# Third Racket Programming Assignment

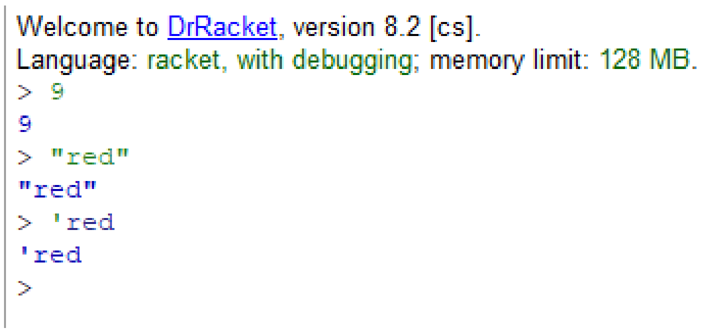
Abstract – This assignment demonstrates the functionality of a variety of LISP calls, demonstrated by mirroring the programming activities from Lesson 7 and 8.

## Task 1 – Historical LISP

Parroting Racket interactions and definitions from "Lesson 7: Historical LISP"

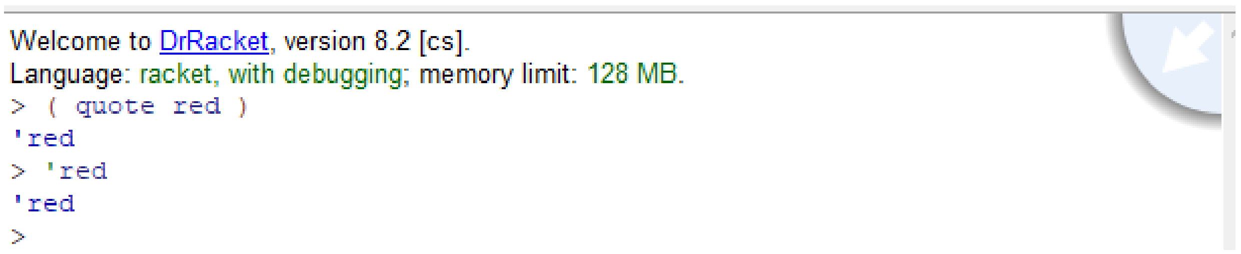
## Quote and Eval

Interactions – constants 9 and "red" and 'red



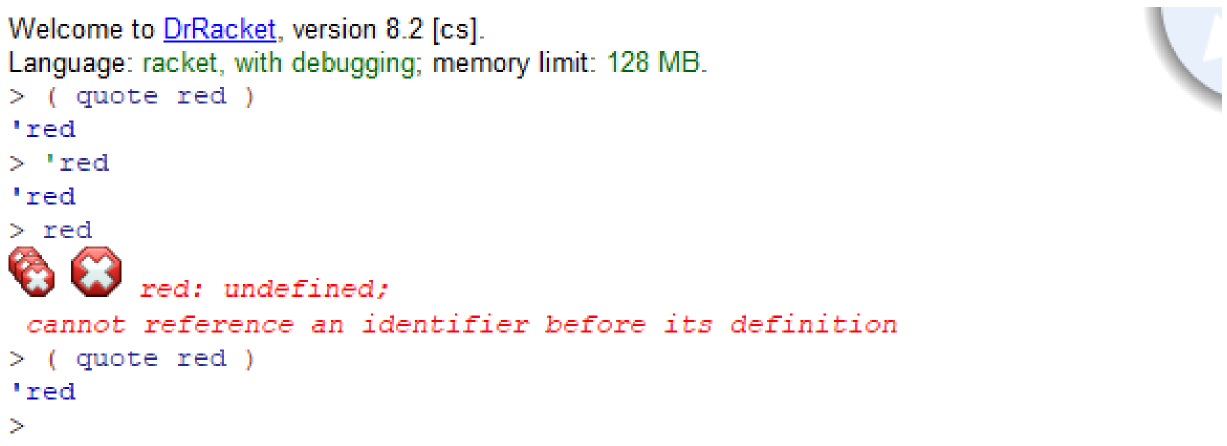
```
Welcome to DrRacket, version 8.2 [cs].
Language: racket, with debugging; memory limit: 128 MB.
> 9
9
> "red"
"red"
> 'red
'red
>
```

Interactions – variants of the quote special form



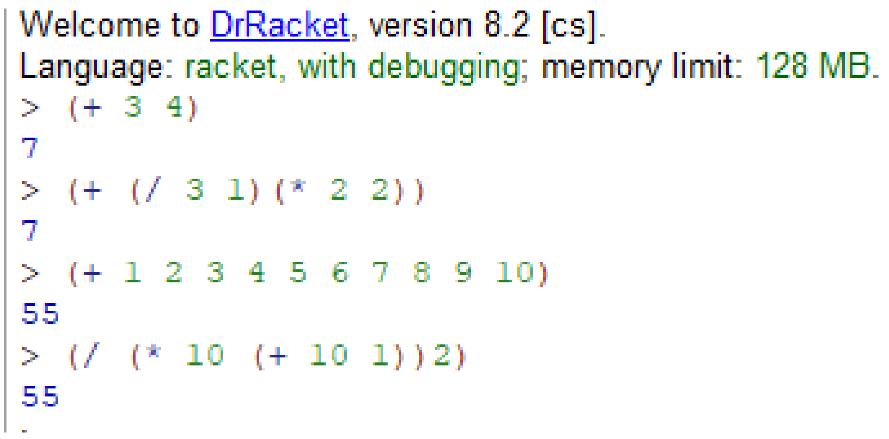
```
Welcome to DrRacket, version 8.2 [cs].
Language: racket, with debugging; memory limit: 128 MB.
> ( quote red )
'red
> 'red
'red
>
```

Interactions – illustrating the "unbound variable" error



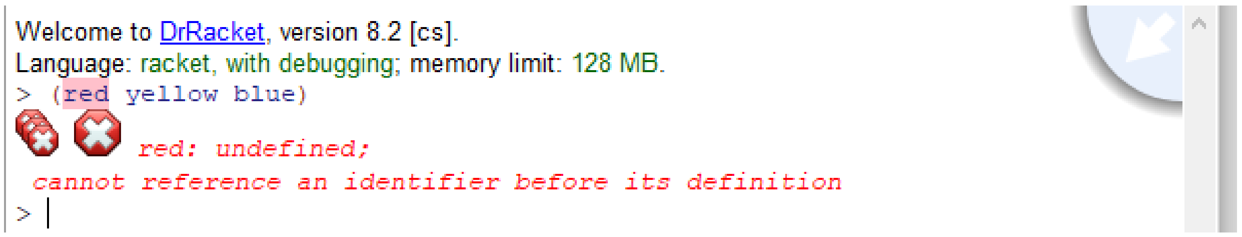
```
Welcome to DrRacket, version 8.2 [cs].
Language: racket, with debugging; memory limit: 128 MB.
> ( quote red )
'red
> 'red
'red
> red
red: undefined;
 cannot reference an identifier before its definition
> ( quote red )
'red
>
```

Interactions – examples of standard form evaluation



```
Welcome to DrRacket, version 8.2 [cs].
Language: racket, with debugging; memory limit: 128 MB.
> (+ 3 4)
7
> (+ (/ 3 1)(* 2 2))
7
> (+ 1 2 3 4 5 6 7 8 9 10)
55
> (/ (* 10 (+ 10 1))2)
55
```

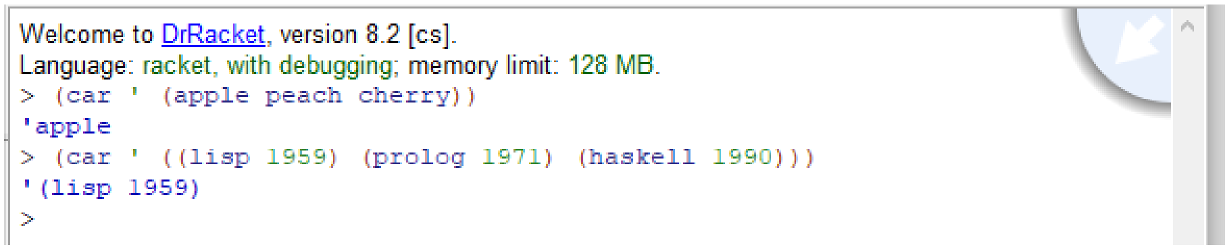
Interactions – illustrating the "unbound function" error



```
Welcome to DrRacket, version 8.2 [cs].
Language: racket, with debugging; memory limit: 128 MB.
> (red yellow blue)
red: undefined;
 cannot reference an identifier before its definition
>
```

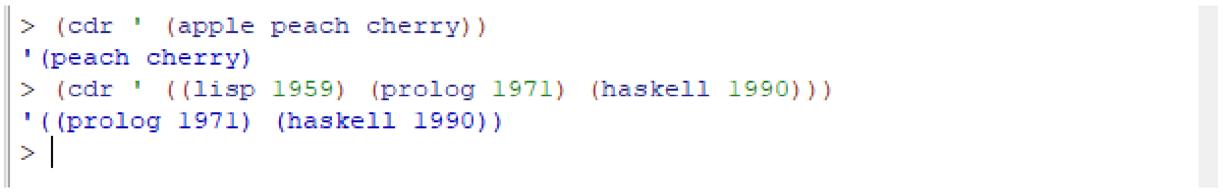
## Car, Cdr, and Cons

Interactions – examples of the car function



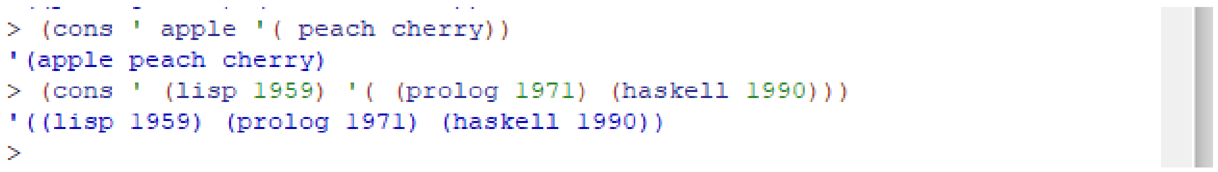
```
Welcome to DrRacket, version 8.2 [cs].
Language: racket, with debugging; memory limit: 128 MB.
> (car ' (apple peach cherry))
'apple
> (car ' ((lisp 1959) (prolog 1971) (haskell 1990)))
'(lisp 1959)
>
```

