Documenting Steps & Experimental Results

Step 1. Defining List of Primitives, & Basic Functions

Step 2. Some basic testing :

LLM : llama3-70b-8192 (temp = 1, max\_tokens = 1024, top\_p = 1)

Prompt :

A new programming language follows the syntax of Scheme using keywords like lambda, define, cond, #t, #f and so on, and has the same basic data types : atom, list and S - expressions. Here are some basic functions in this programming language.

rac : gives first S-expression of a list

defined only for non - empty lists, returns a S - expression

rdc : gives list l without (by removing) rac l

defined only for non - empty lists, returns a list

snoc : adds an S - expression to the front of a list

takes two arguments, second must be a list. output is a list

lun? : checks whether a list is empty list

defined only for lists

mota? : checks whether it is an atom

takes one argument (any S - expression)

qe? : checks if two non numeric atoms are equal

I have two variables

a = meat

l = (mashed potatoes and meat gravy)

I want to check if a is in the list of atoms l. Can you help me write the code for this? Output only the code directly. Let's make it a function so that I just have to pass a and l to it.

Output :

Here is the code:

```

(define (member? a l)

(cond

((lun? l) #f)

((qe? a (rac l)) #t)

(else (member? a (rdc l)))))

```

Let’s try and remove any mention of the language Scheme and instead describe its syntax.

Prompt :

The programming language we're discussing has a unique syntax that is centred around functional programming and recursion. Here's a breakdown of some key elements:

atoms: An atom is a single, indivisible value such as a number, boolean, or symbol.

lists: A list is a compound data structure that contains an ordered sequence of elements, which can be atoms or other lists.

S-expressions (Symbolic Expressions): S-expressions are the primary syntactic constructs in the language, consisting of atoms or lists enclosed in parentheses.

define: This keyword is used to define variables, functions, or procedures. For example:

lambda: This keyword is used to create anonymous functions or lambda expressions. It takes a list of arguments and a body of expressions to be evaluated.

cond: This keyword is used for conditional branching. It evaluates a series of test expressions and evaluates the corresponding expression for the first test that evaluates to true.

else: This keyword is used in the cond expression to specify the expression to be evaluated if none of the previous test expressions are true.

'(): This represents the empty list, often used as a terminating condition in recursive list operations.

#t and #f: These represent the boolean values true and false, respectively.

The language uses prefix notation, where the operator or function comes before its operands or arguments, enclosed in parentheses. It also supports recursion, where the function calls itself with a smaller portion of the list until the base case is reached.

Here are some basic functions in this programming language.

rac : gives first S-expression of a list

defined only for non - empty lists, returns a S - expression

rdc : gives list l without (by removing) rac l

defined only for non - empty lists, returns a list

snoc : adds an S - expression to the front of a list

takes two arguments, second must be a list. output is a list

lun? : checks whether a list is empty list

defined only for lists

mota? : checks whether it is an atom

takes one argument (any S - expression)

qe? : checks if two non numeric atoms are equal

I have two variables

a = meat

l = (mashed potatoes and meat gravy)

I want to check if a is in the list of atoms l. Can you help me write the code for this? Output only the code directly.

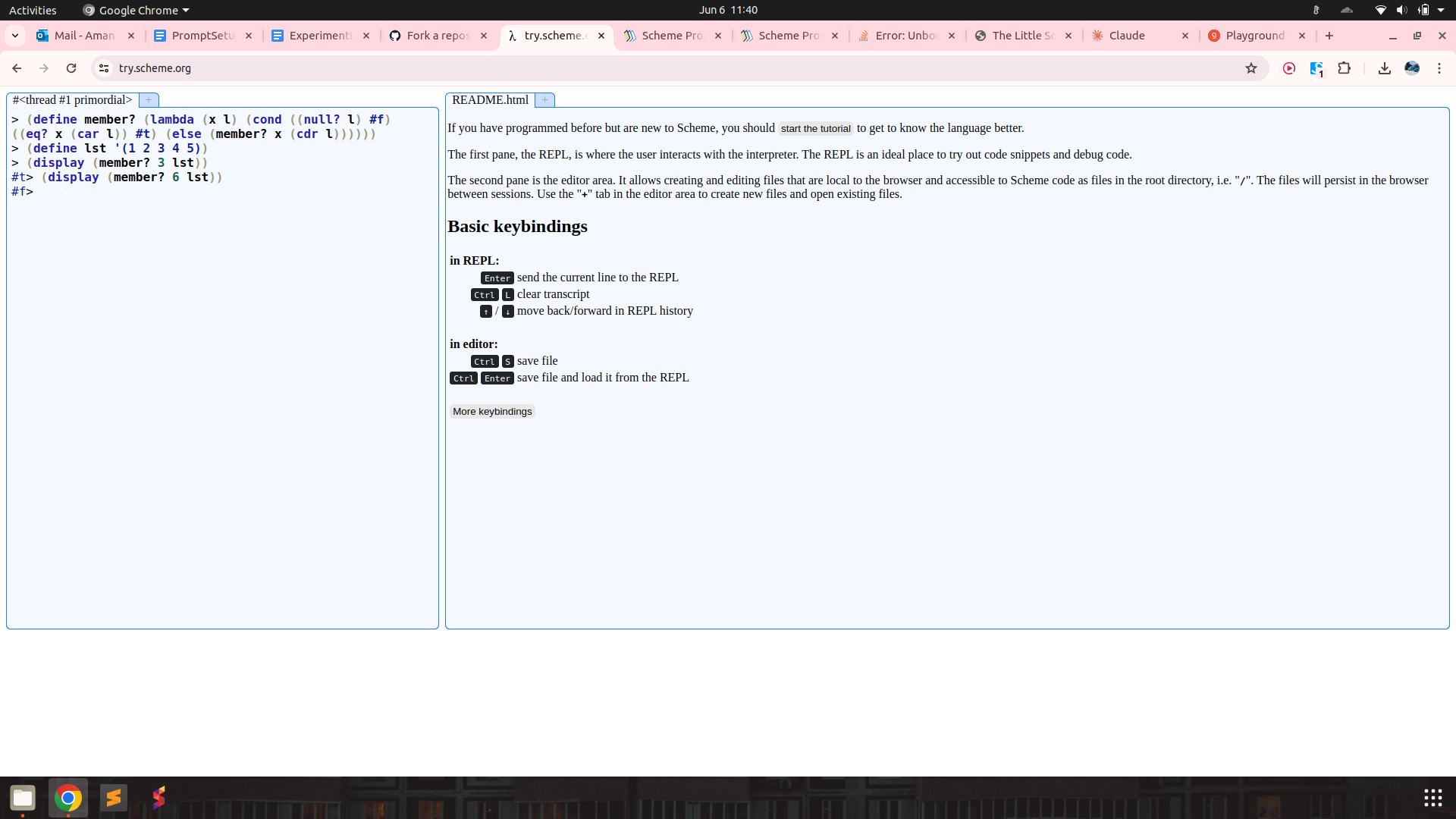
Output :

(define member? (lambda (x l) (cond ((lun? l) #f) ((qe? x (rac l)) #t) (else (member? x (rdc l))))))

(member? a l)

Result :

Decoding primitive names back to original and testing, function works as needed.



Checking if a list l is a list of atoms :

The programming language we're discussing has a unique syntax that is centred around functional programming and recursion. Here's a breakdown of some key elements:

atoms: An atom is a single, indivisible value such as a number, boolean, or symbol.

lists: A list is a compound data structure that contains an ordered sequence of elements, which can be atoms or other lists.

S-expressions (Symbolic Expressions): S-expressions are the primary syntactic constructs in the language, consisting of atoms or lists enclosed in parentheses.

define: This keyword is used to define variables, functions, or procedures. For example:

lambda: This keyword is used to create anonymous functions or lambda expressions. It takes a list of arguments and a body of expressions to be evaluated.

cond: This keyword is used for conditional branching. It evaluates a series of test expressions and evaluates the corresponding expression for the first test that evaluates to true.

else: This keyword is used in the cond expression to specify the expression to be evaluated if none of the previous test expressions are true.

'(): This represents the empty list, often used as a terminating condition in recursive list operations.

#t and #f: These represent the boolean values true and false, respectively.

The language uses prefix notation, where the operator or function comes before its operands or arguments, enclosed in parentheses. It also supports recursion, where the function calls itself with a smaller portion of the list until the base case is reached.

Here are some basic functions in this programming language.

rac : gives first S-expression of a list

defined only for non - empty lists, returns a S - expression

rdc : gives list l without (by removing) rac l

defined only for non - empty lists, returns a list

snoc : adds an S - expression to the front of a list

takes two arguments, second must be a list. output is a list

lun? : checks whether a list is empty list

defined only for lists

mota? : checks whether it is an atom

takes one argument (any S - expression)

qe? : checks if two non numeric atoms are equal

I have a list

l = (mashed potatoes and meat gravy)

I want to check if l is a list of atoms. Can you help me write the code for this? Output only the code directly.

Output :

(define atoms? (lambda (l) (cond ((lun? l) #t) ((mota? (rac l)) (atoms? (rdc l))) (else #f))))

Result :

(define atoms? (lambda (l) (cond ((null? l) #t) ((atom? (car l)) (atoms? (cdr l))) (else #f))))

function is correct.

Finding index of element in a list

Prompt :

The programming language we're discussing has a unique syntax that is centred around functional programming and recursion. Here's a breakdown of some key elements:

atoms: An atom is a single, indivisible value such as a number, boolean, or symbol.

lists: A list is a compound data structure that contains an ordered sequence of elements, which can be atoms or other lists.

S-expressions (Symbolic Expressions): S-expressions are the primary syntactic constructs in the language, consisting of atoms or lists enclosed in parentheses.

define: This keyword is used to define variables, functions, or procedures. For example:

lambda: This keyword is used to create anonymous functions or lambda expressions. It takes a list of arguments and a body of expressions to be evaluated.

cond: This keyword is used for conditional branching. It evaluates a series of test expressions and evaluates the corresponding expression for the first test that evaluates to true.

else: This keyword is used in the cond expression to specify the expression to be evaluated if none of the previous test expressions are true.

