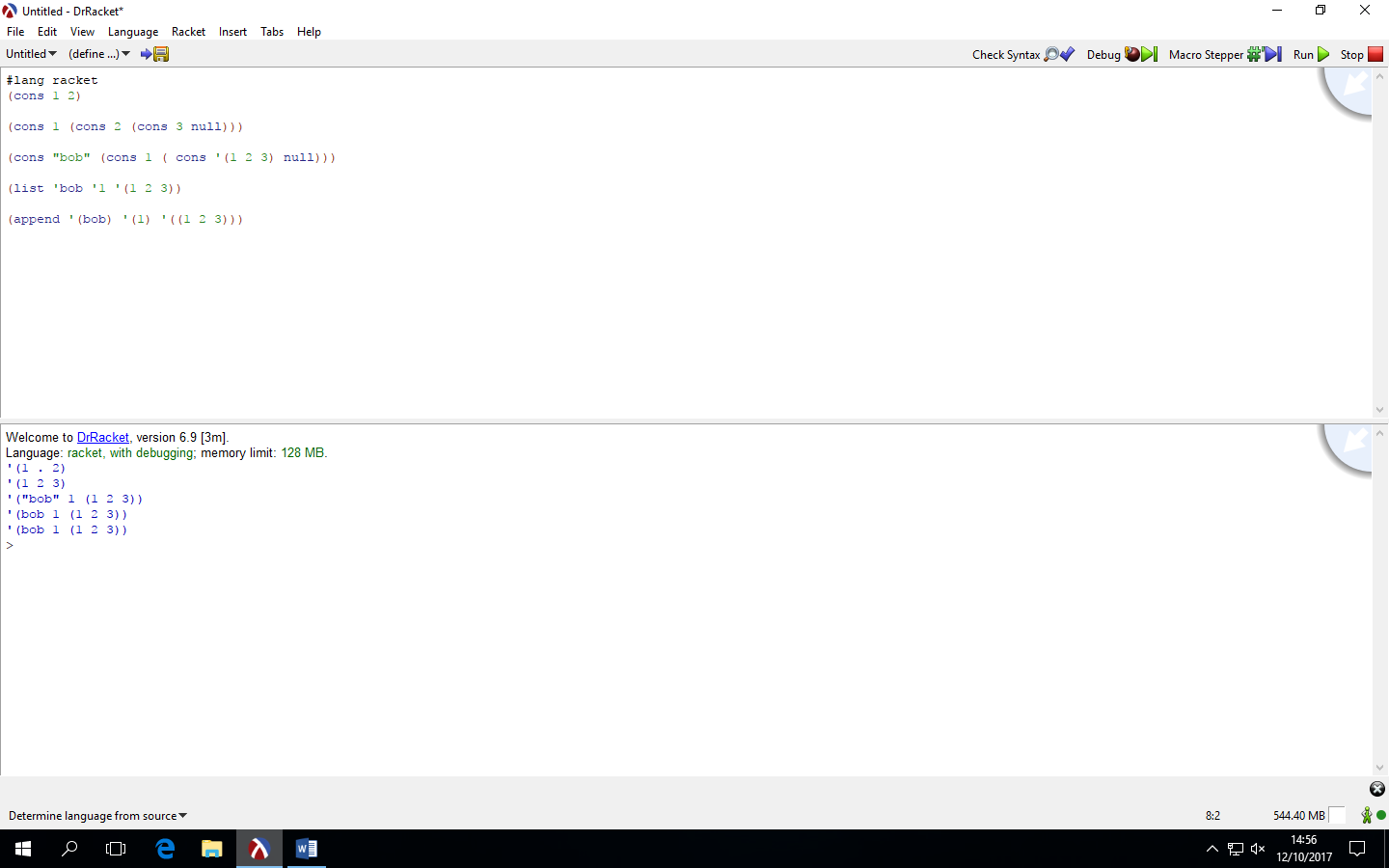
Assignment 2 – Scheme

David Langley

15340201

Q1



Comments on result:

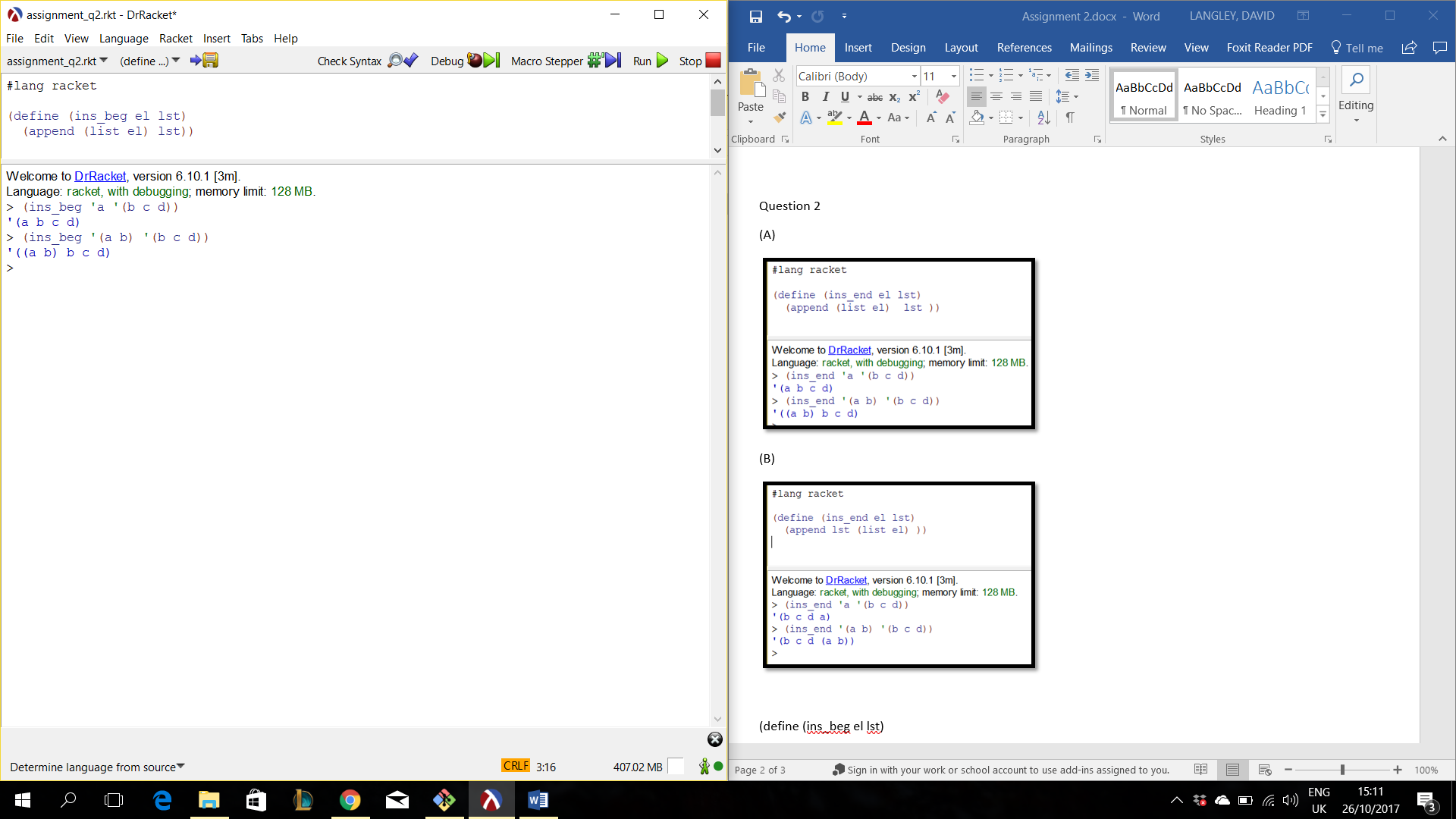
For the cons part, to be able to have multiple numbers using cons properly you need to make sure you ‘loop’ the cons over and over to keep it using the right function that you want it to do. Using cons you need to use double quotes to be allowed to print string whereas the list and append functions don’t need that.

