Evaluate Prefix Notation

- there's a recursive solution & a non-recursive. Something you should be familiar w/ is that you can always implement recursive solutions with stacks & queues. In fact, you should go back & solve Q1 with a stack.
- so, what's a stack? It's a data structure that has pieces of data being put in and taken out in LIFO: last in first out. So, everything you put in will be taken out in exactly the reverse order.

Following the order of the strategy outlined in the overview page: ① understand the question. This includes any clarifying q's you have.

So, one assumption: only TWO operands for any operator. That is:

+ 2 4 is OK

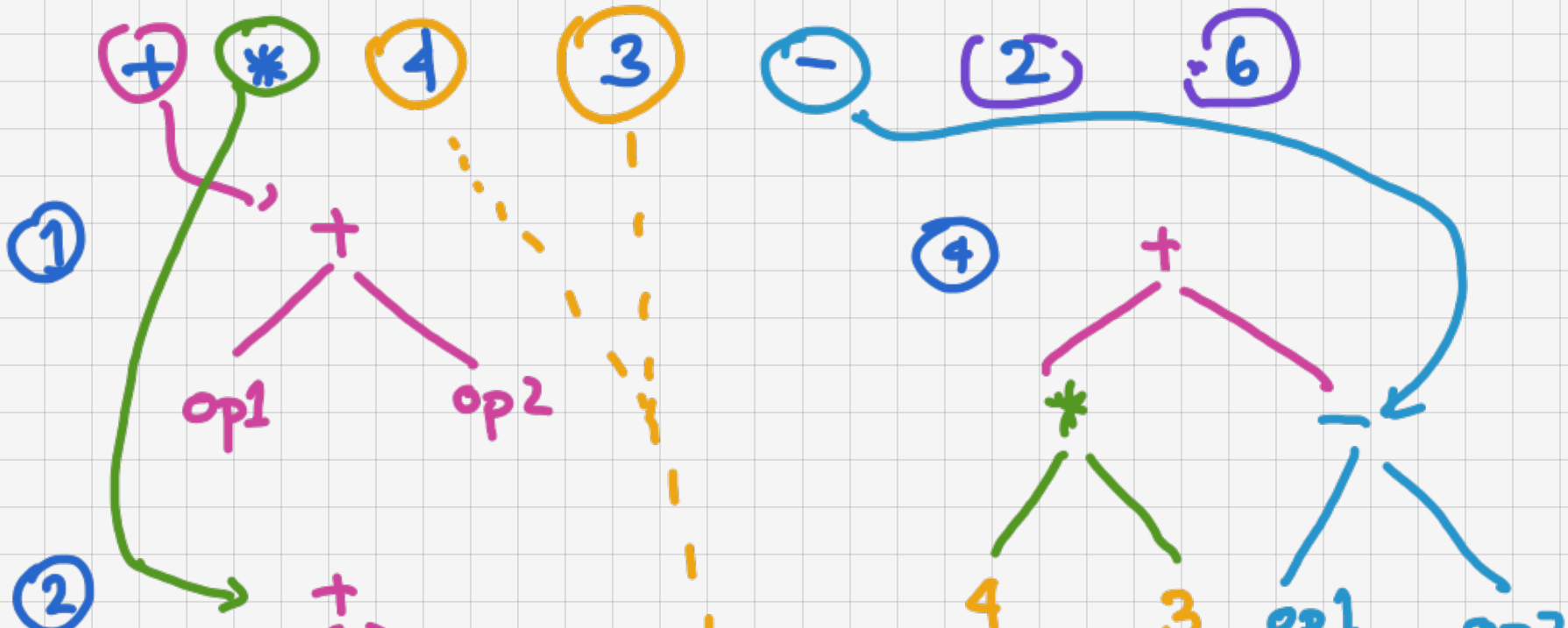
+ 2 4 3 will not work.