'(): This represents the empty list, often used as a terminating condition in recursive list operations.

#t and #f: These represent the boolean values true and false, respectively.

The language uses prefix notation, where the operator or function comes before its operands or arguments, enclosed in parentheses. It also supports recursion, where the function calls itself with a smaller portion of the list until the base case is reached.

Here are some basic functions in this programming language.

rac : gives first S-expression of a list

defined only for non - empty lists, returns a S - expression

rdc : gives list l without (by removing) rac l

defined only for non - empty lists, returns a list

snoc : adds an S - expression to the front of a list

takes two arguments, second must be a list. output is a list

lun? : checks whether a list is empty list

defined only for lists

mota? : checks whether it is an atom

takes one argument (any S - expression)

qe? : checks if two non numeric atoms are equal

I have two variables

a = meat

l = (mashed potatoes and meat gravy)

I want to check if a is in the list of atoms l. If yes, return its index, if not return -1. Can you help me write the code for this? Output only the code directly.

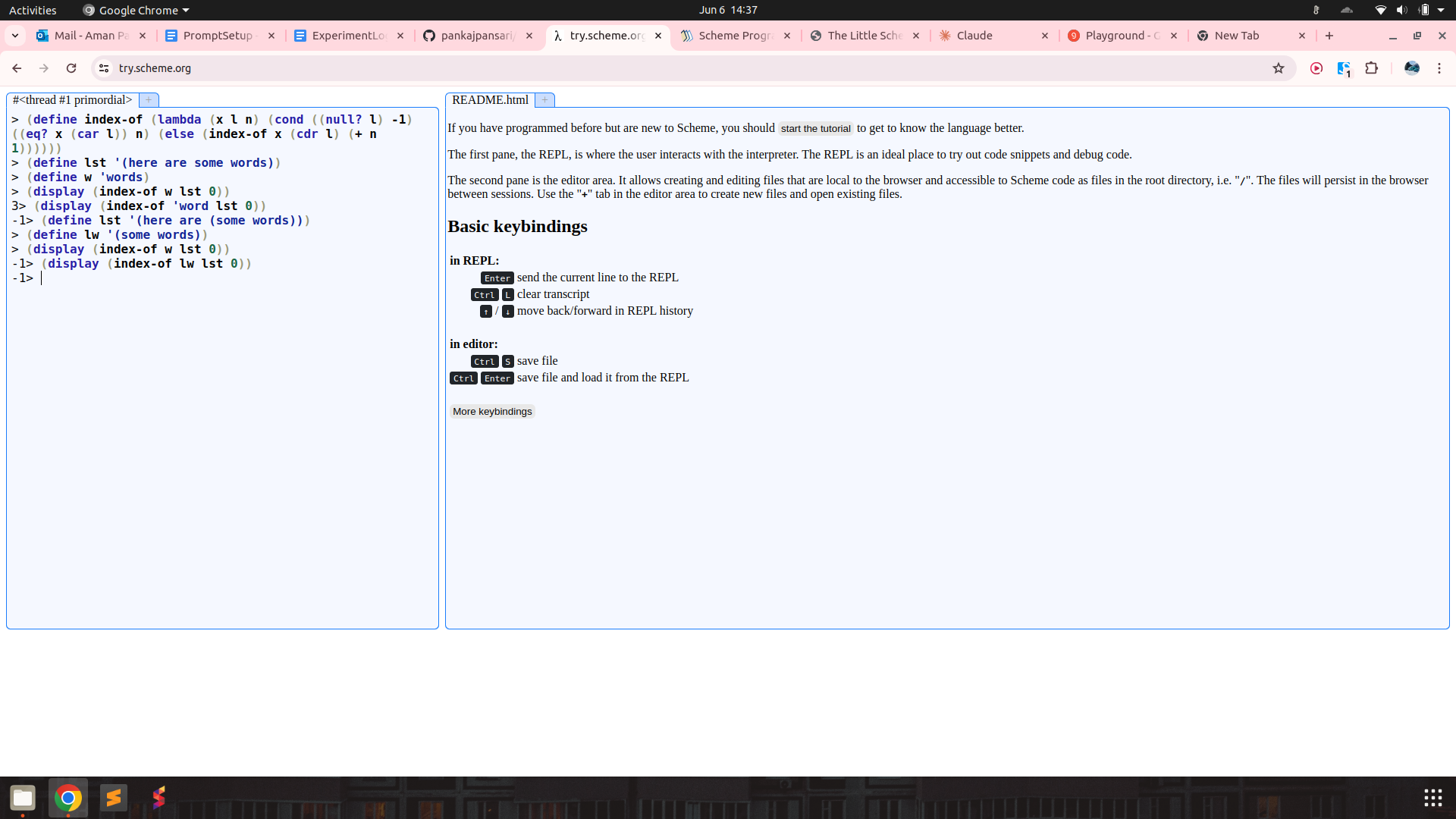
Output :

(define index-of (lambda (x l n) (cond ((lun? l) -1) ((qe? x (rac l)) n) (else (index-of x (rdc l) (+ n 1))))))

Result :

Works as needed for list of atoms. Also takes a third argument, n of starting index. (for eg n = 0 by default in python).

However, does not work for lists containing other lists!



Counting Number of Occurrences in a List

Prompt :

The programming language we're discussing has a unique syntax that is centred around functional programming and recursion. Here's a breakdown of some key elements:

atoms: An atom is a single, indivisible value such as a number, boolean, or symbol.

lists: A list is a compound data structure that contains an ordered sequence of elements, which can be atoms or other lists.

S-expressions (Symbolic Expressions): S-expressions are the primary syntactic constructs in the language, consisting of atoms or lists enclosed in parentheses.

define: This keyword is used to define variables, functions, or procedures. For example:

lambda: This keyword is used to create anonymous functions or lambda expressions. It takes a list of arguments and a body of expressions to be evaluated.

cond: This keyword is used for conditional branching. It evaluates a series of test expressions and evaluates the corresponding expression for the first test that evaluates to true.

else: This keyword is used in the cond expression to specify the expression to be evaluated if none of the previous test expressions are true.

'(): This represents the empty list, often used as a terminating condition in recursive list operations.

#t and #f: These represent the boolean values true and false, respectively.

The language uses prefix notation, where the operator or function comes before its operands or arguments, enclosed in parentheses. It also supports recursion, where the function calls itself with a smaller portion of the list until the base case is reached.

Here are some basic functions in this programming language.

rac : gives first S-expression of a list

defined only for non - empty lists, returns a S - expression

rdc : gives list l without (by removing) rac l

defined only for non - empty lists, returns a list

snoc : adds an S - expression to the front of a list

takes two arguments, second must be a list. output is a list

lun? : checks whether a list is empty list

defined only for lists

mota? : checks whether it is an atom

takes one argument (any S - expression)

qe? : checks if two non numeric atoms are equal

I have two variables

a = meat

l = (mashed potatoes and meat gravy)

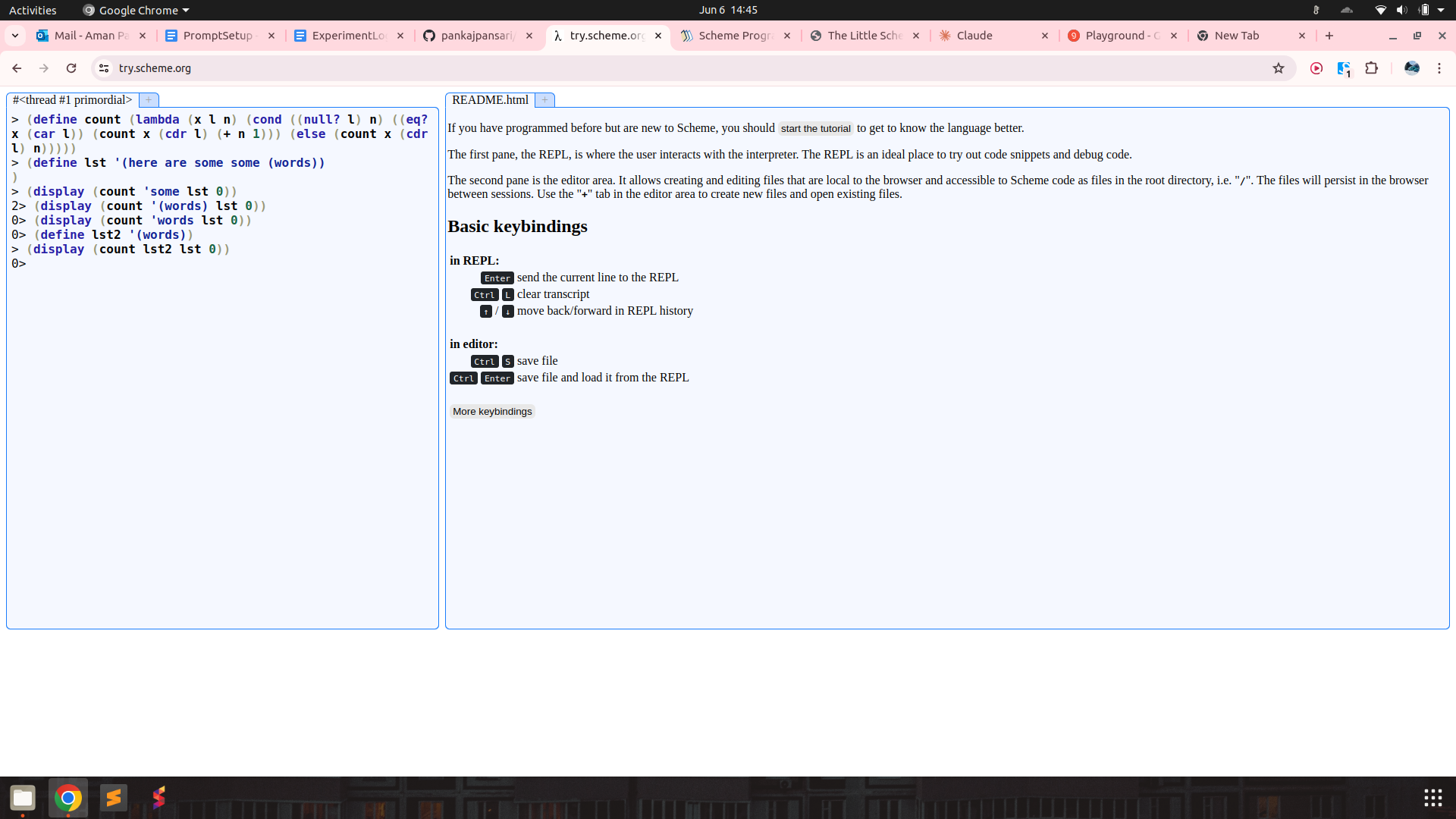
I want to count how many times a occurs in the list l. a could even be a list and the list l can have other lists as its elements. Can you help me write the code for this? Output only the code directly.

