1 What Would Scheme Print?

1. What will Scheme output?

```scheme
(scm> (define pi 3.14))
```

**Solution:** pi

```scheme
(scm> pi)
```

**Solution:** 3.14

```scheme
(scm> 'pi)
```

**Solution:** pi

```scheme
(scm> (+ 1 2))
```

**Solution:** 3

```scheme
(scm> (+ 1 (* 3 4))
```

**Solution:** 13
scm> (if 2 3 4)

Solution: 3

scm> (if 0 3 4)

Solution: 3

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

Solution: 1

scm> (if (= 1 1) 'hello 'goodbye)

Solution: hello

scm> (define (factorial n)
    (if (= n 0)
        1
        (* n (factorial (- n 1)))))

Solution: factorial

scm> (factorial 5)

Solution: 120
2. **Hailstone yet again** Define a program called `hailstone`, which takes in two numbers `seed` and `n`, and returns the `n`th 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
  
  ```scheme
  (quotient 103 10) outputs 10
  ```

- **remainder**: takes two numbers and computes the remainder of dividing the first number by the second
  
  ```scheme
  (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)
Solution:
(define (hailstone seed n)
  (if (= n 0)
      seed
      (if (= 0 (remainder seed 2))
          (hailstone
           (quotient seed 2)
           (- n 1))
          (hailstone
           (+ 1 (* seed 3))
           (- n 1))))))
3. What will Scheme output?

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

Solution:
1
```

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

Solution:
#f
```

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

Solution:
0
```

```
scm> (define a 4)

Solution:
a
```

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

Solution:
7
```

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

Solution:
1
```

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

Solution: 4
```
4. Define \texttt{apply-multiple} which takes in a single argument function \texttt{f}, a nonnegative integer \texttt{n}, and a value \texttt{x} and returns the result of applying \texttt{f} to \texttt{x} a total of \texttt{n} times.

\begin{verbatim}
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
\end{verbatim}

\texttt{(define apply-multiple (f n x)}
(define (apply-multiple f n x)
  (if (= n 0)
      x
      (f (apply-multiple f (- n 1) x))))

Alternate solution:
(define (apply-multiple f n x)
  (if (= n 0)
      x
      (apply-multiple f (- n 1) (f x))))