CSE 262: Quiz #4  
Due November 4th, 2022 at 11:59 PM

The quiz has TWO questions. Please submit your answer by updating this file in the quizzes folder of your Bitbucket account, and then committing and pushing. You should use as much space as you want for each answer. Please be detailed in your answers. Remember: this quiz is worth 9% of your grade, and you will not receive very many points if you do not give detailed answers.

**Question 1:** In Scheme, let is syntactic sugar. Explain how it is possible to get the equivalent of scoped variables without let. Give a code example with let, and then re-write it to the equivalent code without let. Then discuss whether you think it is worth it for Scheme to have added let.

Syntactic sugar is designed to make syntax easier to read or express within a programming language. In the programming language Scheme, let is syntactic sugar to get equivalent scoped variables that could have also been gotten using lambda. Lambda’s body has syntax that is similar to let, but the structure in which the variables, expressions, and bodies are paired and ordered are very different. To start, I would like to talk about the form of the lambda and the let expressions.

**The lambda expression form is: The let expression form is:**

((lambda (<var1> …<varn>) (let ((<var1> <exp1>)

<body>) (<var2> <exp2>)

<exp1> …

… (<varn> <expn))

<expn>) <body>)

As you can see, in the lambda expression, the variables are listed first, then the body, then the expression that the variables contain. On the other hand, the let expression lists the pairs of the variables and the expressions they contain right next to each other first, then the body. Evidently, the two structures are very different. Next, I would like to show an example of a let and lambda function that are identical to further show their difference in syntax. The two functions below are identical in functionality, but different in syntax because of the difference in structures in let and lambda as discussed previously.

Let function (Divide the sum of two numbers by 2) Lambda function (Divide the sum of two numbers by 2)

(define (foo1 a b) (define (foo2 a b)

(let ((c (+ a b))) ((lambda (c)

(/ c 2))) (/ c 2)

(+ a b)

In my opinion, I believe that it was worth it for Scheme to add let because it serves well as syntactic sugar in many cases. For example, if one were to have a function with a ton of variables, it would be hard to keep track of which variable is paired with which expression. If one were to use lambda, they would have to tediously match the number in which the expression and variable was declared. This could get messy. On the other hand, let makes it easier to read by allowing one to see the variable right next to the expression it represents. Overall, I find myself using let much more often than lambda because it’s easy to read.

**Question 2:** It has been said that null is a “billion dollar mistake”. Modern functional languages (and also languages like Rust) do not have null; instead they use things like Option<>. How does this work? How does it solve the problem(s) with null? Give a code example, and discuss the implications of **not** having null, both the positive and the negative.