INTRODUCTION TO SCHEME

CS 61A GROUP MENTORING

July 11, 2018

1 What Would Scheme Print?

1. What will Scheme output?

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

Solution: `pi`

```
scm> pi
```

Solution: `3.14`

```
scm> 'pi
```

Solution: `pi`

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

Solution: `3`

```
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 Code Writing in Scheme

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

     `(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)









)
```

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 Special Forms

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

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

Solution: Error: Zero Division

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

Solution: boom2

```
scm> (boom2)
```

Solution: Error: Zero Division

Why/How are the two "boom" definitions above different?

Solution: The first line is setting boom1 to be equal to the value `(/ 1 0)`, which turns out to be an error. On the other hand, boom2 is defined as a lambda that takes in no arguments that, when called, will evaluate `(/ 1 0)`.

How can we rewrite boom2 without using the lambda operator?

Solution:

```
(define (boom2) (/ 1 0))
```

4 More Code Writing

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.

```
;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
```

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

)

Solution:

```
(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))))
```