And (twice -1)-2Can you define twice?Sure. It's (lambda (x) (* 2 x))What's (map twice '(4 -1))(8 -2)What's (map '(4))(8)What should (map '()) be?() sounds like a good idea.map is part of the language, but here's a definition anyway. (define map (lambda (f Ist) (cond ((mul? Ist) '()) (else (cons (f (car Ist)) (map f (cdr Ist)))))))That's old news, isn't it?What does (map twice (map twice x)) mean?Multiply each element of x by 4.Can you express this formally.Sure. How about (map (lambda (x) (* 4 x)) x)Does this work?Sure. Remember the distinction between free and bound variables. The x inside the lambda expression is bound.Does this work?No. It's too different than the original expression. It would be nice to use twice in the simplification.Does this work?No! (twice twice) produces an error message.How come?Twice expects a number, not a procedure argument.Here's a solution (map twice (map twice x)) == (map (lambda (x) (twice (twice x))) x)Can you see a difference between the two equivalent expressions?The first traverses two lists (x, and the result of the first map application). The second expression traverses only one list.	What is (twice 4)	8
Can you define twice?Sure. It's (Iambda (x) (* 2 x))What's (map twice '(4 -1))(8 -2)What's (map '(4))(8)What's map '(4))(8)What should (map '()) be?() sounds like a good idea.map is part of the language, but here's a definition anyway. (define map (lambda (f Ist) (cond ((mul1? lst) '()) (else (cons (f (car lst)) (map f (cdr lst)))))))That's old news, isn't it?What does (map twice (map twice x)) mean?Multiply each element of x by 4.Can you express this formally.Sure. How about (map (lambda (x) (* 4 x)) x)Does this work?Sure. Remember the distinction between free and bound variables. The x inside the lambda expression is bound.Does (map (lambda (x) (* 4 x)) x) make happy?No. It's too different than the original expression. It would be nice to use twice in the simplification.Does this work?No! (twice twice) produces an error message.How come?Twice expects a number, not a procedure argument.Here's a solution (map (lambda (x) (twice (twice x))) x)Much better.Can you see a difference between the two equivalent expressions?The first traverses two lists (x, and the result of the first map application). The second expression traverses only one list.	And (twice -1)	-2
Image: series of the series	Can you define twice?	Sure. It's
What's (map twice '(4 - 1))(8 - 2)What's (map '(4))(8)What should (map '()) be?() sounds like a good idea.map is part of the language, but here's a definition anyway. (define map ((auhda (f lst) (cond ((aul? lst) '()) (else (cons (f (car lst))(map f (cdr lst)))))))That's old news, isn't it?What does (map twice (map twice x)) mean?Multiply each element of x by 4.Can you express this formally.Sure. How about (map (lambda (x) (* 4 x)) x)Does this work?Sure. Remember the distinction between free and bound variables. The x inside the lambda expression is bound.Does this work?No. It's too different than the original expression. It would be nice to use twice in the simplification.Does this work?No! (twice twice) produces an error message.How come?Twice expects a number, not a procedure argument.Here's a solution (map (lambda (x) (twice (twice x))) x)Much better.Can you see a difference between the two equivalent expressions?The first traverses two lists (x, and the result of the first map application). The second expression traverses only one list.		(lambda (x) (* 2 x))
What's (map '(4))(8)What should (map '()) be?() sounds like a good idea.map is part of the language, but here's a definition anyway. (define map (lambda (f lst) (cond ((null? lst) '()) (else (cons (f (car lst)) (map f (cdr lst)))))))That's old news, isn't it?What does (map twice (map twice x)) mean?Multiply each element of x by 4.Can you express this formally.Sure. How about (map (lambda (x) (* 4 x)) x)Does this work?Sure. Remember the distinction between free and bound variables. The x inside the lambda expression is bound.Does this work?No. It's too different than the original expression. It would be nice to use twice in the simplification.Does this work?No! (twice twice) produces an error message.How come?Twice expects a number, not a procedure argument.Here's a solution (map (lambda (x) (twice (x)) == (map (lambda (x) (twice (wice x))) x)Much better.Can you see a difference between the two equivalent expressions?The first traverses two lists (x, and the result of the first map application). The second expression traverses only one list.	What's (map twice '(4 -1))	(8 - 2)
