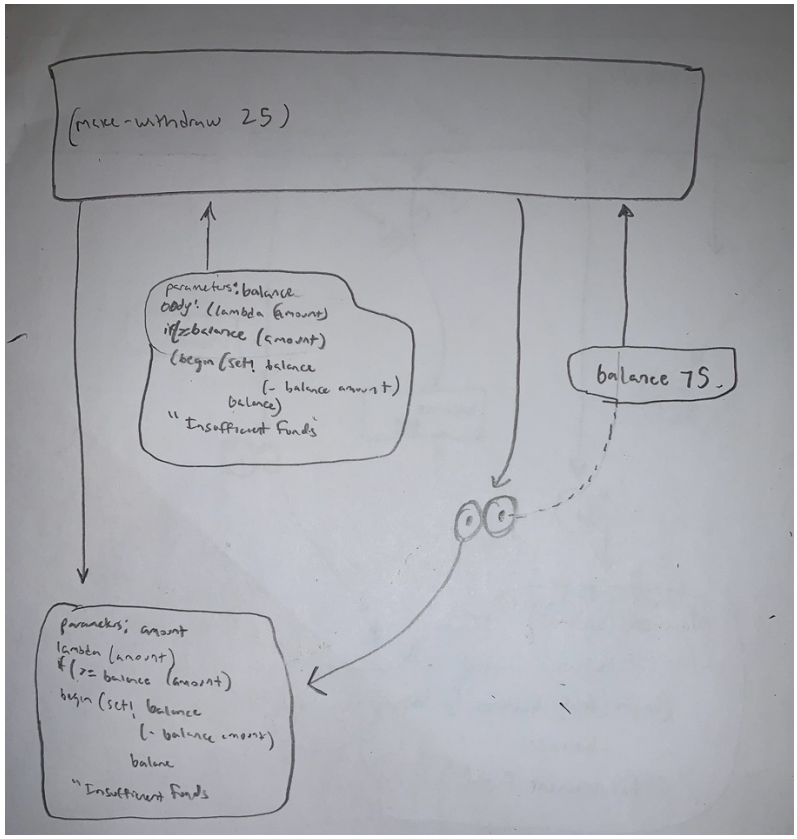
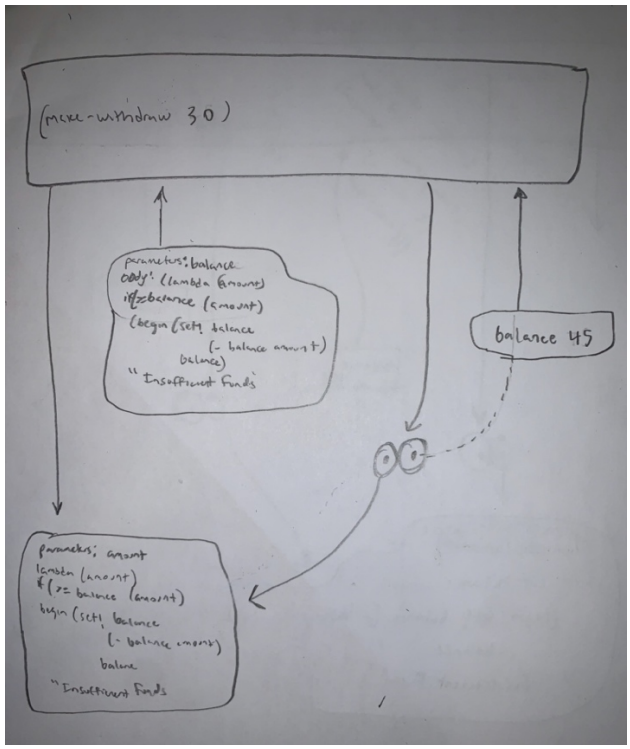
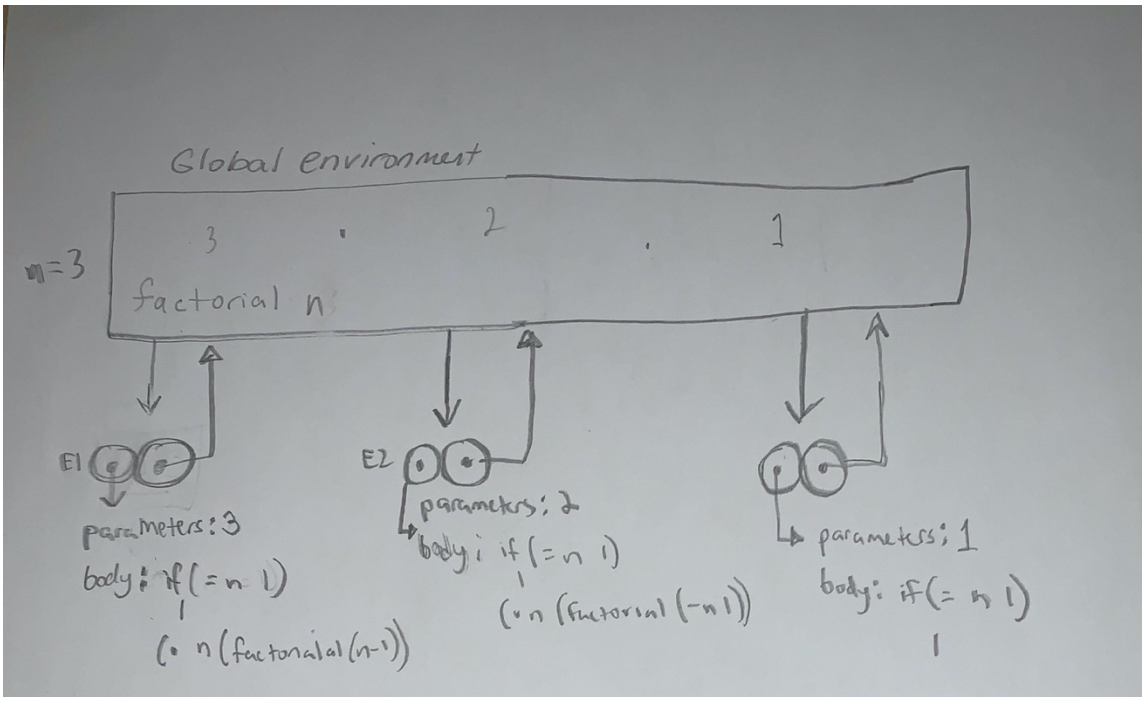
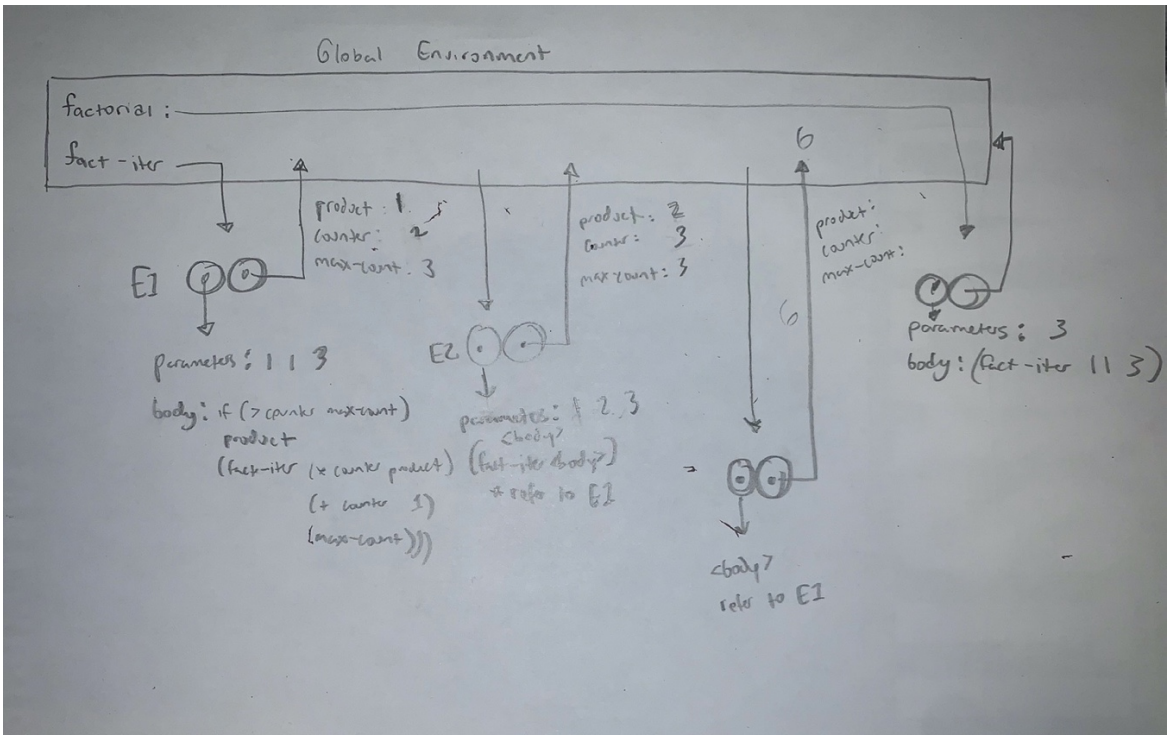
2.)


