Questions 1-3

Bookmark this page

Sometimes a program we design doesn't work properly. The program might get an error or a test might fail. One useful skill in trying to find the problem is to scan over the design elements looking for inconsistencies. Does the purpose match the signature? Do the tests match the signature and purpose? Does the stub match what comes before it? Does the function header match the signature and purpose? Does the function body match the signature, purpose and tests?

Often times looking for an inconsistency between different parts of these is enough to find the bug and make it clear what needs to be fixed.

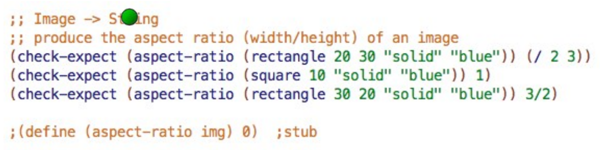
The questions on this page are intended to help you begin to learn this skill.

Question 1

1 point possible (graded)

A "pointing on a picture" problem (also called an "image mapped input" problem) presents an image, and asks you to click on an area in the image. To help ensure accuracy, make sure you click near the middle of what the question asks for.

In the following partially complete function design, one part of the design is inconsistent with the rest. Click on the most specific or smallest part that "doesn't belong".



```
;; Image -> String
;; produce the aspect ratio (width/height) of an image
(check-expect (aspect-ratio (rectangle 20 30 "solid" "blue")) (/ 2 3))
(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
```

Explanation
The signature shows the function as producing String. But this doesn't match the purpose, the check-expects or the stub, all of which have the function producing Number.

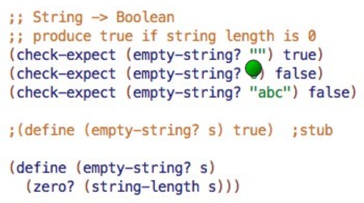
Submit Show Answer

Answers are displayed within the problem

Question 2

1 point possible (graded)

In the following partially complete function design, one part of the design is inconsistent with the rest. Click on the most specific or smallest part that "doesn't belong".



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

;(define (empty-string? s) true)  ;stub

(define (empty-string? s)
  (zero? (string-length s)))
```

Explanation
In the second check-expect, the operand in the call to empty-string? is 0. But 0 is a number, the signature says the function must consume String, and all the other parts of the design have the function consuming String.

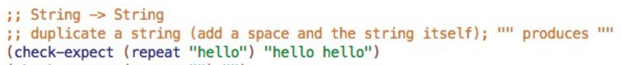
Submit Show Answer

Answers are displayed within the problem

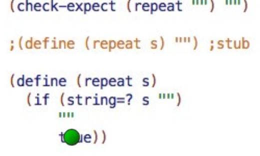
Question 3

1 point possible (graded)

In the following partially complete function design, one part of the design is inconsistent with the rest. Click on the most specific or smallest part that "doesn't belong".



```
;; String -> String
;; duplicate a string (add a space and the string itself); "" produces ""
(check-expect (repeat "hello") "hello hello")
```



```
(check-expect (repeat "") "")

;(define (repeat s) "") ;stub

(define (repeat s)
  (if (string=? s "")
      ""
      true))
```

Explanation

The false answer in the if expression produces true. But true is a boolean, which does not match the signature, purpose or any of the check-expects.

Submit

Show Answer

Answers are displayed within the problem

Previous Next