier before its definition
> (define (area-of-circle diameter)
    (define radius (/ diameter 2) )
    (define radius-squared (square radius ) )
    (define the-area (* pi radius-squared ) )
   the-area)

> |
```

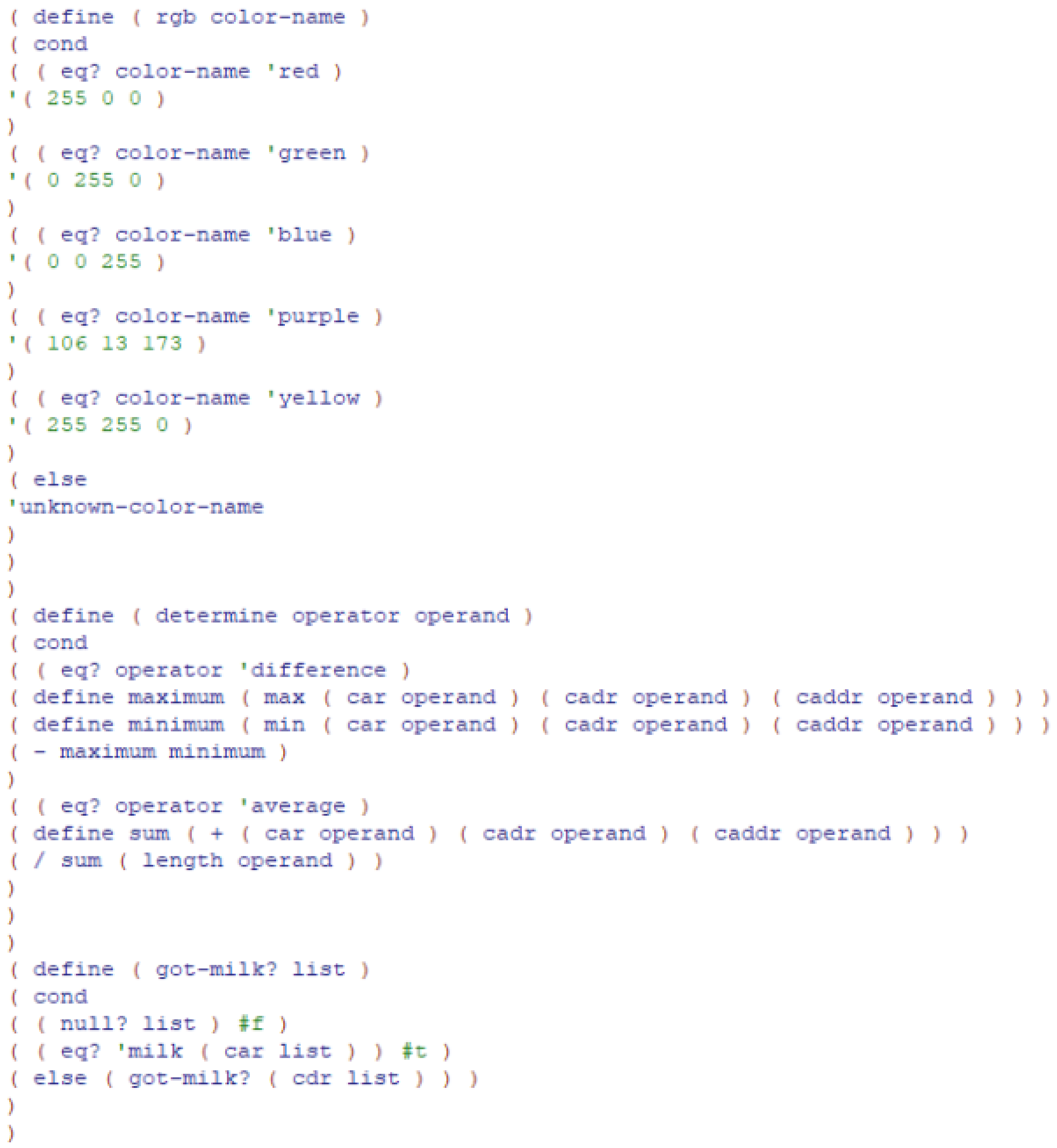
Interactions – Testing the area-of-circle function