(make-withdraw 25)
parameters: balance
body: (lambda (amount)
if (>= balance amount)
(begin (set! balance
(- balance amount))
balance)
"Insufficient Funds"
balance 75
parameters: amount
(lambda (amount)
if (>= balance amount)
begin (set! balance
(- balance amount))
balance
"Insufficient Funds


(make-withdraw 30)
parameters: balance
body: (lambda (amount)
if (>= balance amount)
(begin (set! balance
(- balance amount))
balance)
"Insufficient Funds"
balance 45
parameters: amount
(lambda (amount)
if (>= balance amount)
begin (set! balance
(- balance amount))
balance
"Insufficient Funds

5.)


Global environment
3
2
1
n=3
factorial n
E1
E2
parameters: 3
body: if (= n 1)
1
(* n (factorial (-n 1)))
parameters: 2
body: if (= n 1)
1
(* n (factorial (-n 1)))
parameters: 1
body: if (= n 1)
1


Global Environment
factorial :
fact-iter
6
product: 1
counter: 2
max-count: 3
product: 2
counter: 3
max-count: 3
product:
counter:
max-count:
E1
E2
parameters: 1 1 3
body: if (> counter max-count)
product
(fact-iter (* counter product)
(+ counter 1)
max-count)))
parameters: 1 2 3
<body>
(fact-iter body)
* refer to E1
6
<body>
refer to E1
parameters: 3
body: (fact-iter 1 1 3)

Part2

1.) Some difficulties I ran into when designing this homework assignment was trying to figure out how to iterate through lists. Another problem was trying to figure out how to store multiple variables in a function. I still need to learn how to get into a functional programming mindset and to think recursively because this assignment took way too long and I didn't even finish past min naïve cost. Another struggle I have was trying to stay sane when things don't work the way I expected them too. In the end, I am grateful to have figured out min-cost-naïve and to take steps at becoming a better functional programmer. Please have mercy on me when grading this.

2.) I know that the min-cost-naïve should have ran much slower than the dynamic programming implementation because of memoization. Wish I could have figured this out to show.