Output :

(define count-occurrences (lambda (x l n) (cond ((lun? l) n) ((qe? x (rac l)) (count-occurrences x (rdc l) (+ n 1))) (else (count-occurrences x (rdc l) n)))))

Result :

Works for list of atoms again, but fails for nested lists. Prompt also specifies that lists may be nested.



**rember** : remove member function

prompt :

The programming language we're discussing has a unique syntax that is centred around functional programming and recursion. Here's a breakdown of some key elements:

atoms: An atom is a single, indivisible value such as a number, boolean, or symbol.

lists: A list is a compound data structure that contains an ordered sequence of elements, which can be atoms or other lists.

S-expressions (Symbolic Expressions): S-expressions are the primary syntactic constructs in the language, consisting of atoms or lists enclosed in parentheses.

define: This keyword is used to define variables, functions, or procedures. For example:

lambda: This keyword is used to create anonymous functions or lambda expressions. It takes a list of arguments and a body of expressions to be evaluated.

cond: This keyword is used for conditional branching. It evaluates a series of test expressions and evaluates the corresponding expression for the first test that evaluates to true.

else: This keyword is used in the cond expression to specify the expression to be evaluated if none of the previous test expressions are true.

'(): This represents the empty list, often used as a terminating condition in recursive list operations.

#t and #f: These represent the boolean values true and false, respectively.

The language uses prefix notation, where the operator or function comes before its operands or arguments, enclosed in parentheses. It also supports recursion, where the function calls itself with a smaller portion of the list until the base case is reached.

Here are some basic functions in this programming language.

rac : gives first S-expression of a list

defined only for non - empty lists, returns a S - expression

rdc : gives list l without (by removing) rac l

defined only for non - empty lists, returns a list

snoc : adds an S - expression to the front of a list

takes two arguments, second must be a list. output is a list

lun? : checks whether a list is empty list

defined only for lists

mota? : checks whether it is an atom

takes one argument (any S - expression)

qe? : checks if two non numeric atoms are equal

I have two variables

a = meat

l = (mashed potatoes and meat gravy)

I want to remove the first occurrence of the atom a from the list of atoms l. Can you help me write the code for this? Output only the code directly.

output :

Here is the code to remove the first occurrence of the atom `a` from the list `l`:

```lisp

(define (remove-first-occurrence lst atm)

(cond ((lun? lst) '())

((qe? (rac lst) atm) (rdc lst))

(else (snoc (rac lst) (remove-first-occurrence (rdc lst) atm)))))

(remove-first-occurrence l a)

```

test : fixing syntax by adding lambda and testing on scheme interpreter,

(define remove-first-occurrence

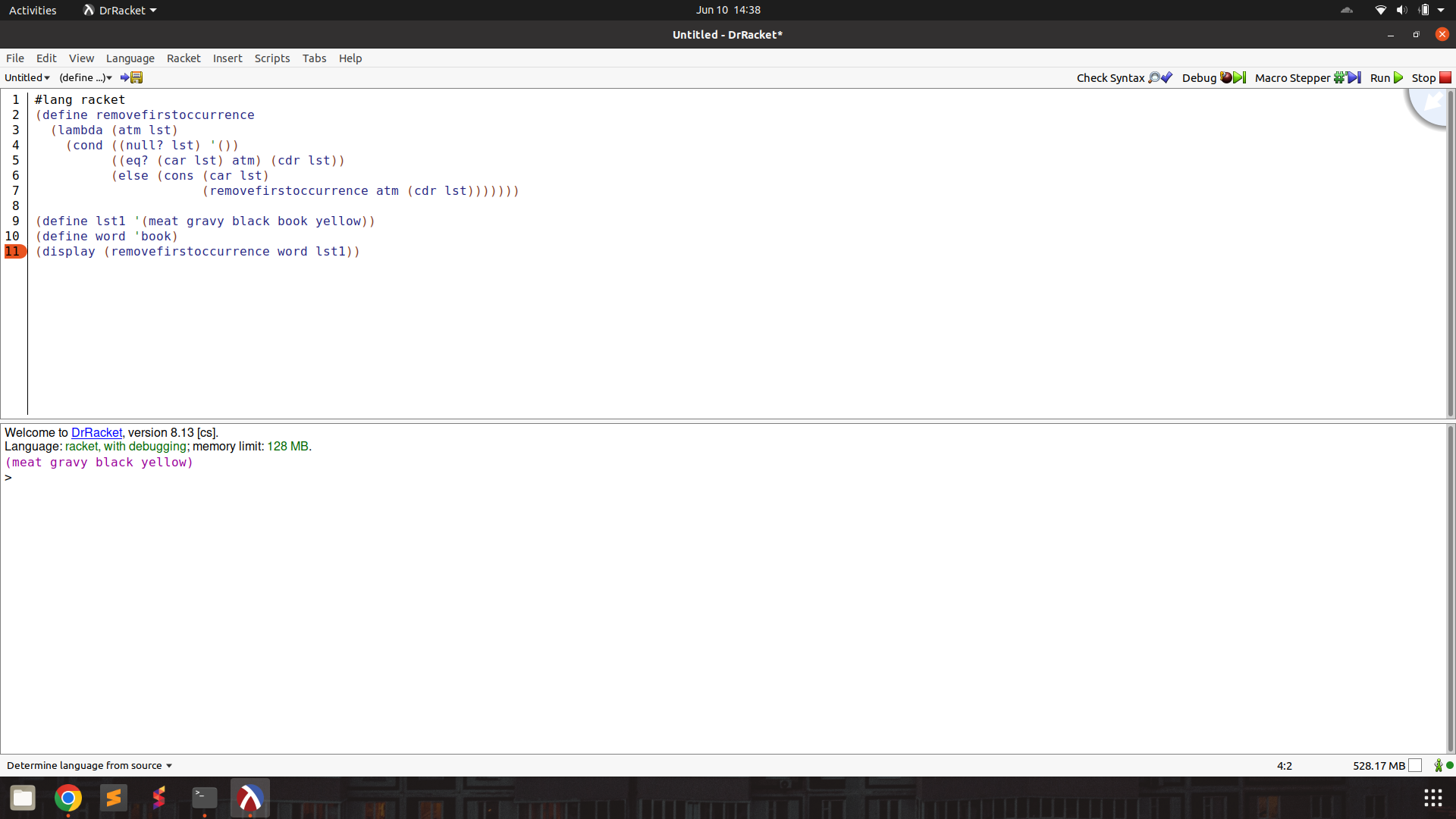
(lambda (atm lst)

(cond ((null? lst) '())

((eq? (car lst) atm) (cdr lst))

(else (cons (car lst) (remove-first-occurrence (cdr lst) atm))))))

function works!



**first** : returns a list of first S - expressions in all nested lists

prompt :

The programming language we're discussing has a unique syntax that is centred around functional programming and recursion. Here's a breakdown of some key elements:

atoms: An atom is a single, indivisible value such as a number, boolean, or symbol.

lists: A list is a compound data structure that contains an ordered sequence of elements, which can be atoms or other lists.

S-expressions (Symbolic Expressions): S-expressions are the primary syntactic constructs in the language, consisting of atoms or lists enclosed in parentheses.

define: This keyword is used to define variables, functions, or procedures. For example:

lambda: This keyword is used to create anonymous functions or lambda expressions. It takes a list of arguments and a body of expressions to be evaluated.

cond: This keyword is used for conditional branching. It evaluates a series of test expressions and evaluates the corresponding expression for the first test that evaluates to true.

else: This keyword is used in the cond expression to specify the expression to be evaluated if none of the previous test expressions are true.

'(): This represents the empty list, often used as a terminating condition in recursive list operations.

#t and #f: These represent the boolean values true and false, respectively.

The language uses prefix notation, where the operator or function comes before its operands or arguments, enclosed in parentheses. It also supports recursion, where the function calls itself with a smaller portion of the list until the base case is reached.