```
> (area-of-circle 20)
314.1592653589793
>
```

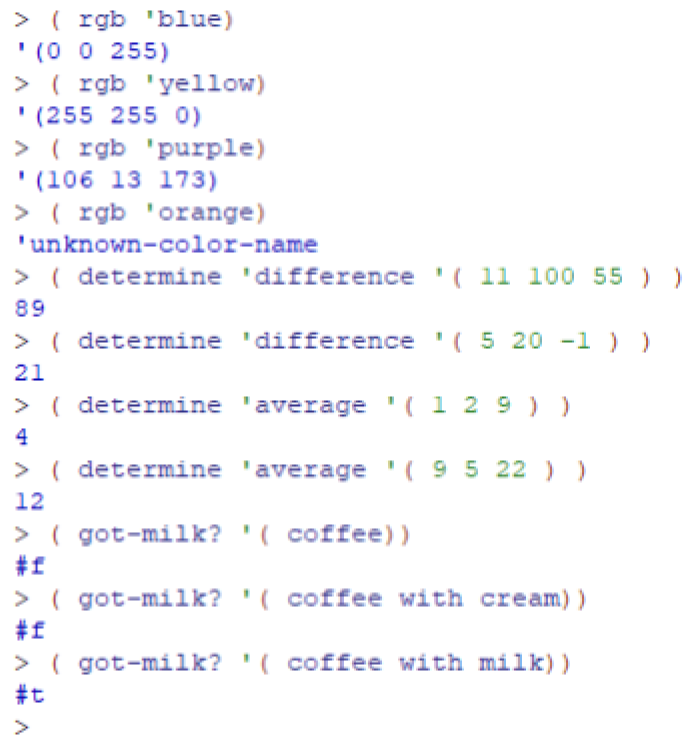
Cond

Definitions – Defining the rgb, determine, and got-milk function



```
( define ( rgb color-name )
( cond
( ( eq? color-name 'red )
'( 255 0 0 )
)
( ( eq? color-name 'green )
'( 0 255 0 )
)
( ( eq? color-name 'blue )
'( 0 0 255 )
)
( ( eq? color-name 'purple )
'( 106 13 173 )
)
( ( eq? color-name 'yellow )
'( 255 255 0 )
)
( else
'unknown-color-name
)
)
)
( define ( determine operator operand )
( cond
( ( eq? operator 'difference )
( define maximum ( max ( car operand ) ( cadr operand ) ( caddr operand ) ) )
( define minimum ( min ( car operand ) ( cadr operand ) ( caddr operand ) ) )
( - maximum minimum )
)
( ( eq? operator 'average )
( define sum ( + ( car operand ) ( cadr operand ) ( caddr operand ) ) )
( / sum ( length operand ) )
)
)
)
( define ( got-milk? list )
( cond
( ( null? list ) #f )
( ( eq? 'milk ( car list ) ) #t )
( else ( got-milk? ( cdr list ) ) )
)
)
```

Interactions – Mimicking the demo illustrating application of the three functions

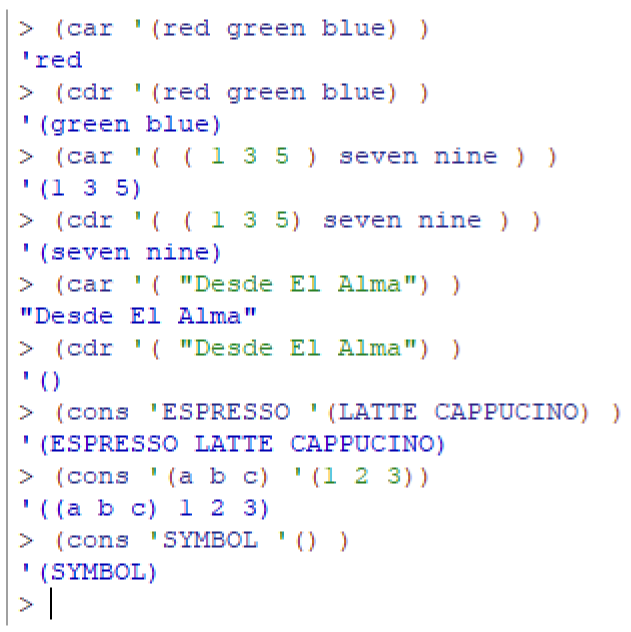