Interactions – examples of the cdr function



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

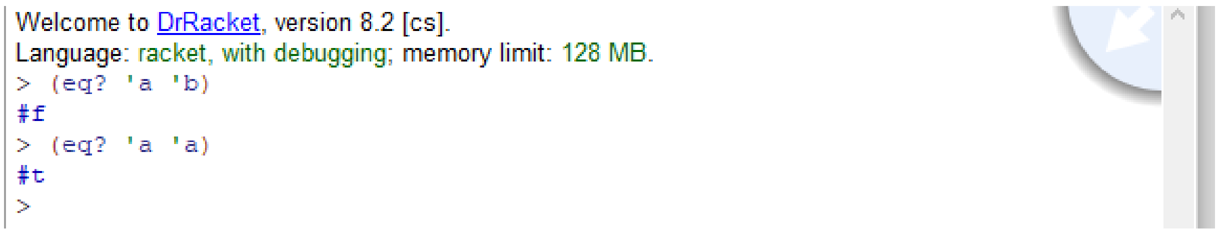
Interactions – examples of the cons function



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

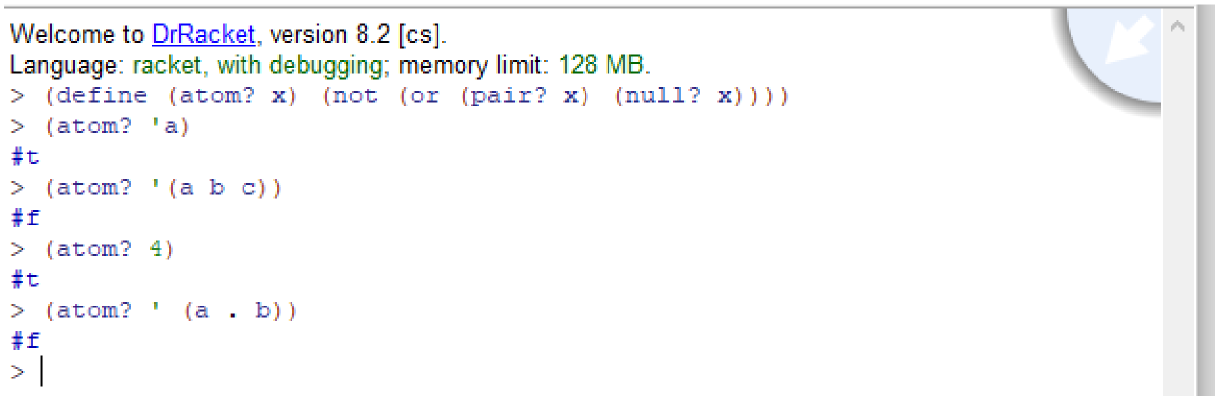
## Eq and Atom

Interactions – examples of the eq? function



```
Welcome to DrRacket, version 8.2 [cs].
Language: racket, with debugging; memory limit: 128 MB.
> (eq? 'a 'b)
#f
> (eq? 'a 'a)
#t
>
```

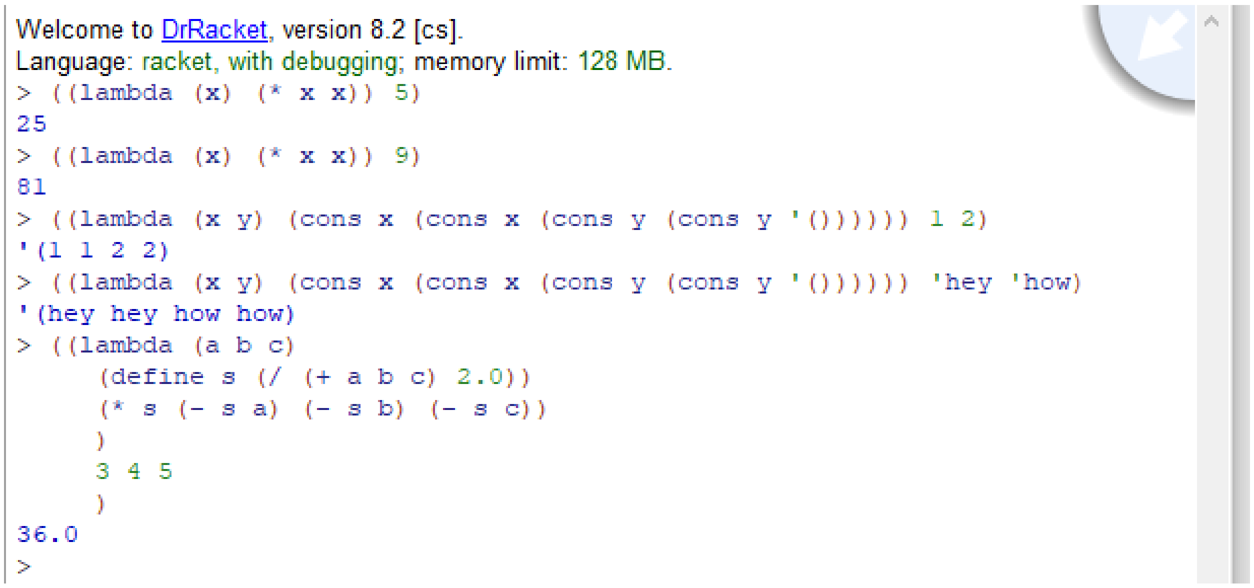
Interactions – examples of the atom? Function



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

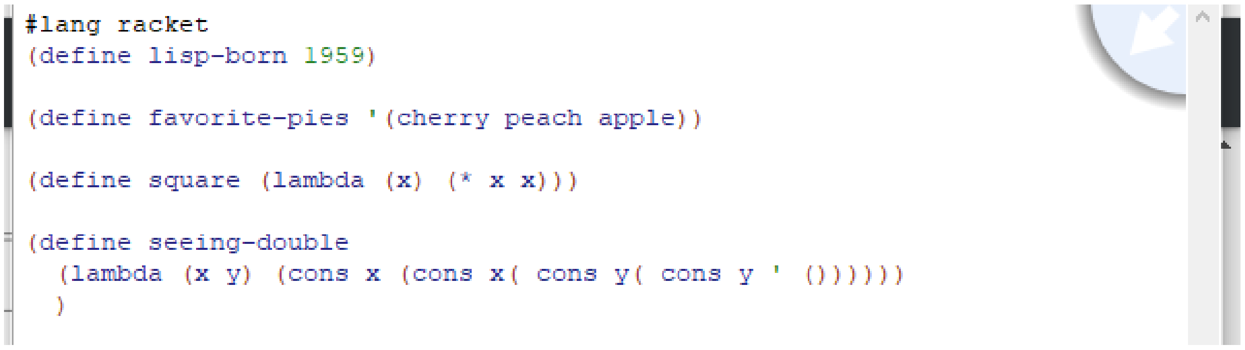
Interactions – Interactions featuring lambda function application



```
Welcome to DrRacket, version 8.2 [cs].
Language: racket, with debugging; memory limit: 128 MB.
> ((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 'how)
'(hey hey how how)
> ((lambda (a b c)
      (define s (/ (+ a b c) 2.0))
      (* s (- s a) (- s b) (- s c))
      )
      3 4 5
      )
36.0
>
```