What should (map '()) be?() sounds like a good idea.map is part of the language, but here's a definition anyway. (define map (lambda (f lst) (cond ((null? lst) '()) (else (cons (f (car lst)) (map f (cdr lst)))))))That's old news, isn't it?What does (map twice (map twice x)) mean?Multiply each element of x by 4.Can you express this formally.Sure. How about (map (lambda (x) (* 4 x)) x)Does this work?Sure. Remember the distinction between free and bound variables. The x inside the lambda expression is bound.Does (map (lambda (x) (* 4 x)) x) make happy?No. It's too different than the original expression. It would be nice to use twice in the simplification.Does this work?No! (twice twice) produces an error measage.How come?Twice expects a number, not a procedure argument.Here's a solution (map twice (map twice x))) == (map (lambda (x) (twice (twice x))) x)Much better.Can you see a difference between the two equivalent expressions?The first traverses two lists (x, and the result of the first map application). The second expression traverses only one list.	What's (map '(4))	(8)
map is part of the language, but here's a definition anyway. (define map (lambda (f lst) (cond ((null? lst) '()) (else (cons (f (car lst)) (map f (cdr lst)))))))That's old news, isn't it?What does (map twice (map twice x)) mean?Multiply each element of x by 4.Can you express this formally.Sure. How about (map (lambda (x) (* 4 x)) x)Does this work?Sure. Remember the distinction between free and bound variables. The x inside the lambda expression is bound.Does (map (lambda (x) (* 4 x)) x) make happy?No. It's too different than the original expression. It would be nice to use twice in the simplification.Does this work?No! (twice twice) produces an error (map (twice twice) x)Does this work?No! (twice twice) produces an error (map (twice twice) x)Here's a solution (map twice (map twice x)) == (map (lambda (x) (twice (twice x))) x)Much better.Can you see a difference between the two equivalent expressions?The first traverses two lists (x, and the result of the first map application). The second expression traverses only one list.	What should (map '()) be?	() sounds like a good idea.
definition anyway. (define map (lambda (f lst) (cond ((mull? lst) '()) (else (cons (f (car lst)) (map f (cdr lst)))))))Here's a solution (map (lambda (x) (* a x)) x)What does (map twice (map twice x)) mean?Multiply each element of x by 4.Can you express this formally.Sure. How about (map (lambda (x) (* 4 x)) x)Does this work?Sure. Remember the distinction between free and bound variables. The x inside the lambda expression is bound.Does (map (lambda (x) (* 4 x)) x) make happy?No. It's too different than the original expression. It would be nice to use twice in the simplification.Does this work?No! (twice twice) produces an error (map (twice twice) x)How come?Twice expects a number, not a procedure argument.Here's a solution (map twice (map twice x)) == (map (lambda (x) (twice twice x)) x)Much better.Can you see a difference between the two equivalent expressions?The first traverses two lists (x, and the result of the first map application). The second expression traverses only one list.	map is part of the language, but here's a	That's old news, isn't it?
(define map (lambda (f lst) (cond ((null? lst) '()) (else (cons (f (car lst)) (map f (cdr lst)))))))Multiply each element of x by 4.What does (map twice (map twice x)) mean?Multiply each element of x by 4.Can you express this formally.Sure. How about (map (lambda (x) (* 4 x)) x)Does this work?Sure. Remember the distinction between free and bound variables. The x inside the lambda expression is bound.Does (map (lambda (x) (* 4 x)) x) make happy?No. It's too different than the original expression. It would be nice to use twice in the simplification.Does this work?No! (twice twice) produces an error message.How come?Twice expects a number, not a procedure argument.Here's a solution (map (lambda (x) (twice (twice x))) x)Much better.Can you see a difference between the two equivalent expressions?The first traverses two lists (x, and the result of the first map application). The second expression traverses only one list.	definition anyway.	