```
> ( rgb 'blue)
'(0 0 255)
> ( rgb 'yellow)
'(255 255 0)
> ( rgb 'purple)
'(106 13 173)
> ( rgb 'orange)
'unknown-color-name
> ( determine 'difference '( 11 100 55 ) )
89
> ( determine 'difference '( 5 20 -1 ) )
21
> ( determine 'average '( 1 2 9 ) )
4
> ( determine 'average '( 9 5 22 ) )
12
> ( got-milk? '( coffee))
#f
> ( got-milk? '( coffee with cream))
#f
> ( got-milk? '( coffee with milk))
#t
>
```

Task 2 - References and Constructors

Racket Session featuring CAR, CDR, and CONS

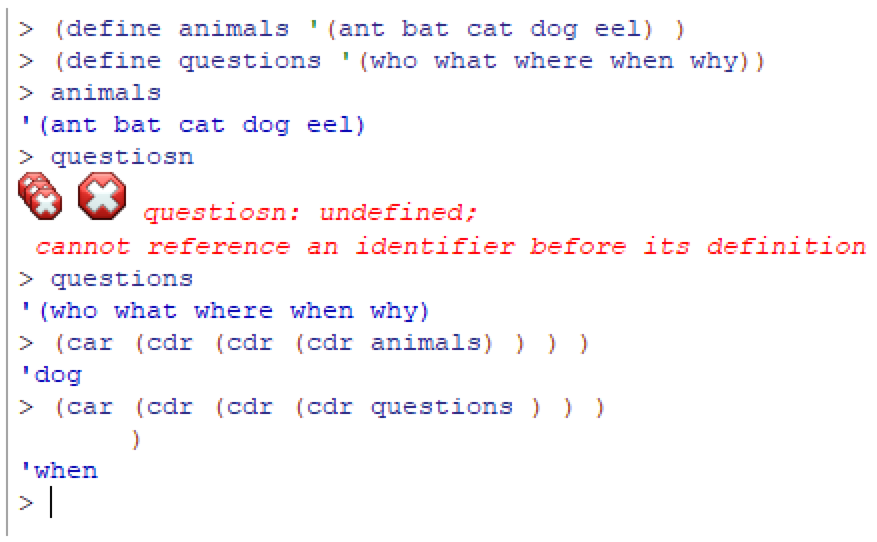
Interactions – Applying CAR, CDR, and CONS



```
> (car '(red green blue) )
'red
> (cdr '(red green blue) )
'(green blue)
> (car '( ( 1 3 5 ) seven nine ) )
'(1 3 5)
> (cdr '( ( 1 3 5) seven nine ) )
'(seven nine)
> (car '( "Desde El Alma") )
"Desde El Alma"
> (cdr '( "Desde El Alma") )
'()
> (cons 'ESPRESSO '(LATTE CAPPUCINO) )
'(ESPRESSO LATTE CAPPUCINO)
> (cons '(a b c) '(1 2 3))
'((a b c) 1 2 3)
> (cons 'SYMBOL '() )
'(SYMBOL)
> 
```

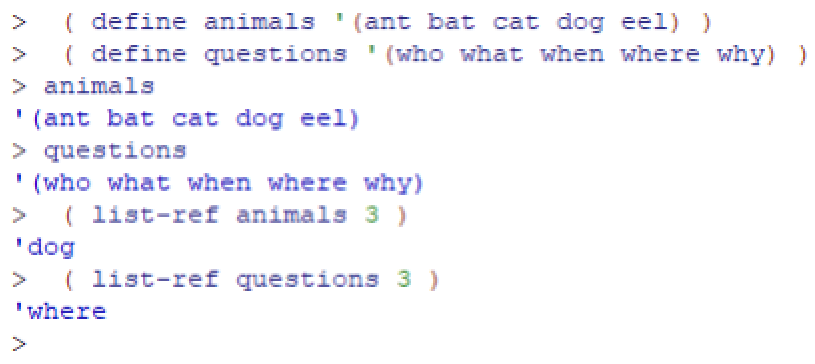
Referencing a list element

Interactions – Referencing a list element from scratch