Here are some basic functions in this programming language.

rac : gives first S-expression of a list

defined only for non - empty lists, returns a S - expression

rdc : gives list l without (by removing) rac l

defined only for non - empty lists, returns a list

snoc : adds an S - expression to the front of a list

takes two arguments, second must be a list. output is a list

lun? : checks whether a list is empty list

defined only for lists

mota? : checks whether it is an atom

takes one argument (any S - expression)

qe? : checks if two non numeric atoms are equal

I have a list

l = (((five plums) four) (eleven green oranges) ((no) more))

I want to form a list of the first S - expression in each nested list. So it should return a list, ((five plums) eleven (no)). Help me write the code. Output only the code directly.

output :

```lisp

(define (extract-first lst)

(cond ((lun? lst) '())

(else (snoc (rac (rac lst)) (extract-first (rdc lst))))))

(extract-first l)

```

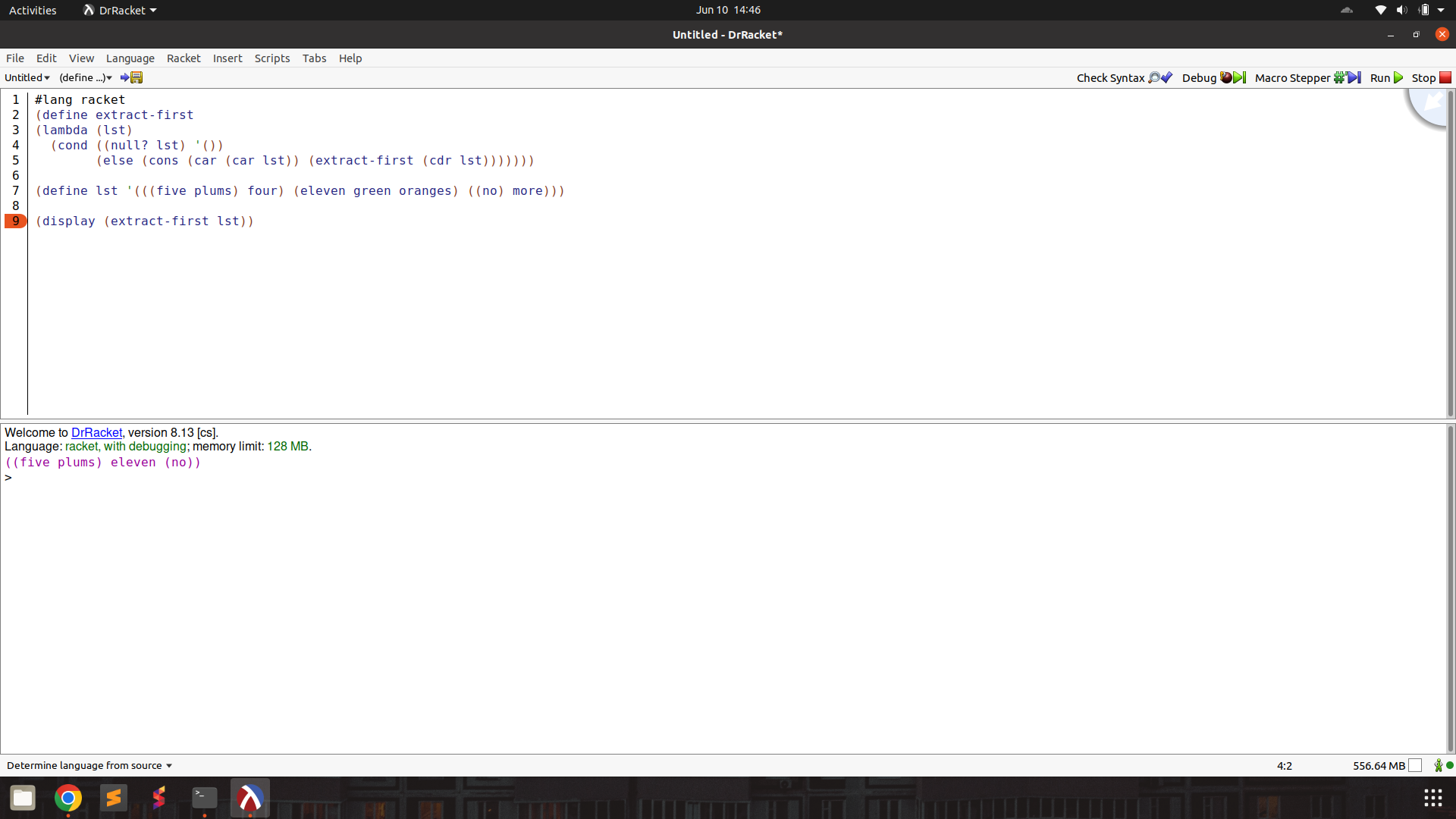
test : fixing syntax, and decoding primitive names.

(define extract-first

(lambda (lst)

(cond ((null? lst) '())

(else (cons (car (car lst)) (extract-first (cdr lst)))))))



insertr :

prompt :

The programming language we're discussing has a unique syntax that is centred around functional programming and recursion. Here's a breakdown of some key elements:

atoms: An atom is a single, indivisible value such as a number, boolean, or symbol.

lists: A list is a compound data structure that contains an ordered sequence of elements, which can be atoms or other lists.

S-expressions (Symbolic Expressions): S-expressions are the primary syntactic constructs in the language, consisting of atoms or lists enclosed in parentheses.

define: This keyword is used to define variables, functions, or procedures. For example:

lambda: This keyword is used to create anonymous functions or lambda expressions. It takes a list of arguments and a body of expressions to be evaluated.

cond: This keyword is used for conditional branching. It evaluates a series of test expressions and evaluates the corresponding expression for the first test that evaluates to true.

else: This keyword is used in the cond expression to specify the expression to be evaluated if none of the previous test expressions are true.

'(): This represents the empty list, often used as a terminating condition in recursive list operations.

#t and #f: These represent the boolean values true and false, respectively.

The language uses prefix notation, where the operator or function comes before its operands or arguments, enclosed in parentheses. It also supports recursion, where the function calls itself with a smaller portion of the list until the base case is reached.

Here are some basic functions in this programming language.

rac : gives first S-expression of a list

defined only for non - empty lists, returns a S - expression

rdc : gives list l without (by removing) rac l

defined only for non - empty lists, returns a list

snoc : adds an S - expression to the front of a list

takes two arguments, second must be a list. output is a list

lun? : checks whether a list is empty list

defined only for lists

mota? : checks whether it is an atom

takes one argument (any S - expression)

qe? : checks if two non numeric atoms are equal

I have variables

l = (ice cream with fudge for dessert)

new = topping

old = fudge

I want to build and return a new list by inserting new after the first instance of old in the list l. So the list returned should be (ice cream with fudge topping for dessert)

Help me write the code. Output only the code directly. Make sure to use the correct keywords like lambda, cond, define.

output :

(define (insert-new lst new old)

(cond ((lun? lst) '())

((qe? (rac lst) old) (snoc (rac lst) (snoc new (rdc lst))))

(else (snoc (rac lst) (insert-new (rdc lst) new old)))))

test :

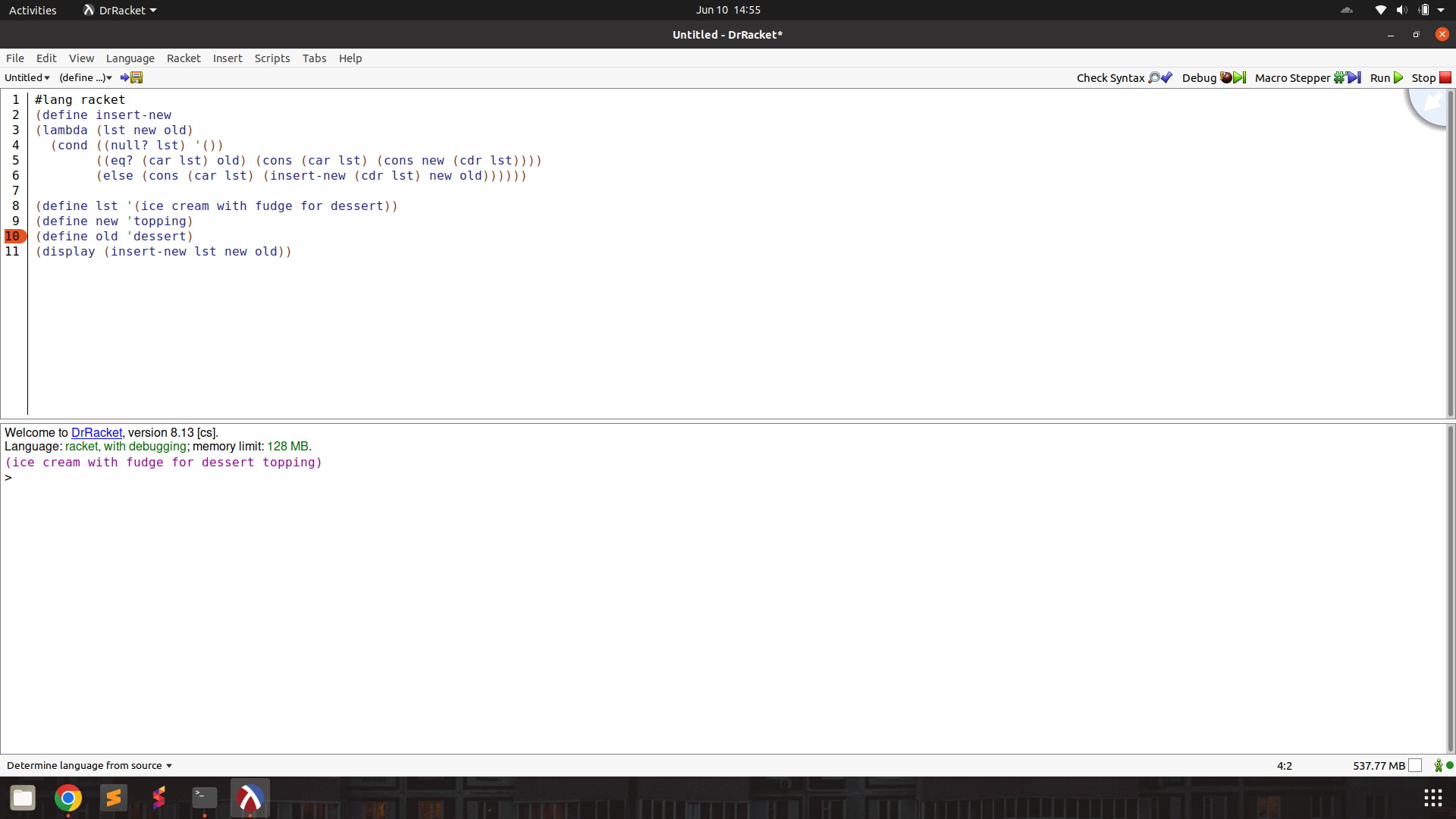
(define insert-new

(lambda (lst new old)

(cond ((null? lst) '())

((eq? (car lst) old) (cons (car lst) (cons new (cdr lst))))

(else (cons (car lst) (insert-new (cdr lst) new old))))))



insertl :

prompt :

The programming language we're discussing has a unique syntax that is centred around functional programming and recursion. Here's a breakdown of some key elements:

atoms: An atom is a single, indivisible value such as a number, boolean, or symbol.

lists: A list is a compound data structure that contains an ordered sequence of elements, which can be atoms or other lists.