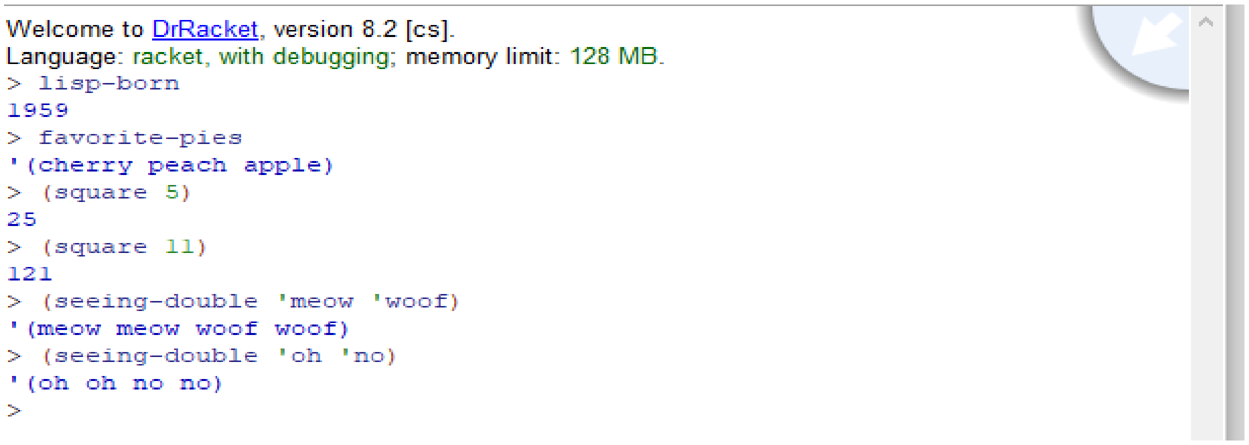
## Define

Definitions – defining four items, two variables, and two functions



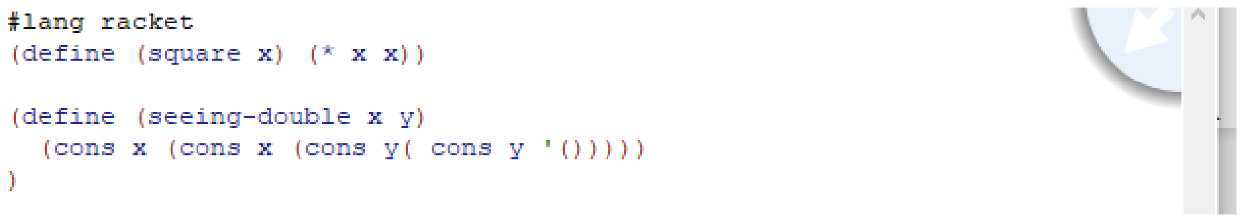
```
#lang racket
(define lisp-born 1959)

(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



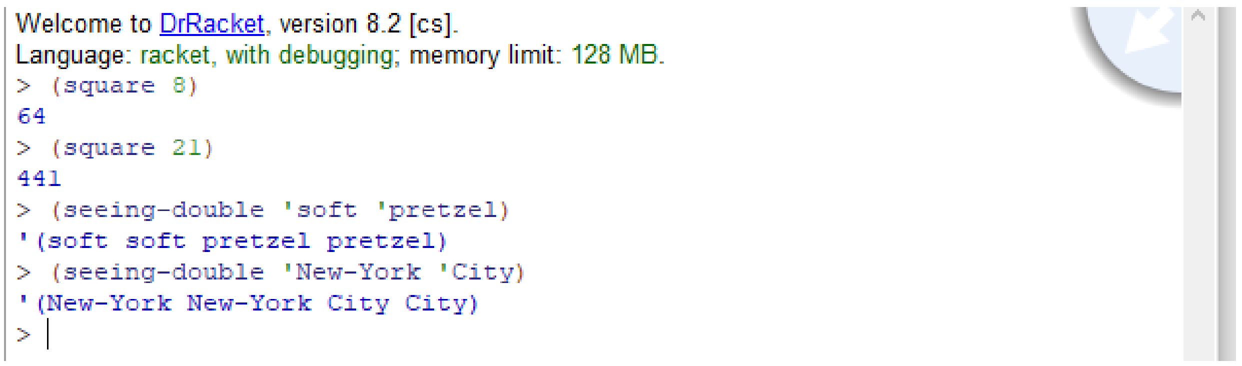
```
Welcome to DrRacket, version 8.2 [cs].
Language: racket, with debugging; memory limit: 128 MB.
> lisp-born
1959
> favorite-pies
'(cherry peach apple)
> (square 5)
25
> (square 11)
121
> (seeing-double 'meow 'woof)
'(meow meow woof woof)
> (seeing-double 'oh 'no)
'(oh oh no no)
>
```

Definitions – redefining the two functions



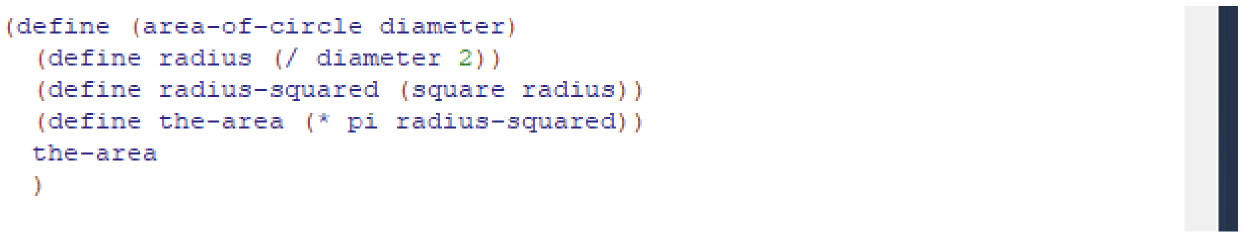
```
#lang racket
(define (square x)  (* x x))

(define (seeing-double x y)
  (cons x (cons x (cons y( cons y '())))))
)
```

Interactions – illustrating the applications of these functions



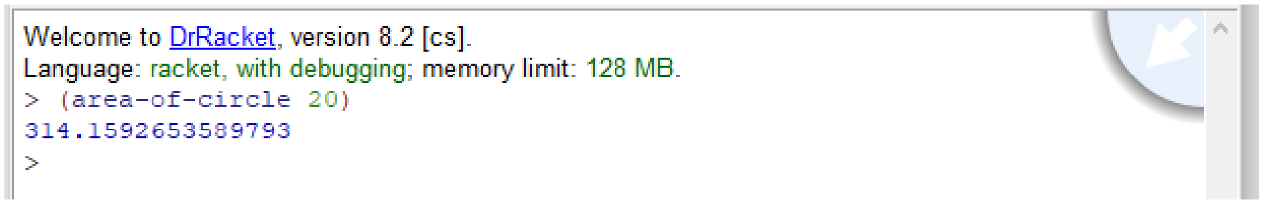
```
Welcome to DrRacket, version 8.2 [cs].
Language: racket, with debugging; memory limit: 128 MB.
> (square 8)
64
> (square 21)
441
> (seeing-double 'soft 'pretzel)
'(soft soft pretzel pretzel)
> (seeing-double 'New-York 'City)
'(New-York New-York City City)
> 
```

Definitions- defining the area-of-circle



```
(define (area-of-circle diameter)
  (define radius (/ diameter 2))
  (define radius-squared (square radius))
  (define the-area (* pi radius-squared))
  the-area
  )
```

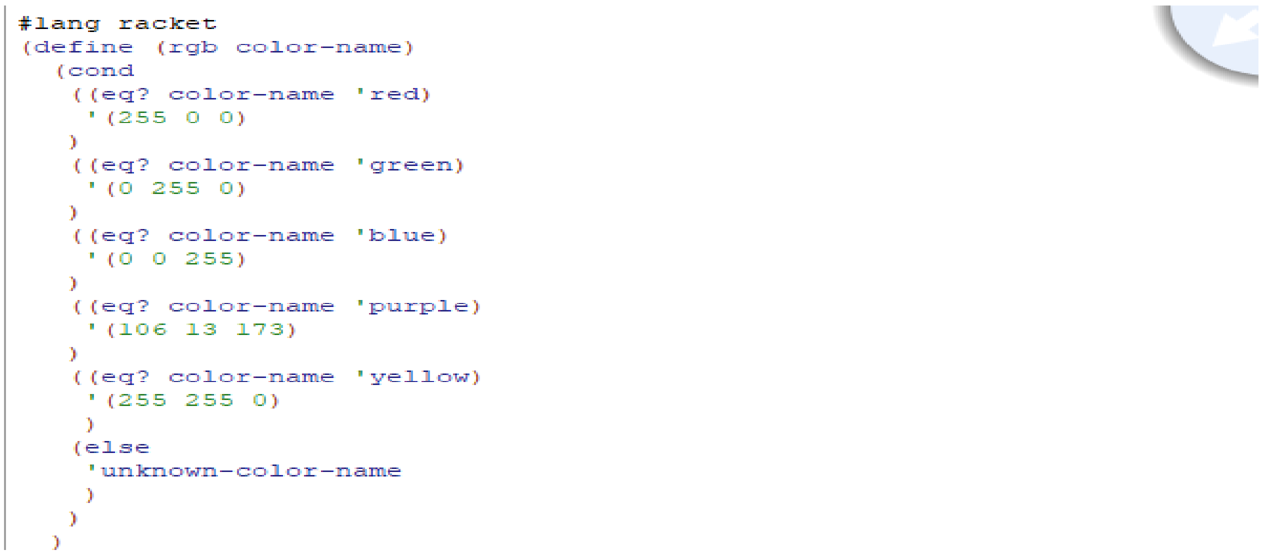
Interactions – testing the area-of-circle



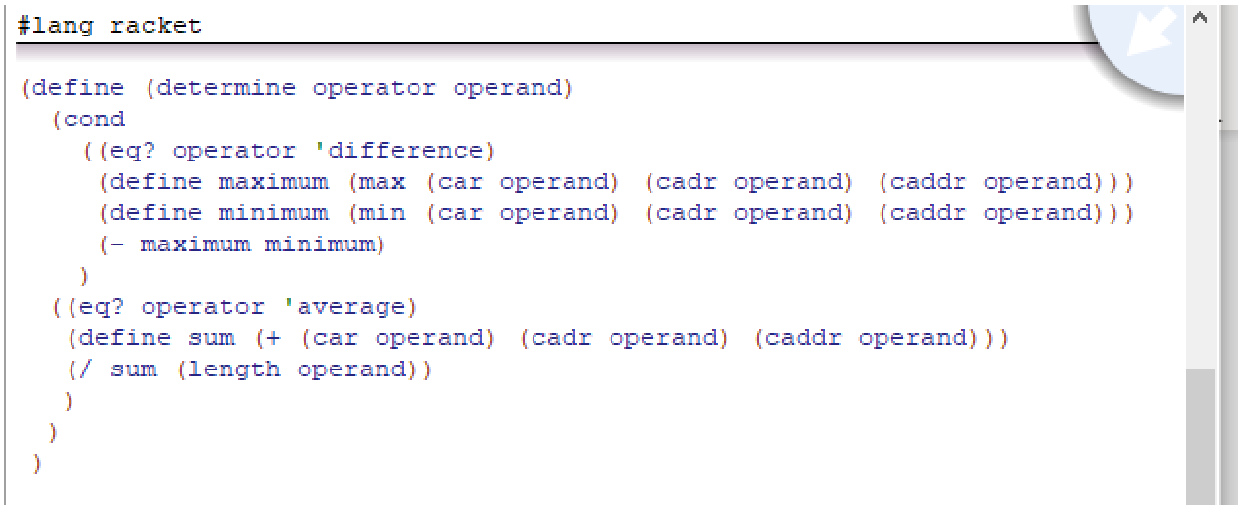
```
Welcome to DrRacket, version 8.2 [cs].
Language: racket, with debugging; memory limit: 128 MB.
> (area-of-circle 20)
314.1592653589793
>
```

## Cond

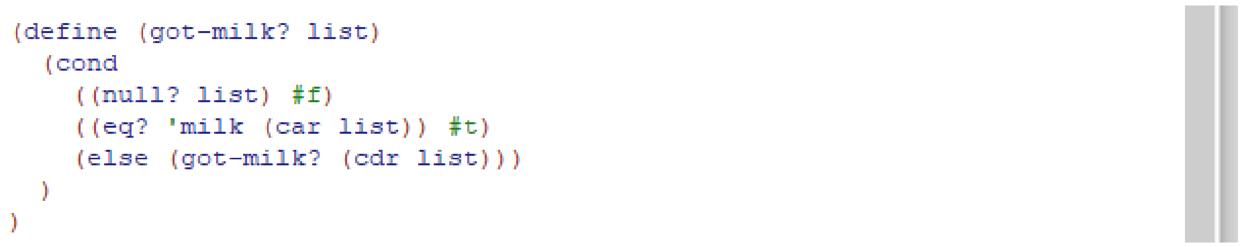
Definitions – defining the rgb, determine, and got-milk? Functions