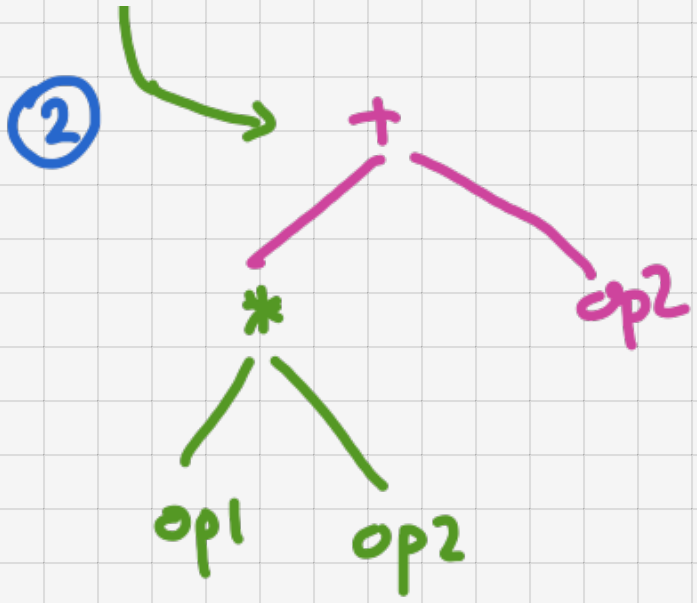
Assumption 2: what's the input? I'll leave it up to you. I'll use a list of strings.

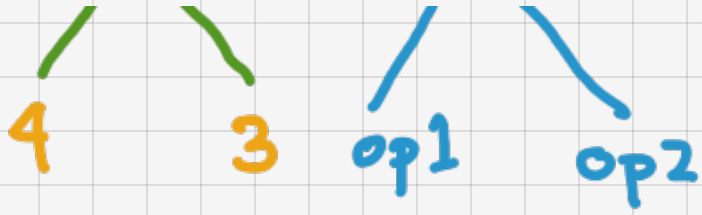
Assumption 3: only use '+', '-', '*', '/'.

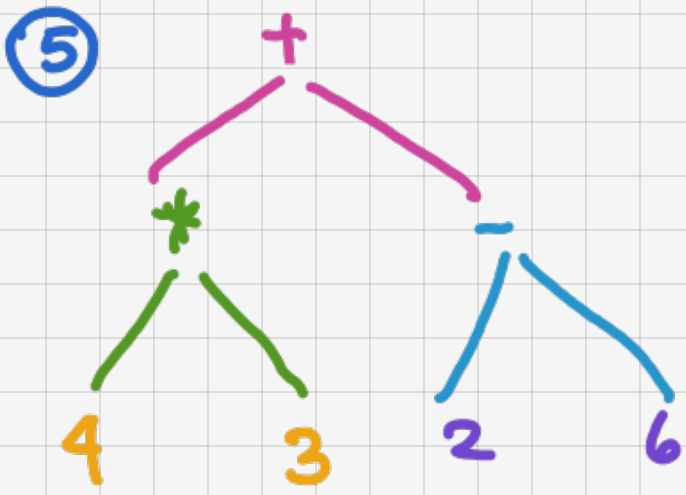
Now, examples: Let's say

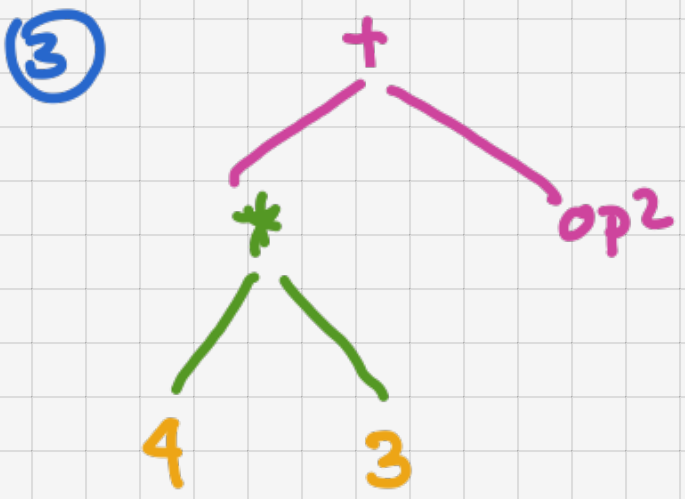
input = ['+', '*', '4', '3', '-', '2', '6']