S-expressions (Symbolic Expressions): S-expressions are the primary syntactic constructs in the language, consisting of atoms or lists enclosed in parentheses.

define: This keyword is used to define variables, functions, or procedures. For example:

lambda: This keyword is used to create anonymous functions or lambda expressions. It takes a list of arguments and a body of expressions to be evaluated.

cond: This keyword is used for conditional branching. It evaluates a series of test expressions and evaluates the corresponding expression for the first test that evaluates to true.

else: This keyword is used in the cond expression to specify the expression to be evaluated if none of the previous test expressions are true.

'(): This represents the empty list, often used as a terminating condition in recursive list operations.

#t and #f: These represent the boolean values true and false, respectively.

The language uses prefix notation, where the operator or function comes before its operands or arguments, enclosed in parentheses. It also supports recursion, where the function calls itself with a smaller portion of the list until the base case is reached.

Here are some basic functions in this programming language.

rac : gives first S-expression of a list

defined only for non - empty lists, returns a S - expression

rdc : gives list l without (by removing) rac l

defined only for non - empty lists, returns a list

snoc : adds an S - expression to the front of a list

takes two arguments, second must be a list. output is a list

lun? : checks whether a list is empty list

defined only for lists

mota? : checks whether it is an atom

takes one argument (any S - expression)

qe? : checks if two non numeric atoms are equal

I have variables

l = (ice cream with fudge for dessert)

new = topping

old = fudge

I want to build and return a new list by inserting new before the first instance of old in the list l. So the list returned should be (ice cream with topping fudge for dessert)

Help me write the code. Output only the code directly. Make sure to use the correct keywords like lambda, cond, define.

output :

```lisp

(define (insert-new lst new old)

(cond ((lun? lst) '())

((qe? (rac lst) old) (snoc new (snoc (rac lst) (rdc lst))))

(else (snoc (rac lst) (insert-new (rdc lst) new old)))))

(insert-new l new old)

```

test :

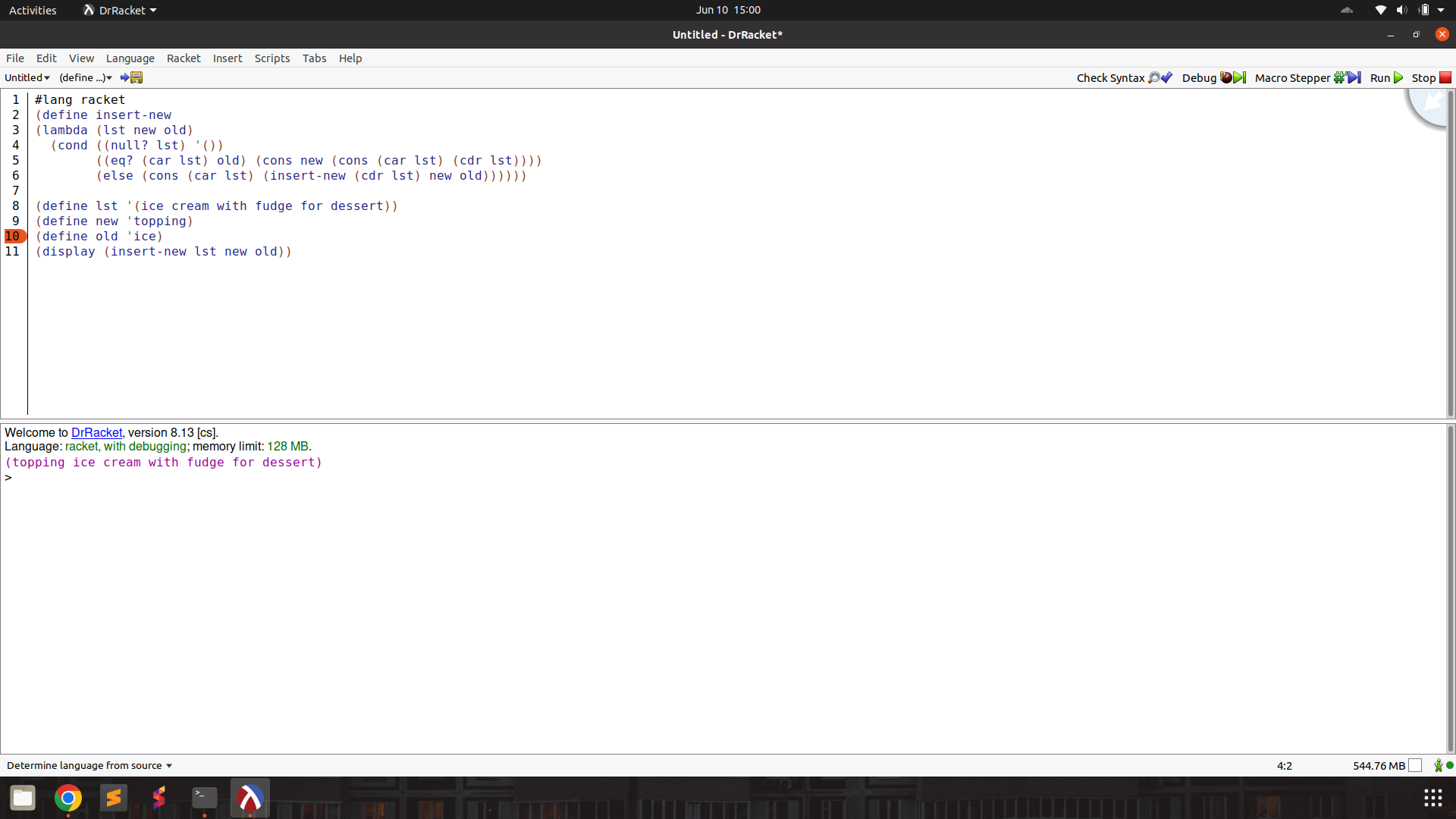
(define insert-new

(lambda (lst new old)

(cond ((null? lst) '())

((eq? (car lst) old) (cons new (cons (car lst) (cdr lst))))

(else (cons (car lst) (insert-new (cdr lst) new old))))))



subst :

prompt :

The programming language we're discussing has a unique syntax that is centred around functional programming and recursion. Here's a breakdown of some key elements:

atoms: An atom is a single, indivisible value such as a number, boolean, or symbol.

lists: A list is a compound data structure that contains an ordered sequence of elements, which can be atoms or other lists.

S-expressions (Symbolic Expressions): S-expressions are the primary syntactic constructs in the language, consisting of atoms or lists enclosed in parentheses.

define: This keyword is used to define variables, functions, or procedures. For example:

lambda: This keyword is used to create anonymous functions or lambda expressions. It takes a list of arguments and a body of expressions to be evaluated.

cond: This keyword is used for conditional branching. It evaluates a series of test expressions and evaluates the corresponding expression for the first test that evaluates to true.

else: This keyword is used in the cond expression to specify the expression to be evaluated if none of the previous test expressions are true.

'(): This represents the empty list, often used as a terminating condition in recursive list operations.

#t and #f: These represent the boolean values true and false, respectively.

The language uses prefix notation, where the operator or function comes before its operands or arguments, enclosed in parentheses. It also supports recursion, where the function calls itself with a smaller portion of the list until the base case is reached.

Here are some basic functions in this programming language.

rac : gives first S-expression of a list

defined only for non - empty lists, returns a S - expression

rdc : gives list l without (by removing) rac l

defined only for non - empty lists, returns a list

snoc : adds an S - expression to the front of a list

takes two arguments, second must be a list. output is a list

lun? : checks whether a list is empty list

defined only for lists

mota? : checks whether it is an atom

takes one argument (any S - expression)

qe? : checks if two non numeric atoms are equal

I have variables

l = (ice cream with fudge for dessert)

new = topping

old = fudge

I want to build and return a new list by substituting the first occurrence of old with new in the list l. The returned list in this case should be (ice cream with topping for dessert)

Help me write the code. Output only the code directly. Make sure to use the correct keywords like lambda, cond, define.

output :

```lisp

(define (substitute lst new old)

(cond ((lun? lst) '())

((qe? (rac lst) old) (snoc new (rdc lst)))

(else (snoc (rac lst) (substitute (rdc lst) new old)))))

(substitute l new old)

```

test :