```
> (define animals '(ant bat cat dog eel) )
> (define questions '(who what where when why))
> animals
'(ant bat cat dog eel)
> questiosn
 questiosn: undefined;
 cannot reference an identifier before its definition
> questions
'(who what where when why)
> (car (cdr (cdr (cdr animals) ) ) )
'dog
> (car (cdr (cdr (cdr questions ) ) )
       )
'when
> 
```

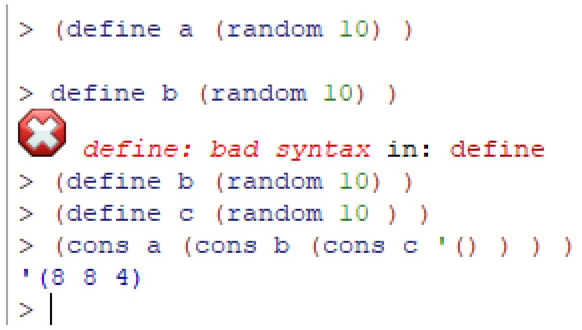
Interactions – Referencing a list element from using list-ref



```
>  ( define animals '(ant bat cat dog eel) )
>  ( define questions '(who what when where why) )
> animals
'(ant bat cat dog eel)
> questions
'(who what when where why)
>  ( list-ref animals 3 )
'dog
>  ( list-ref questions 3 )
'where
>
```

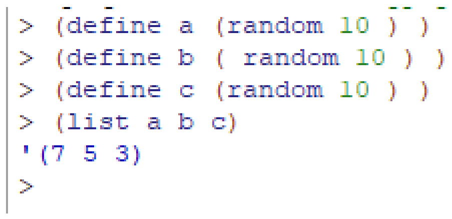
Creating a list

Interactions – Creating a list from scratch



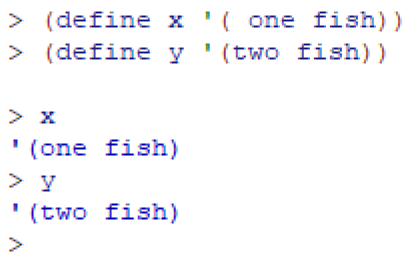
```
> (define a (random 10) )

> define b (random 10) )
  define: bad syntax in: define
> (define b (random 10) )
> (define c (random 10 ) )
> (cons a (cons b (cons c '() ) ) )
'(8 8 4)
> |
```

Interactions – Creating a list using list



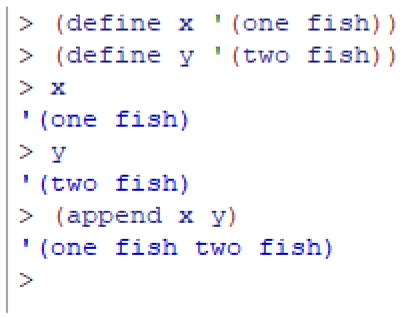
```
> (define a (random 10 ) )
> (define b ( random 10 ) )
> (define c (random 10 ) )
> (list a b c)
'(7 5 3)
>
```

Interactions – Appending two lists from scratch



```
> (define x '( one fish))
> (define y '(two fish))

> x
'(one fish)
> y
'(two fish)
>
```

Interactions – Appending two lists using append

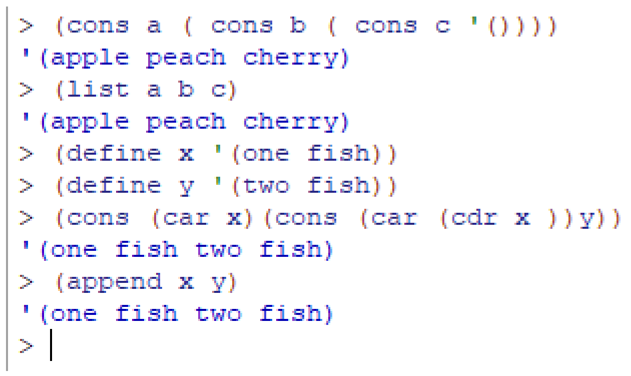


