Multiple Choice Quiz

Below is a quiz on the material you've learned this week. Unlike the lecture questions, you will only have one attempt at each question, so think carefully before submitting your answers.

Question 1

1/1 point (graded)

```
(define L1 (cons "Apple" (cons "Banana" (cons "Orange" (cons "Pear" empty)))))
```

Which of the following expessions will produce the value "Orange"?

- (first (first (rest L1)))
- (rest (rest (first L1)))
- (rest (rest L1))
- (first (rest (rest L1))) ✔

Submit You have used 1 of 1 attempt

✔ Correct (1/1 point)

Question 2

1/1 point (graded)

Which of the following self-referential type comments are well formed (choose all that apply)?

- [x]
```
;; ListOne is one of:
;; - empty
;; - (cons Natural ListOne)
```

- [ ]
```
;; ListTwo is one of:
;; - (cons String ListTwo)
;; - (cons Natural ListTwo)
```

- [x]
```
;; ListThree is one of:
;; - (cons "end" empty)
;; - (cons String ListThree)
```

- [ ]
```
;; ListFour is one of:
;; - (cons Natural ListFour)
```

- [x]
```
;; ListFive is one of:
;; - empty
;; - (cons 0 ListFive)
;; - (cons 1 ListFive)
```

✔

Submit You have used 1 of 1 attempt

✔ Correct (1/1 point)

Questions 3-5

3/3 points (graded)

Consider the following partially obscured types comments:

```
;; ListA  is one of:
;; - empty
;; - (cons "yes" ListA)
;; - (cons "no" ListA)
;; interp.

;; ListB  is one of:
;; - empty
;; (cons String ListB)
;; interp.

;; ListC  is one of:
;; - (cons String empty)
;; - (cons String ListC)
;; interp.
```

The templates for these three types comments have also been partially obscured.

Template 1:

```
(define (fn-for-list l)
  (cond [(empty? (rest l))(...)]
        [else
         (... (first l)
              (fn-for-list (rest l)))]))
```

Template 2:

```
(define (fn-for-list l)
  (cond [(empty? l)(...)]
        [(                    )(... (first l)
                                    (fn-for-list (rest l)))]
        [else
         (... (first l)
              (fn-for-list (rest l)))]))
```

Template 3:

```
(define (fn-for-list l)
  (cond [(empty? l)(...)]
        [else
         (... (first l)
              (fn-for-list (rest l)))]))
```

Which template belongs with ListA?

- Template 1
- Template 2
- Template 3

✔

Which template belongs with ListB?

- Template 1
- Template 2
- Template 3

✔

Which template belongs with ListC?

- Template 1
- Template 2
- Template 3

✔

Submit You have used 1 of 1 attempt

✔ Correct (3/3 points)

Questions 6-8

3/3 points (graded)

For each list, choose which of the three data definition defined above it is a valid example for (choose all that apply).

```
(define L0 empty)
```

- [x] ListA
- [x] ListB
- [ ] ListC

✔

```
(define L1 (cons "yes" (cons "no" (cons "yes" empty))))
```

- [x] ListA
- [x] ListB
- [x] ListC

✔

```
(define L2 (cons "yes" (cons "no" (cons "maybe" empty))))
```

- [ ] ListA
- [x] ListB
- [x] ListC

✔

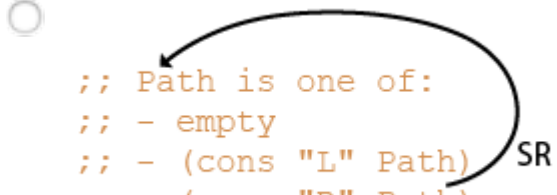
Submit You have used 1 of 1 attempt

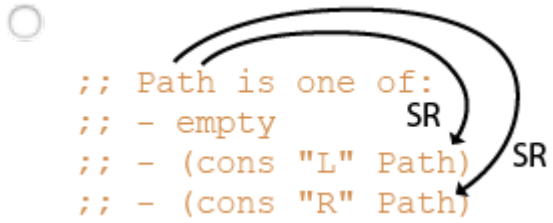
✔ Correct (3/3 points)