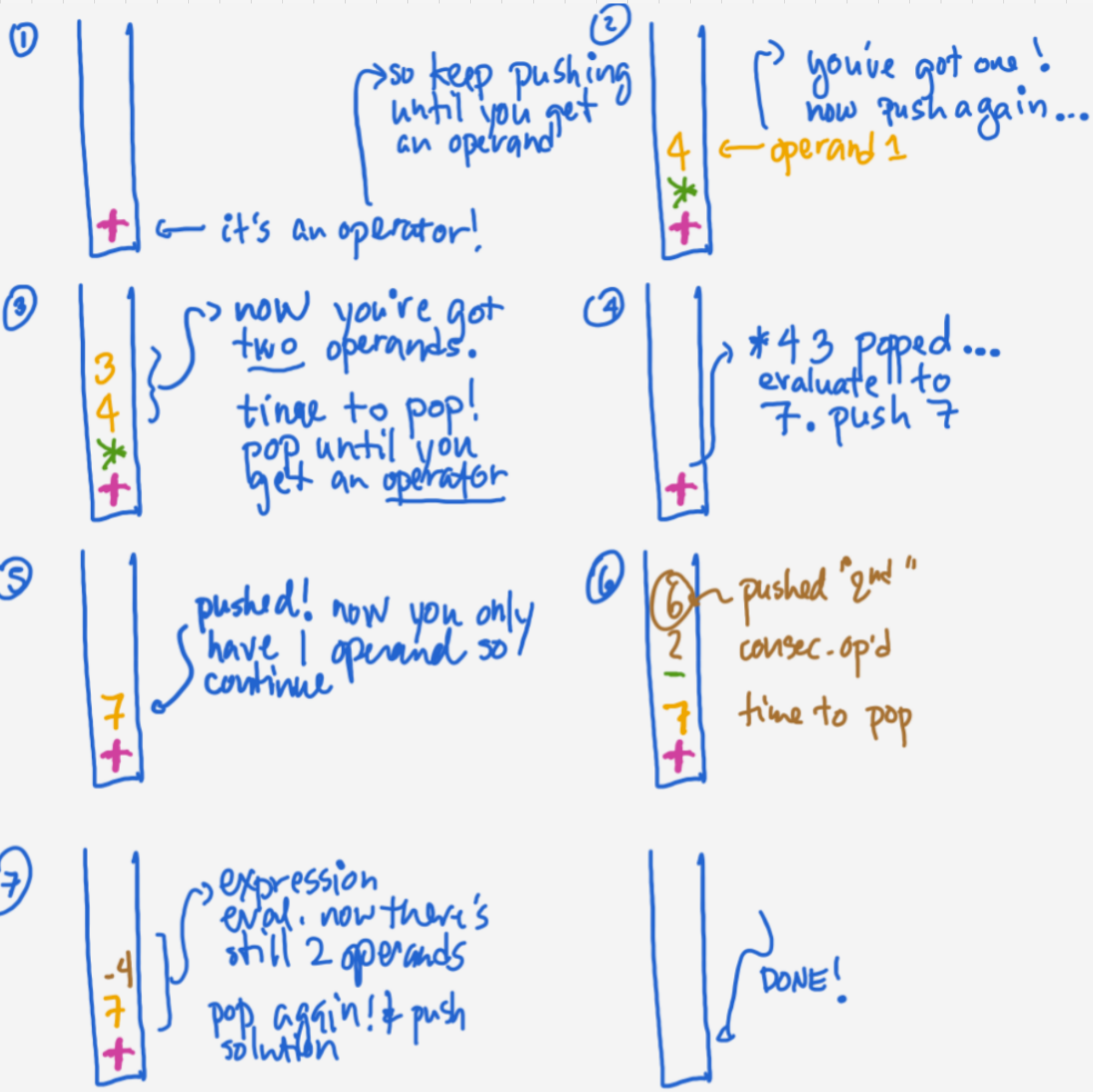






Above, I actually parsed and added operands at once, but in reality, you would add operations.

So, how do you do it with stacks?



So now you know the basic concept! Pseudocode:

1. create an empty stack
2. keep track of if you're on second operand
3. start loop through input:
   a.) if an operator: push it on stack
   you're on operand 1

```
def evalprefix ( input ):
    ops = set ( [ '+','-','*','/'] )
    stack = []
    first_operand = True
    for i in input :
        if i in ops :
            stack.append (i)
            first_operand = True
        elif first_operand
            stack.append (i)
            first_operand = False
        else:
            op2 = i
            op1 = stack.pop()
            op = stack.pop()
            stack.push (eval (op, op1, op2)
    return stack.pop()
```