For the list function, it is a better cons function allowing for easier access to the values.

Then the append function is again another better version of the list function allowing for more values to be listed in the function without error.

Question 2

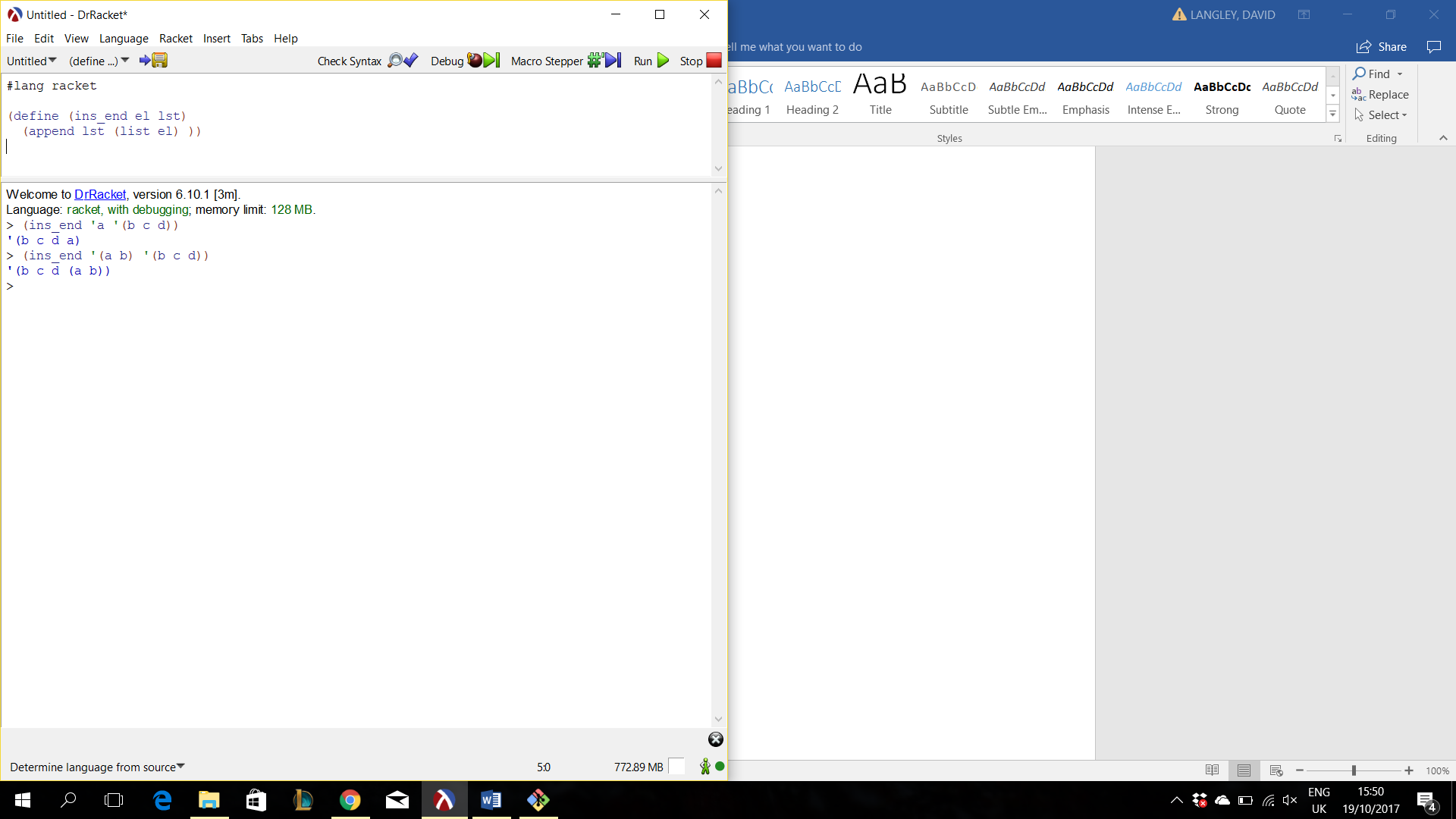
(A)



(define (ins\_beg el lst)

(append (list el) lst))

