Instructions: For this exercise you are encouraged to work in groups of two so that you can discuss the problems, help each other when you get stuck and check your partners work. This lab introduces the lambda calculus including beta-reduction, encoding numbers and Booleans, and recursion using the Y combinatory.

1. Given the functions

```lisp
(define (sumf f n)
  (if (= n 0)
      0
      (+ (f n) (sumf f (- n 1))))
)

(define (nth-power n)
  (lambda (x)
    (power x n)))
```

evaluate (sumf (nth-power 3) 2) using the substitution model of function evaluation. Assume that (power x n) computes x^n.

2. Show that the and, or, and not functions using the lambda calculus in are correct by showing that the correct result is obtained for all combinations of the inputs.

3. Using beta-reduction show that (succ two) is three, where two and three are the Church encodings the numbers 2 and 3.

4. Use induction to prove that ((cn add1) 0) = the number n, where cn is the Church encoding of n. You may assume that add1 correctly adds 1 to its input.

5. Assume that m and n are the encodings of the numbers 2 and 3 as Church numerals. Using the substitution model of computation to show that (mulc m n) returns the
Church encoding of product \( m \times n = 6 \). You may assume that \((\text{addc} \ a \ b)\) returns the Church numeral representing the sum \( a+b \), where inputs \( a \) and \( b \) are Church encodings for numbers \( a \) and \( b \).

6. Trace through the expansion (use beta reduction) of \((g \ (Y \ g) \ 3)\) using the functions \( g \) and \( Y \) from the discussion on the \( Y \) combinator.

Try running this in the lazy evaluator (why won’t this work in Racket?). Note you will need to use currying so that \( g \) is a function of one variable.

For extra credit implement beta-reduction. For extra credit show how to compute and implement the predecessor of a Church number.