Quiz 3

Questions 1 - 3 (6 versions for #1, 4 versions for #2-3)

1.
```
(define (f1 x)
 (cond [(null? x) 0]
       [(number? x) x]
       [(string? x) (string-length x)]
       [(list? x) (+ (f1 (car x)) (f1 (cdr x)))]
       [#t 0]))
(define x (cons ??? list′))
(define y (f1 x))
(define ans (= z′ y))
```

list′	**z′**	**???**
(list 1 "ab" (list "cde" #f) 5)	14	3
(list 1 "ab" (list "cde" #f) 5)	18	7
(list 1 "ab" (list "cde" #f) 5)	11	0
(list 3 "xyz" (list #t 4) "uw")	14	2
(list 3 "xyz" (list #t 4) "uw")	12	0
(list 3 "xyz" (list #t 4) "uw")	20	8

2. *Question 2 had a bug rendering it impossible to answer without mutation, which was not intended. All students received full credit for this question, regardless of response.*

3.
```
(define r 5)
(define (f3 s t)
 (let* ([t t′]
        [r t])
   (+ r s t)))
(define q (f3 ??? 10))
(define ans3 (= z′ q))
```

t′	**z′**	**???**
2	10	6
2	6	2
5	12	2
5	10	0

Question 4-6 (5 versions for #4)

4.
```
(define x ???)
(define y (foo x))
(define ans (equal? y (cons a′ b′)))
```

a′	b′	???
0	3	`(list 1 3 5)` *or any list with 0 even numbers and 3 odd numbers*
2	0	`(list 2 4)` *or any list with 2 even numbers and 0 odd numbers*
3	1	`(list 2 3 4 6)` *or any list with 3 even numbers and 1 odd number*
2	4	`(list 1 2 3 4 5 7)` *or any list with 2 even numbers and 4 odd numbers*
3	3	`(list 1 2 3 4 5 6)` *or any list with 3 even numbers and 3 odd numbers*

5. `(list 1 2 3 4 5 6)`
or any list with no sublists containing numbers

6. `(list 1 2 (list 3 4 (list 5)) 6)`
or any list with a sublist that contains numbers

Questions 7-8 (4 versions for #8)

7.
```
(define (stream-map f s)
  (lambda () (cons (f (car (s))) (stream-map f (cdr (s))))))
```

8. Write an expression to go in place of `???` so that `ans` results in a stream containing the same values as ***s′***. Assume `stream-map` works as described above, regardless of what you wrote in the previous problem.

```
(define ans (stream-map ??? t′))
```

s′	**t′**	**???**
negs	nats	`(lambda (n) (* n -1))`
evens	nats	`(lambda (n) (* n 2))`
odds	evens	`(lambda (n) (- n 1))`
evens	odds	`(lambda (n) (+ n 1))`

Questions 9-15 (questions were shuffled)

A type system that rejects all programs	Sound but not complete
A type system that rejects any program that contains a `first` expression or a `second` expression, and accepts all other programs	Sound but not complete
A type system that rejects any program that contains a `first` or `second` expression where the argument is not an `apair` expression and accepts all other programs	Sound but not complete
A type system that rejects any program that contains a `first` or `second` expression where the argument is a `call` expression and accepts all other programs	Neither sound nor complete
A type system that rejects any program that contains a `first` or `second` expression where the argument is an `int` expression, an `add` expression, or an `munit` expression and accepts all other programs	Complete but not sound
A type system that rejects any program that contains a `first` expression and accepts all other programs	Neither sound nor complete
A type system that accepts all programs	Complete but not sound

Questions 16-17

16.
```
[(ispos? e)
  (let ([v1 (eval-exp (ispos-e e))])
    (if (const? v1)
        (bool (> (const-int v1) 0))
        (error "ispos applied to non-number")))]
```

17.
```
(define (gt e1 e2) (ispos (add e1 (negate e2))))
```