```racket
#lang racket
(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
    )
   )
  )
```

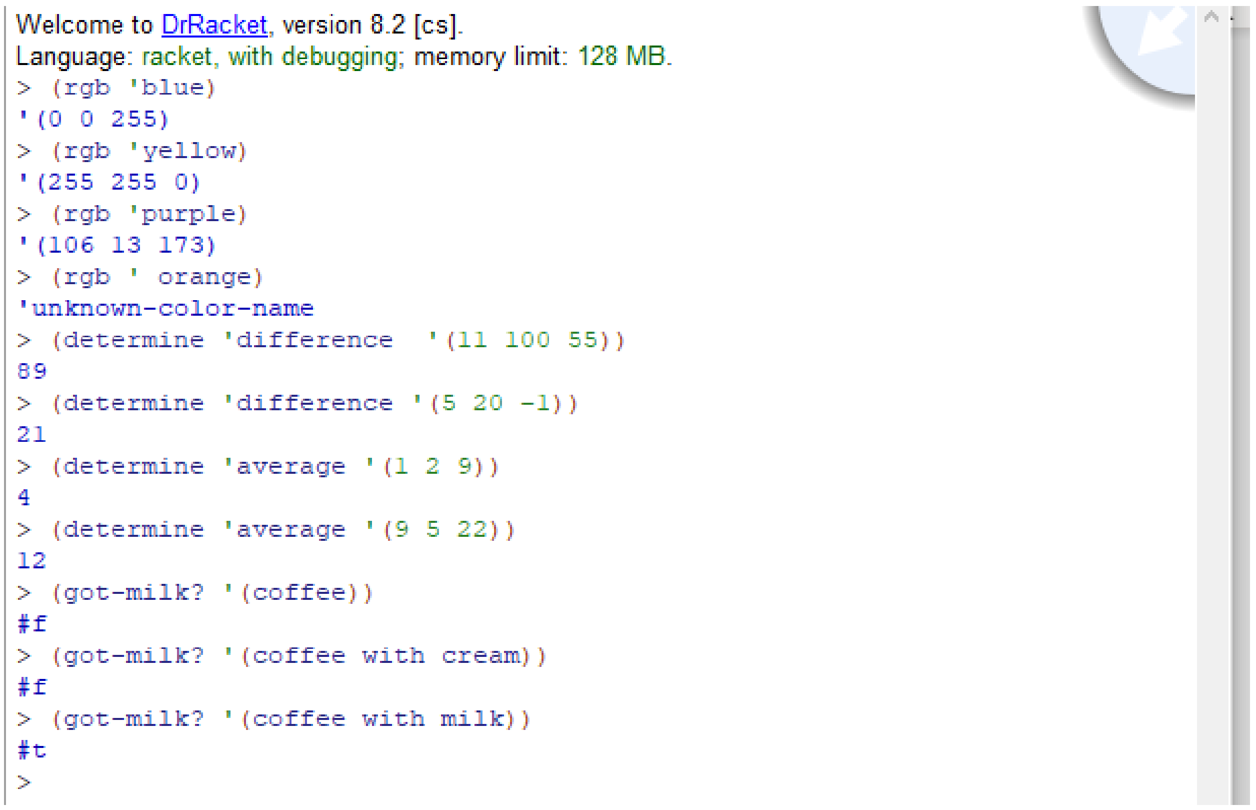


```racket
#lang racket

(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))
   )
  )
 )
```



```racket
(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



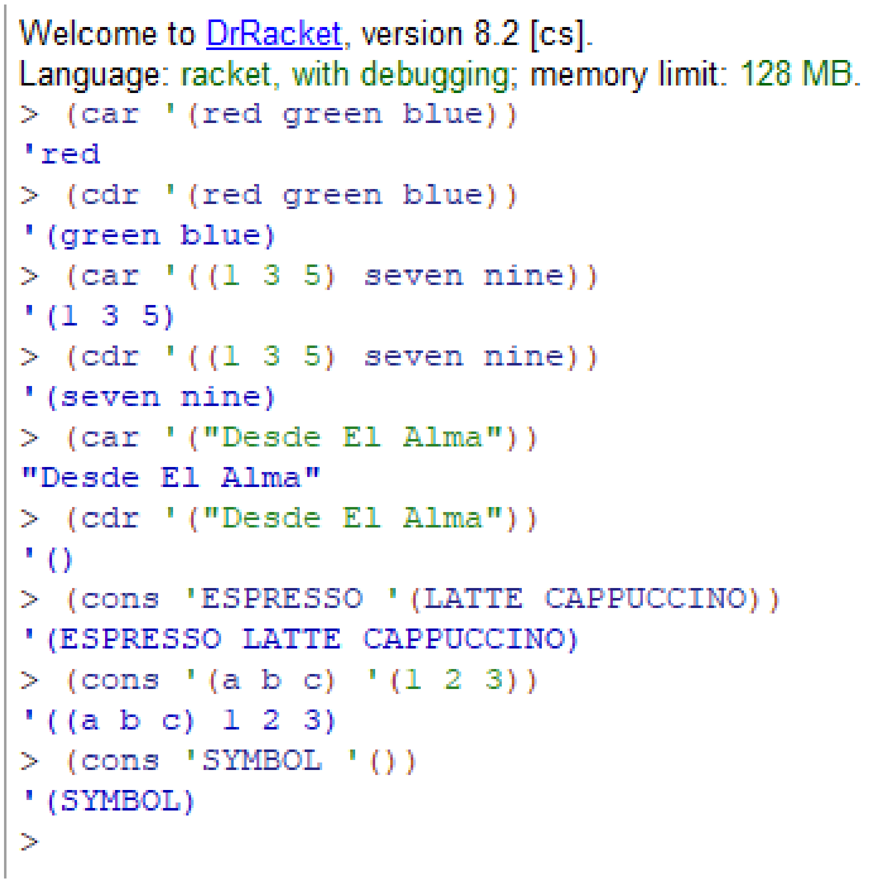
```
Welcome to DrRacket, version 8.2 [cs].
Language: racket, with debugging; memory limit: 128 MB.
> (rgb 'blue)
'(0 0 255)
> (rgb 'yellow)
'(255 255 0)
> (rgb 'purple)
'(106 13 173)
> (rgb ' orange)
'unknown-color-name
> (determine 'difference  '(11 100 55))
89
> (determine 'difference '(5 20 -1))
21
> (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 – Referencers and Constructors

Parroting Racket interactions and definitions from “Lesson 8: Basic List Processing” that pertain expressly to referencers and constructors

## Racket Session featuring CAR, CDR, and CONS

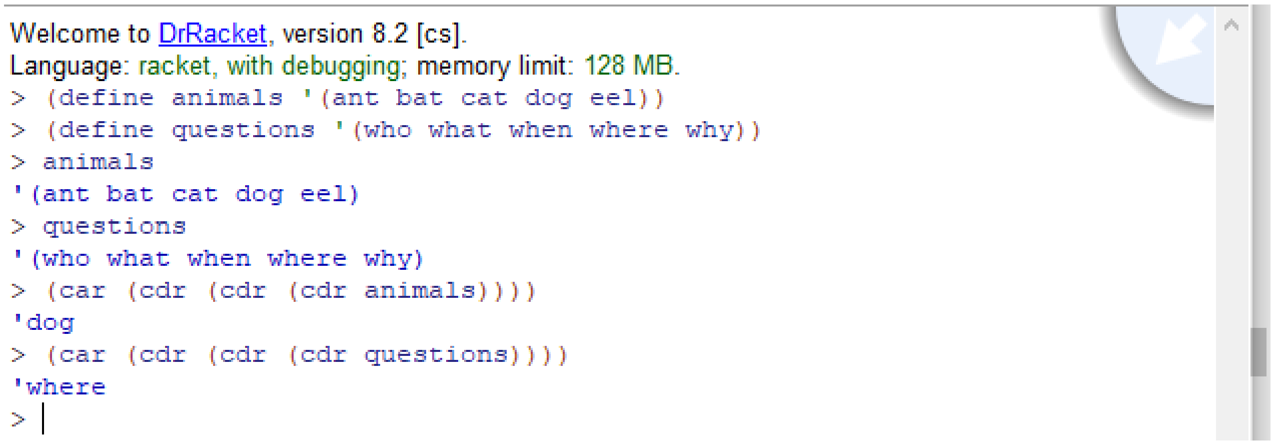
Interactions - applying CAR, CDR, and CONS