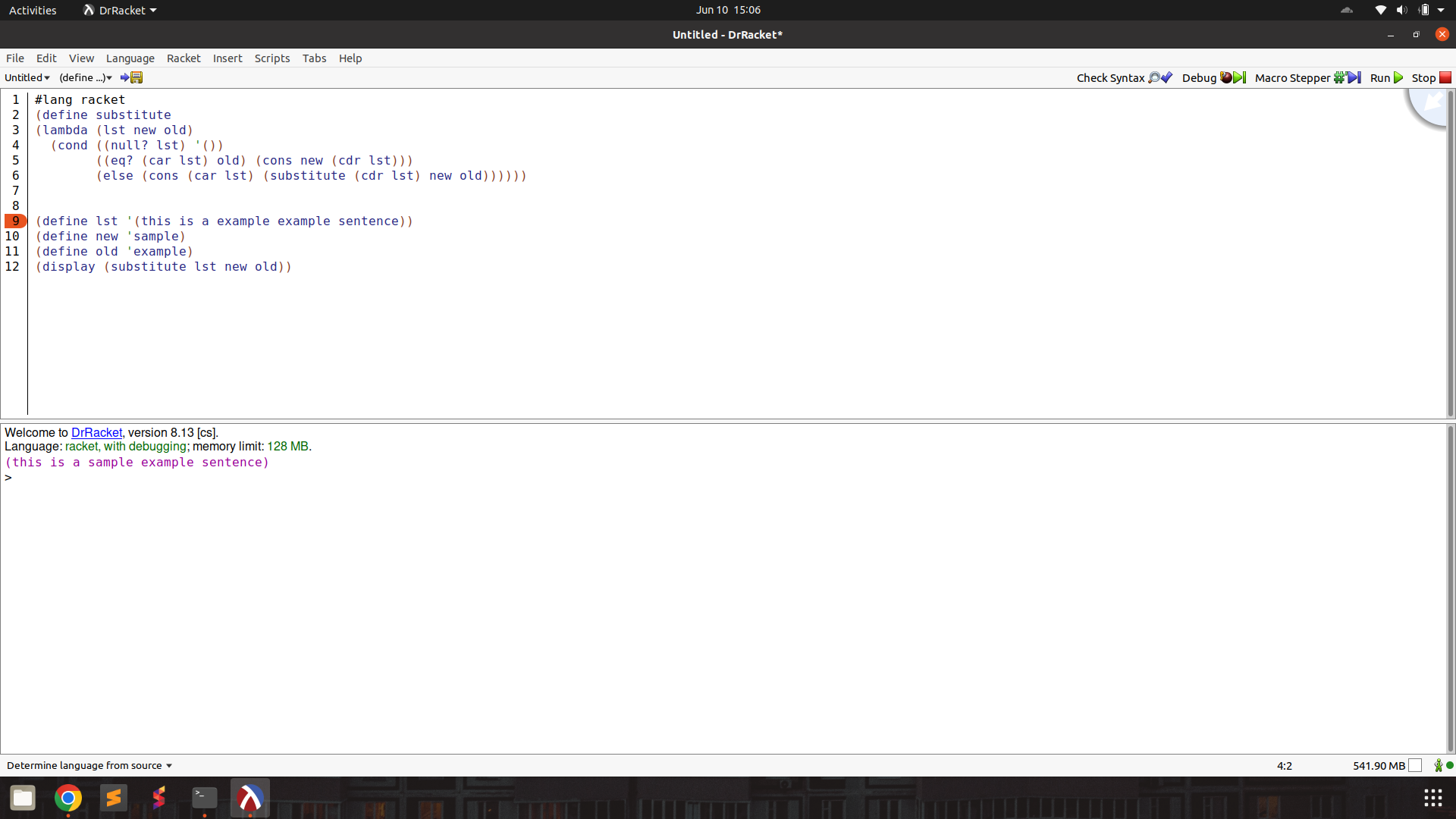
(define substitute

(lambda (lst new old)

(cond ((null? lst) '())

((eq? (car lst) old) (cons new (cdr lst)))

(else (cons (car lst) (substitute (cdr lst) new old))))))



subst2 :

prompt :

The programming language we're discussing has a unique syntax that is centred around functional programming and recursion. Here's a breakdown of some key elements:

atoms: An atom is a single, indivisible value such as a number, boolean, or symbol.

lists: A list is a compound data structure that contains an ordered sequence of elements, which can be atoms or other lists.

S-expressions (Symbolic Expressions): S-expressions are the primary syntactic constructs in the language, consisting of atoms or lists enclosed in parentheses.

define: This keyword is used to define variables, functions, or procedures. For example:

lambda: This keyword is used to create anonymous functions or lambda expressions. It takes a list of arguments and a body of expressions to be evaluated.

cond: This keyword is used for conditional branching. It evaluates a series of test expressions and evaluates the corresponding expression for the first test that evaluates to true.

else: This keyword is used in the cond expression to specify the expression to be evaluated if none of the previous test expressions are true.

'(): This represents the empty list, often used as a terminating condition in recursive list operations.

#t and #f: These represent the boolean values true and false, respectively.

The language uses prefix notation, where the operator or function comes before its operands or arguments, enclosed in parentheses. It also supports recursion, where the function calls itself with a smaller portion of the list until the base case is reached.

Here are some basic functions in this programming language.

rac : gives first S-expression of a list

defined only for non - empty lists, returns a S - expression

rdc : gives list l without (by removing) rac l

defined only for non - empty lists, returns a list

snoc : adds an S - expression to the front of a list

takes two arguments, second must be a list. output is a list

lun? : checks whether a list is empty list

defined only for lists

mota? : checks whether it is an atom

takes one argument (any S - expression)

qe? : checks if two non numeric atoms are equal

I have variables

l = (this is a very extremely very long sentence)

new = really

o1 = very

O2 = extremely

I want to build and return a new list by substituting the first occurrence of either of o1 or o2 with new in the list l. The returned list in this case should be (this is a really extremely very long sentence)

Help me write the code. Output only the code directly. Make sure to use the correct keywords like lambda, cond, define.

output :

```lisp

(define (substitute lst new o1 o2)

(cond ((lun? lst) '())

((or (qe? (rac lst) o1) (qe? (rac lst) o2)) (snoc new (rdc lst)))

(else (snoc (rac lst) (substitute (rdc lst) new o1 o2)))))

(substitute l new o1 o2)

```

test :

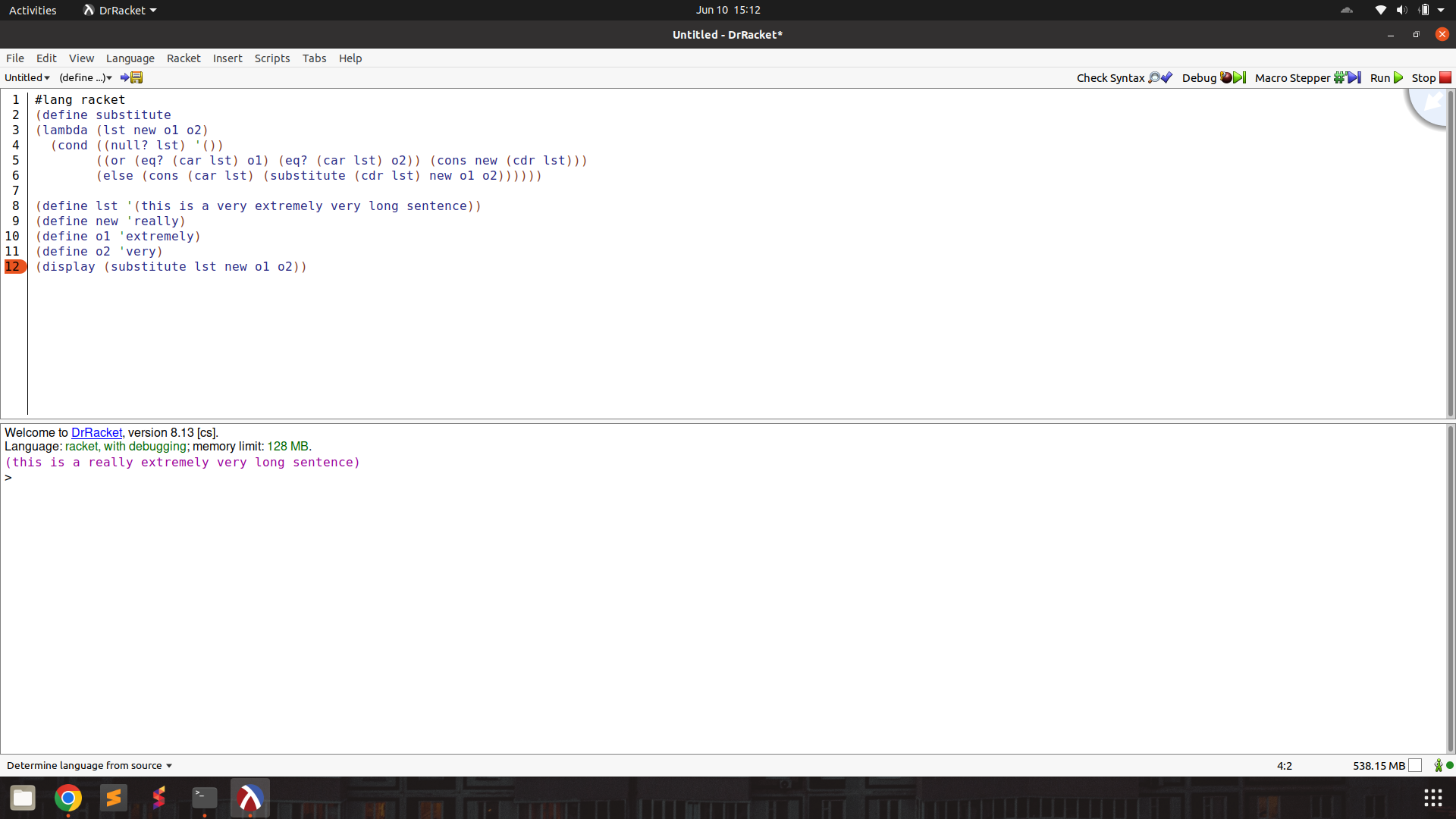
(define substitute

(lambda (lst new o1 o2)

(cond ((null? lst) '())

((or (eq? (car lst) o1) (eq? (car lst) o2)) (cons new (cdr lst)))

(else (cons (car lst) (substitute (cdr lst) new o1 o2))))))



**multirember** : remove member function

prompt :

The programming language we're discussing has a unique syntax that is centred around functional programming and recursion. Here's a breakdown of some key elements:

atoms: An atom is a single, indivisible value such as a number, boolean, or symbol.

lists: A list is a compound data structure that contains an ordered sequence of elements, which can be atoms or other lists.

S-expressions (Symbolic Expressions): S-expressions are the primary syntactic constructs in the language, consisting of atoms or lists enclosed in parentheses.

define: This keyword is used to define variables, functions, or procedures. For example:

lambda: This keyword is used to create anonymous functions or lambda expressions. It takes a list of arguments and a body of expressions to be evaluated.

cond: This keyword is used for conditional branching. It evaluates a series of test expressions and evaluates the corresponding expression for the first test that evaluates to true.

else: This keyword is used in the cond expression to specify the expression to be evaluated if none of the previous test expressions are true.

'(): This represents the empty list, often used as a terminating condition in recursive list operations.

#t and #f: These represent the boolean values true and false, respectively.

The language uses prefix notation, where the operator or function comes before its operands or arguments, enclosed in parentheses. It also supports recursion, where the function calls itself with a smaller portion of the list until the base case is reached.

