CS135 Tutorial 03

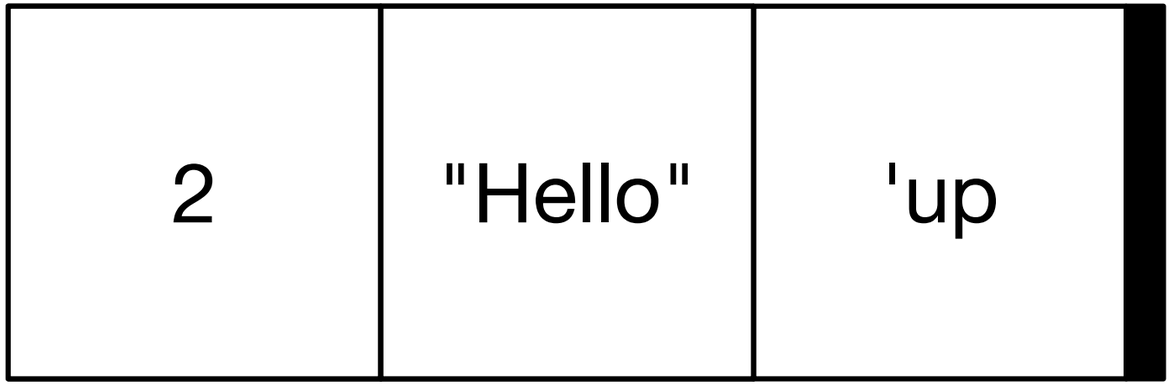
Review of list functions

List Values	List Functions
• `empty`: an empty list • `(cons v lst)`: where `v` is a value and `lst` is a list (which includes `empty`)	• `(cons v lst)`: Consumes a value and a list; produces a new, longer list. • `(first (cons a b)) => a` • `(rest (cons a b)) => b` • `(empty? empty) => true` • `(empty? a) => false` where `a` is any value other than `empty` • `(cons? (cons a b)) => true` • `(cons? a) => false` where a is any Racket value not created using `cons`

`a` and `b` are values. `a` can be any value; `b` is a list value.

Creating List Values

- Write Racket code for this list:



Contracts

What would valid contracts be for the following?

```
(define (foo a b c) (max a b (first c))

(define (bar a b)
   (cond [(string<? a b) (cons a (cons b empty))]
         [else (cons b (cons a empty))]))

(define (qux a)
   (cond [(empty? a) 0]
         [(empty? (rest a)) 1]
         [else 2]))
```

List-of-three?

Write a function, `list-of-three?`, which consumes a value and produces `true` if it is a list with exactly three elements and `false` otherwise. Just for fun, do it without using `length`.

Recall the design recipe steps:

1. Draft a purpose statement
2. Construct examples
3. Write the function definition header and contract
4. Finalize the purpose with parameter names
5. Write the definition body
6. Write additional tests, if needed

Three-of-a-kind?

Evaluate the following Racket code with a person near you. List as many improvements as you can.

```
;; three-of-a-kind produces true if the list has exactly three symbols
;; and they are all the same.

(check-expect (three-of-a-kind? (cons 'a (cons 'a (cons 'a empty)))) true)

;; (ListOf Symbol) -> Bool
(define (three-of-a-kind? x)
  (and (symbol=? (first x) (first (rest x)))
       (symbol=? (first x) (first (rest (rest x))))))
```