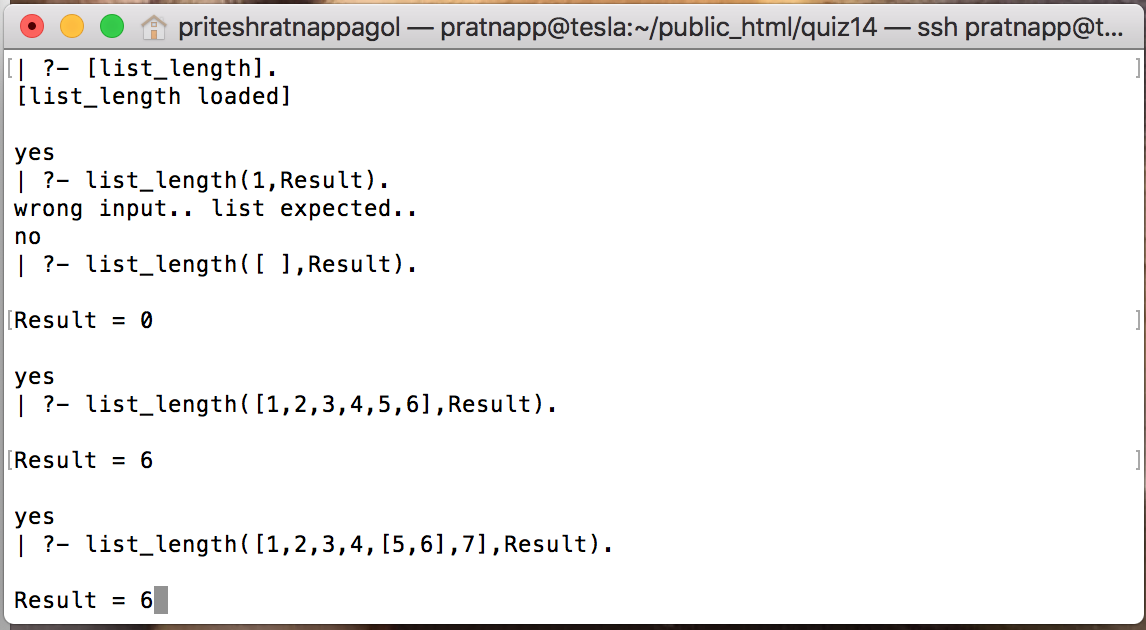
**Name : Pritesh Ratnappagol**

1. **Length of the list**

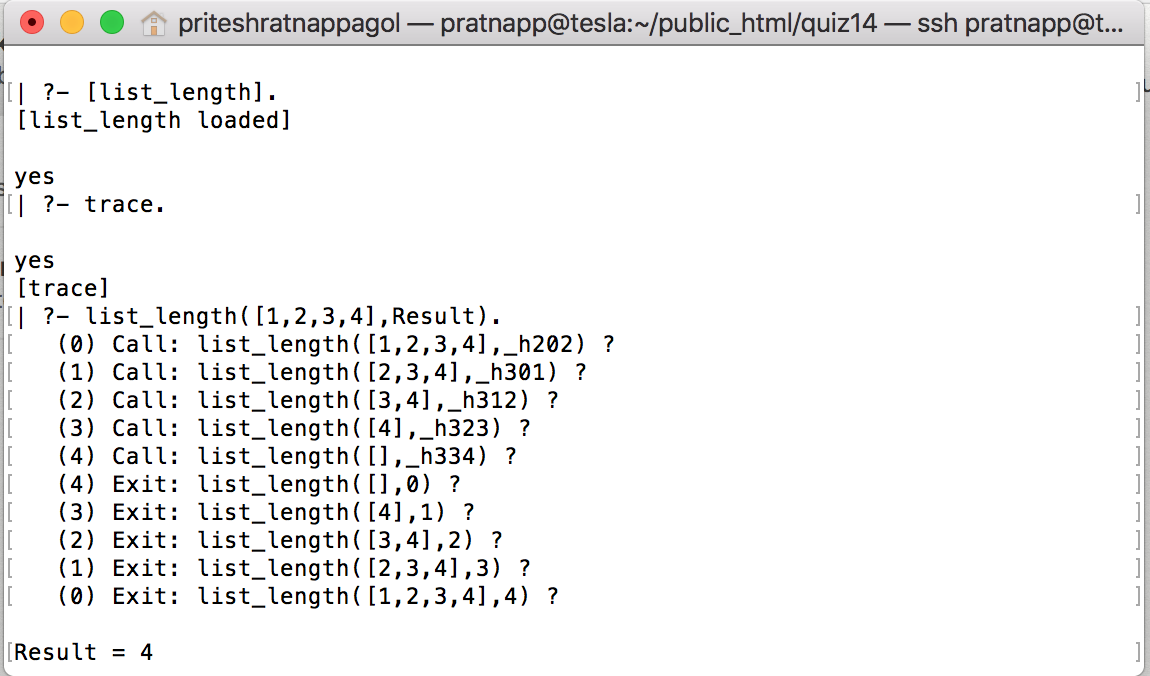
The program can be used to compute the length of the given list of various sizes.

Program also checks if there are no elements in the list then returns “ 0 ” (i.e. zero) else calculates the length of the list accordingly and returns the value.

To implement this I am first checking if it is a list or not, if not I am printing a error message; If no elements it will return zero else it will calculate the length of the list by recursively performing addition operation on the tail of the list.