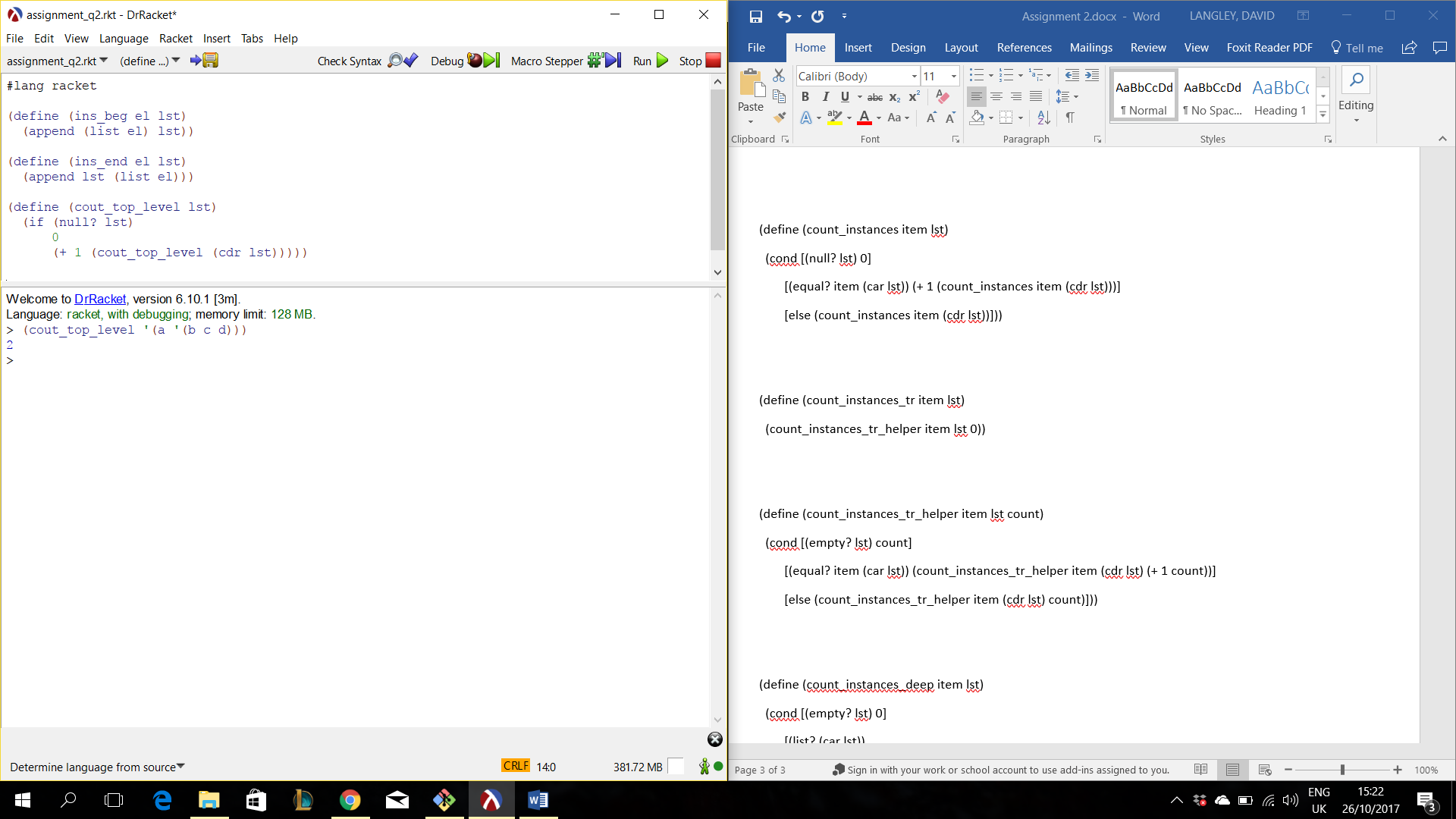
(B)



(define (ins\_end el lst)

(append lst (list el)))

(C)



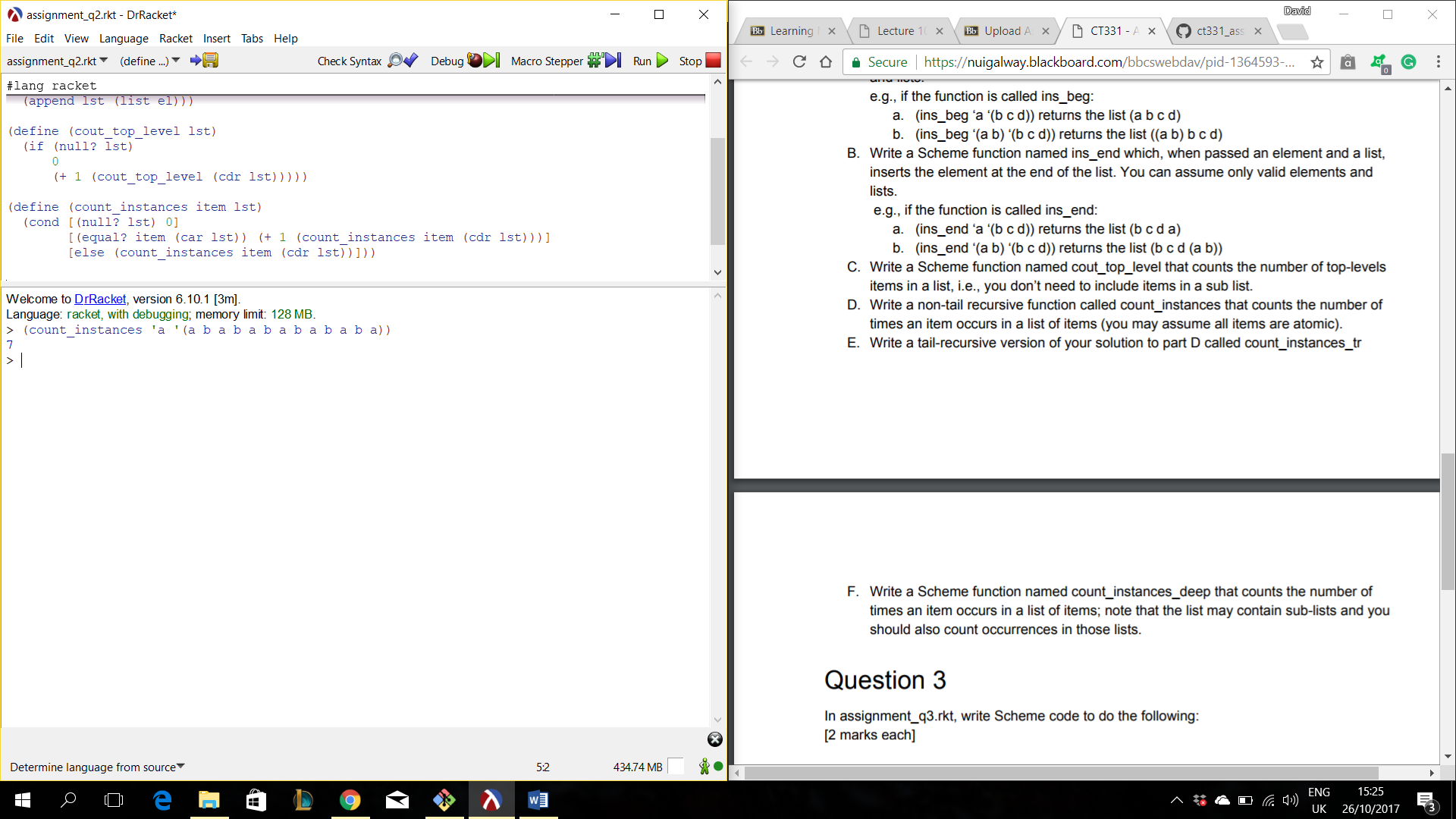
(define (cout\_top\_level lst)