(Iambda (f Ist) (cond ((null? Ist) '()) (else (cons (f (car Ist)) (map f (cdr Ist)))))))Multiply each element of x by 4.What does (map twice (map twice x)) mean?Multiply each element of x by 4.Can you express this formally.Sure. How about (map (lambda (x) (* 4 x)) x)Does this work?Sure. Remember the distinction between free and bound variables. The x inside the lambda expression is bound.Does (map (lambda (x) (* 4 x)) x) make happy?No. It's too different than the original expression. It would be nice to use twice in the simplification.Does this work?No! (twice twice) produces an error (map (twice twice) x)How come?Twice expects a number, not a procedure argument.Here's a solution (map (lambda (x) (twice (twice x))) x)Much better.Can you see a difference between the two equivalent expressions?The first traverses two lists (x, and the result of the first map application). The second expression traverses only one list.	(define map	
(cond ((null? lst) '()) (else (cons (f (car lst)) (map f (cdr lst)))))))Multiply each element of x by 4.What does (map twice (map twice x)) mean?Multiply each element of x by 4.Can you express this formally. (map (lambda (x) (* 4 x)) x)Sure. How about (map (lambda (x) (* 4 x)) x)Does this work?Sure. Remember the distinction between free and bound variables. The x inside the lambda expression is bound.Does (map (lambda (x) (* 4 x)) x) make happy?No. It's too different than the original expression. It would be nice to use twice in the simplification.Does this work? (map (twice twice) x)No! (twice twice) produces an error message.How come?Twice expects a number, not a procedure argument.Here's a solution (map twice (map twice x)) == (map (lambda (x) (twice (twice x))) x)Much better.Can you see a difference between the two equivalent expressions?The first traverses two lists (x, and the result of the first map application). The second expression traverses only one list.	(lambda (f lst)	
((null? lst) '()) (else (cons (f (car lst)) (map f (cdr lst)))))))Multiply each element of x by 4.What does (map twice (map twice x)) mean?Multiply each element of x by 4.Can you express this formally.Sure. How about (map (lambda (x) (* 4 x)) x)Does this work?Sure. Remember the distinction between free and bound variables. The x inside the lambda expression is bound.Does (map (lambda (x) (* 4 x)) x) make happy?No. It's too different than the original expression. It would be nice to use twice in the simplification.Does this work?No! (twice twice) produces an error (map (twice twice) x)How come?Twice expects a number, not a procedure argument.Here's a solution (map twice (map twice x)) == (map (lambda (x) (twice (twice x))) x)Much better.Can you see a difference between the two equivalent expressions?The first traverses two lists (x, and the result of the first map application). The second expression traverses only one list.	(cond	
(else (cons (f (car lst)) (map f (cdr lst))))))Multiply each element of x by 4.What does (map twice (map twice x)) mean?Multiply each element of x by 4.Can you express this formally.Sure. How about (map (lambda (x) (* 4 x)) x)Does this work?Sure. Remember the distinction between free and bound variables. The x inside the lambda expression is bound.Does (map (lambda (x) (* 4 x)) x) make happ?No. It's too different than the original expression. It would be nice to use twice in the simplification.Does this work?No! (twice twice) produces an error (map (twice twice) x)How come?Twice expects a number, not a procedure argument.Here's a solution (map twice (map twice x)) == (map (lambda (x) (twice (twice x))) x)Much better.Can you see a difference between the two equivalent expressions?The first traverses two lists (x, and the result of the first map application). The second expression traverses only one list.	((null? lst) '())	
What does (map twice (map twice x)) mean?Multiply each element of x by 4.Can you express this formally.Sure. How about (map (lambda (x) (* 4 x)) x)Does this work?Sure. Remember the distinction between free and bound variables. The x inside the lambda expression is bound.Does (map (lambda (x) (* 4 x)) x) make happy?No. It's too different than the original expression. It would be nice to use twice in the simplification.Does this work?No! (twice twice) produces an error (map (twice twice) x)How come?Twice express a number, not a procedure argument.Here's a solution (map twice (map twice x)) == (map (lambda (x) (twice (twice x))) x)Much better.Can you see a difference between the two equivalent expressions?The first traverses two lists (x, and the result of the first map application). The second expression traverses only one list.	(else (cons (f (car lst)) (map f (cdr lst))))))	
mean?Sure. How about (map (lambda (x) (* 4 x)) x)Can you express this formally.Sure. How about (map (lambda (x) (* 4 x)) x)Does this work?Sure. Remember the distinction between free and bound variables. The x inside the lambda expression is bound.Does (map (lambda (x) (* 4 x)) x) make happy?No. It's too different than the original expression. It would be nice to use twice in the simplification.Does this work?No! (twice twice) produces an error (map (twice twice) x)How come?Twice express a number, not a procedure argument.Here's a solution (map twice (map twice x)) == (map (lambda (x) (twice (twice x))) x)Much better.Can you see a difference between the two equivalent expressions?The first traverses two lists (x, and the result of the first map application). The second expression traverses only one list.	What does (map twice (map twice x))	Multiply each element of x by 4.