Screenshot 1. Instance of finding length of the list

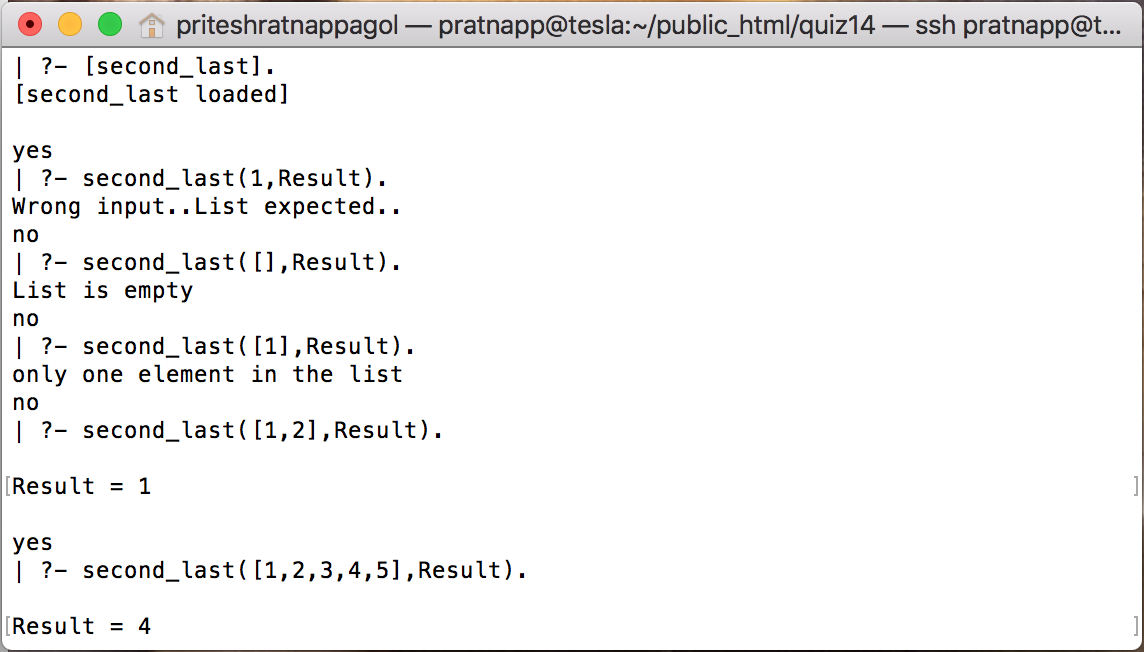


Screenshot 2. Trace of finding length of the list

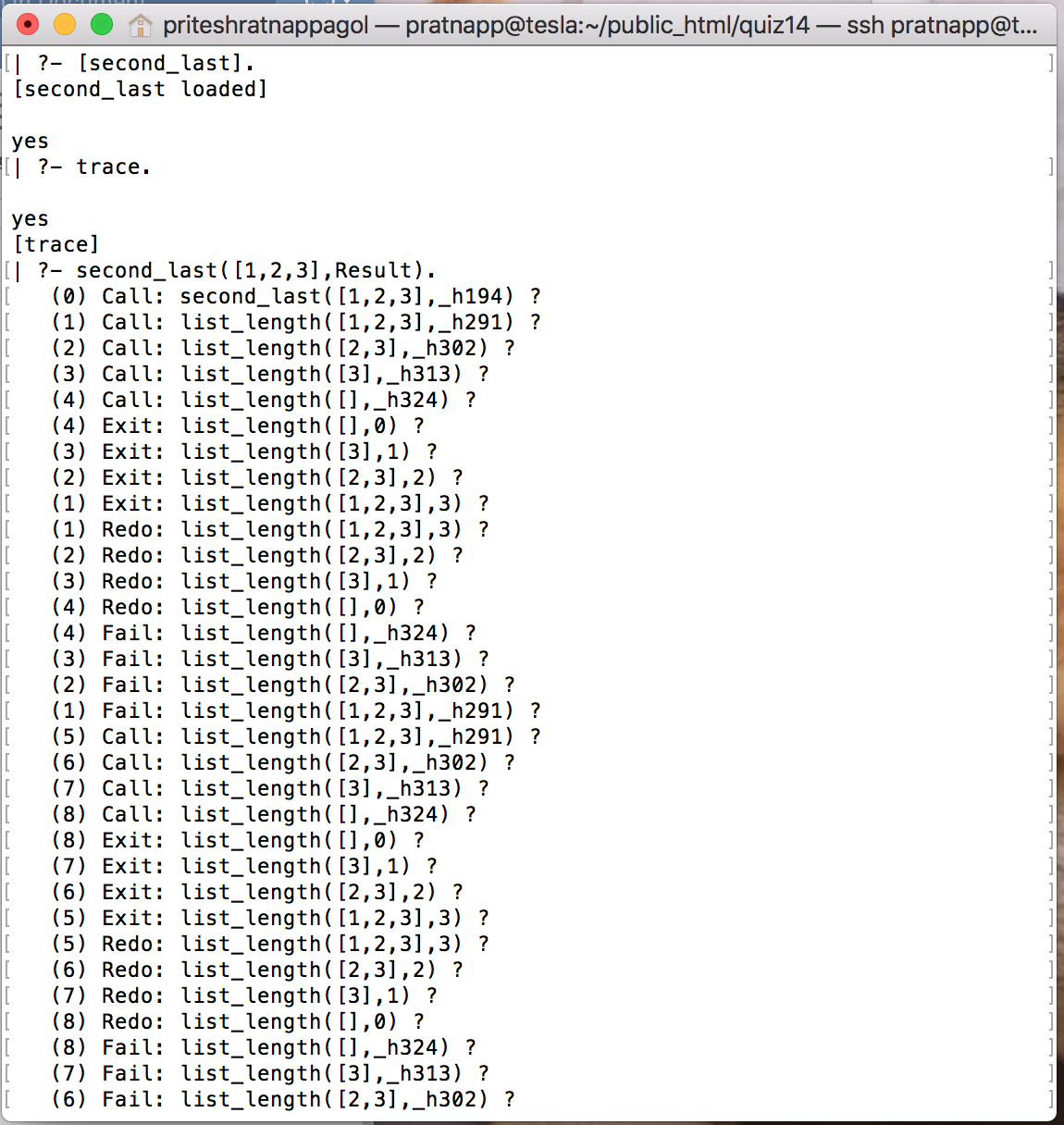
1. **Finding the second last element**

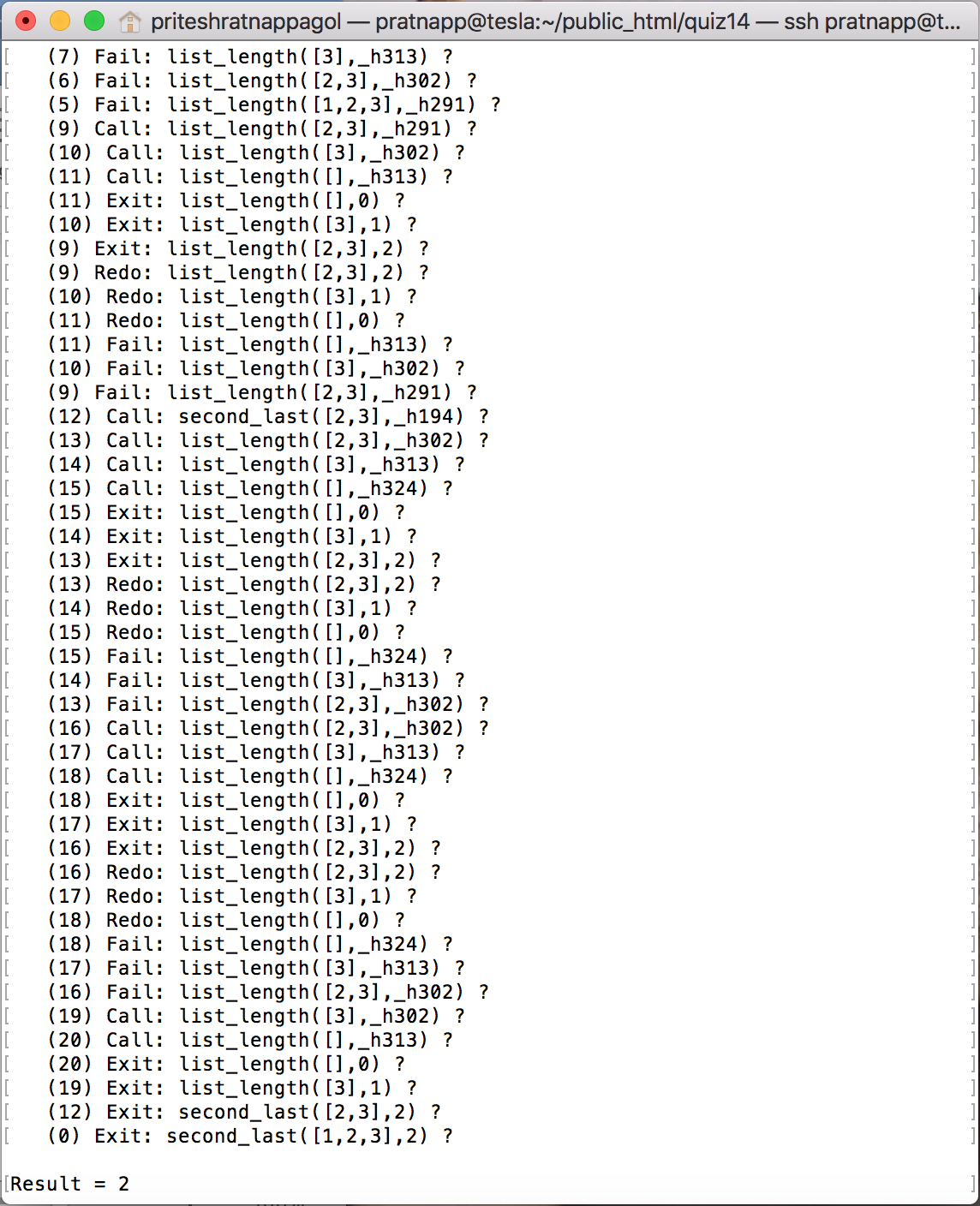
I have implemented it using four conditions:

1. Program first checks if the input is list or not.
2. Program checks if list is empty and if so it returns the statement “List is empty” and exits the program
3. If list is not empty and contains only one atom in it, it returns “List has only one item” and exits the program