```
Welcome to DrRacket, version 8.2 [cs].
Language: racket, with debugging; memory limit: 128 MB.
> (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 CAPPUCCINO))
'(ESPRESSO LATTE CAPPUCCINO)
> (cons '(a b c) '(1 2 3))
'((a b c) 1 2 3)
> (cons 'SYMBOL '())
'(SYMBOL)
>
```

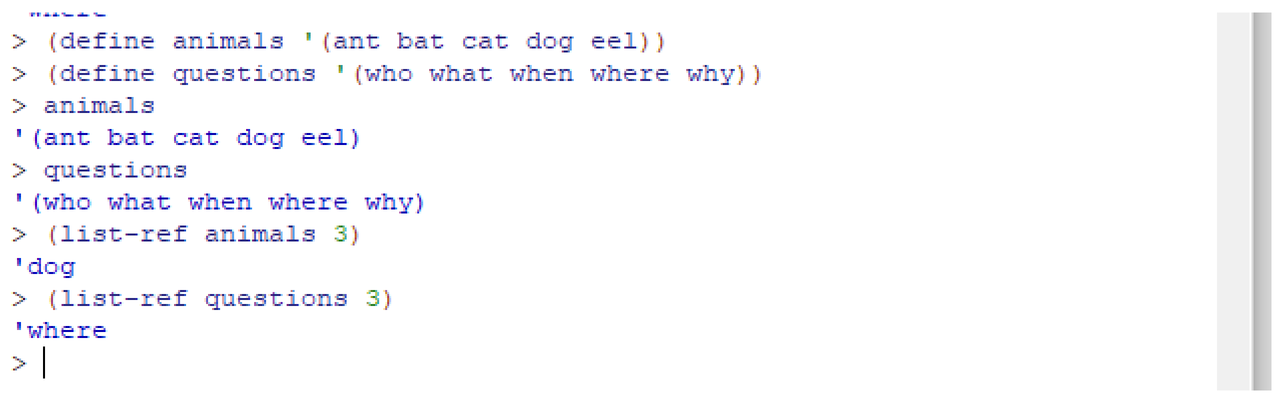
## Referencing a list element

Interactions – Referencing a list element from scratch



```
Welcome to DrRacket, version 8.2 [cs].
Language: racket, with debugging; memory limit: 128 MB.
> (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)
> (car (cdr (cdr (cdr animals))))
'dog
> (car (cdr (cdr (cdr questions))))
'where
> 
```

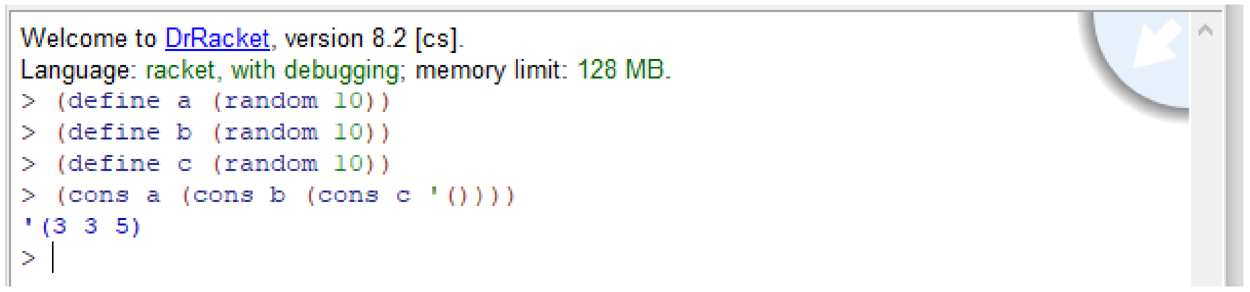
Interactions – referencing a list element using list-ref



```
> (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
> 
```

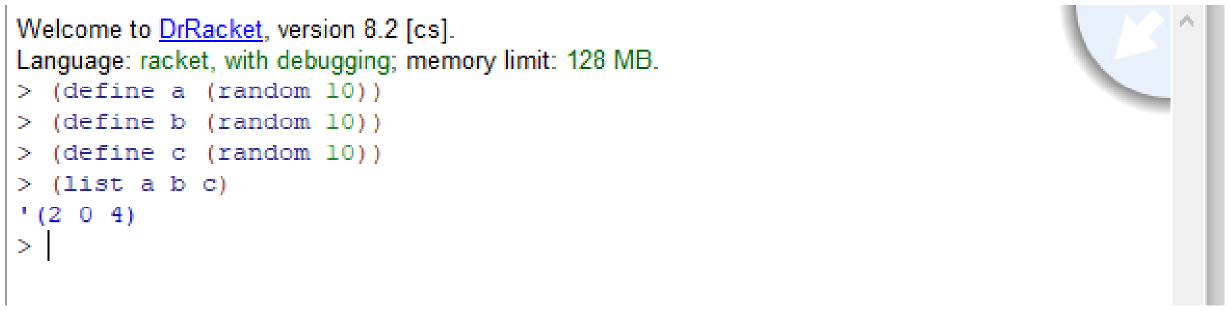
## Creating a list

Interactions – creating a list from scratch



```
Welcome to DrRacket, version 8.2 [cs].
Language: racket, with debugging; memory limit: 128 MB.
> (define a (random 10))
> (define b (random 10))
> (define c (random 10))
> (cons a (cons b (cons c '())))
'(3 3 5)
> 
```

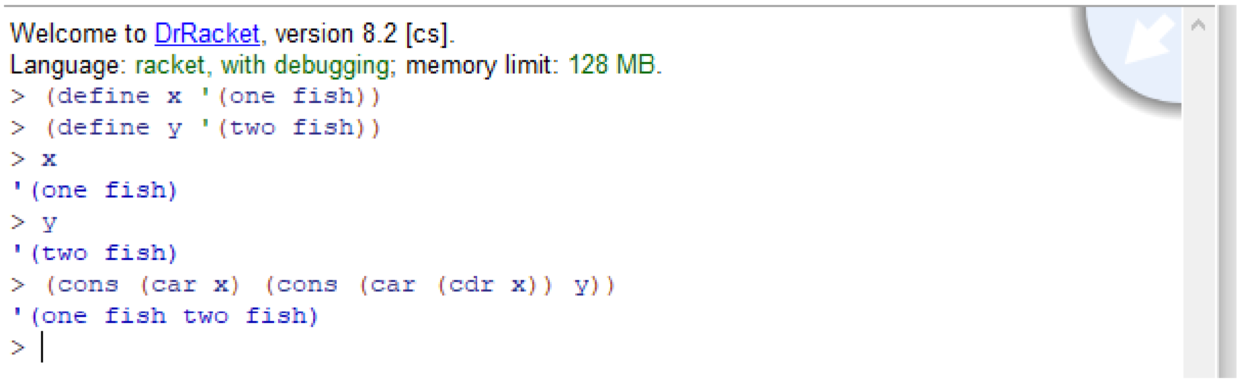
Interactions – creating a list using list



```
Welcome to DrRacket, version 8.2 [cs].
Language: racket, with debugging; memory limit: 128 MB.
> (define a (random 10))
> (define b (random 10))
> (define c (random 10))
> (list a b c)
'(2 0 4)
> 
```

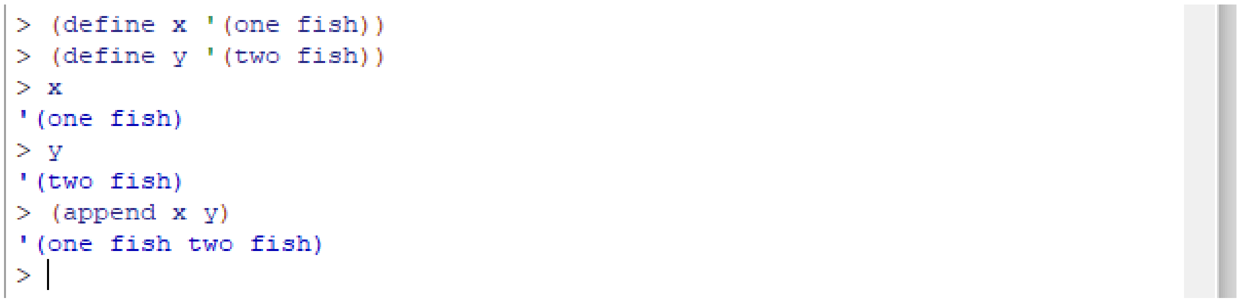
## Appending one list to another list

Interactions – appending two lists from scratch