(if (null? lst)

0

(+ 1 (cout\_top\_level (cdr lst)))))

(D)



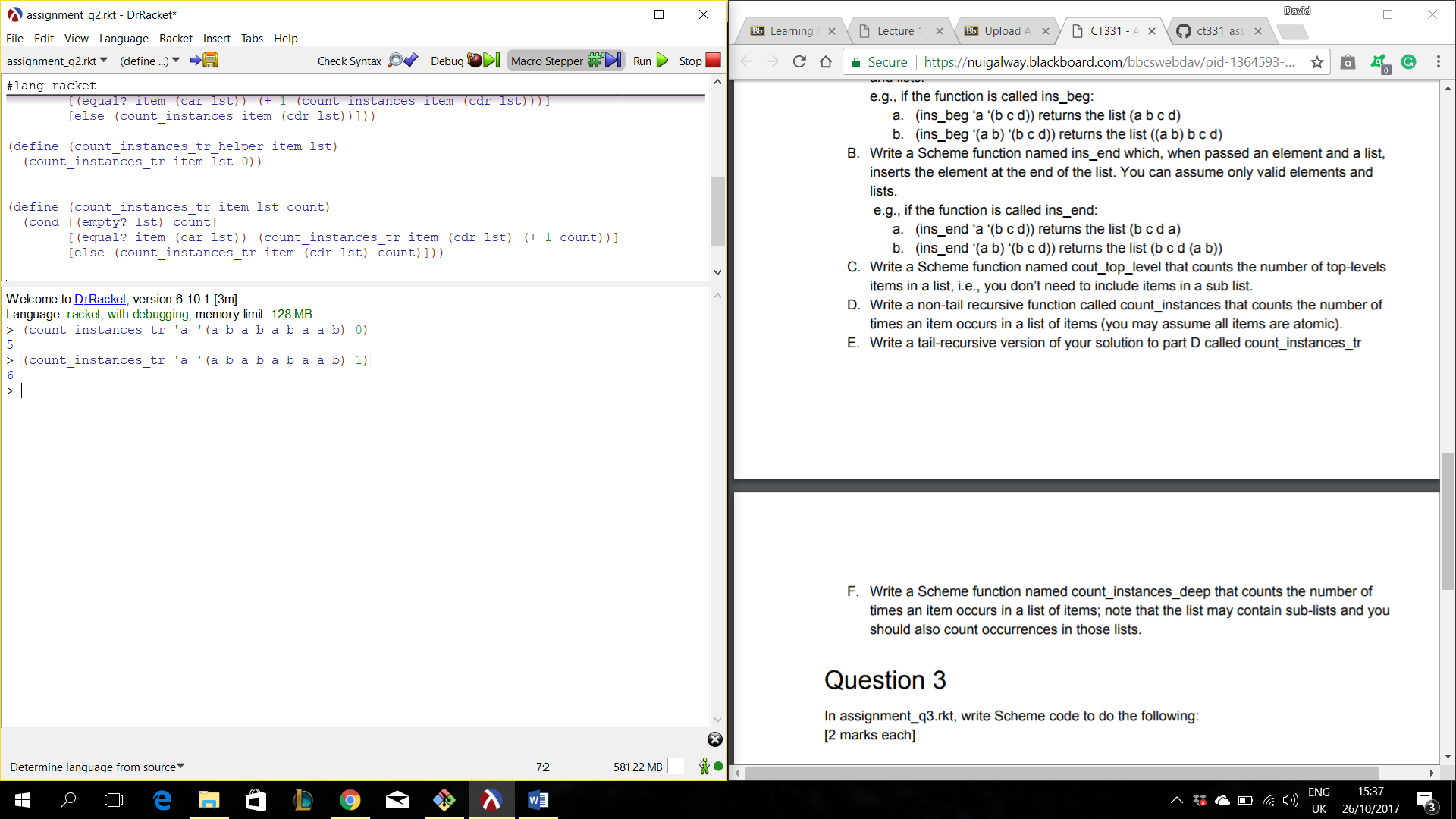
(define (count\_instances item lst)

