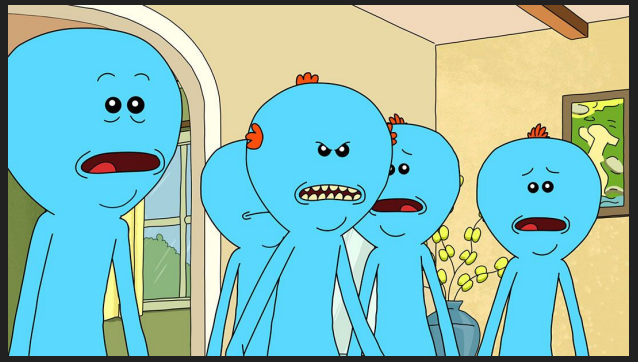
CS61A Discussion 8

Interpreters/Tail Recursion

Calculator (Worksheet Time!)

Tail Call Optimization

- Recursion requires a lot of lingering frames
  - This is because we typically need to do extra work on the return of our recursive call
- Solution: Tail Call Optimization!



What is a Tail Call?

- The last expression to be evaluated in a form (for specifics see worksheet)
- Keep the order of evaluation in mind!
  - Ex. (+ 1 (helper 2))
    - Last thing to be evaluated here is +

Tail Call Optimization

- Scheme takes advantage of tail calls and drops the previous frame if no extra work needs to be done (recursive call was in a tail context)
- Typically to achieve this, we can fiddle with the arguments of our helper functions to include our result
- Try factorial: https://scheme.cs61a.org/

7. (6 points) Counting stars

(a) (4 pt) Implement a tail-recursive Scheme function called `count`, which takes in a number and a well-formed Scheme list of numbers. `count` returns the number of times the number appears in the list.

```
scm> (count 3 (1 2 3 4 3))
2
scm> (count 42 (3 4 2))
0
```

Remember, **your solution must be tail recursive**. A reasonable solution uses 3 to 4 additional lines of code, but you can use fewer or more if you want. Make sure to fill in the blanks in the second-to-last line.

```
(define (count num lst)
    (define (helper lst total)









    )
    (helper ____________ ____________)
)
```

7. (6 points) Counting stars

(a) (4 pt) Implement a tail-recursive Scheme function called `count`, which takes in an item and a well-formed Scheme list. `count` returns the number of times the item appears in the list.

```
scm> (count 3 (1 2 3 4 3))
2
scm> (count 42 (3 4 2))
0
```

Remember, **your solution must be tail recursive**. A reasonable solution uses 3 to 4 additional lines of code, but you can use fewer or more if you want. Make sure to fill in the blanks in the second-to-last line.

```
(define (count elem lst)

    (define (helper lst total)

        (cond ((null? lst) total)

              ((= (car lst) elem) (helper (cdr lst) (+ total 1)))

              (else (helper (cdr lst))))

    )

    (helper lst 0)

)
```