```
> (define x '(one fish))
> (define y '(two fish))
> x
'(one fish)
> y
'(two fish)
> (append x y)
'(one fish two fish)
>
```

Redacted Racket Session Featuring Referencers and Constructors

Interactions – Mindfully doing the redacted session, for real

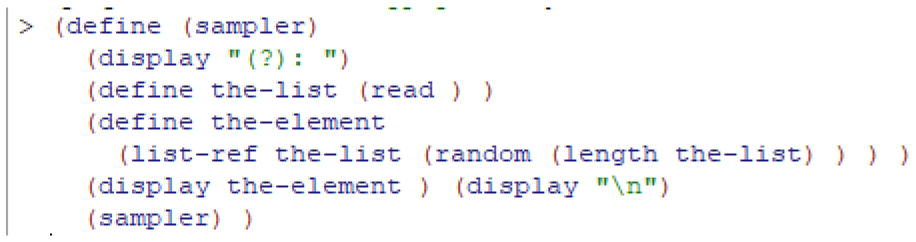
```
> (define languages '(racket prolog haskell rust))
> languages
'(racket prolog haskell rust)
> 'languages
'languages
> ( quote languages)
'languages
> ( car languages)
'racket
> ( cdr languages)
'(prolog haskell rust)
> ( car ( cdr languages))

'prolog
> ( cdr ( cdr languages))
'(haskell rust)
> ( cadr languages)
'prolog
> ( cddr languages)
'(haskell rust)
> (first languages)
'racket
> (second languages)
'prolog
> (third languages)
'haskell
> (list-ref languages 2)
'haskell
> (define numbers '(1 2 3))
> (define letters '(a b c))
> ( cons numbers letters)
'((1 2 3) a b c)
> (list numbers letters)
'((1 2 3) (a b c))
> (append numbers letters)

'(1 2 3 a b c)
> (define animals '(ant bat cat dot eel))
> (car ( cdr ( cdr ( cdr animals))))
'dot
> (caddr animals)
'cat
> (list-ref animals 3)
'dot
> (define a 'apple)
> (define b 'peach)
> (define c 'cherry)
```