4. If list is not empty and contains two element, it returns the second last element by simply using the car function for the given list.
5. If list is not empty and contains more than two elements it recursively performs the above steps on tail of the list till it is reaches the step 4 and returns the second last element in the list.



Screenshot 3. Instance of finding second last element

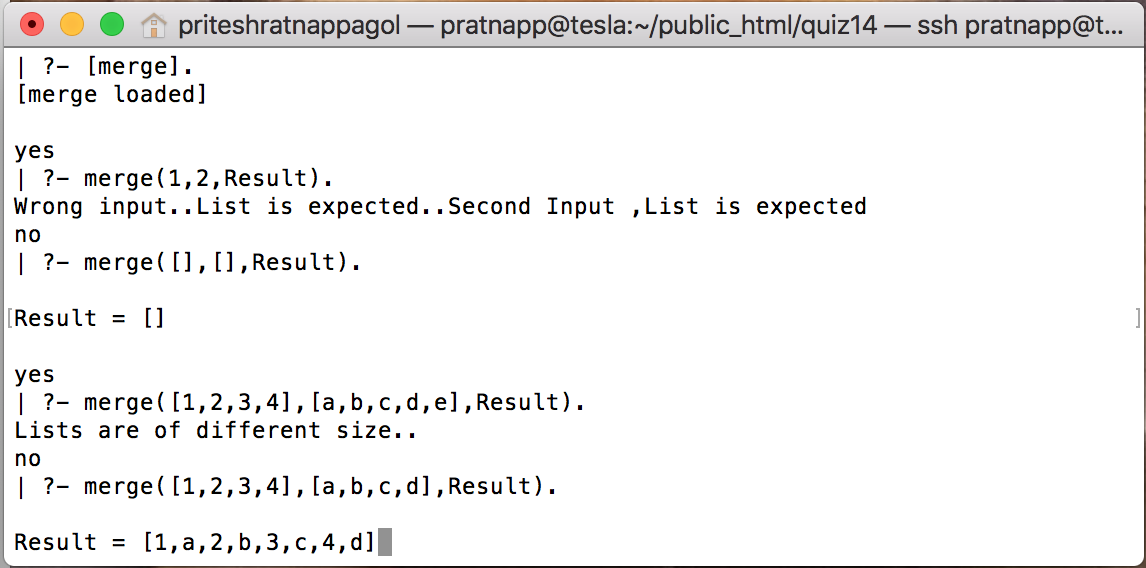




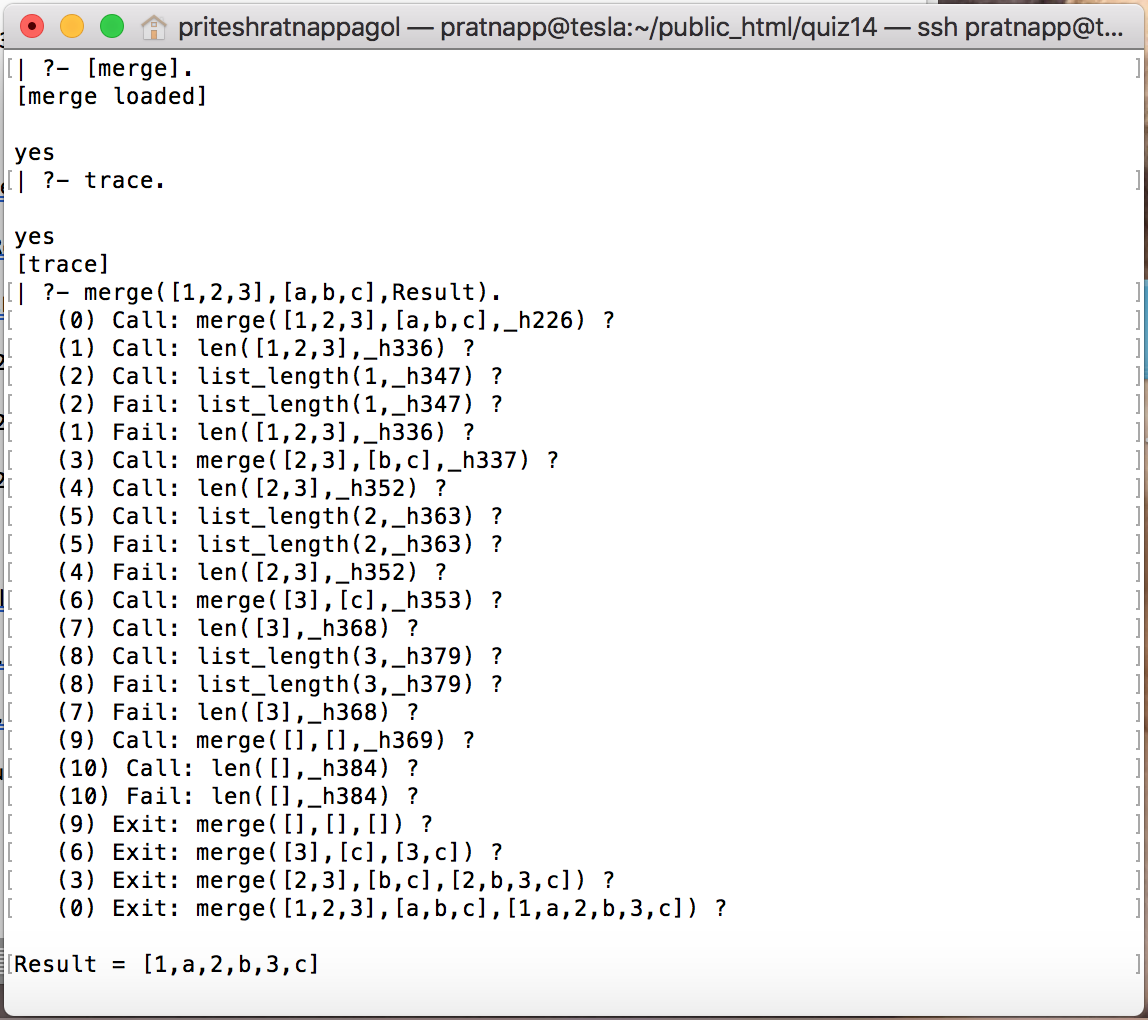
Screenshot 4. Trace of finding second last element