```
Welcome to DrRacket, version 8.2 [cs].
Language: racket, with debugging; memory limit: 128 MB.
> (define x '(one fish))
> (define y '(two fish))
> x
'(one fish)
> y
'(two fish)
> (cons (car x) (cons (car (cdr x)) y))
'(one fish two fish)
> 
```

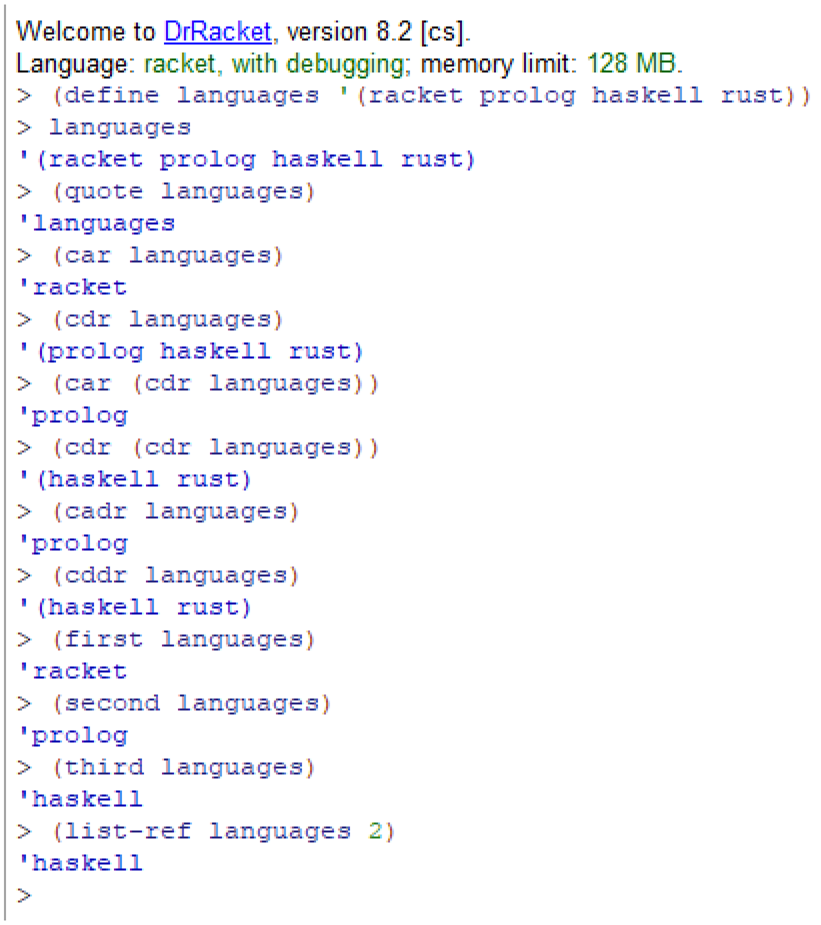
Interactions – appending two list using append



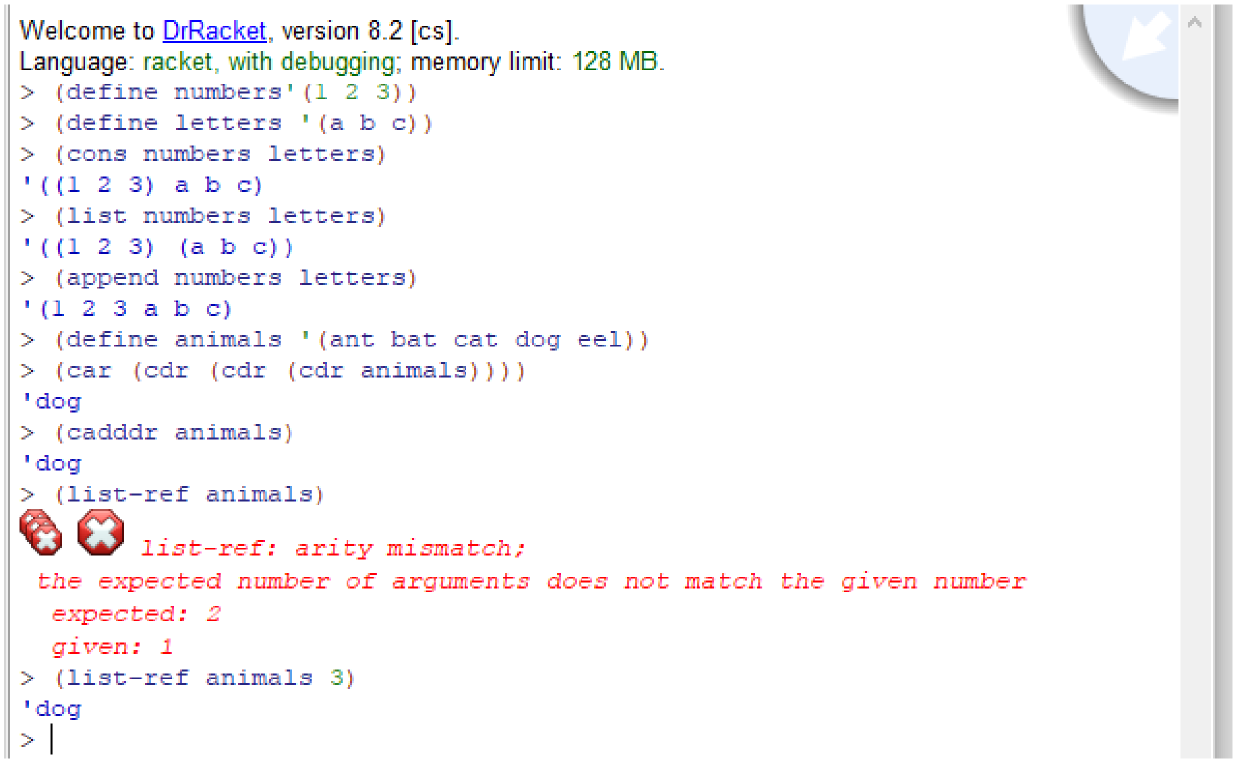
```
> (define x '(one fish))
> (define y '(two fish))
> x
'(one fish)
> y
'(two fish)
> (append x y)
'(one fish two fish)
> 
```

## Redacted Racket Session Featuring Referencers and Constructors

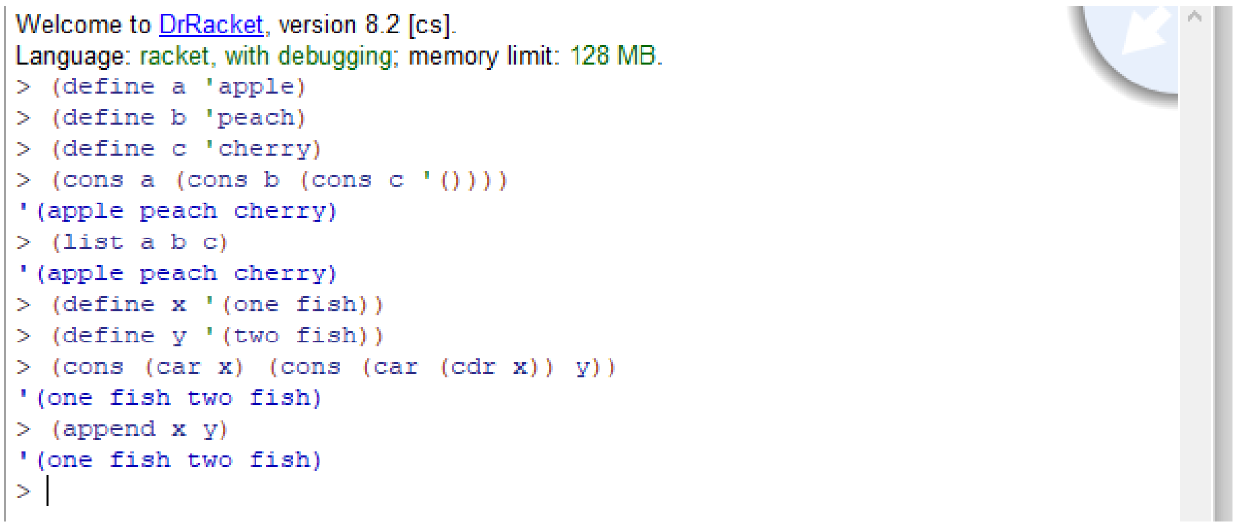
Interactions – mindfully doing the redacted session, for real



```
Welcome to DrRacket, version 8.2 [cs].
Language: racket, with debugging; memory limit: 128 MB.
> (define languages '(racket prolog haskell rust))
> languages
'(racket prolog haskell rust)
> (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
>
```



```
Welcome to DrRacket, version 8.2 [cs].
Language: racket, with debugging; memory limit: 128 MB.
> (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 dog eel))
> (car (cdr (cdr (cdr animals))))
'dog
> (cadddr animals)
'dog
> (list-ref animals)
list-ref: arity mismatch;
 the expected number of arguments does not match the given number
  expected: 2
  given: 1
> (list-ref animals 3)
'dog
>
```

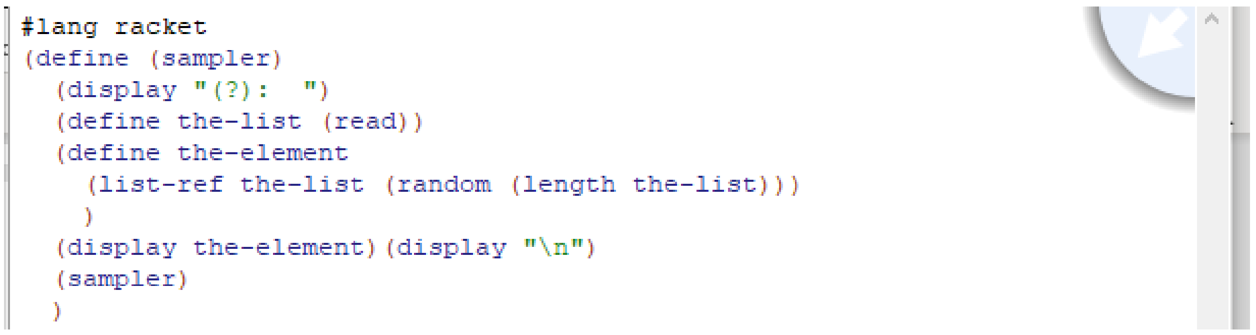