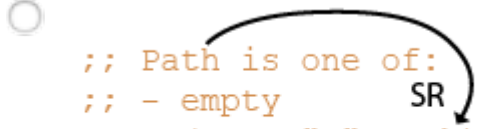
Question 9

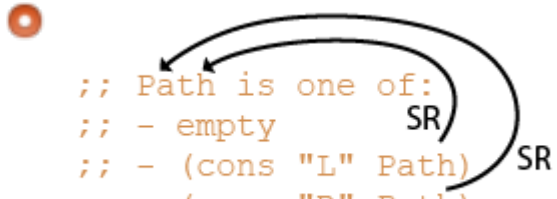
1/1 point (graded)

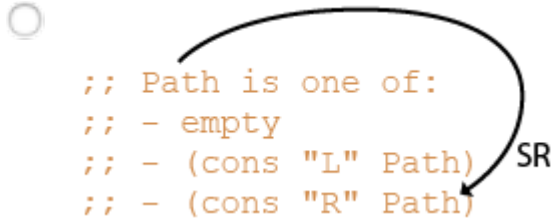
Which of the following shows the correct way to draw reference arrows for the type `Path`?

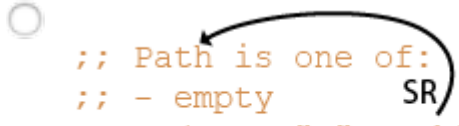

;; Path is one of:
;; - empty
;; - (cons "L" Path)
;; - (cons "R" Path)
SR


;; Path is one of:
;; - empty
;; - (cons "L" Path)
;; - (cons "R" Path)
SR
SR


;; Path is one of:
;; - empty
;; - (cons "L" Path)
;; - (cons "R" Path)
SR


;; Path is one of:
;; - empty
;; - (cons "L" Path)
;; - (cons "R" Path)
SR
SR


;; Path is one of:
;; - empty
;; - (cons "L" Path)
;; - (cons "R" Path)
SR


;; Path is one of:
;; - empty
;; - (cons "L" Path)
;; - (cons "R" Path)
SR

Submit You have used 1 of 1 attempt

✔ Correct (1/1 point)

Question 10

1/1 point (graded)

Consider the following Data Definitions:

```
(define-struct concert (artist venue))
;; Concert is (make-concert String String)
;; interp. a concert with the band playing, and the venue they're playing at
(define C1 (make-concert "Shakey Graves" "Commodore Ballroom"))
(define C2 (make-concert "Tallest Man On Earth" "Orpheum Theatre"))
#;
(define (fn-for-concert c)
  (... (concert-artist c)
       (fn-for-venue (concert-venue c))))

;; ListOfConcert is one of:
;; - empty
;; - (cons Concert ListOfConcert)
;; interp. a list of concerts
(define LOC1 empty)
(define LOC2 (cons C1 (cons C2 empty)))
#;
(define (fn-for-loc loc)
  (cond [(empty? loc)(...)]
        [else
         (... (fn-for-concert (first loc))
              (fn-for-loc (rest loc)))]))

(define-struct festival (name shows))
;; Festival is (make-festival String ListOfConcert)
;; interp. a festival with name, and list of shows that are part of the festival
(define CANCELLED-FESTIVAL (make-festival "" empty))
(define VFMF (make-festival "Vancouver Folk Music Festival"
                            (cons (make-concert "Hawksley Workman" "Main Stage")
                                  (cons (make-concert "Grace Petrie" "Stage 1")
                                        (cons (make-concert "Mary Gauthier"
"Stage 5") empty)))))
#;
(define (fn-for-festival f)
  (... (festival-name f))
       (cond [(empty? (festival-shows f))(...)]
             [else
              (... (fn-for-concert (first (festival-shows f)))
                   (fn-for-festival (rest (festival-shows f))))]))
```

Which of the above templates are completely correct (select all that apply)?