1. **Merging the two lists.**

To implement this function I am taking two lists of equal length from the user and checking the condition if length of the two list is equal or not and creating a new list by using by getting the first element from both the lists and recursively calling merging the two lists.



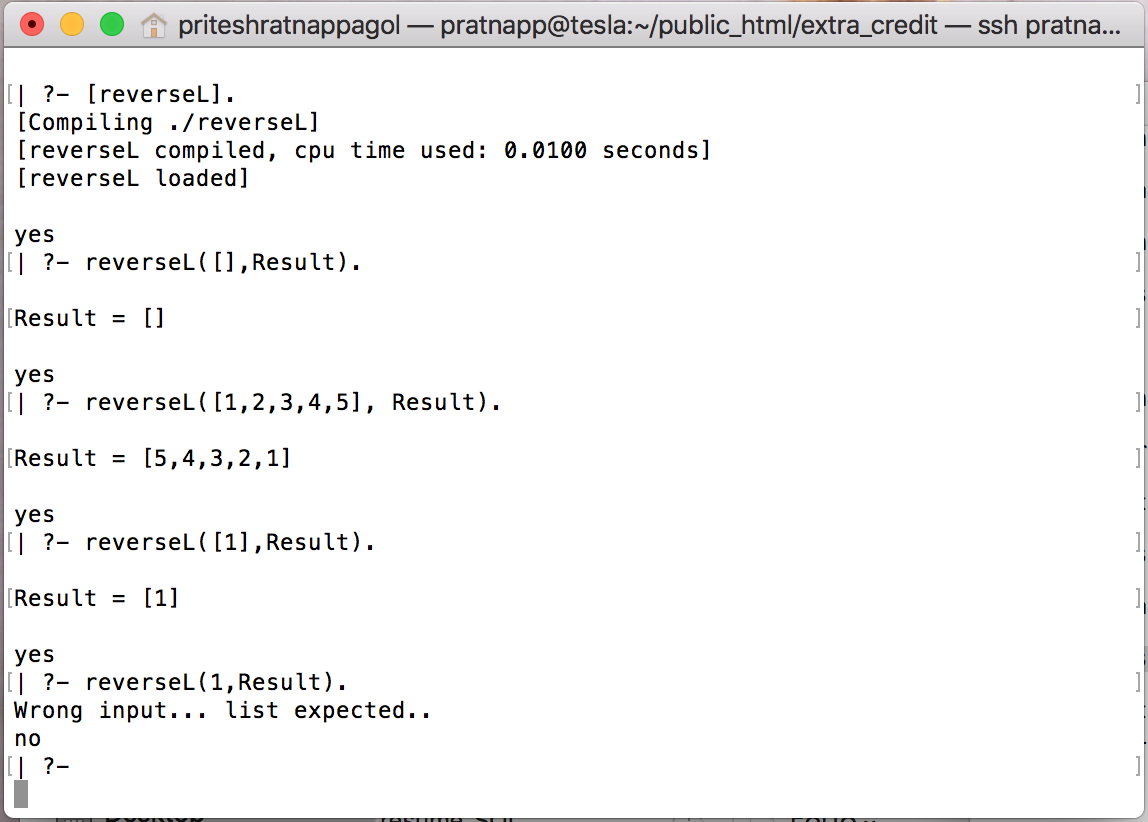
Screenshot 5. Instance of merging the list



