0. Question `((lambda (x) (- (count x) 1)) 0)`

A correct spelling of my TA's name is _____.
My discussion section **number** is _____.

1. The `Scheme-` trix

What will Scheme print in response to the following expressions? If an expression produces an error message, you may just write "error"; you don't have to provide the exact text of the message. If the value of an expression is a procedure, just write "procedure"; you don't have to show the form in which Scheme prints procedures.

	`> (butfirst (butlast (se '(cooler?) 'jeff '(chung) )))`
	`> (first '(word 1 2 3))`
	`> (se (first 3) (bf 3))`
	`> ((23))`
	`> ((define (square x) (* x x)) 3)`
	`> ((first 'first) 'butfirst)`
	`> ((lambda (x y) x) '(1 2))`

	`> (define (f) (g))`
	`> (define (g) f)`
	`> (f)`

	`> (cond (first 'hello)`
	`(first '(1 2 3 4))`
	`(else 19))`

	`> (define (a b c)`
	`(define (b a) a)`
	`(b c))`
	`> (a 4 2)`

	`> (every ((lambda (x) (x 3)) +) '(1 2 3 4))`
	`> (keep (lambda (x) x) '(false = true))`

2. RUNNING TOTAL... In theaters now!

Write a function `running-total`, that takes in non-empty sentence of numbers and returns another sentence of the running totals. In other words, the first number in the resulting sentence should be the first number of the argument sentence; the second number in the resulting sentence should be the sum of the first and second numbers in the argument sentence, and so on.

```
> (running-total '(1 2 3 4))
(1 3 6 10)
> (running-total '(-5 0 -22 18 55))
(-5 -5 -27 -9 46)
> (running-total '(3))
(3)
```

See if you can write this function without defining any helpers!

3. CASw6e1sAome

Define a function `interleave` that takes two words as arguments and returns a new word with their letters interleaved. The words need not be of the same length. *You may use exactly one* `if` *in your solution and no other conditionals.*

```
> (interleave ‘conniving ‘chung)
ccohnunnigving
> (interleave ‘david ‘kevin)
dkaevviidn
> (interleave “” ‘jeff) ;; “” is the empty word
jeff
(interleave ‘iscool “”)
iscool
```

What would happen if you passed two sentences as arguments to `interleave`?

4. Enough

Suppose we're interested in the `count-change` problem again, but this time, we have a limited number of coins for each denomination. `count-change` now takes in three arguments `--` the amount to make change for, a sentence of denomination values, and a sentence of coin availability. Your task is to write this new version of `count-change`. For example,

`(count-change 100 '(25 10) '(3 7))` is the number of ways to make change for 100 using three quarters and seven dimes.

5. I Heart Huckabees

Write a function `keep-head` that, given a function `f` of one argument, returns a new function of one argument that returns the same value as `(f some-x)`, except the value is the *first* of what `(f some-x)` would return. For example:

```
(define foo (keep-head square))
> (foo 8)
6
> (foo 12)
1
```

6. Coach Carter

Here are some functions:

```
(define double (lambda (x) (* x 2)))
(define olympics (lambda (x)
(lambda (g) (g (g x)))))
```

And here is a Scheme expression. All it’s missing are parentheses; insert parentheses where needed to make the expression return a number.

```
olympics    olympics    2    double    double
```

7. Constantine

Write a procedure named `constant-fn?` that takes two arguments: a function of one argument, and a sentence of numbers. It should return `#t` if the value returned by the function is always the same for all of the numbers in the argument sentence. For example:

```
> (constant-fn? sqrt '(2 3 4 5 6))
#f
> (constant-fn? (lambda (x) 4) '(5 3 88 2 100 7 8 9))
#t
> (constant-fn? (lambda (a) (< a 10)) '(22 23 24 25 26 27))
#t
> (constant-fn? (lambda (a) (< a 10)) '(5 7 9 11))
#f
```

8. Freddy Vs. Jason

Given the definitions below, decide which of the following expressions return the same value in both normal and applicative order of evaluation. If both result in an error, this is considered returning the same value.

```
(define (f x y) (/ 10 x))
(define (g) (lambda (y a) (y a)))
(define (h x) (lambda () (random x)))
```

(a) `(f 0 2)`

(b) `(f 10 ((g) 2 2))`

(c) `(h (f 0 2))`

(d) `((g) h 0)`

9. Campus Phone

Consider the following procedure.

```
(define (biggest sent)
  (define (big-iter big s)
    (cond ((empty? s) big)
          ((> (first s) big) (big-iter (first s) (bf s)))
          (else (big-iter big (bf s)))))
  (big-iter (first sent) sent))
```

(a) What is the domain of `biggest`? Be specific and concise.

(b) In a few words, describe the invariant for `big-iter` above that would be most useful in explaining how the procedure works.

Consider the procedure below. `selsort` is supposed to sort a sentence of numbers into a sentence in increasing order by adding a number to the head of the sorted sentence on each iteration.

```
(define (selsort seq)
  (define (iter sorted unsorted)
    (if (null? unsorted)
        sorted
        (iter (cons (biggest unsorted) sorted) (cdr unsorted))))
  (iter '() seq))
```

(c) The procedure `selsort` has a bug. In one short English sentence, say what it is. You need not say how to fix it.

(d) Which of the following statements is/are necessary to prove that `selsort` terminates (i.e. doesn't run forever) despite the bug? Check all that apply.

__ `sorted` is initially empty.
__ The length of `unsorted` gets smaller in every iteration.
__ When invoked by `selsort`, `biggest` terminates.
__ The input `sent` is a sentence that contains only positive integers.
__ The length of `sorted` plus the length of `unsorted` is a constant.

9. Title 9

```
(define (garply n)
  (if (< n 20)
    n
   (+ (foo n)(garply (- n 1)) )) )
```

Assuming foo is defined somewhere, please circle `TRUE` or `FALSE` and provide a one- sentence explanation of your choice.

1. `TRUE` or `FALSE`: We have enough information to determine the order of growth of garply.

2. `TRUE` or `FALSE`: No matter how `foo` is defined, `garply` will always have an order of growth greater than or equal to $\Theta(n)$

3. `TRUE` or `FALSE`: `garply` has an order of growth of $\Theta(n^2)$ if `foo` is defined as follows:

```
(define (foo n)
  (if (< n 100)
    121
    (+ (* n 100) (foo (- n 1)))))
```

4. `TRUE` or `FALSE`: `garply` generates a iterative process.

5. `TRUE` or `FALSE`: You can find the largest number of a sentence of numbers in linear time.

6. `TRUE` or `FALSE`: You can find out whether a sentence contains two equal words (for example, the sentence `(the cat in the hat)` contains "the" twice) in linear time.

7. `TRUE` or `FALSE`: You can find out whether or not all the words in a sentence are equal in linear time.