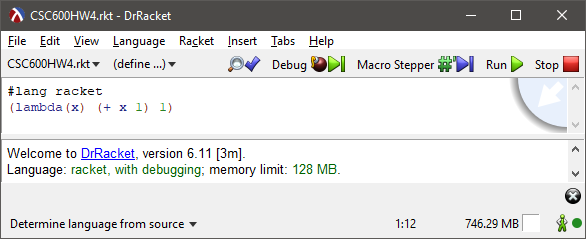
**CSC 600**

**Homework #4**

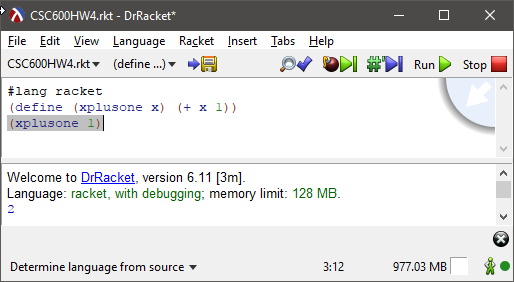
Parker Gray

11/10/2017

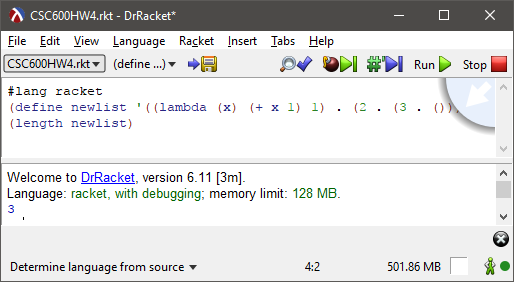
**1a.)** Below you see the function lambda (x). lambda (x) is an anonymous function that takes any number- in this case 1- and adds 1 to it. lambda (x) is an anonymous function because it contains no name to be called by- it doesn’t use the (define <funct>) form but it manages to be an object all its own despite this. Thus, this lambda (x) function is an example of an anonymous literal value.



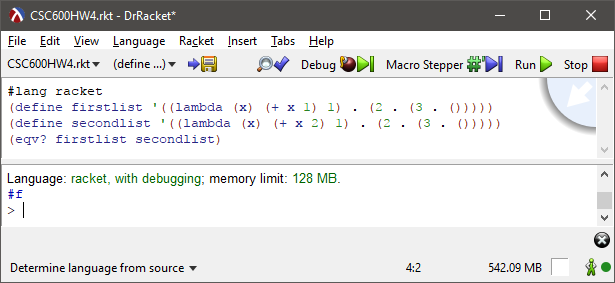
**1b.)** Below, a function (+ x 1) is given the variable name (xplusone x), or a reference with argument x. This demonstrates how functions may be stored as variables with names like xplusone.



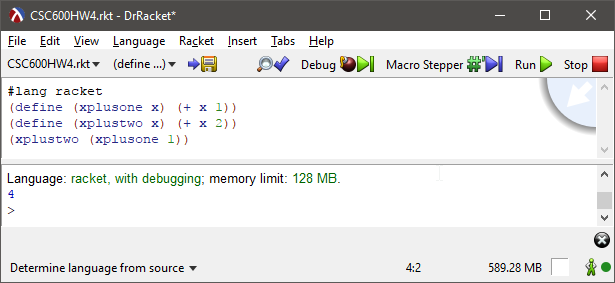
**1c.)** newlist is a list that contains an anonymous function as its first member. When the length function is ran, it reports that newlist has a length of 3, thereby acknowledging the anonymous lambda function as a member of the function, demonstrating that functions can be contained within lists.



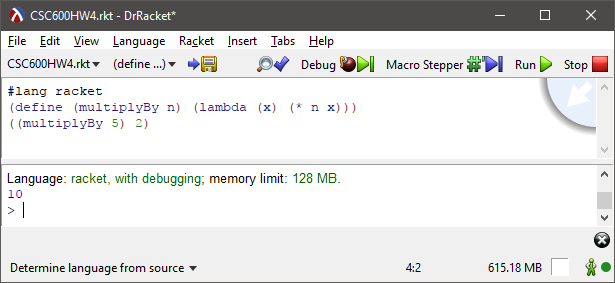
**1d.)** Here we have two separate lists that we run eqv? on. The only difference between the two lists is that the anonymous function for secondlist contains the arithmetic expression x + 2 rather than x + 1. Racket correctly deduces this to be the case and prints false.



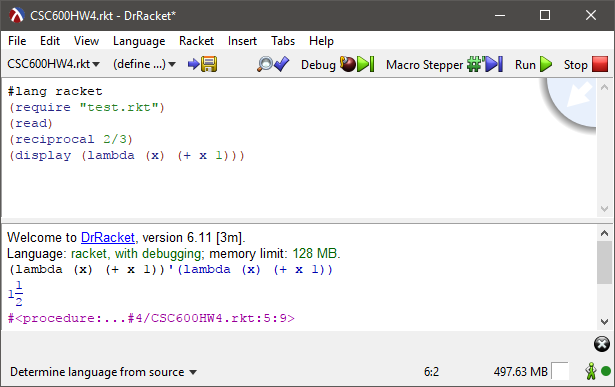
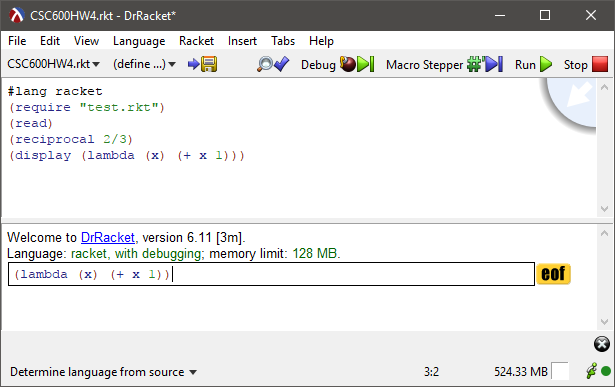
**1e.)** Two functions are defined- xplusone as before and xplustwo. After both are defined, xplusone is used as the argument for xplus two and returns the correct value 4. This demonstrates that functions can be passed as an argument.

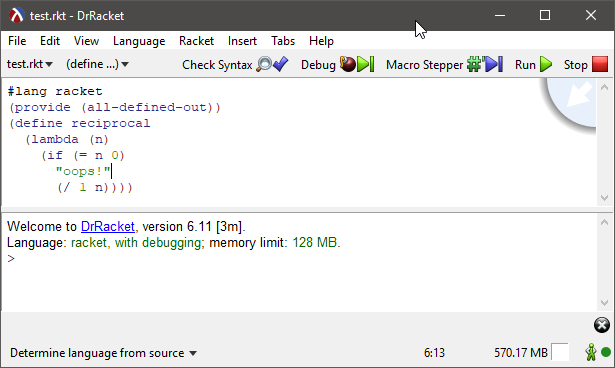


**1f.)** Below, a function multiplyBy multiplies function n by x- the following line (multiplyBy 5) 2) solves the sequence by passing both 5 and 2 as arguments to multiplyBy and are passed to the lambda function sequentially- returning a function to a function in other words.



**1g.)** Below I have the three different types of input specified displayed. I load “test.rkt” at the beginning of the file and then invoke the function within test.rkt (named reciprocal) on line 4. The contents of what I put into (read) are in the first image below. Finally, display prints out the ‘type’ of lambda (x) aka a procedure. The code contained in test.rkt is on the next page.





**2.)**