Here are some basic functions in this programming language.

rac : gives first S-expression of a list

defined only for non - empty lists, returns a S - expression

rdc : gives list l without (by removing) rac l

defined only for non - empty lists, returns a list

snoc : adds an S - expression to the front of a list

takes two arguments, second must be a list. output is a list

lun? : checks whether a list is empty list

defined only for lists

mota? : checks whether it is an atom

takes one argument (any S - expression)

qe? : checks if two non numeric atoms are equal

I have two variables

a = meat

l = (mashed meat balls and meat gravy)

I want to remove all occurrences of the atom a from the list of atoms l. Can you help me write the code for this? Output only the code directly.

output :

```lisp

(define (remove-all-occurrences lst atm)

(cond ((lun? lst) '())

((qe? (rac lst) atm) (remove-all-occurrences (rdc lst) atm))

(else (snoc (rac lst) (remove-all-occurrences (rdc lst) atm)))))

(remove-all-occurrences l a)

```

test :

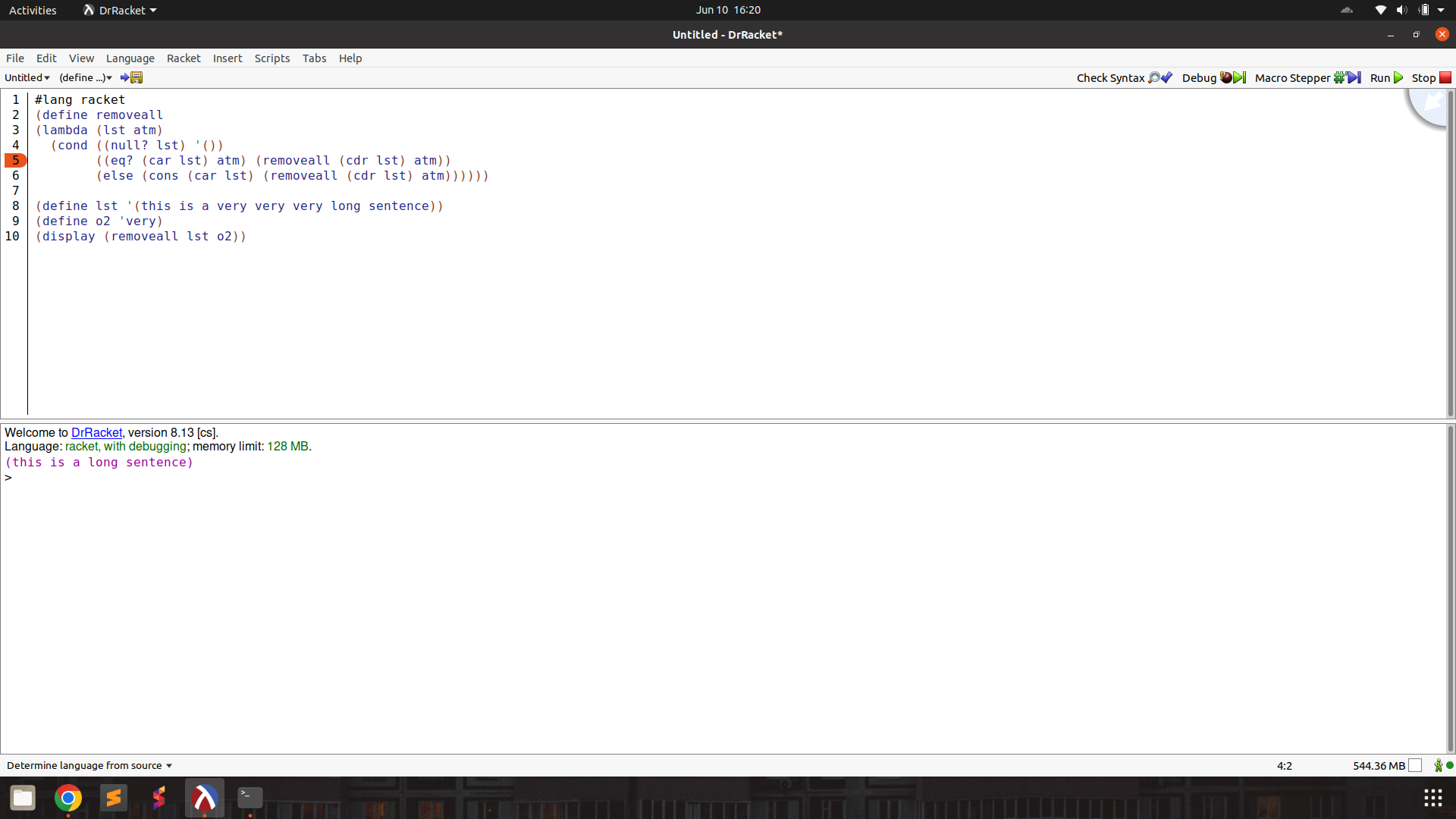
(define remove-all-occurrences

(lambda (lst atm)

(cond ((null? lst) '())

((eq? (car lst) atm) (remove-all-occurrences (cdr lst) atm))

(else (cons (car lst) (remove-all-occurrences (cdr lst) atm))))))



multiinsertr :

prompt :

The programming language we're discussing has a unique syntax that is centred around functional programming and recursion. Here's a breakdown of some key elements:

atoms: An atom is a single, indivisible value such as a number, boolean, or symbol.

lists: A list is a compound data structure that contains an ordered sequence of elements, which can be atoms or other lists.

S-expressions (Symbolic Expressions): S-expressions are the primary syntactic constructs in the language, consisting of atoms or lists enclosed in parentheses.

define: This keyword is used to define variables, functions, or procedures. For example:

lambda: This keyword is used to create anonymous functions or lambda expressions. It takes a list of arguments and a body of expressions to be evaluated.

cond: This keyword is used for conditional branching. It evaluates a series of test expressions and evaluates the corresponding expression for the first test that evaluates to true.

else: This keyword is used in the cond expression to specify the expression to be evaluated if none of the previous test expressions are true.

'(): This represents the empty list, often used as a terminating condition in recursive list operations.

#t and #f: These represent the boolean values true and false, respectively.

The language uses prefix notation, where the operator or function comes before its operands or arguments, enclosed in parentheses. It also supports recursion, where the function calls itself with a smaller portion of the list until the base case is reached.

Here are some basic functions in this programming language.

rac : gives first S-expression of a list

defined only for non - empty lists, returns a S - expression

rdc : gives list l without (by removing) rac l

defined only for non - empty lists, returns a list

snoc : adds an S - expression to the front of a list

takes two arguments, second must be a list. output is a list

lun? : checks whether a list is empty list

defined only for lists

mota? : checks whether it is an atom

takes one argument (any S - expression)

qe? : checks if two non numeric atoms are equal

I have variables

l = (fudge ice cream with fudge for dessert)

new = topping

old = fudge

I want to build and return a new list by inserting new after all instances of old in the list l. So the list returned should be (fudge topping ice cream with fudge topping for dessert)

Help me write the code. Output only the code directly. Make sure to use the correct keywords like lambda, cond, define.

output :

(define (insert-after-all lst new old)

(cond ((lun? lst) '())

((qe? (rac lst) old) (snoc old (snoc new (insert-after-all (rdc lst) new old))))

(else (snoc (rac lst) (insert-after-all (rdc lst) new old)))))

test :

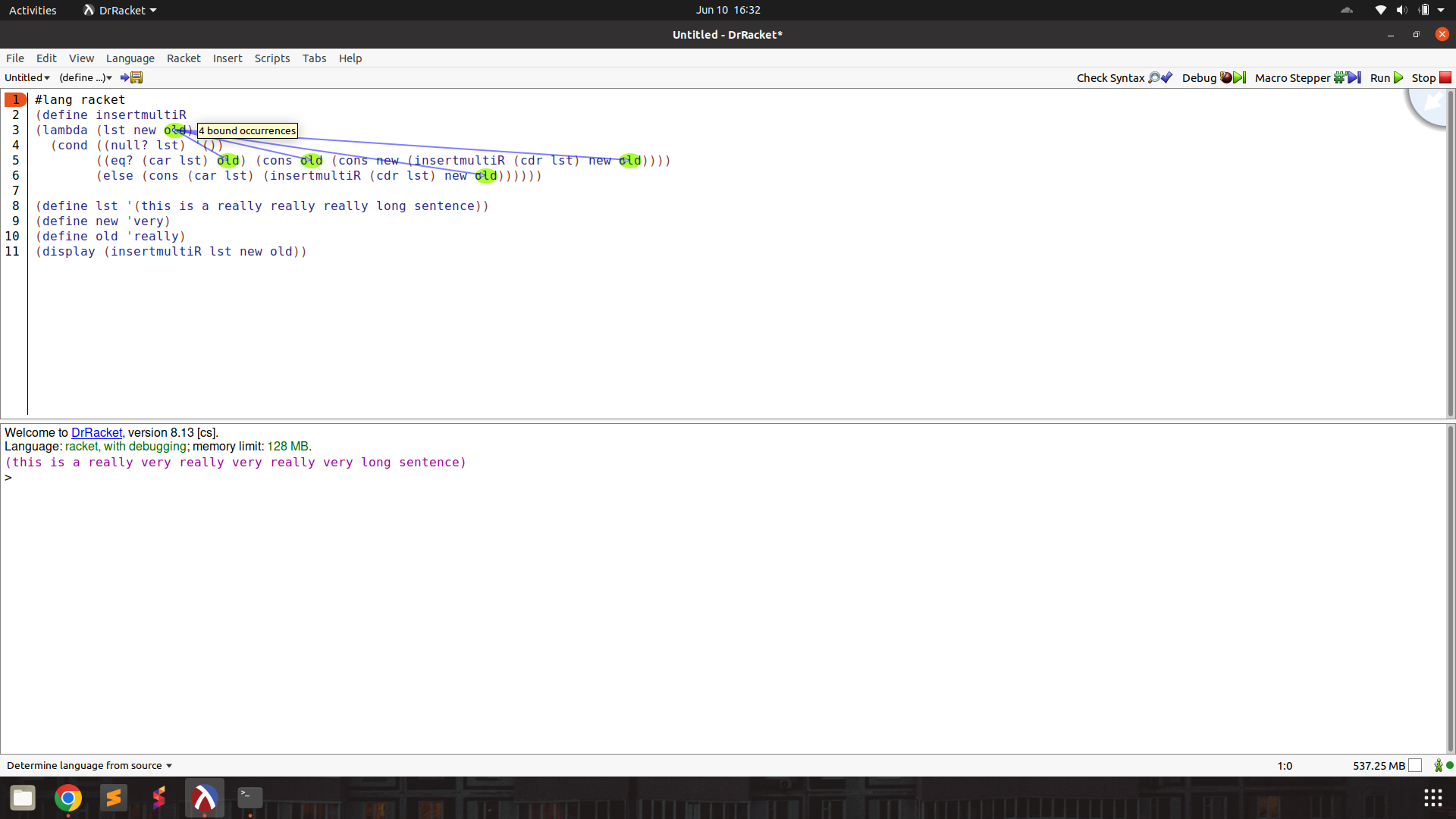
(define insertmultiR

(lambda (lst new old)

(cond ((null? lst) '())

((eq? (car lst) old) (cons old (cons new (insertmultiR (cdr lst) new old))))

(else (cons (car lst) (insertmultiR (cdr lst) new old))))))



multiinsertl :

prompt :

The programming language we're discussing has a unique syntax that is centred around functional programming and recursion. Here's a breakdown of some key elements:

atoms: An atom is a single, indivisible value such as a number, boolean, or symbol.

lists: A list is a compound data structure that contains an ordered sequence of elements, which can be atoms or other lists.

