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

Review

`how-many-pizzas`

Problem: Design a function that pluralizes a given word. For simplicity you may assume that just adding s is enough to pluralize a word.

Which of the following purpose statements is best?

A.
```
; Pluralize s.
```

B.
```
; Produce plural string.
```

C.
```
; Add "s".
```

D.
```
; Produce the given string with "s"
added to the end.
```

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") "grasss")`

E. More than one of the above

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

```
;; Image -> String                                          [A]
;; produce the aspect ratio (width/height) of an image [B]
(check-expect
  (aspect-ratio (rectangle 20 30 "solid" "blue"))
  (/ 2 3))                                                ;[C]
(check-expect
  (aspect-ratio (square 10 "solid" "blue"))
  1)
(check-expect
  (aspect-ratio (rectangle 30 20 "solid" "blue"))
  3/2)

; define (aspect-ratio img) 0) ;stub                      [D]
```

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)))
```

Review

`first-world-problems`

How many `check-expects` do we need?

`tall?`

`cond` expressions

`aspect-ratio`

Step through the following function call by hand:

```
(define (absval n)
  (cond
    [(> n 0) n]
    [(< n 0) (* -1 n)]
    [else 0]))

(absval -3)
```

Fill in the blanks!

```
(define (mag x)
  (cond
    [______ ______]
    [______ ______]
    [______ ______]))
```

`else`

`"positive"`

`(< x 0)`

`(> x 0)`

`"zero"`

`"negative"`

Attendance!

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