1 What Would Scheme Print?

1. What will Scheme output?

```
scm> 3.14

scm> pi

scm> (define pi 3.14)

scm> pi

scm> 'pi

scm> (+ 1 2)

scm> (+ 1 (* 3 4))

scm> (if 2 3 4)

scm> (if 0 3 4)

scm> (- 5 (if #f 3 4))

scm> (if (= 1 1) 'hello 'goodbye)
```
scm> (define (factorial n)
  (if (= n 0)
      1
      (* n (factorial (- n 1)))))

scm> (factorial 5)
2. Hailstone yet again Define a program called hailstone, which takes in two numbers seed and n, and returns the nth hailstone number in the sequence starting at seed. Assume the hailstone sequence starting at seed is longer or equal to n. As a reminder, to get the next number in the sequence, if the number is even, divide by two. Else, multiply by 3 and add 1.

Useful procedures

- quotient: floor divides, much like // in python
  
  (quotient 103 10) outputs 10

- remainder: takes two numbers and computes the remainder of dividing the first number by the second
  
  (remainder 103 10) outputs 3

; The hailstone sequence starting at seed = 10 would be
; 10 => 5 => 16 => 8 => 4 => 2 => 1

; Doctests
> (hailstone 10 0) 10
> (hailstone 10 1) 5
> (hailstone 10 2) 16
> (hailstone 5 1) 16

(define (hailstone seed n) )
3 Special Forms

3. What will Scheme output?

```scheme
scm> (if 1 1 (/ 1 0))
```

```scheme
scm> (and 1 #f (/ 1 0))
```

```scheme
scm> (or #f #f 0 #f (/ 1 0))
```

```scheme
scm> (define a 4)
```

```scheme
scm> ((lambda (x y) (+ a x y)) 1 2)
```

```scheme
scm> ((lambda (x y z) (y x z)) 2 / 2)
```

```scheme
scm> ((lambda (x) (x x)) (lambda (y) 4))
```

```scheme
scm> (define boom1 (/ 1 0))
```

```scheme
scm> (define boom2 (lambda () (/ 1 0)))
```

```scheme
scm> (boom2)
```

Why/How are the two “boom” definitions above different?

How can we rewrite boom2 without using the lambda operator?
4. Define `apply-multiple` which takes in a single argument function $f$, a nonnegative integer $n$, and a value $x$ and returns the result of applying $f$ to $x$ a total of $n$ times.

```scheme
; doctests
scm> (apply-multiple (lambda (x) (* x x)) 3 2)
256
scm> (apply-multiple (lambda (x) (+ x 1)) 10 1)
11
scm> (apply-multiple (lambda (x) (* 1000 x)) 0 5)
5
```

```scheme
(define apply-multiple (f n x)
)