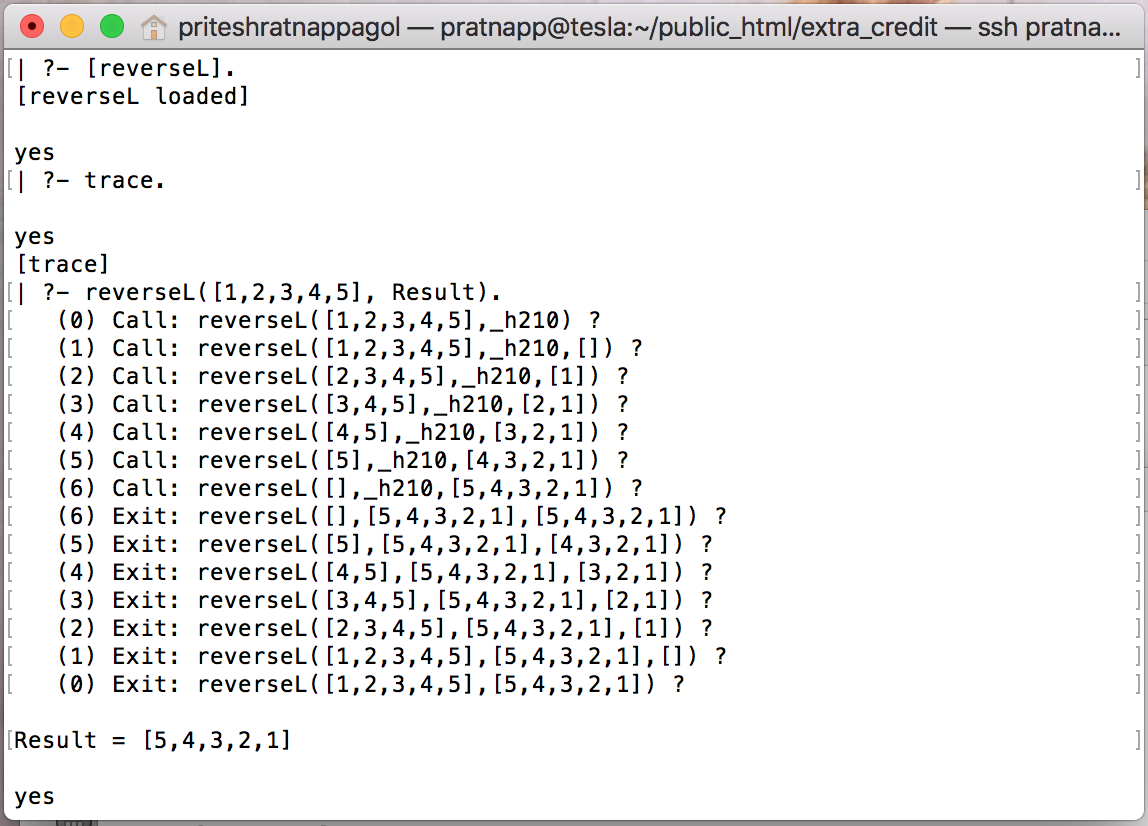
Screenshot 6. Trace of merging the list

1. Reversing the list.

* The implemented program first checks if the input given is list or not, and if wrong input is given then displays the error message accordingly.
* If we reverse the empty list, we will get the empty list in return (base case).
* If we reverse the list [H|T] , we will get the list obtained by reversing T and concatenating with [H] recursively . Trace of this is shown in Screenshot 2.
* Instances of this are shown Screenshot 1.



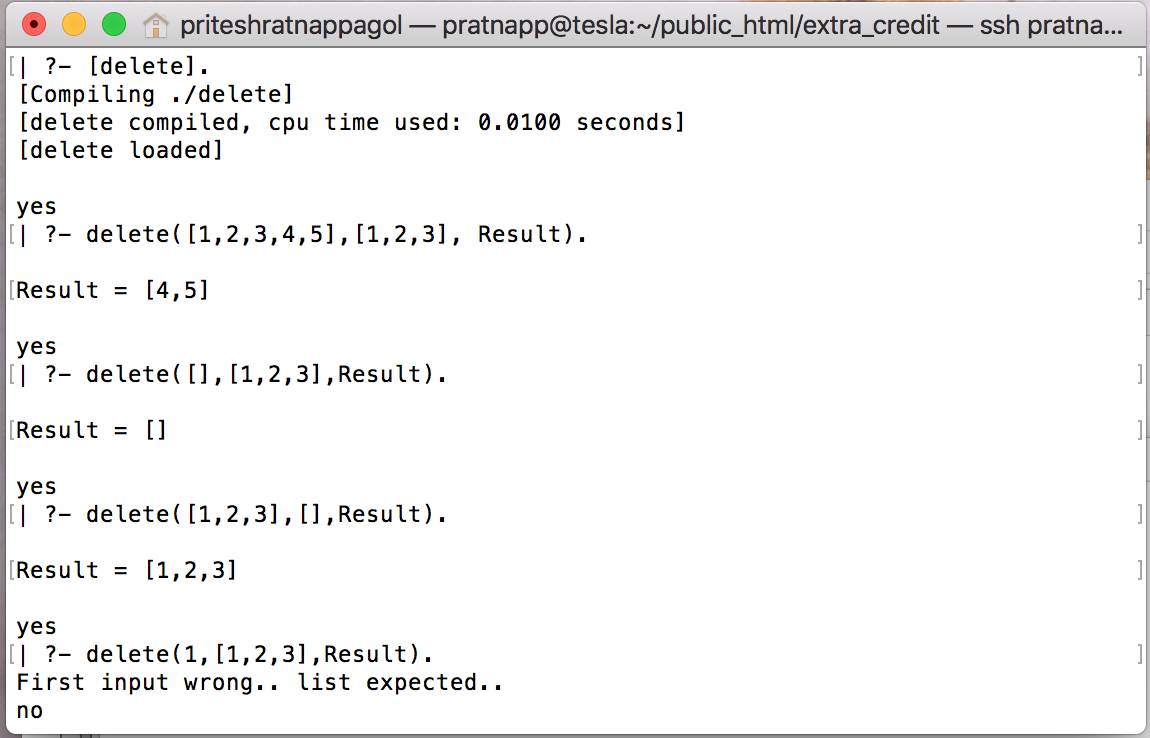
**Screenshot 1. Instances of the reverse program**



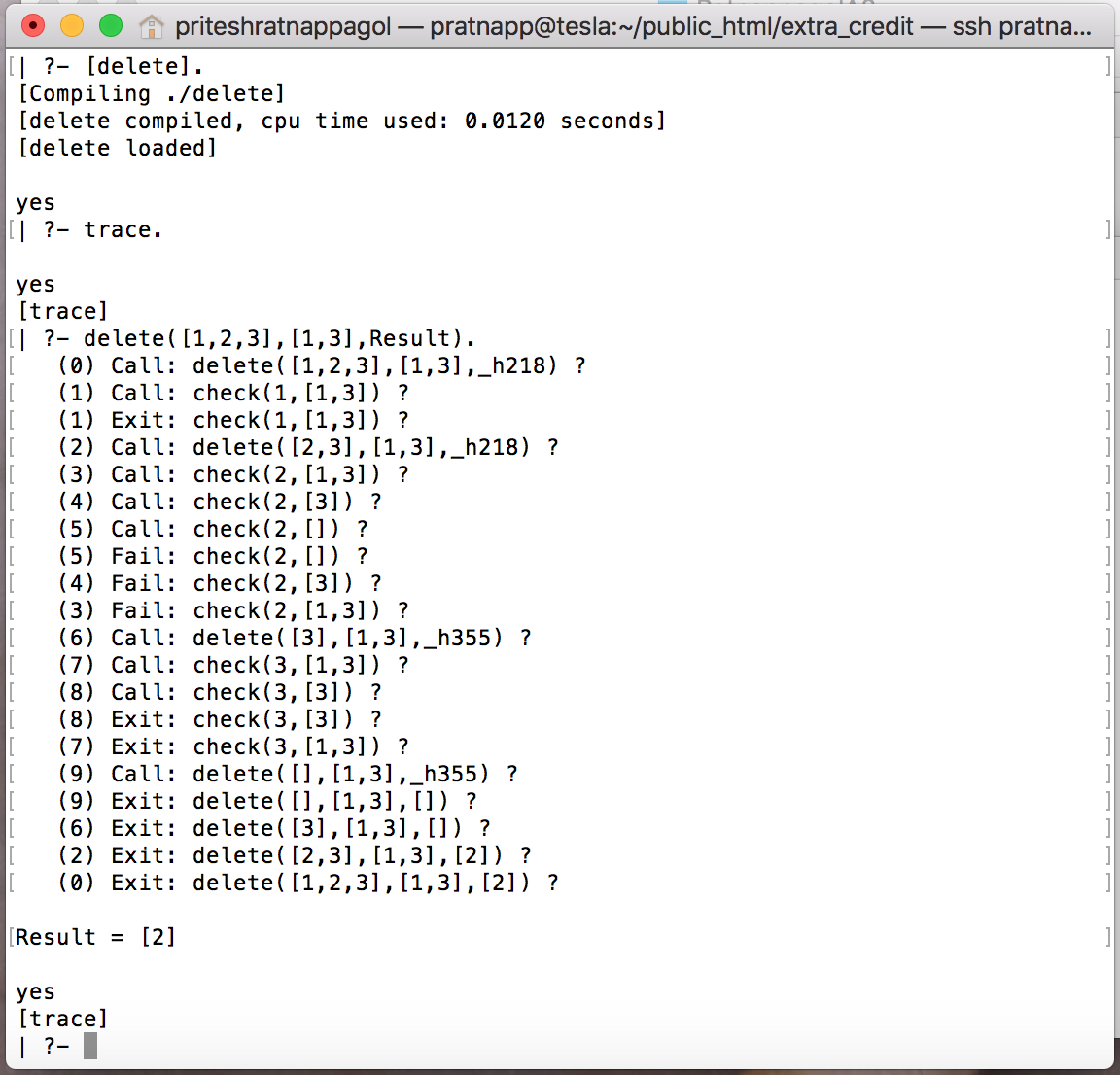
**Screenshot 2. Trace of the reverse program**