Can you express this formally.Sure. How about (map (lambda (x) (* 4 x)) x)Does this work?Sure. Remember the distinction between free and bound variables. The x inside the lambda expression is bound.Does (map (lambda (x) (* 4 x)) x) make happy?No. It's too different than the original expression. It would be nice to use twice in the simplification.Does this work?No! (twice twice) produces an error (map (twice twice) x)Now come?Twice express a number, not a procedure argument.Here's a solution (map twice (map twice x)) == (map (lambda (x) (twice (twice x))) x)Much better.Can you see a difference between the equivalent expressions?The first traverses two lists (x, and the result of the first map application). The second expression traverses only one list.	mean?	
Image: Construct of the second expression is work?(map (lambda (x) (* 4 x)) x)Does this work?Sure. Remember the distinction between free and bound variables. The x inside the lambda expression is bound.Does (map (lambda (x) (* 4 x)) x) make happy?No. It's too different than the original expression. It would be nice to use twice in the simplification.Does this work?No! (twice twice) produces an error (map (twice twice) x)Mere's a solution (map twice (map twice x)) == (map (lambda (x) (twice twice x))) x)Much better.Can you see a difference between the two equivalent expressions?The first traverses two lists (x, and the result of the first map application). The second expression traverses only one list.	Can you express this formally.	Sure. How about
Does this work?Sure. Remember the distinction between free and bound variables. The x inside the lambda expression is bound.Does (map (lambda (x) (* 4 x)) x) make happy?No. It's too different than the original expression. It would be nice to use twice in the simplification.Does this work?No! (twice twice) produces an error (map (twice twice) x)How come?Twice expects a number, not a procedure argument.Here's a solution (map (lambda (x) (twice (twice x))) x)Much better.Can you see a difference between the two equivalent expressions?The first traverses two lists (x, and the result of the first map application). The second expression traverses only one list.		(map (lambda (x) (* 4 x)) x)
free and bound variables. The x inside the lambda expression is bound.Does (map (lambda (x) (* 4 x)) x) make happy?No. It's too different than the original expression. It would be nice to use twice in the simplification.Does this work?No! (twice twice) produces an error (map (twice twice) x)How come?Twice expects a number, not a procedure argument.Here's a solution (map twice (map twice x)) == (map (lambda (x) (twice (twice x))) x)Much better.Can you see a difference between the two equivalent expressions?The first traverses two lists (x, and the result of the first map application). The second expression traverses only one list.	Does this work?	Sure. Remember the distinction between
Image:		free and bound variables. The x inside the
Does (map (lambda (x) (* 4 x)) x) make happy?No. It's too different than the original expression. It would be nice to use twice in the simplification.Does this work? (map (twice twice) x)No! (twice twice) produces an error message.How come?Twice expects a number, not a procedure argument.Here's a solution (map twice (map twice x)) == (map (lambda (x) (twice (twice x))) x)Much better.Can you see a difference between the two equivalent expressions?The first traverses two lists (x, and the result of the first map application). The second expression traverses only one list.		lambda expression is bound.
happy?expression. It would be nice to use twice in the simplification.Does this work?No! (twice twice) produces an error(map (twice twice) x)message.How come?Twice expects a number, not a procedure argument.Here's a solutionMuch better.(map twice (map twice x)) == (map (lambda (x) (twice (twice x))) x)Much better.Can you see a difference between the two equivalent expressions?The first traverses two lists (x, and the second expression traverses only one list.	Does (map (lambda (x) (* 4 x)) x) make	No. It's too different than the original
Image: constraint of the simplification.Does this work?No! (twice twice) produces an error(map (twice twice) x)message.How come?Twice expects a number, not a procedure argument.Here's a solutionMuch better.(map twice (map twice x)) == (map (lambda (x) (twice (twice x))) x)Much better.Can you see a difference between the two equivalent expressions?The first traverses two lists (x, and the result of the first map application). The second expression traverses only one list.	happy?	expression. It would be nice to use twice
Does this work?No! (twice twice) produces an error message.(map (twice twice) x)message.How come?Twice expects a number, not a procedure argument.Here's a solution (map twice (map twice x)) == (map (lambda (x) (twice (twice x))) x)Much better.Can you see a difference between the two equivalent expressions?The first traverses two lists (x, and the result of the first map application). The second expression traverses only one list.		in the simplification.
