1 Learning Goals

- Practice writing some macros
- Review for the final

2 Macros

2.1 Write a macro that takes an expression and a number `n` and repeats the expression `n` times. For example, `(repeat-n expr 2)` should behave the same as `(twice expr)`. Note that it's possible to pass in a combination as the second argument (e.g. `(+ 1 2)`) as long as it evaluates to a number. Be sure that you evaluate this expression in your macro so that you don't treat it as a list.

Complete the implementation below, making use of the `replicate` function given below. The `replicate` function takes in a value `x` and a number `n` and returns a list with `x` repeated `n` times.

```scheme
(define (replicate x n)
  (if (= n 0) nil
      (cons x (replicate x (- n 1)))))

(define-macro (repeat-n expr n)
```

```scheme
scm> (repeat-n (print '(resistance is futile)) 2)
(resistance is futile)
(resistance is futile)
scm> (repeat-n (print (+ 3 3)) (+ 1 1))  ; Pass a call expression in as n
6
6
```

2.2 Write a macro that takes in two expressions and `or`'s them together (applying short-circuiting rules). However, do this without using the `or` special form. You may also assume the name `v1` doesn't appear anywhere outside of our macro. Fill in the implementation below.

```scheme
(define-macro (or-macro expr1 expr2)

  `(let ((v1 _____))

     (if  _____

        _____)))
```

```scheme
scm> (or-macro (print 'bork) (/ 1 0))
bork
scm> (or-macro (= 1 0) (+ 1 2))
3
```

3   Final Exam Prep

3.1   Fall 2020 Final, Question 2a

4 *Review!*

3.2 Fall 2020 Final, Question 3a