```
Welcome to DrRacket, version 8.2 [cs].
Language: racket, with debugging; memory limit: 128 MB.
> (define a 'apple)
> (define b 'peach)
> (define c 'cherry)
> (cons a (cons b (cons c '())))
'(apple peach cherry)
> (list a b c)
'(apple peach cherry)
> (define x '(one fish))
> (define y '(two fish))
> (cons (car x) (cons (car (cdr x)) y))
'(one fish two fish)
> (append x y)
'(one fish two fish)
> 
```

## Task 3 – Random Selection

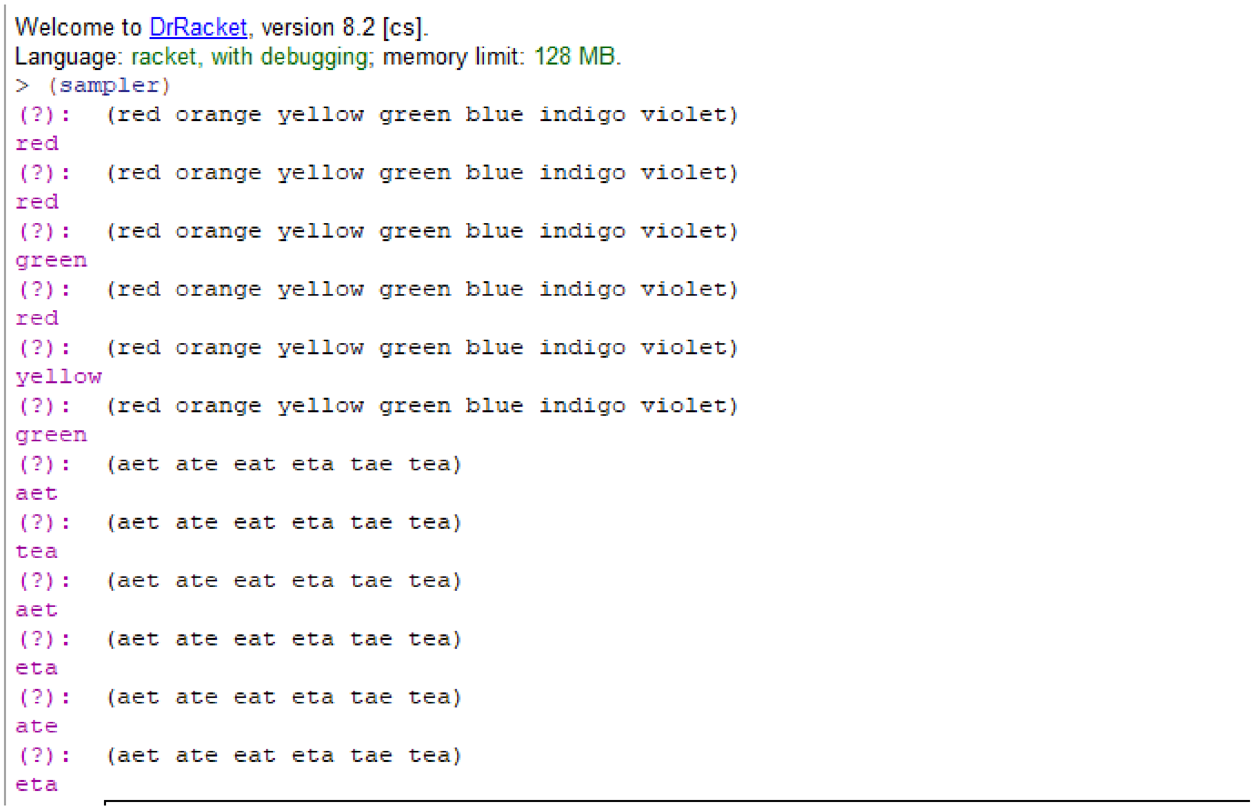
The simple little program presented selects an element at random from a given list. The list is provided by means of the read function, which will read any S-expression, including a list

Definitions – defining the sampler program

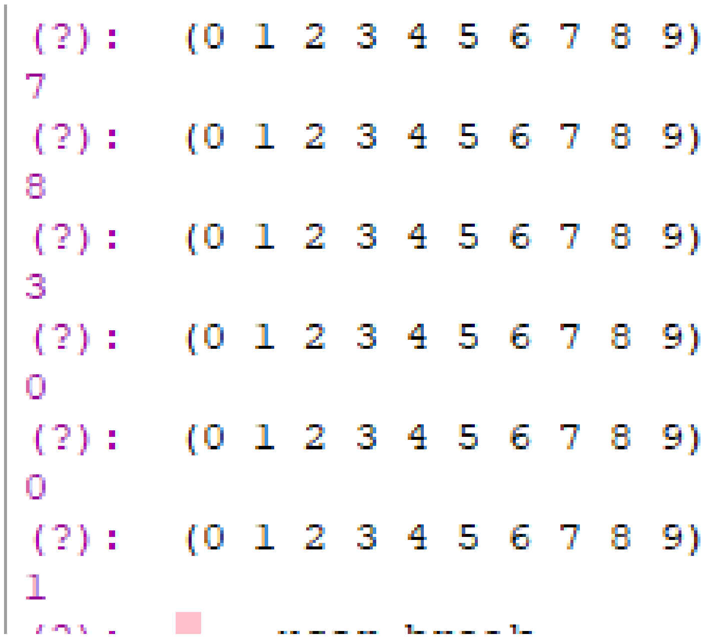


```
#lang racket
(define (sampler)
  (display "(?):  ")
  (define the-list (read))
  (define the-element
    (list-ref the-list (random (length the-list)))
    )
  (display the-element)(display "\n")
  (sampler)
  )
```

Interactions – mimicking the sampler program demo



```
Welcome to DrRacket, version 8.2 [cs].
Language: racket, with debugging; memory limit: 128 MB.
> (sampler)
(?):  (red orange yellow green blue indigo violet)
red
(?):  (red orange yellow green blue indigo violet)
red
(?):  (red orange yellow green blue indigo violet)
green
(?):  (red orange yellow green blue indigo violet)
red
(?):  (red orange yellow green blue indigo violet)
yellow
(?):  (red orange yellow green blue indigo violet)
green
(?):  (aet ate eat eta tae tea)
aet
(?):  (aet ate eat eta tae tea)
tea
(?):  (aet ate eat eta tae tea)
aet
(?):  (aet ate eat eta tae tea)
eta
(?):  (aet ate eat eta tae tea)
ate
(?):  (aet ate eat eta tae tea)
eta
```

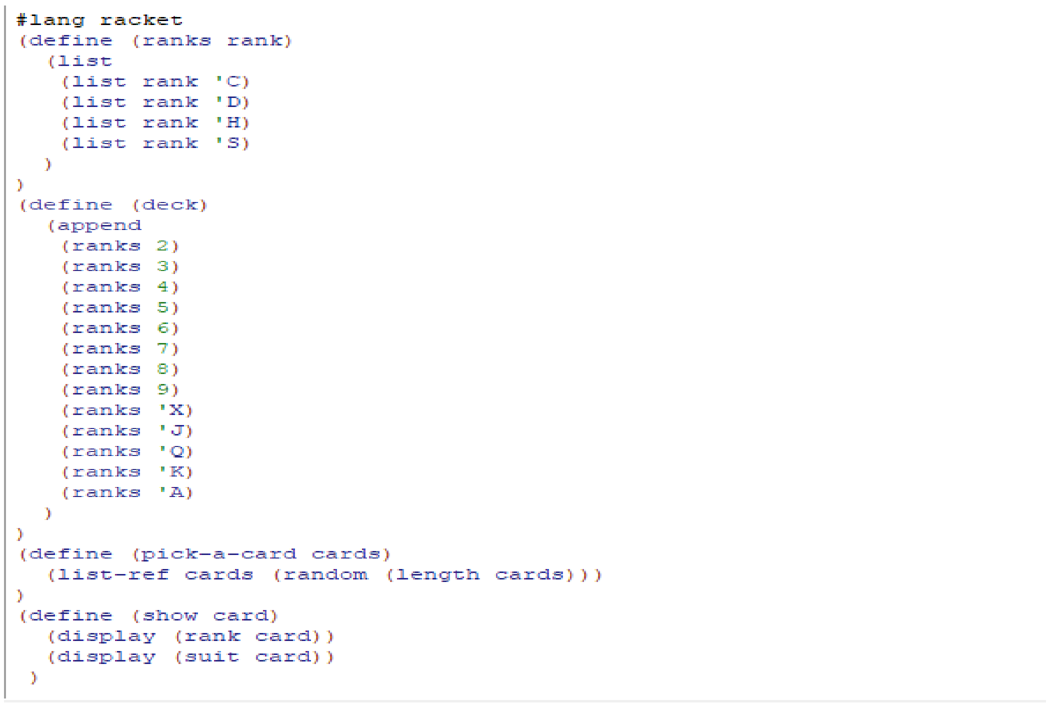