1. Creating a new list after deleting List 2 from List 1.

* The implemented program first checks if both the input given are list or not, and if wrong input is given then displays the error message accordingly.
* If empty list is deleted from the empty list, we will get the empty list in return (base case).
* To delete the elements in the list I have wrote a function that checks if a variable, atom or term is in the list or not. Trace of this is shown in Screenshot 4.
* Instances of this are shown Screenshot 3.



**Screenshot 3. Instances of the delete function**



**Screenshot 4. Trace of the delete function**

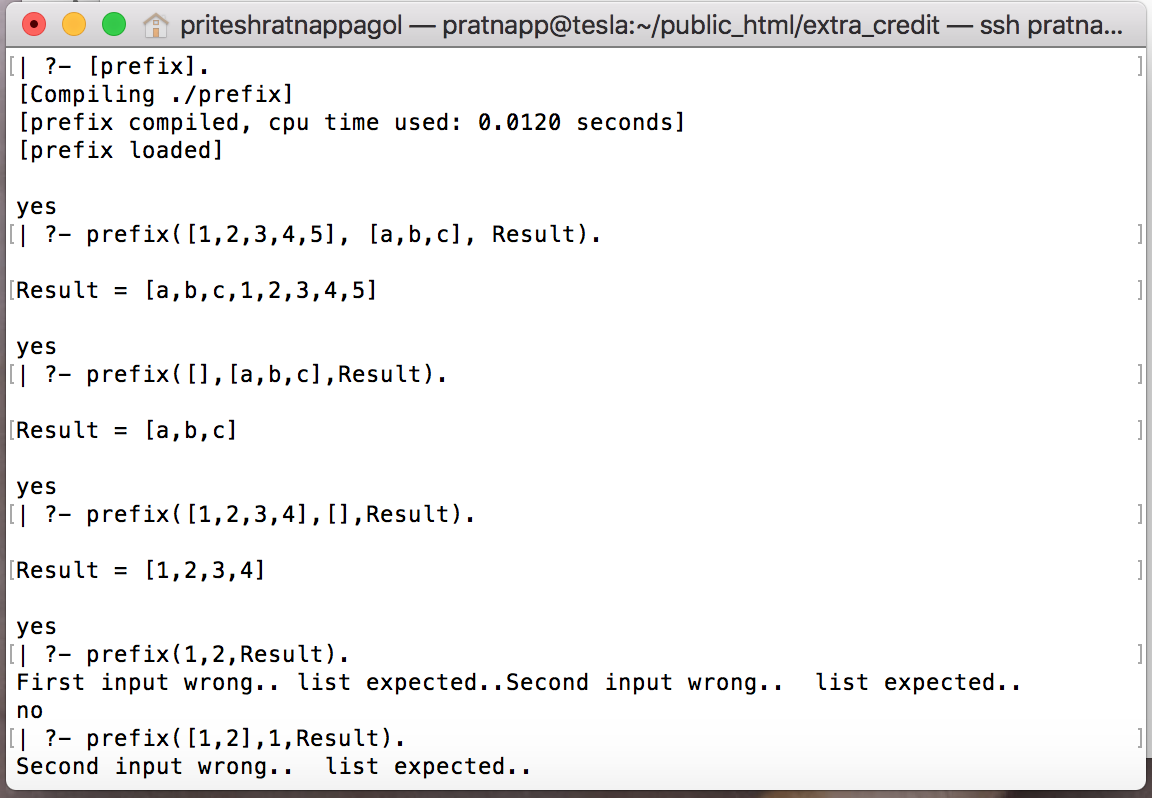
1. Prefix

The implemented program first checks if both the input given are list or not, and if wrong input is given then displays the error message accordingly.

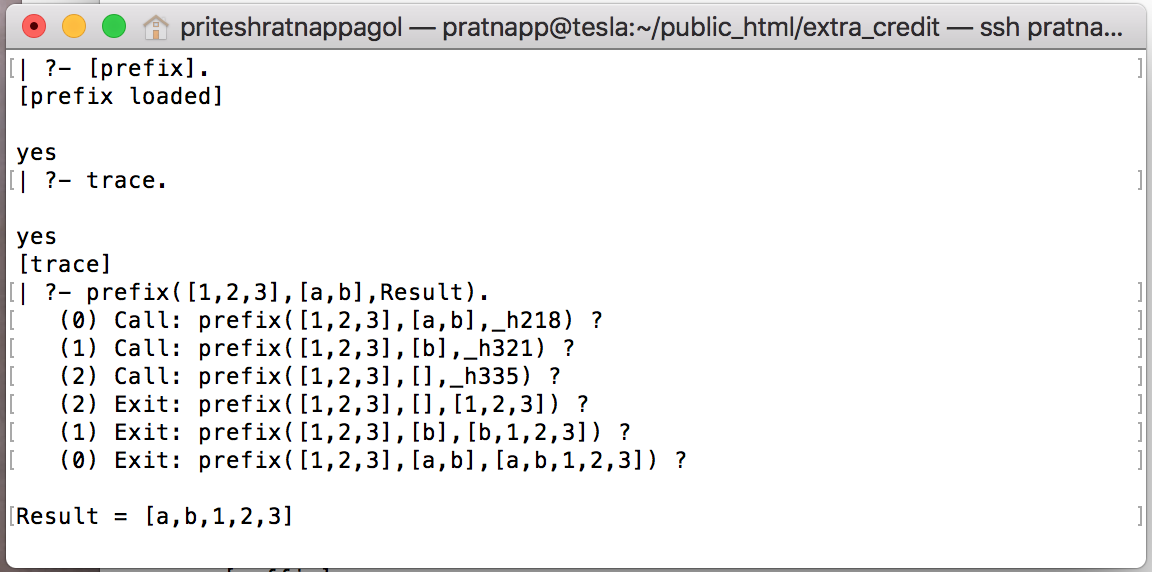
When empty list is appended to other list, the result is the other list.(Base case)

