Computer Science I

CMPE/CSCI 1370 - 01

http://bit.ly/1370abcde

Design recipe

1. Data definitions
2. Signature, purpose, stub
3. Examples
4. Template
5. Function definition
6. Test and debug

Which of the following are appropriate examples?

```
;; String -> String
;; Produce the given string with "s" added to the end.

(define (pluralize str) "")  ;stub
```

A. `(check-expect (pluralize "cat") "s")`

B. `(check-expect (pluralize "cat") "cat")`

C. `(check-expect (pluralize "dog") "dogs")`

D. `(check-expect (pluralize "grass") "grassss")`

E. More than one of the above

Which part of the partially-completed design is inconsistent from the rest?

```
;; String -> Boolean                                           [A]
;; produce true if string length is 0                          [B]
(check-expect (empty-string? "") true)                        ;[C]
(check-expect (empty-string? 0) false)
(check-expect (empty-string? "abc") false)

;(define (empty-string? s) true) ;stub                         [D]

(define (empty-string? s)                                     ;[E]
  (zero? (string-length s)))
```

How many `check-expects` do we need?

`tall?`

Step through `letter-grade`

Boolean operators

a	b	(and a b)	(or a b)	(not a)
#true	#true			
#true	#false			
#false	#true			
#false	#false			

```
(and
  (> 7 4)
  (or
    (not (> 7 8))
    (= 7 5)))
```

A. `#true`

B. `#false`

C. Error

D. It depends

E. I don't know

Why data definitions?

`next-color`

From: UBCx: HtC1x

```
(define (next-color c)
  (cond [(= c 0) 2]
        [(= c 1) 0]
        [(= c 2) 1]))
```

```racket
;; Natural -> Natural
;; produce next color of traffic light
(check-expect (next-color 0) 2)
(check-expect (next-color 1) 0)
(check-expect (next-color 2) 1)

;(define (next-color c) 0)   ;stub

;(define (next-color c)      ;template
;  (... c))

(define (next-color c)
  (cond [(= c 0) 2]
        [(= c 1) 0]
        [(= c 2) 1]))
```

```
;; Data definitions:

;; TLColor is one of:
;;  - "red"
;;  - "yellow"
;;  - "green"
;; interp. "red" means red, "yellow" yellow, "green" green

(define (fn-for-tlcolor c)
  (cond [(string=? c "red") (...)]
        [(string=? c "yellow") (...)]
        [(string=? c "green") (...)]))
```

How to design data

1. Structure definition

2. Type comment

3. Interpretation

4. Examples

5. Template

From: UBCx: HtC1x

```
;; Functions

;; TLColor -> TLColor
;; produce next color of traffic light
(check-expect (next-color "red") "green")
(check-expect (next-color "yellow") "red")
(check-expect (next-color "green") "yellow")

;(define (next-color c) "red")  ;stub

; Template from TLColor

(define (next-color c)
  (cond [(string=? c "red")     "green"]
        [(string=? c "yellow") "red"]
        [(string=? c "green")  "yellow"]))
```

Data definition: Atomic non-distinct

`FirstName`

Attendance!

http://bit.ly/1370-1rollcall