

## AI 2 Documentation

Me: Im working on a project but I cant seem to figure out the issues in my rkt code. Here is the error message: if: missing an "else" expression in: (if (assoc (second expr) state) (let-values (((val new-state) (eval (third expr) state))) (if (failure? val) (values val state) (values (success (second expr)) (update-var (second expr) (from-success #f val) new-state))) (values (failure (list "Error: variable not defined" (second expr))) state)))

> and here is the code:

[Code Submitted]

AI 2: Thanks for sharing your code and the error message, Kaleb. The Racket error:

if: missing an "else" expression in:

```
(if (assoc (second expr) state)
    (let-values (((val new-state) (eval (third expr) state)))
      (if (failure? val)
          (values val state)
          (values (success (second expr))
                  (update-var (second expr) (from-success #f val) new-state)))
    (values (failure (list "Error: variable not defined" (second expr))) state)))
```

...tells us that one of your if expressions is missing its else clause — meaning you wrote (if test then-part) instead of the full (if test then-part else-part).

Looking into your code, the issue is right here in your 'assign clause:

racket

```
[(equal? (first expr) 'assign)
 (if (assoc (second expr) state)
     (let-values (((val new-state) (eval (third expr) state)))
       (if (failure? val)
           (values val state)
           (values (success (second expr))
                   (update-var (second expr) (from-success #f val) new-state)))
     (values (failure (list "Error: variable not defined" (second expr))) state)))
```

```


(update-var (second expr) (from-success #f val) new-state)))

;; ← THIS LINE is inside the (if (assoc ...)) but lacks the matching else

(values (failure (list "Error: variable not defined" (second expr))) state))]
```

 Fix:

You need to add an else clause to the outer if. Currently, your else branch ((values (failure ...))) is floating outside of the if. You can fix it by wrapping everything in the proper branches:

 Corrected version:

racket

```

[(equal? (first expr) 'assign)

(if (assoc (second expr) state)

  (let-values (((val new-state) (eval (third expr) state)))

    (if (failure? val)

      (values val state)

      (values (success (second expr))

                (update-var (second expr) (from-success #f val) new-state))))

  (values (failure (list "Error: variable not defined" (second expr))) state))]
```

Notice how the final (values ...) is now clearly inside the else branch of the outer if.

Let me know if you want help testing it with specific expressions or adding support for new features.

Me: Okay I did this and fixes a few more things but now I have a different issue. Here is the error message: if: bad syntax in: (if (assoc (second expr) state) (let-values (((val new-state) (eval (third expr) state))) (if (failure? val) (values val state) (values (success (second expr)) (update-var (second expr) (from-success #f val) new-state))) (values (failure (list "Error: variable not defined" (second expr))) state)) ((equal? (first expr) (quote remove)) (values (success (second expr)) (remove-var (second expr) state))) (else (values (failure "Error: unknown operation") state))))

AI2: Thanks for sharing your code and the error message, Kaleb. The issue here is with your use of the if expression in the assign case of your eval function. Specifically, the syntax error message:

perl

if: bad syntax in: (if (assoc (second expr) state) (let-values ...) ...)

The problem

Your assign clause in the cond block ends like this:

racket

```
[(equal? (first expr) 'assign)
 (if (assoc (second expr) state)
     (let-values ... ) ;; happy path
     (values ...))] ;; error case
```

But you are missing a closing parenthesis on the let-values expression inside the then branch. This causes the if to think you're trying to use more than 3 expressions, which isn't allowed in Racket's if (it only takes a condition, then-branch, and else-branch).

✅ Fix

Here's the corrected version of that part of the eval function:

racket

```
[(equal? (first expr) 'assign)
 (if (assoc (second expr) state)
     (let-values ([(val new-state) (eval (third expr) state)])
       (if (failure? val)
           (values val state)
           (values (success (second expr))
                    (update-var (second expr) (from-success #f val) new-state))))
     (values (failure (list "Error: variable not defined" (second expr))) state))]
```

✅ Note the additional closing parenthesis just after the inner (update-var ...) line — that properly closes the let-values.

Let me know if you want help testing a few expressions with this evaluator to ensure it behaves as expected.

Me: I made some changes to my code and I got a new error: new-state: unbound identifier in: new-state

AI2: Kaleb, the error new-state: unbound identifier in: new-state is happening because you're trying to use the variable new-state in a scope where it hasn't been defined yet. This usually happens when a let-values block or a similar form doesn't properly bind a variable before it's used.

Looking at the end of your eval function, specifically inside the (equal? (first expr) 'assign) case:

```
r
```

```
(let-values ([ (val new-state) (eval (third expr) state)])
```

You then use new-state here:

```
racket
```

```
(update-var (second expr) (from-success #f val) new-state)
```

This is fine, assuming this block is complete. But from your pasted code, the final expression looks cut off right after:

```
racket
```

```
(values (failure (list "Error: variable not defined" (second expr)))...
```

That line ends with ..., indicating that the full expression wasn't provided. If the else branch of the assign clause is missing its complete body, Racket might be getting confused, especially if it's expecting more expressions and never sees new-state get bound properly in that case.