Appending [head | List\_2] to another list, List\_1 results in a new List, whose first element is the head and tail is List\_3 such that List\_3 is a result of appending List\_2 and List\_1. Trace is shown in screenshot 6.

Instances are shown in Screenshot 5.



**Screenshot 5. Instances of the prefix function**



**Screenshot 6. Trace of the prefix function**

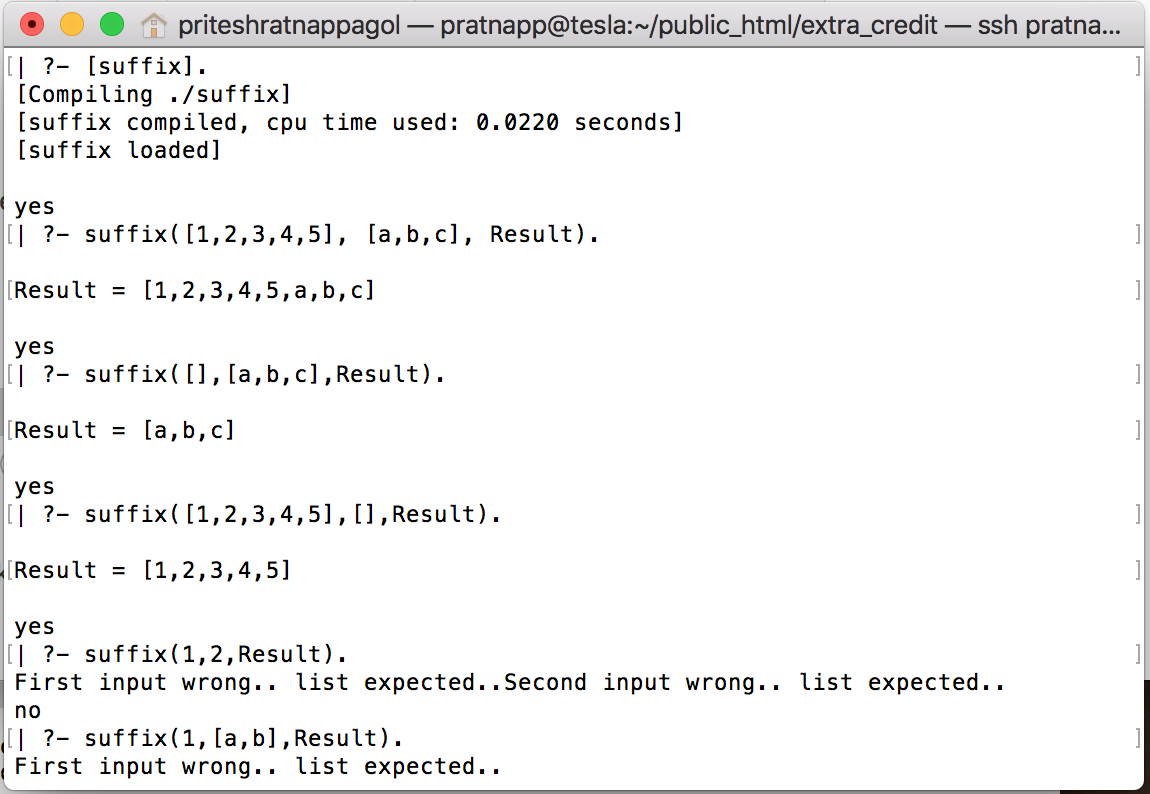
1. Suffix

The implemented program first checks if both the input given are list or not, and if wrong input is given then displays the error message accordingly.

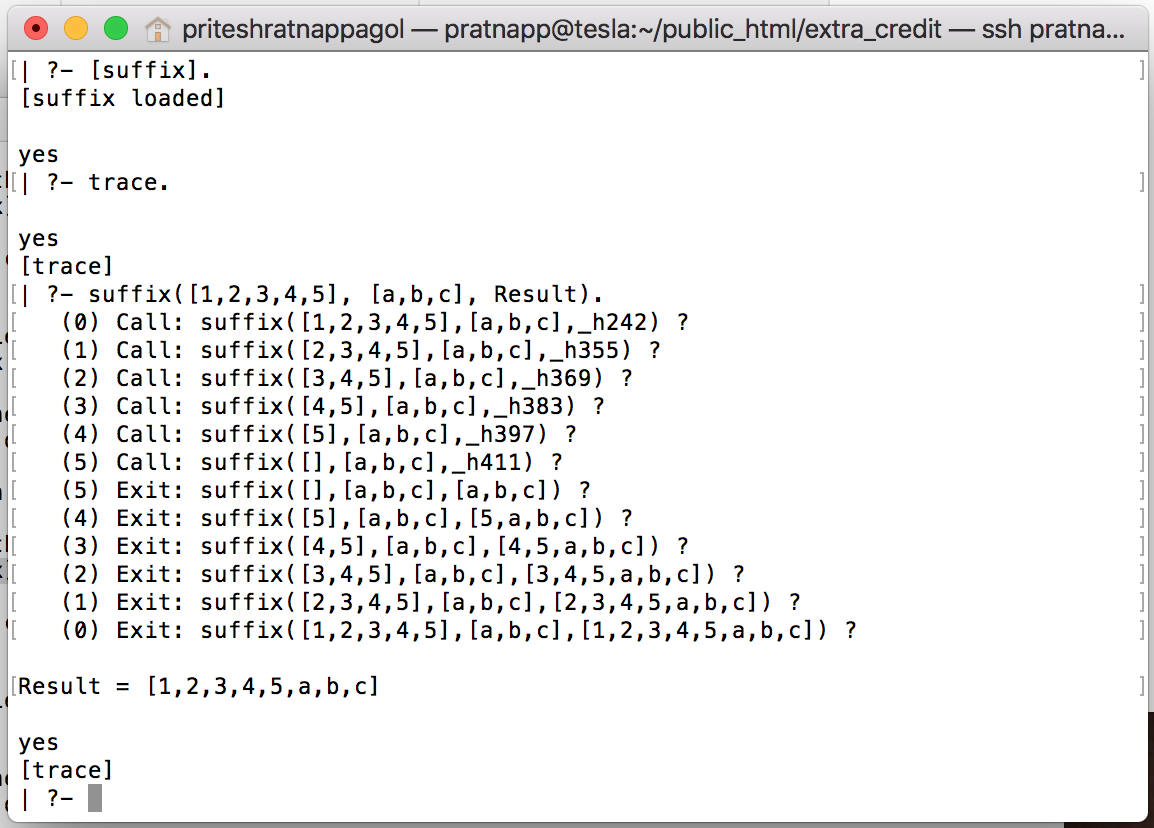
When empty list is appended to other list, the result is the other list.(Base case).

