Quiz

For this multiple choice design quiz, download the starter and complete the problems.

Once you have finished, answer the multiple choice questions about your design.

Unlike the lecture questions, you will only have one attempt to answer each question in the quiz, so make sure to read each answer carefully before selecting one and pressing submit.

Question 1

1/1 point (graded)

The first three questions deal with problem 1 in the starter file.

What is the correct Signature for `arrange-all` function you designed?

- `(Image Image -> Image) Image (listof Image) -> Image`
- `(X X -> X) X (listof X) -> X`
- `(X Y -> Y) Y (listof X) -> Y` ✔
- `(X Y -> X) X (listof X) -> X`

Explanation
The `combine-all` function has a combining function and a base, and produces the same type as the base, which can be different than the original type in the list.

Submit You have used 1 of 1 attempt

Show Answer

Answers are displayed within the problem

Question 2

1/1 point (graded)

Which of the following are valid check-expects for `arrange-all`?

- ☑
```
(check-expect (arrange-all above empty-image (list (square 10 "solid" "pink") (triangle 20 "solid" "red")))
              (above (square 10 "solid" "pink") (triangle 20 "solid" "red")))
```
- ☑
```
(check-expect (arrange-all cons empty (list "a" "b" "c"))
              (cons "a" (cons "b" (cons "c" empty))))
```
- ☐
```
(check-expect (arrange-all string-append 0 empty) empty)
```
- ☑
```
(check-expect (arrange-all + 0 (list 1 2 3 4 5)) 15)
```

Explanation
(check-expect (arrange-all string-append 0 empty) empty) is not valid, because `string-append` produces a String, but the base 0, is a Number.

Submit You have used 1 of 2 attempts

Save Show Answer

Answers are displayed within the problem

Question 3

1/1 point (graded)

Which built in abstract function is equivalent to `arrange-all`?

- map
- **foldr**
- filter
- andmap

✔

Explanation

`combine-all` and `fold` have the same siganture, and do the same thing.

Submit You have used 1 of 1 attempt

Show Answer

Answers are displayed within the problem

Question 4-8

5/5 points (graded)

The next five questions have to do with Problem 2 in the starter file.

Which of the functions you designed call `map`? Select all that apply.

- [x] `lengths`
- [ ] `odd-only`
- [ ] `all-odd?`
- [x] `minus-n`
- [ ] none of them

✔

Which of the functions you designed call `ormap`? Select all that apply.

- [ ] `lengths`
- [ ] `odd-only`
- [ ] `all-odd?`
- [ ] `minus-n`
- [x] none of them

✔

Which of the functions you designed call `andmap`? Select all that apply.

- [ ] `lengths`
- [ ] `odd-only`
- [x] `all-odd?`
- [ ] `minus-n`
- [ ] none of them

✔

Which of the functions you designed call `filter`? Select all that apply.

- [ ] `lengths`
- [x] `odd-only`

- [ ] `all-odd?`
- [ ] `minus-n`
- [ ] none of them

✔

Which of the functions you designed require(s) a closure? Select all that apply.

- [ ] `lengths`
- [ ] `odd-only`
- [ ] `all-odd?`
- [x] `minus-n`
- [ ] none of them

✔

Explanation
Here are the completed function designs:

```
;; (listof String) -> (listof Natural)
;; produces a list of the lengths of each string in los
(define (lenths lon)
  (map string-length lon))

;; (listof Natural) -> (listof Natural)
;; produces a list of just the odd elements of lon
(define (odd-only lon)
  (filter odd? lon))

;; (listof Natural -> Boolean
;; produce true if all elements of the list are odd
(define (all-odd? lon)
  (andmap odd? lon))

;; (listof Natural) -> (listof Natural)
;; subtracts n from each element of the list
(define (minus-n lon n)
  (local [(define (-n i) (- i n))]
    (map -n lon)))
```

Submit You have used 1 of 1 attempt

Show Answer

Answers are displayed within the problem

Question 9

1/1 point (graded)

The remaining questions deal with Problem 3 in the starter file

How many arguments does your fold function consume?

9 ✔ **Answer:** 9

9

Explanation
There should be 9 arguments. If you answered this question incorrectly, please take another look at your solution before attempting the remaining questions.

Submit You have used 1 of 1 attempt

Show Answer

Answers are displayed within the problem

Question 10

1/1 point (graded)

Which of the following is a correct signature for `fold-region`? (Be sure you followed the guidelines so that your arguments are in the same order as ours)

- ○ `;; (String X Z -> Y) (Y Z -> Z) X X X X X Z Region -> Z`
- ◉ `;; (String X Z -> Y) (Y Z -> Z) X X X X X Z Region -> Y`
- ○ `;; (String X Z -> Y) (Y Z -> Z) Z Region -> Y`

- ;; (String X Z -> Y) (Y Z -> Z) X X X X X Z -> Y

✔

Explanation
Here is our design for `fold-region`:

```
(define (fold-region c1 c2 b1 b2 b3 b4 b5 b6 r)
  (local [(define (fn-for-region r)
            (c1 (region-name r)
                (fn-for-type (region-type r))
                (fn-for-lor (region-subregions r))))

          (define (fn-for-type t)
            (cond [(string=? t "Continent") b1]
                  [(string=? t "Country") b2]
                  [(string=? t "Province") b3]
                  [(string=? t "State") b4]
                  [(string=? t "City") b5]))

          (define (fn-for-lor lor)
            (cond [(empty? lor) b6]
                  [else
                   (c2 (fn-for-region (first lor))
                       (fn-for-lor (rest lor)))]))]
    (fn-for-region r)))
```

Submit You have used 1 of 1 attempt

Show Answer

Answers are displayed within the problem

Question 11

1/1 point (graded)

What should the result of this check-expect be?

```
(check-expect (fold-region make-region cons "Continent" "Country" "Province" "State" "City" empty CANADA) ____)
```

- this violates the signature of `fold-region`
- `(list "Canada" "British Columbia" "Vancouver" "Victoria" "Alberta" "Calgary" "Edmonton")`
- `(list CANADA BC VANCOUVER VICTORIA ALBERTA CALGARY EDMONTON)`
- `CANADA`

✔

Explanation
This check-expect should produce the region CANADA

Submit You have used 1 of 1 attempt

Show Answer

Answers are displayed within the problem

Question 12

1/1 point (graded)

Which of the following is a correct function definition for `all-regions`?

```
(define (all-regions r)
  (fold-region region-name cons "" "" "" "" "" empty r)))
```

```
(define (all-regions r)
      (fold-region region-name append "" "" "" "" "" empty r)))
```

```
(define (all-regions r)
  (local [(define (c1 n t r) (cons n r))]
      (fold-region c1 cons "" "" "" "" "" empty r)))
```

```
(define (all-regions r)
  (local [(define (c1 n t r) (cons n r))]
      (fold-region c1 append "" "" "" "" "" empty r)))
```

✔

Explanation
b1 through b5 are never used in this function, so they can be any value. This function produces a list, so b6 must be empty. c1 must consume three arguments and produce the first, the regions name, so it must be locally defined, or defined as a helper function. c2 combines two lists, so it must be append.

Submit You have used 1 of 1 attempt

Show Answer

Answers are displayed within the problem

Previous

Next