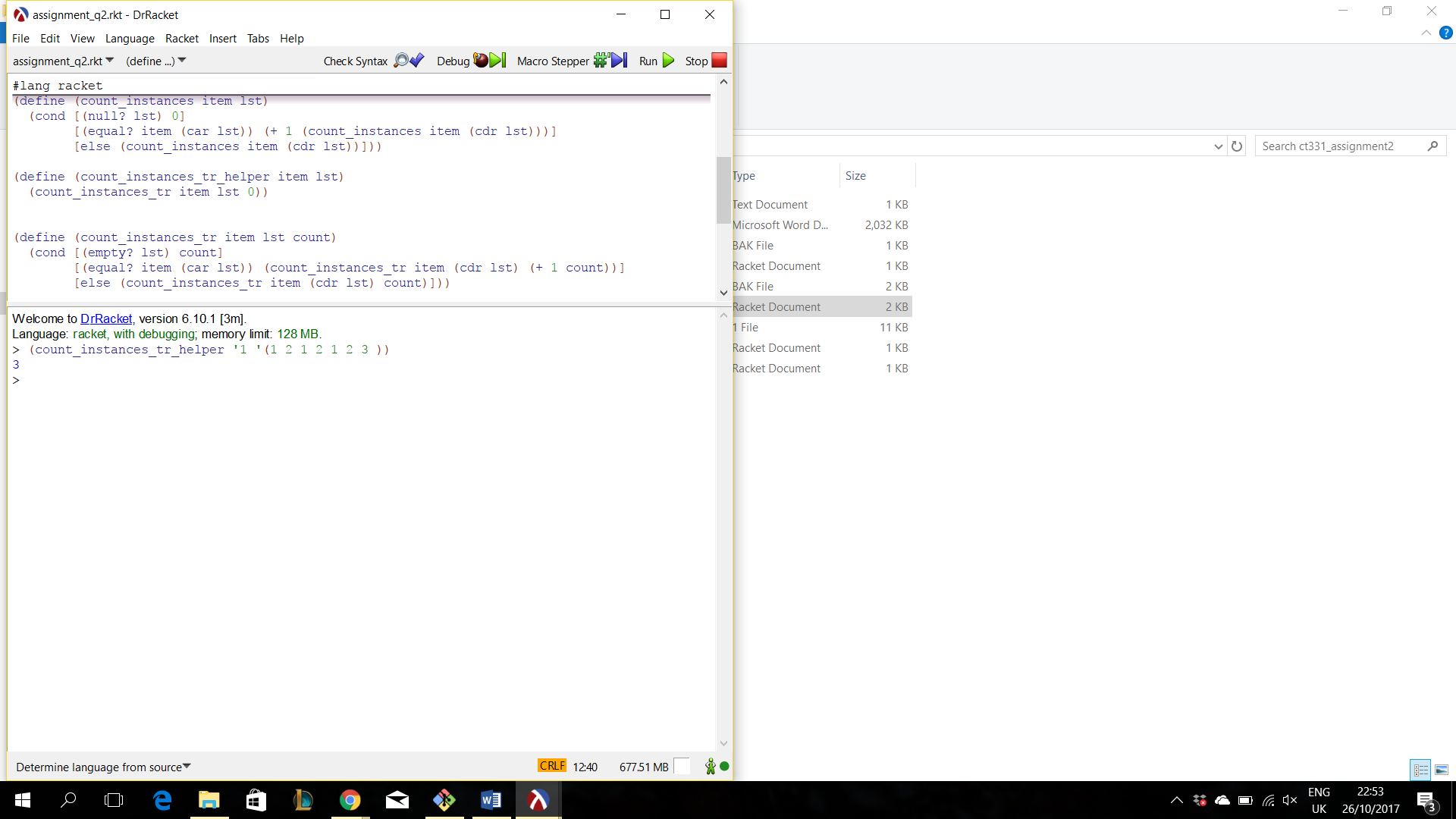
(cond [(null? lst) 0]

[(equal? item (car lst)) (+ 1 (count\_instances item (cdr lst)))]

[else (count\_instances item (cdr lst))]))

(E)





(define (count\_instances\_tr\_helper item lst)

(count\_instances\_tr item lst 0))

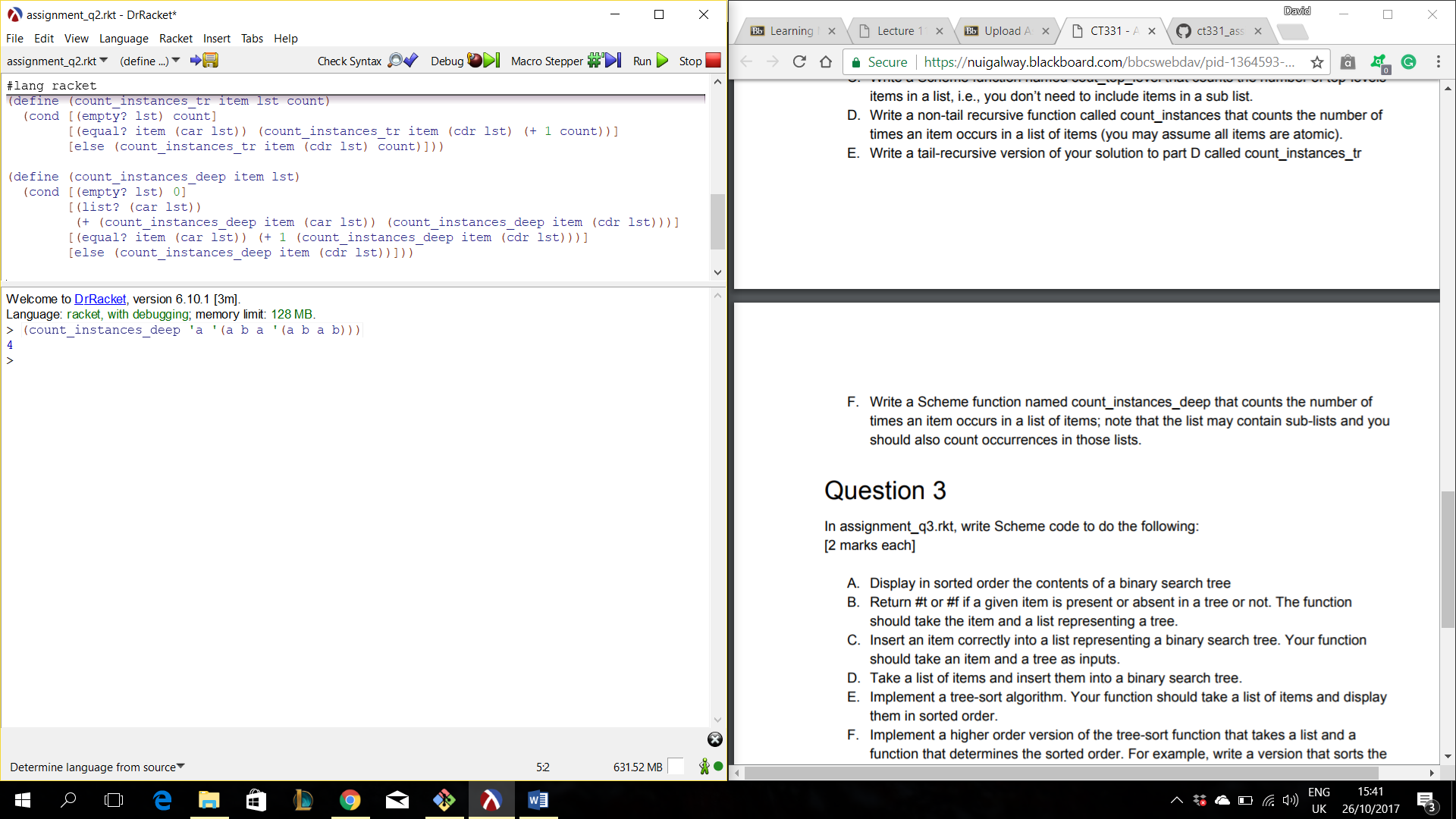
(define (count\_instances\_tr item lst count)