- [ ] Concert
- [x] ListOfConcert
- [ ] Festival
- [ ] none are correct

✔

Submit You have used 1 of 2 attempts

✔ Correct (1/1 point)

Question 11

1/1 point (graded)

Use the options below to create the correct template rules for ListOfConcert.

You can use up to six template rules, but you do not need to use all the space.

Note, if you drap the options out of the bar at the bottom, they become bigger and you will be able to read the template rules.

```
;; Template Rules Used
;; - One of: 2 cases
;; - atomic distinct: empty
;; - compound: (cons Concert ListOfConcert)
;; - reference: (first loc) is Concert
;; - self-reference: (rest loc) is ListOfConcert
```

✔

Submit You have used 1 of 1 attempt

✔ Correct (1/1 point)

Question 12

1/1 point (graded)

We want to design a function called festival-schedule that produces the lineup of a festival as a list of the bands and where they're performing as a list of strings.

So `(festival-schedule VFMF)` should produce

```
(cons "Hawksley Workman: Main Stage" (cons "Grace Petrie: Stage 1" (cons
"Mary Gauthier: Stage 1" empty)))
```

```
;; Festival -> ListOfString
;; produces a list of each band paired with where they are performing
(check-expect (festival-schedule (make-festival "" empty)) empty)
(check-expect (festival-schedule (make-festival "CFMF"
                                                (cons (make-concert "Father John
Misty" "Main Stage") empty)))
              (cons "Father John Misty: Main Stage" empty))
(define (festival-schedule f) f)   ; stub
```

Which of the following statements is true?

- The check-expects are correct and there are enough of them
- There are enough check-expects, but they are incorrect or poorly formed
- The individual check-expects are each correct, but we need additional or different one(s)
- There are not enough check-expects and they are incorrect

✔

Submit You have used 1 of 1 attempt

✔ Correct (1/1 point)

Question 13

1/1 point (graded)

Assume you have a function called display-concert with the following signature, purpose and stub.

```
;; Concert -> String
;; produce the name and venue of a concert, separated with a colon, as a string
(define (display-concert c) "")  ;stub
```

You design the function `festival-schedule` and wish for a function called `schedule-loc`. The completed function body for `festival-schedule is`:

```
(define (festival-schedule f)
    (schedule-loc (festival-shows f)))
```

Your wish list entry for `schedule-loc` is:

```
;; ListOfConcert -> ListOfString
;; given a list of concerts, produce a list of each artist paired with where
they are performing.
(define (schedule-loc loc) empty)  ;stub
```

Which of the following definitions is correct for `schedule-loc`?

```
(define (schedule-loc loc)
  (cons (display-concert (first loc))
        (schedule-loc (rest loc))))
```

```
(define (schedule-loc loc)
  (cond [(empty? loc) empty]
        [else
         (cons (string-append (concert-artist (first loc)) ":" (concert-
venue (first loc)))
               (schedule-loc (rest loc)))]))
```

```
(define (schedule-loc loc)
  (cond [(empty? loc) empty]
        [else
         (cons (display-concert (first loc))
               (schedule-loc (rest loc)))]))
```

```
(define (schedule-loc loc)
  (cons (string-append (concert-artist (first loc)) ":" (concert-venue
(first loc)))
        (schedule-loc (rest loc))))
```

✔

Submit You have used 1 of 2 attempts

✔ Correct (1/1 point)

Question 14

0/1 point (graded)

Suppose you want to change the definition of festival to have three fields: the name, the headlining Concert, and all the other shows in the festival.

The new type is:

```
(define-struct festival (name headliner shows))
;; Festival is (make-festival String Concert ListOfConcert)
```

Write the new template for `Festival`.

Which elements are included in the new template (select all that apply)?

- [x] `fn-for-concert`
- [ ] `concert-artist`
- [x] `empty?`
- [x] `festival-headliner`
- [x] `fn-for-loc`
- [ ] `concert-venue`
- [x] `festival-name`
- [x] `first`
- [x] `rest`
- [x] `festival-shows`

✖

Submit You have used 1 of 1 attempt

✖ Incorrect (0/1 point)