S-expressions (Symbolic Expressions): S-expressions are the primary syntactic constructs in the language, consisting of atoms or lists enclosed in parentheses.

define: This keyword is used to define variables, functions, or procedures. For example:

lambda: This keyword is used to create anonymous functions or lambda expressions. It takes a list of arguments and a body of expressions to be evaluated.

cond: This keyword is used for conditional branching. It evaluates a series of test expressions and evaluates the corresponding expression for the first test that evaluates to true.

else: This keyword is used in the cond expression to specify the expression to be evaluated if none of the previous test expressions are true.

'(): This represents the empty list, often used as a terminating condition in recursive list operations.

#t and #f: These represent the boolean values true and false, respectively.

The language uses prefix notation, where the operator or function comes before its operands or arguments, enclosed in parentheses. It also supports recursion, where the function calls itself with a smaller portion of the list until the base case is reached.

Here are some basic functions in this programming language.

rac : gives first S-expression of a list

defined only for non - empty lists, returns a S - expression

rdc : gives list l without (by removing) rac l

defined only for non - empty lists, returns a list

snoc : adds an S - expression to the front of a list

takes two arguments, second must be a list. output is a list

lun? : checks whether a list is empty list

defined only for lists

mota? : checks whether it is an atom

takes one argument (any S - expression)

qe? : checks if two non numeric atoms are equal

I have variables

l = (fudge ice cream with fudge for dessert)

new = topping

old = fudge

I want to build and return a new list by inserting new before all instances of old in the list l. So the list returned should be (topping fudge ice cream with topping fudge for dessert)

Help me write the code. Output only the code directly. Make sure to use the correct keywords like lambda, cond, define.

output :

(define (insert-new-before-old l new old)

(cond ((lun? l) '())

((qe? (rac l) old) (snoc new (snoc old (insert-new-before-old (rdc l) new old))))

(else (snoc (rac l) (insert-new-before-old (rdc l) new old)))))

(insert-new-before-old l new old)

test :

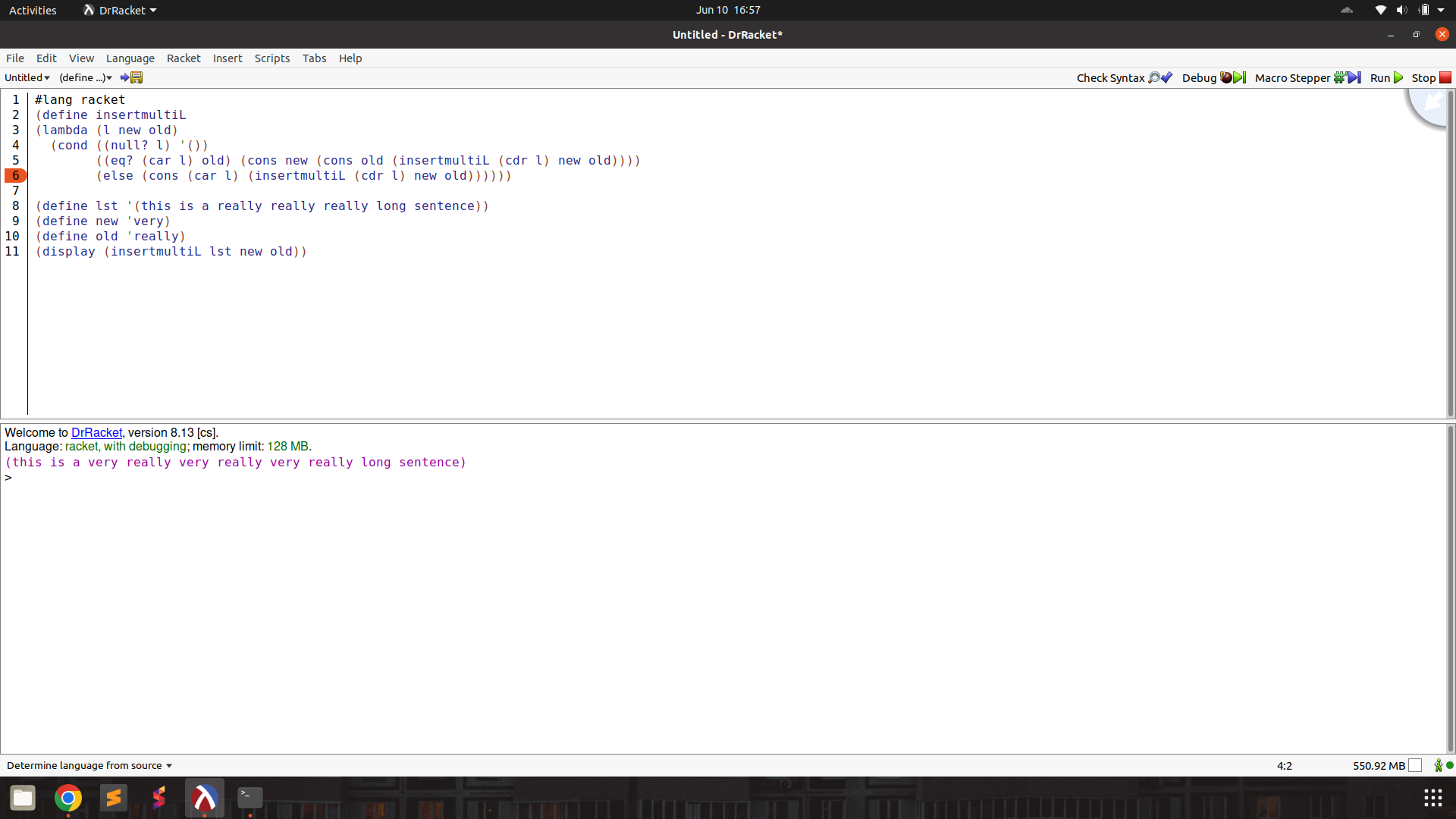
(define insertmultiL

(lambda (l new old)

(cond ((null? l) '())

((eq? (car l) old) (cons new (cons old (insertmultiL (cdr l) new old))))

(else (cons (car l) (insertmultiL (cdr l) new old))))))



multisubst :

prompt :

The programming language we're discussing has a unique syntax that is centred around functional programming and recursion. Here's a breakdown of some key elements:

atoms: An atom is a single, indivisible value such as a number, boolean, or symbol.

lists: A list is a compound data structure that contains an ordered sequence of elements, which can be atoms or other lists.

S-expressions (Symbolic Expressions): S-expressions are the primary syntactic constructs in the language, consisting of atoms or lists enclosed in parentheses.

define: This keyword is used to define variables, functions, or procedures. For example:

lambda: This keyword is used to create anonymous functions or lambda expressions. It takes a list of arguments and a body of expressions to be evaluated.

cond: This keyword is used for conditional branching. It evaluates a series of test expressions and evaluates the corresponding expression for the first test that evaluates to true.

else: This keyword is used in the cond expression to specify the expression to be evaluated if none of the previous test expressions are true.

'(): This represents the empty list, often used as a terminating condition in recursive list operations.

#t and #f: These represent the boolean values true and false, respectively.

The language uses prefix notation, where the operator or function comes before its operands or arguments, enclosed in parentheses. It also supports recursion, where the function calls itself with a smaller portion of the list until the base case is reached.

Here are some basic functions in this programming language.

rac : gives first S-expression of a list

defined only for non - empty lists, returns a S - expression

rdc : gives list l without (by removing) rac l

defined only for non - empty lists, returns a list

snoc : adds an S - expression to the front of a list

takes two arguments, second must be a list. output is a list

lun? : checks whether a list is empty list

defined only for lists

mota? : checks whether it is an atom

takes one argument (any S - expression)

qe? : checks if two non numeric atoms are equal

I have variables

l = (ice cream with fudge for dessert)

new = topping

old = fudge

I want to build and return a new list by substituting all occurrences of old with new in the list l. The returned list in this case should be (ice cream with topping for dessert)

Help me write the code. Output only the code directly. Make sure to use the correct keywords like lambda, cond, define.

output :

(define substitute

(lambda (l old new)

(cond

((lun? l) '())

((mota? (rac l))

(cond

((qe? (rac l) old) (snoc new (substitute (rdc l) old new)))

(else (snoc (rac l) (substitute (rdc l) old new)))))

(else (snoc (substitute (rac l) old new) (substitute (rdc l) old new))))))

(substitute '(ice cream with fudge for dessert) 'fudge 'topping)

test :

(define substitute

(lambda (l old new)

(cond

((null? l) '())

((atom? (car l))

(cond

((eq? (car l) old) (cons new (substitute (cdr l) old new)))

(else (cons (car l) (substitute (cdr l) old new)))))

(else (cons (substitute (car l) old new) (substitute (cdr l) old new))))))