(cond [(empty? lst) count]

[(equal? item (car lst)) (count\_instances\_tr item (cdr lst) (+ 1 count))]

[else (count\_instances\_tr item (cdr lst) count)]))

(F)

(define (count\_instances\_deep item lst)

(cond [(empty? lst) 0]

[(list? (car lst))

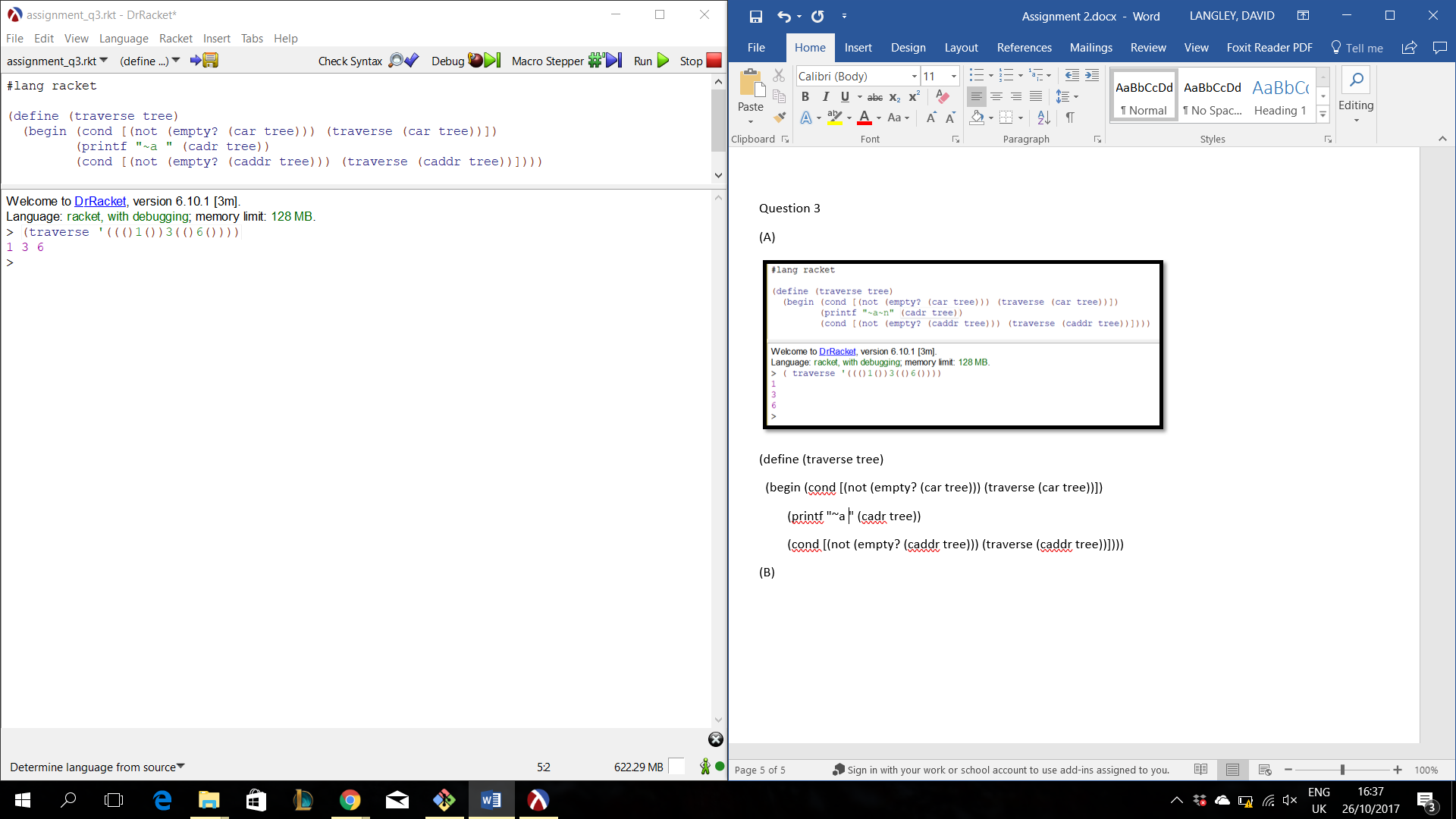
(+ (count\_instances\_deep item (car lst)) (count\_instances\_deep item (cdr lst)))]

[(equal? item (car lst)) (+ 1 (count\_instances\_deep item (cdr lst)))]

[else (count\_instances\_deep item (cdr lst))]))

Question 3

(A)



