This is my introduction.

Racket is a multi-paradigm programming language in the Lisp-Scheme family. Part of its goal is to operate as a platform for language creation, design, and implementation. The platform provides a development environment called DrRacket which is written in Racket itself. The IDE is used in the "ProgramByDesign" outreach program in an endeavor to turn programming into an essential part of the liberal arts curriculum. Racket's core language includes modules, lexical closures, tail calls, delimited continuations, parameters and more.

The language also comes with primitives, which regulates resource administration and enables the language to act like an operating system for loading and organizing other programs.

Further extensions to the language are developed using powerful macro systems, which can control all properties of a language. Distinct from other programming languages that have an absence of macro systems, language constructs in Racket are created on top of the base language using macros.\\

Lisp is a family of programming languages with a unique, fully parenthesized prefix notation. Only Fortran is older than lisp making it the second-oldest high-level programming language still in prevalent use today. Lisp has changed a lot all since it began and today, the best-known purpose Lisp dialect is Scheme.

Scheme is a programming language that provisions several paradigms, and that includes functional programming which is one of the main dialects of Lisp. Distinct from Common Lisp, Scheme follows a plain design attitude stipulating a small standard core with powerful tools for language extension.\\

Functional programming languages are especially devised to manage symbolic computation and list processing applications. Functional programming is grounded upon mathematical functions. Some of the prevalent functional programming languages involve: Lisp, Python, Erlang, Haskell, Clojure, and more.

In order to explain Multirember, I will start by explaining Rember.

Rember is a function that removes the first instance of the first parameter from the second parameter.

We begin by defining the function rember and assigning it with an anonymous function "lambda" that allows us to write local functions that can be passed as arguments or returned as the value of function calls.

We allocate "lambda" with 2 arguments a and lat, so that the first parameter can be removed from the second one.

Within that function we start a conditional statement. Our conditional statement contains 2 expressions, the first expression is checking if lat is null and if it is then the body of that expression is outputting "quote()" which is an empty list '() and ignoring the rest of the expressions.\\

If lat is not null, the second expression comes to check if the first atom of lat is equal to our first parameter of "lambda" which is a, and if it is equal, the body of the second expression comes to output the cdr of lat and ignoring any other expressions.\\

While if both expressions turn out to be false then the else statement which is always true comes to action by constructing a new list with first atom of lat, since the first atom of lat is not equal to the first parameter of "lambda" and then calling the "rember" function again but only on the cdr of lat and this loop will keep on going until the first instance of atom "a" has been removed from the "lat" list and the function will then output the new list which excludes the first instance of atom "a".

Unlike rember, multirember is a function that removes all instances of the first parameter from the second parameter and not just the first instance.

Similar to the rember function, we begin by defining the function multirember and assigning it with an anonymous function "lambda" that allows us to write local functions that can be passed as arguments or returned as the value of function calls.

We allocate "lambda" with 2 arguments a and lat, so that the first parameter can be removed from the second one.\\

The first parameter "a" refers to an atom, and the second parameter "lat" refers to a list that the atom "a" will be removed from.

Within the multirember function, we start a conditional statement.\\

The first expression in the conditional statement checks if the list "lat" is null, if the list "lat" is null then there is no point checking if anything is equal to our first parameter "a" because our list is empty, so the body of the expression would output "quote()" which is an empty list '() and the rest of the function would be ignored.\\

If the list "lat" is not null, an else statement which is always true will run. In the else statement another conditional statement will appear. The first expression of the second conditional statement will check if the first atom of the list "lat" is equal to the first parameter of the lambda function "a", and if it is, the body of that expression will call the multirember function again but only on the cdr of the list "lat".\\

The way it goes is that the function will loop as many times as there are atoms in the list plus 1. So for example if there are 5 atoms in the list "lat, the function will run 6 times each time checking if the first atom in the list is equal to the first parameter "a", and when it finds a match it will return only the cdr of the list so excluding the car of the list since it found a match of it being equal to "a", and it will call multirember again on this new constructed list that contains the cdr, so that the second atom that we had in the old list has now become the first atom in our new constructed list.

When it doesn't find a match in the first atom of the list, it will construct a new list containing that first atom that is not equal to the first parameter "a", and calling multirember again but only on the cdr of the list "lat.

This will keep going on until the list is empty, and when the function runs again it will stop at the condition where the list "lat" is null and output the new constructed list that excludes all instances of the atom from the first parameter "a".