✅ Fix

Make sure you complete the assign clause. Here's how that whole block should look, with the missing piece filled in:

racket

```
[(equal? (first expr) 'assign)
 (if (assoc (second expr) state)
     (let-values ([ (val new-state) (eval (third expr) state)])
       (if (failure? val)
           (values val state)
           (values (success (second expr))
                    (update-var (second expr) (from-success #f val) new-state))))
     (values (failure (list "Error: variable not defined" (second expr))) state))]
```

Make sure that last state in the else clause is included.

Let me know if you're seeing any other errors after this!

Me: I still get the same error

AI2: Thanks for confirming, Kaleb. If you're still getting the new-state: unbound identifier in: new-state error even after completing the assign clause, then the likely issue is variable shadowing or scoping—most likely in this section of your eval function:

racket

```
[(member (first expr) '(div add sub mult))
 (let-values ([ (x new-state) (eval (second expr) state)])
   (if (failure? x)
       (values x state)
       (let-values ([ (y new-state2) (eval (third expr) new-state)])
         ...
```

In this block, you're binding new-state, and then reusing new-state2 inside it, which is correct.

But: outside of this clause, new-state is not defined. So if any other case tries to use new-state without defining it first, the error will occur.

### ✅ Solution Steps

Double-check all your clauses in eval to make sure every use of new-state comes from a let, let\*, or let-values.

Here's a checklist:

### ✅ Good:

racket

```
(let-values ([ (x new-state) ... ] ...)
```

### ❌ Bad (causes your error):

r

```
(update-var id val new-state) ; <- if new-state isn't bound yet here, boom: unbound identifier
```

### 🔍 Common Fix Area

From your code, it's very likely that the (member '(div add sub mult)) clause looks like this:

racket

```
[(member (first expr) '(div add sub mult))  
 (let-values ([ (x new-state) (eval (second expr) state) ])  
   ...  
   (let-values ([ (y new-state2) ... ])  
     ...  
     (values (some-result) new-state2)))]
```

That's all good.

But now check other branches (like 'assign, 'define) to make sure you're not doing this:

racket

```
;; BAD: update-var using unbound new-state
```

```
(values (success ...) (update-var id val new-state))
```

Instead, make sure the whole structure is:

racket

```
(let-values ([ (val new-state) (eval expr state)])
```

```
... use new-state ...)
```

#### Final Tip

To catch it easily, search your code for `new-state` and verify that every usage is inside a `let`, `let-values`, or directly in scope.

Would you like me to go line-by-line through your pasted `eval` function and flag every problematic use of

Me: could it be happening here:

[Code Submitted]

```
(define (repl)
```

```
  (let loop ((state (initial-state)))
```

```
    (display(format "State: ~a"
```

```
      (map (λ (pair)
```

```
        (list (car pair)
```

```
          (if (equal? (cdr pair) 'undefined) 'undefined (cdr pair)))new-state))))
```

```
    (flush-output)
```

AI2:

Yes, Kaleb — that's exactly where your error is coming from.

You're using `new-state` inside your `repl` function, specifically here:

racket

```
(map (λ (pair)
```

```
  (list (car pair)
```

```
(if (equal? (cdr pair) 'undefined) 'undefined (cdr pair)))  
new-state)
```

But nowhere in the repl function is new-state defined — your loop parameter is named state.

✅ Fix

You should change new-state to state in that expression:

```
racket  
(define (repl)  
  (let loop ((state (initial-state)))  
    (display (format "State: ~a"  
      (map (λ (pair)  
        (list (car pair)  
          (if (equal? (cdr pair) 'undefined) 'undefined (cdr pair))))  
      state))) ; ← change here  
    (flush-output)  
    ...))
```

Once you make that change, the unbound identifier in: new-state error should go away.

Want help checking the rest of the repl function too?

Me: I have a new error: map: arity mismatch; the expected number of arguments does not match the given number expected: at least 2 given: 1

AI2: Great catch, Kaleb — the new error is caused by a small but subtle parentheses issue in your map call.

🔍 The problem:

```
racket  
(map (λ (pair)
```



```
(list (car pair)
```

```
      (if (equal? (cdr pair) 'undefined) 'undefined (cdr pair)))
```

```
state) ; ← this closing paren is in the wrong place
```

You accidentally closed the lambda too early, so Racket thinks map only got 1 argument, not 2.

✅ The fix:

You just need to move one parenthesis so the function and the list are both passed to map properly:

```
racket
```

```
(map (λ (pair)
```

```
      (list (car pair)
```

```
            (if (equal? (cdr pair) 'undefined) 'undefined (cdr pair))))
```

```
state)
```

Full corrected snippet:

```
racket
```

```
(define (repl)
```

```
  (let loop ((state (initial-state)))
```

```
    (display (format "State: ~a"
```

```
              (map (λ (pair)
```

```
                    (list (car pair)
```

```
                          (if (equal? (cdr pair) 'undefined) 'undefined (cdr pair))))
```

```
state)))
```

```
(flush-output)
```

```
(let ((input (read)))
```

```
  (when (not (equal? input 'quit))
```

```
    (let-values ([(result new-state) (eval input state)])
```

```
(if (success? result)
  (begin
    (displayln (format "Success: ~a" (from-success #f result)))
    (displayln (format "State: ~a" new-state)))
  (begin
    (displayln (format "Failure: ~a" (from-failure #f result)))
    (displayln (format "State: ~a" state))))
(loop new-state))))
```

Let me know if you'd like help improving the REPL interaction further, like adding prompts or catching read errors.