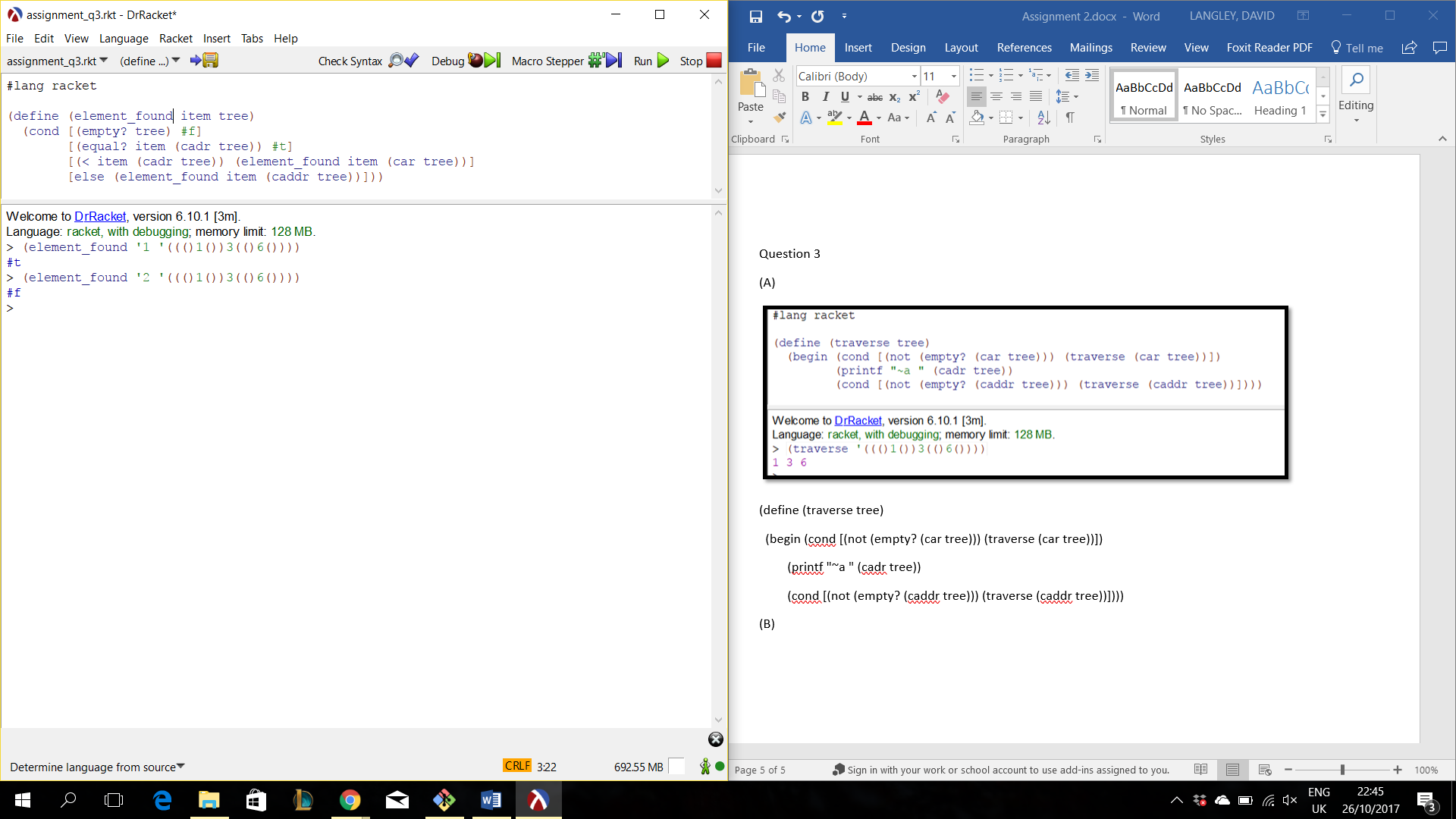
(define (traverse tree)

(begin (cond [(not (empty? (car tree))) (traverse (car tree))])

(printf "~a " (cadr tree))

(cond [(not (empty? (caddr tree))) (traverse (caddr tree))])))

(B)



(define (element\_found item tree)

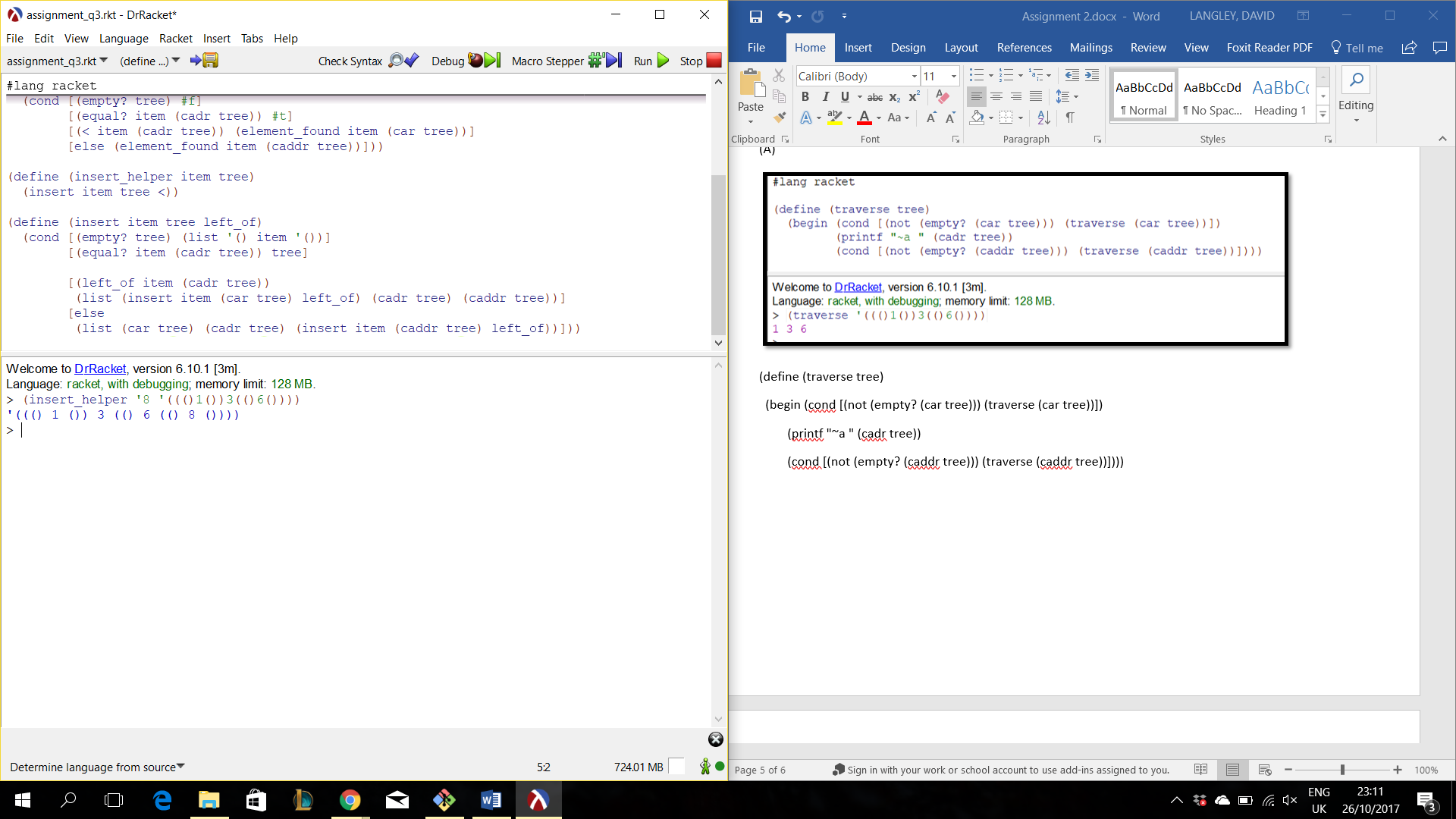
(cond [(empty? tree) #f]