Appending [head | List\_1] to another list, List\_2 results in a new List, whose first element is the head and tail is List\_3 such that List\_3 is a result of appending List\_1 and List\_2. Trace is shown in screenshot 8.

Instances are shown in Screenshot 7.



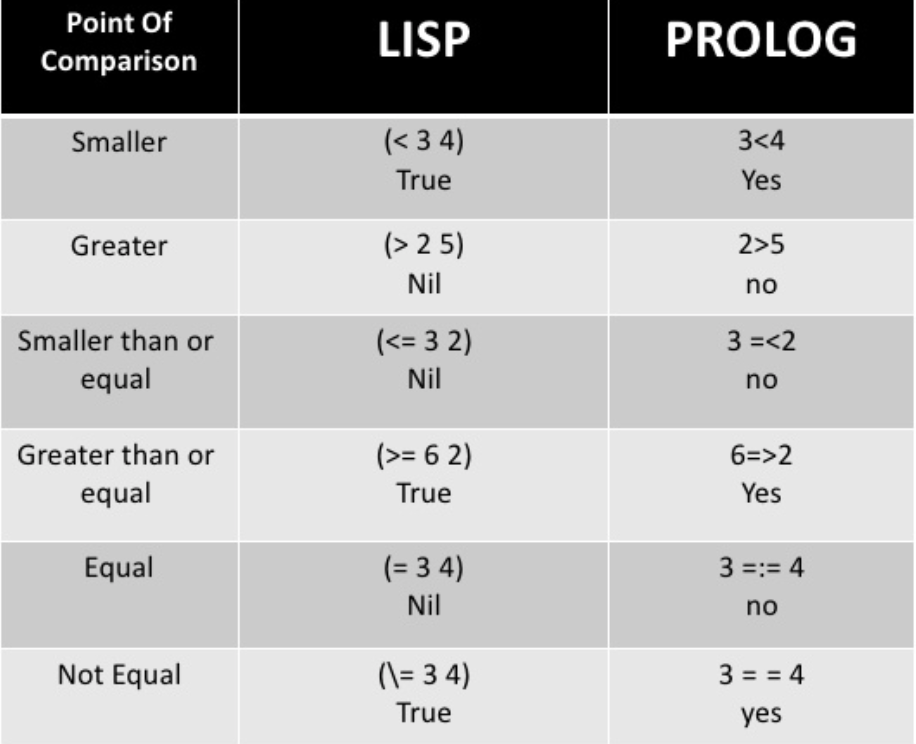
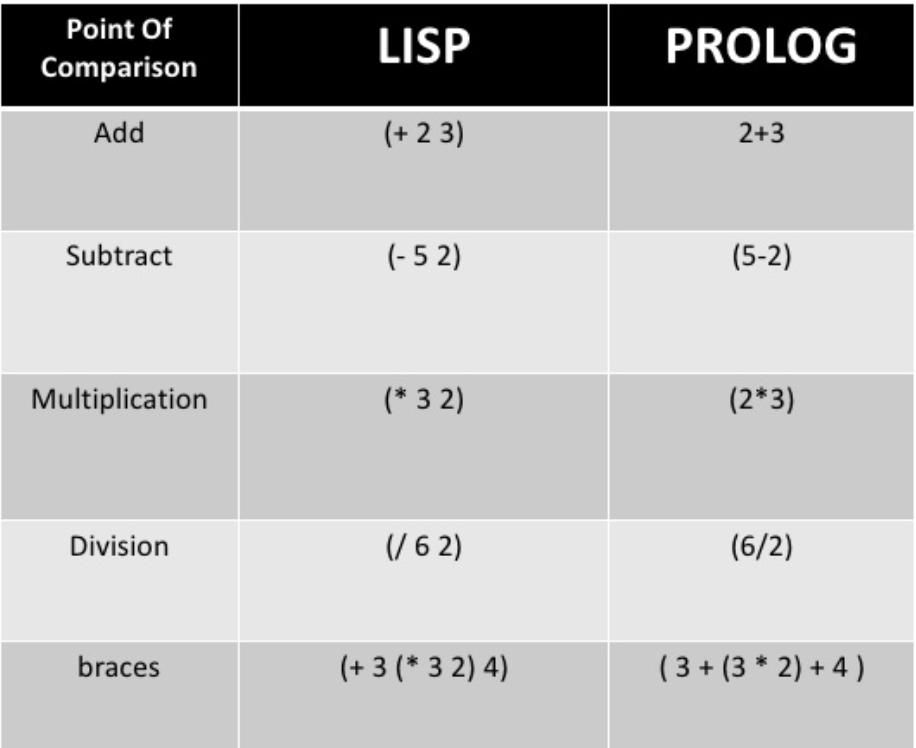
**Screenshot 7. Instances of the suffix function**



**Screenshot 8. Trace of the suffix function**

1. **Comparison between clisp and prolog.**

* Prolog is a declarative language, while CLISP is a functional language.
* Prolog is used most for logic and reasoning problems, while CLISP is used for problems with rapid prototyping needs.
* Prolog is smaller language and easier to learn , while CLISP handles variety of tasks and easier to use.
* Prolog is an interpreter and LISP is a compiler as prolog compiles the program line by line and LISP compiles the whole program at a time.
* Arithmetic and logical operations in Lisp and prolog.



**Finding the head and tail in prolog. Finding the head and tail in Clisp.**

[X|Y] denotes a list with head X and tail Y. head (called car) and the tail (called cdr)

| - used to create a list or dismantle a list.

Example: Example:

[H|T] = [a, b, c]. car((A.B))=A , car((A.B) .C)) = (A.B)

H= a, T = [b, c] cdr((A.B)) = B , cdr(((C.B).D)) is D

[a|T] = [H, b, c].

H= a, T = [b, c]