(map (twice twice) x)message.How come?Twice expects a number, not a procedure argument.Here's a solutionMuch better.(map twice (map twice x)) == (map (lambda (x) (twice (twice x))) x)Much better.Can you see a difference between the two equivalent expressions?The first traverses two lists (x, and the result of the first map application). The second expression traverses only one list.	Does this work?	No! (twice twice) produces an error
How come?Twice expects a number, not a procedure argument.Here's a solutionMuch better.(map twice (map twice x)) == (map (lambda (x) (twice (twice x))) x)Much better.Can you see a difference between the two equivalent expressions?The first traverses two lists (x, and the result of the first map application). The second expression traverses only one list.	(map (twice twice) x)	message.
Image: definitionargument.Here's a solutionMuch better.(map twice (map twice x)) ==	How come?	Twice expects a number, not a procedure
Here's a solutionMuch better.(map twice (map twice x)) ==(map (lambda (x) (twice (twice x))) x)Can you see a difference between the twoThe first traverses two lists (x, and the result of the first map application). The second expression traverses only one list.		argument.
(map twice (map twice x)) ==Image: Additional and a state of the first traverses two lists (x, and the first map application). The first traverses only one list.(map (lambda (x) (twice (twice x))) x)The first traverses two lists (x, and the first map application). The first traverses only one list.	Here's a solution	Much better.
(map (lambda (x) (twice (twice x))) x)Image: Can you see a difference between the twoThe first traverses two lists (x, and the result of the first map application). The second expression traverses only one list.	(map twice (map twice x)) ==	
Can you see a difference between the twoThe first traverses two lists (x, and theequivalent expressions?result of the first map application). Thesecond expression traverses only one list.	(map (lambda (x) (twice (twice x))) x)	
equivalent expressions?result of the first map application). The second expression traverses only one list.	Can you see a difference between the two	The first traverses two lists (x, and the
second expression traverses only one list.	equivalent expressions?	result of the first map application). The
		second expression traverses only one list.

(map (o twice twice) x)
Well, it should be
((o (map twice) (map twice)) x)
No!
Map expects two arguments, and it is
only given one: twice.
Yes, we can do something similar.
Do it!
No, but you asked for it!
Should work.
Here's our explanation:
On the left hand side, we apply the
function (let's call it f) to each element of
x, and produce an list of results. We then
apply f to all the elements in the result
list. On the right hand side we apply f
twice to each element of x. The result is
the same.

And if instead of one function we had	That works too.
two?	
(map f (map g x)) ==	
(map (lambda (x) (f (g x))) x)	
Here a new function	Looks pretty meaningless to us.
(define so-of-twice	
(let ((counter 0))	
(lambda (x)	
(set! counter (+ 1 counter))	
(+ (* 2 x) counter))))	
Bare with us, please.	We get (10 17 24)
What's	
(map son-of-twice	
(map son-of-twice '(1 2 3)))	
What's	(8 18 28)
(map (o twice son-of-twice) '(1 2 3))	
See the problem?	These results should have been equal!
But you promised!	It was you! I am only doing what I am
	told.
Can we explain what happened?	Son-of-Twice behaves a little differently
	each time it is invoked, because counter
	keeps on changing.
Right. This is called a side-effect.	We don't like those, don't we?
Sure don't.	Good, I had a feeling this sort of thing
	can make a girl cry.
Not to mention grown up programmers.	Don't get me started on the traffic.
Was this the only problem we had today?	
No, we are talking about our	Well, there's were these problems with
manipulation of map expressions.	(twice (twice)), and (map twice).
All in a days work.	But can't we do better?

Of course we can, that's why we will use	But only after we had some pizza, right?
the Haskell language.	
And a tall Chocolate Brownie Frappuccino®	Our's was tasty. How was yours?