```
(?):  (0 1 2 3 4 5 6 7 8 9)
7
(?):  (0 1 2 3 4 5 6 7 8 9)
8
(?):  (0 1 2 3 4 5 6 7 8 9)
3
(?):  (0 1 2 3 4 5 6 7 8 9)
0
(?):  (0 1 2 3 4 5 6 7 8 9)
0
(?):  (0 1 2 3 4 5 6 7 8 9)
1
```

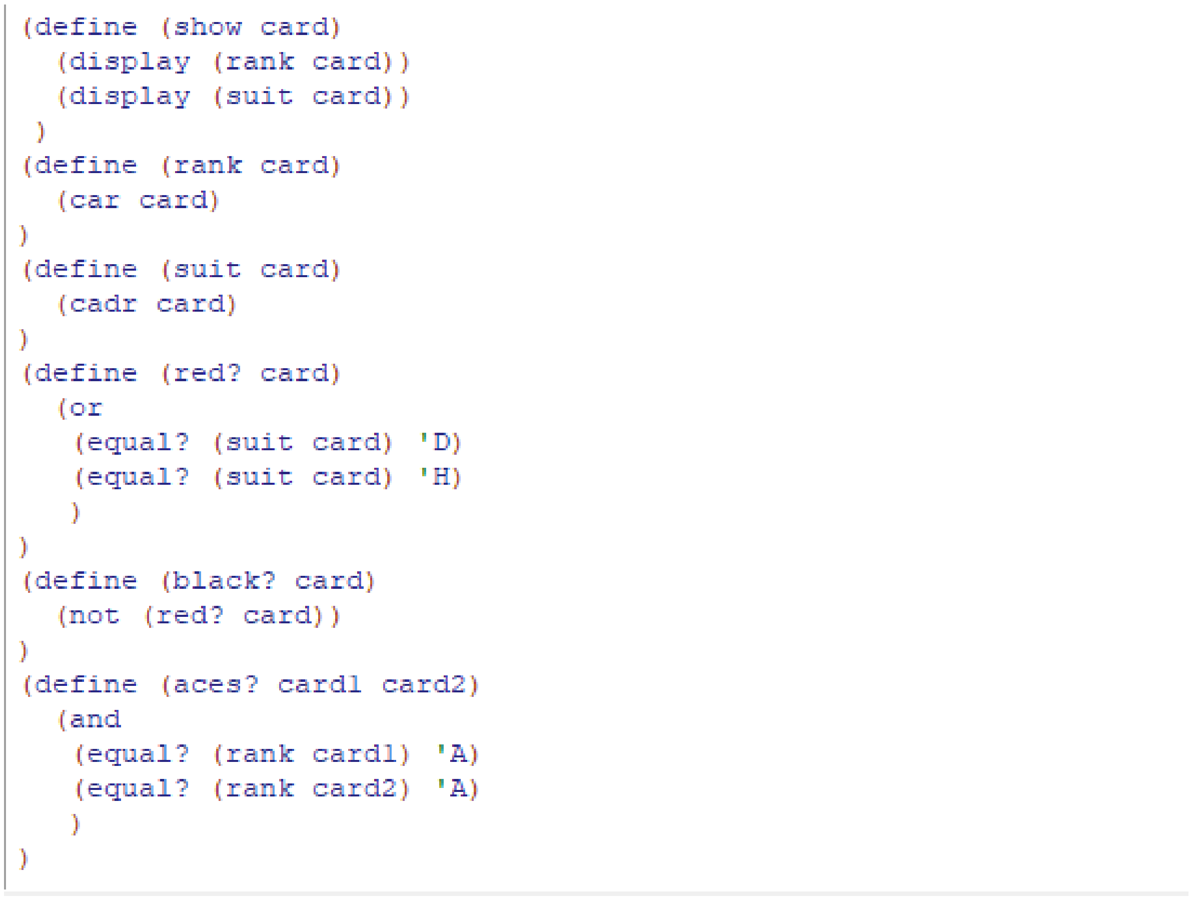
## Task 4 – Playing Card Programming Challenge

The code and demo for the playing card programming challenge presented at the end of lesson 8 is presented here

Definitions – programming the card playing functionality

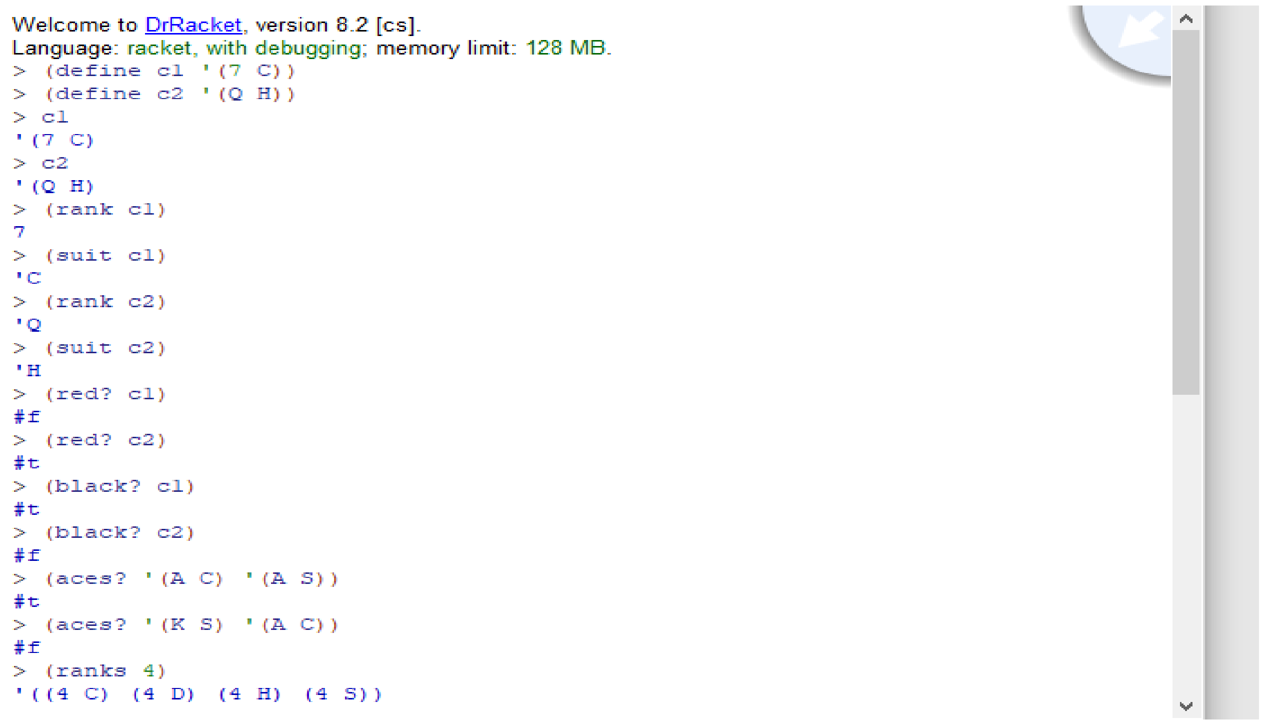


```
#lang racket
(define (ranks rank)
  (list
   (list rank 'C)
   (list rank 'D)
   (list rank 'H)
   (list rank 'S)
  )
)
(define (deck)
  (append
   (ranks 2)
   (ranks 3)
   (ranks 4)
   (ranks 5)
   (ranks 6)
   (ranks 7)
   (ranks 8)
   (ranks 9)
   (ranks 'X)
   (ranks 'J)
   (ranks 'Q)
   (ranks 'K)
   (ranks 'A)
  )
)
(define (pick-a-card cards)
  (list-ref cards (random (length cards)))
)
(define (show card)
  (display (rank card))
  (display (suit card))
 )
```

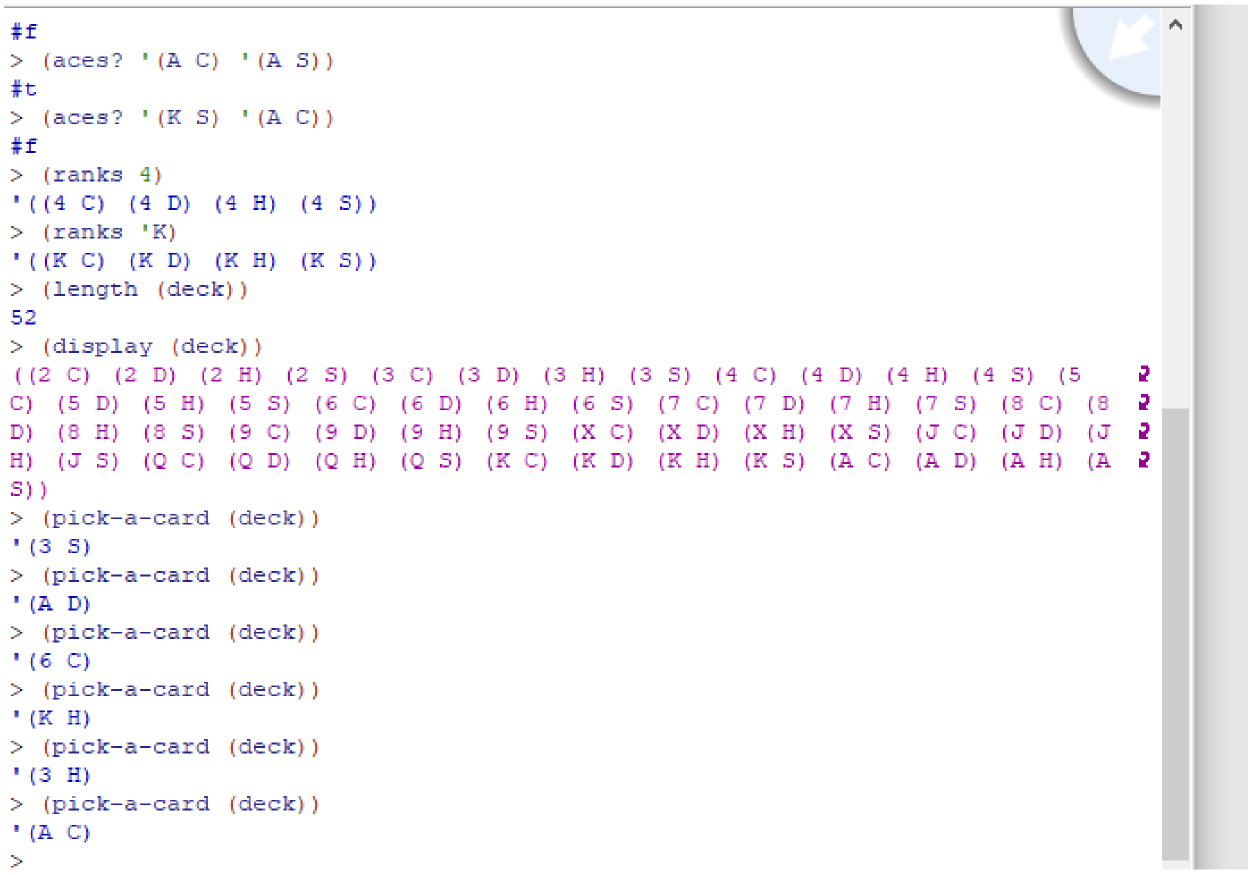


```
(define (show card)
  (display (rank card))
  (display (suit card))
 )
(define (rank card)
  (car card)
)
(define (suit card)
  (cadr card)
)
(define (red? card)
  (or
   (equal? (suit card) 'D)
   (equal? (suit card) 'H)
   )
)
(define (black? card)
  (not (red? card))
)
(define (aces? card1 card2)
  (and
   (equal? (rank card1) 'A)
   (equal? (rank card2) 'A)
   )
)
```

Interactions – mimicking the card functionality demo



```
Welcome to DrRacket, version 8.2 [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))
```



```
#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))
'(3 S)
> (pick-a-card (deck))
'(A D)
> (pick-a-card (deck))
'(6 C)
> (pick-a-card (deck))
'(K H)
> (pick-a-card (deck))
'(3 H)
> (pick-a-card (deck))
'(A C)
>
```