如果是lambda，就转化成closure

如果是closure

如果是function expression的形式 （<expr> <expr> <expr> <expr> …）

1.如果第一个expr已经是closure了，

-如果没有多余的argument了，就直接evaluate

-如果还有多余的argument，把下一个argument放进closure的env里，再把加了 新env的closure和余下的argument继续放入recursion

2.如果第一个expr是lambda，要把lambda转换成closure（我们之后才可以往它里面放env），转化成closure后，带着closure和list of argument继续放入recursion。

要注意的地方：racket好像没有pointer，你在原本的list里的一个hash table里改了东西出来那个function后就不会跟着改变。我们只能重新把我们要改的东西复制过去。

(let\* ((n 10) (f (lambda (x) (\* x n)))) (f 5))

一开始进let\*，把{n:10}放进了外面的hash table

再进recursion，进只有一个binding那层，把lambda变成了closure后，{f：lambda}放进了外面的hash table, 此时closure里的hash table只有{n:10}

再进recursion，剩下(f 5) with外面的hash table {n:10, f:closure}

单独evaluate f with given hash table，就把closure of lambda赋值给了f了

(（closure (lambda (x) (\* x n)))) ENV）5)

把5放进closure 里的env了，此时就应该有{n:10，x:5}，expr也变成

((closure (lambda (x) (\* x n)))) ENV）

我在这个exercise里吃的最大的亏就是

(list (eval-calc (first expr) env)) (rest expr))

(append (list (eval-calc (first expr) env)) (rest expr))

是不一样的。。。。

第一个是把rest expr有作为一个inner list放进去了，

而第二个只是把rest expr里面的东西放进去。。。。。

妈的。。。