```
> (cons a ( cons b ( cons c '())))
'(apple peach cherry)
> (list a b c)
'(apple peach cherry)
> (define x '(one fish))
> (define y '(two fish))
> (cons (car x)(cons (car (cdr x ))y))
'(one fish two fish)
> (append x y)
'(one fish two fish)
> 
```

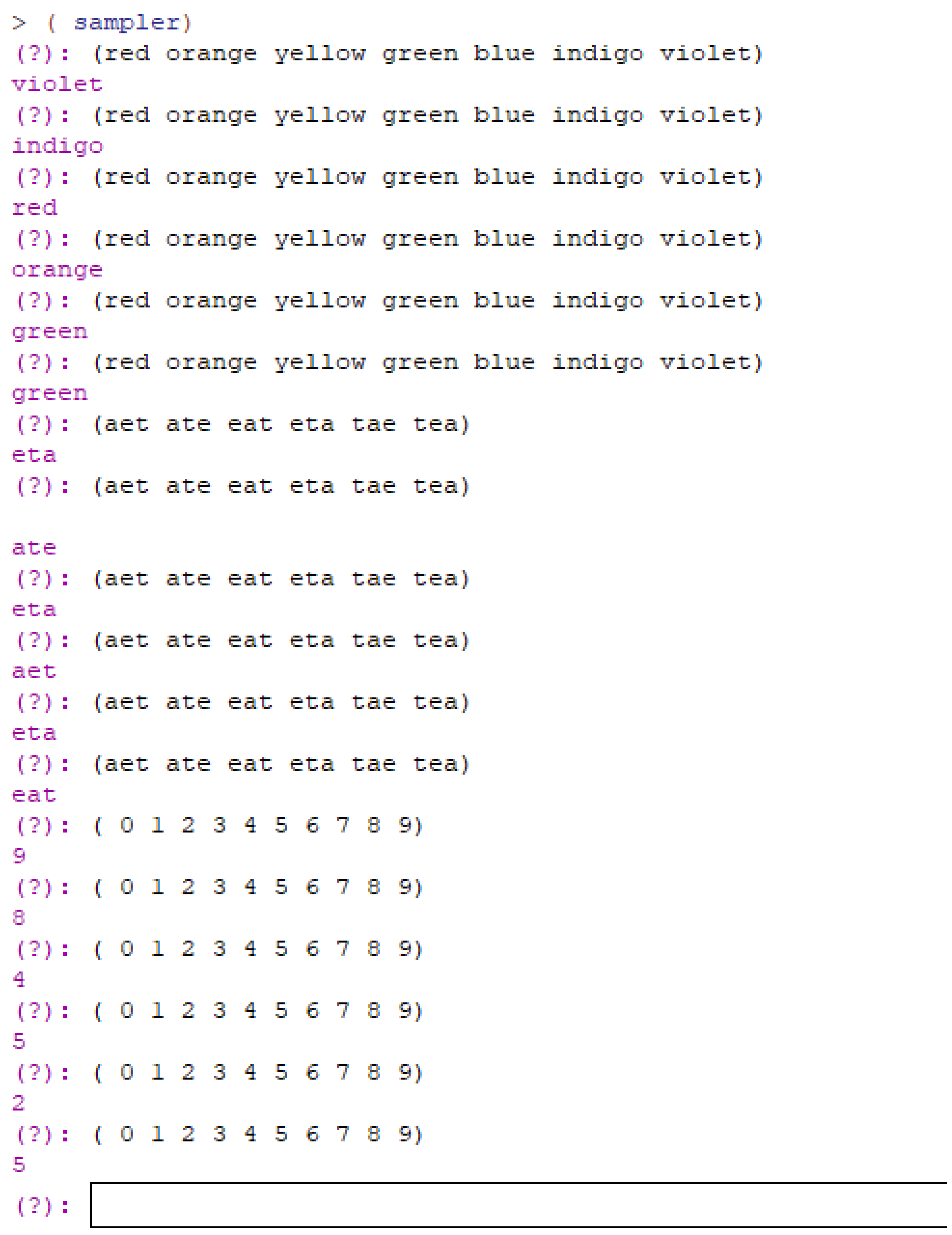
Task 3 - Random Selection

Definitions – Defining the sampler program



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

Interactions – Mimicking the sampler program demo



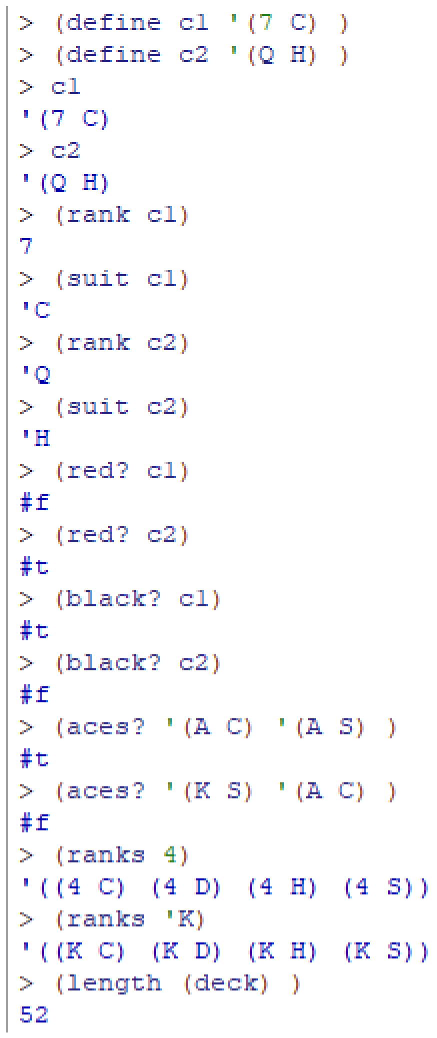
```
> ( sampler)
(?): (red orange yellow green blue indigo violet)
violet
(?): (red orange yellow green blue indigo violet)
indigo
(?): (red orange yellow green blue indigo violet)
red
(?): (red orange yellow green blue indigo violet)
orange
(?): (red orange yellow green blue indigo violet)
green
(?): (red orange yellow green blue indigo violet)
green
(?): (aet ate eat eta tae tea)
eta
(?): (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)