[(equal? item (cadr tree)) #t]

[(< item (cadr tree)) (element\_found item (car tree))]

[else (element\_found item (caddr tree))]))

(C)



(define (insert\_helper item tree)

(insert item tree <))

(define (insert item tree left\_of)

(cond [(empty? tree) (list '() item '())]

[(equal? item (cadr tree)) tree]

[(left\_of item (cadr tree))

(list (insert item (car tree) left\_of) (cadr tree) (caddr tree))]

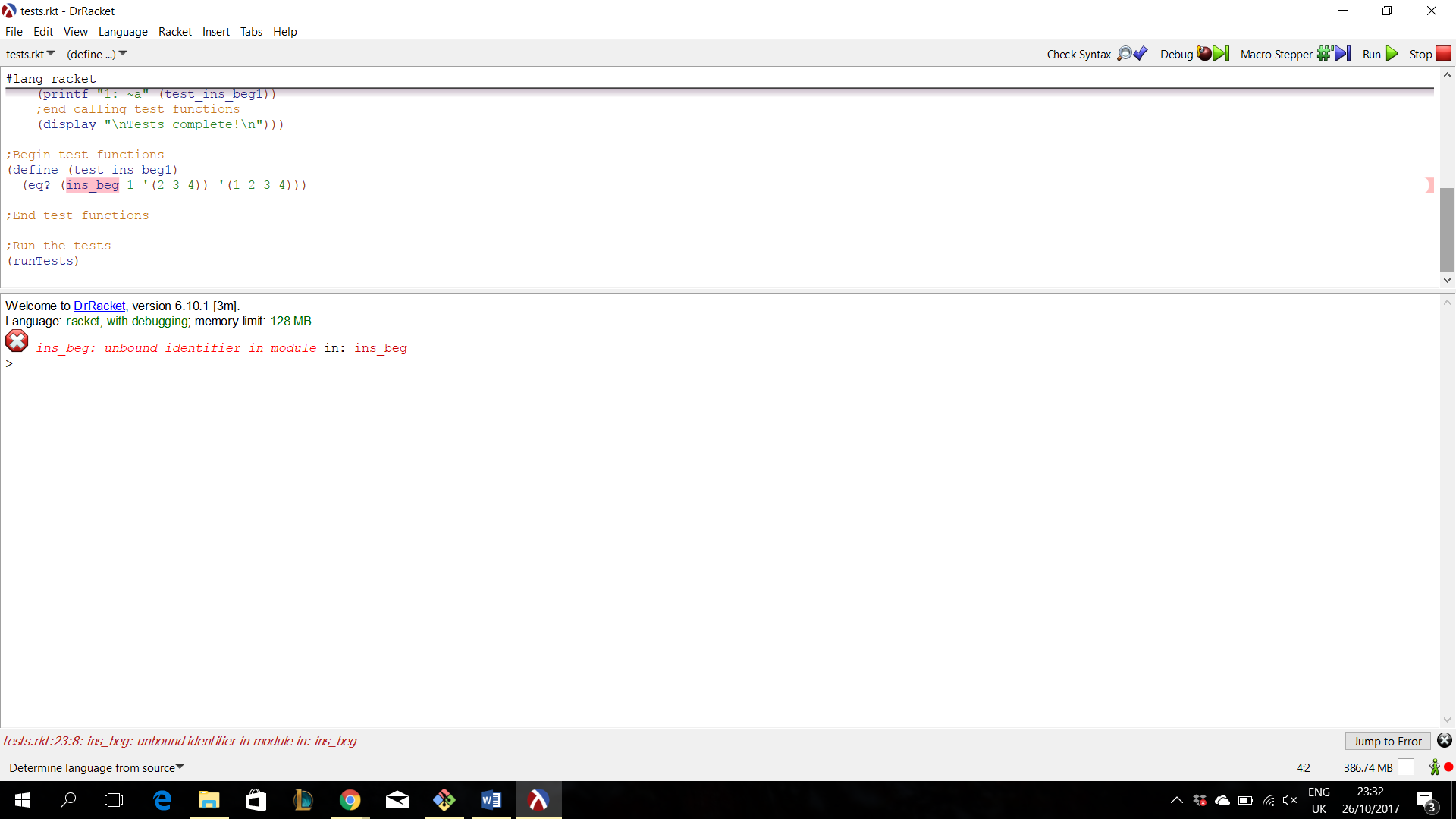
[else (list (car tree) (cadr tree) (insert item (caddr tree) left\_of))]))

(D)

(E)

(F)

Unit Testing



I don’t suppose there’s a problem there?

For the extra credit am I meant to fix this or something I can’t quite work it out.