eta
(?): (aet ate eat eta tae tea)
eat
(?): ( 0 1 2 3 4 5 6 7 8 9)
9
(?): ( 0 1 2 3 4 5 6 7 8 9)
8
(?): ( 0 1 2 3 4 5 6 7 8 9)
4
(?): ( 0 1 2 3 4 5 6 7 8 9)
5
(?): ( 0 1 2 3 4 5 6 7 8 9)
2
(?): ( 0 1 2 3 4 5 6 7 8 9)
5
(?):
```

Task 4 - Playing Card Programming Challenge

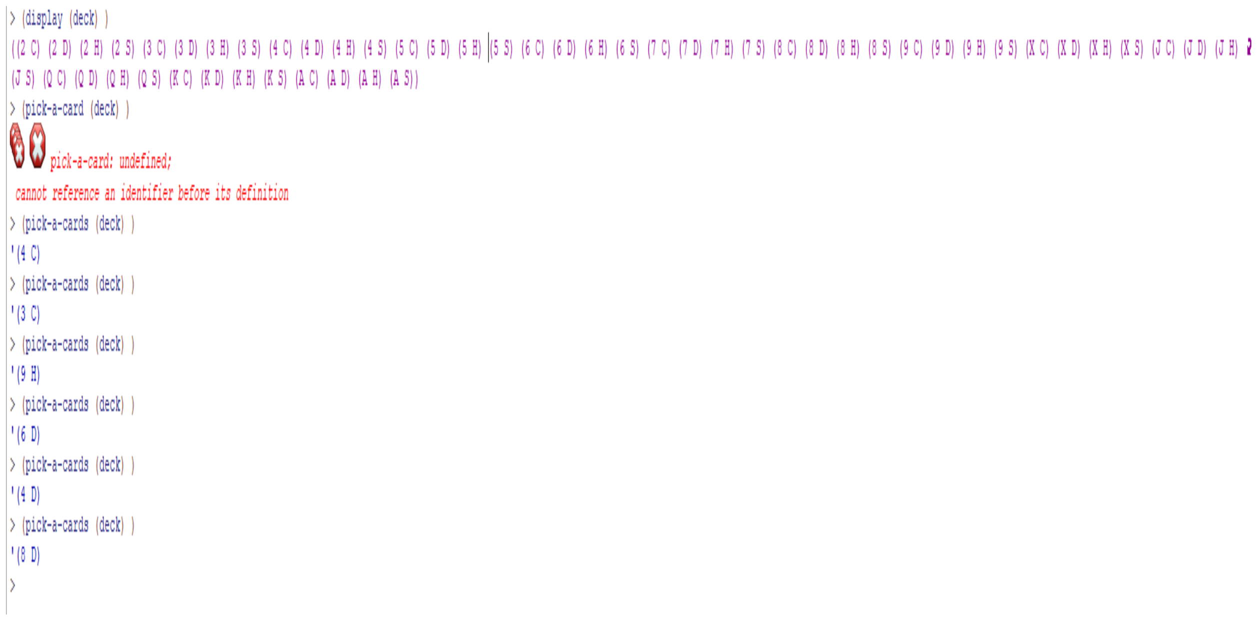
Definitions – Programming the card playing functionality

```
> ( 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-cards cards)
 (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)
 )
 )
```

Interactions – Mimicking the card playing functionality demo



```
> (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 (deck) )
pick-a-card: undefined;
 cannot reference an identifier before its definition
> (pick-a-cards (deck) )
'(4 C)
> (pick-a-cards (deck) )
'(3 C)
> (pick-a-cards (deck) )
'(9 H)
> (pick-a-cards (deck) )
'(6 D)
> (pick-a-cards (deck) )
'(4 D)
> (pick-a-cards (deck) )
'(8 D)
>
```