*Exercise 1:*

a, **Algorithm**:

* Intialize count as 0;
* Intalize a node pointer, current== head
* Use the following while is not NULL:
  + - Growth count;
    - Current=current=>next(move current to next node);
* Return count;

B, **Algorithm**:

* Initialize count as 0;
* Initialize a node pointer, current == head;
* Loop through linked list:
  + If count == k:
    - return the current node;
  + else if count > k:
    - return None;
  + current = current -> next (move current to next node);
* Growth count

C, **Algorithm**:

* Initialize count as 0;
* Initialize node pointer, current == head;
* Loop through the linked list:
  + if count == k:
    - Allocate memory for a new node;
    - current node pointer -> new node;
    - new node pointer -> next node;
  + current = current -> next (move current to next node);
  + Increment count;

D, **Algorithm**

* Initialize count as 0;
* Initialize node pointer, current == head;
* Keep track of the previous position of the current node
* Loop through the linked list:
  + if count == k - 1:
    - previous -> next node of current node;
    - change current to the next node;
    - free memory for previous current node;
    - break from loop;
  + previous node -> current;
  + current = current -> next (move current to next node);
  + Rise count;

e, Algorithm

* Initialize M pointer pointing to a designated node;
* Initialize a new linked list with the pointer P pointing to the first node;
* Initialize current node = head;
* Loop through the P linked list:
  + if next node == NULL:
    - return last node of P = current node;
  + current node -> next node;
* last node of P -> M;

1. **Algorithm**:

* Find the node pointed to by the M pointer;
* Loop through the L linked list:
  + If current node is the node pointed to by the M pointer:
    - Initialize the next node of the current node to be a head of a new linked list;
    - current node -> NONE;

1. **Algorithm**:

Set cursor prev = None;

Set the cursor current = the first button of the list;

* Loop through the L linked list:
* Set the next\_node = next pointer of the current node
* Place the next cursor of the current button pointing to the previous button (prev)
* Set cursor prev = current button
* Set cursor current = next node (next\_node)

Set the last button as the first button of the list (self.head = prev)

Exercise 2:

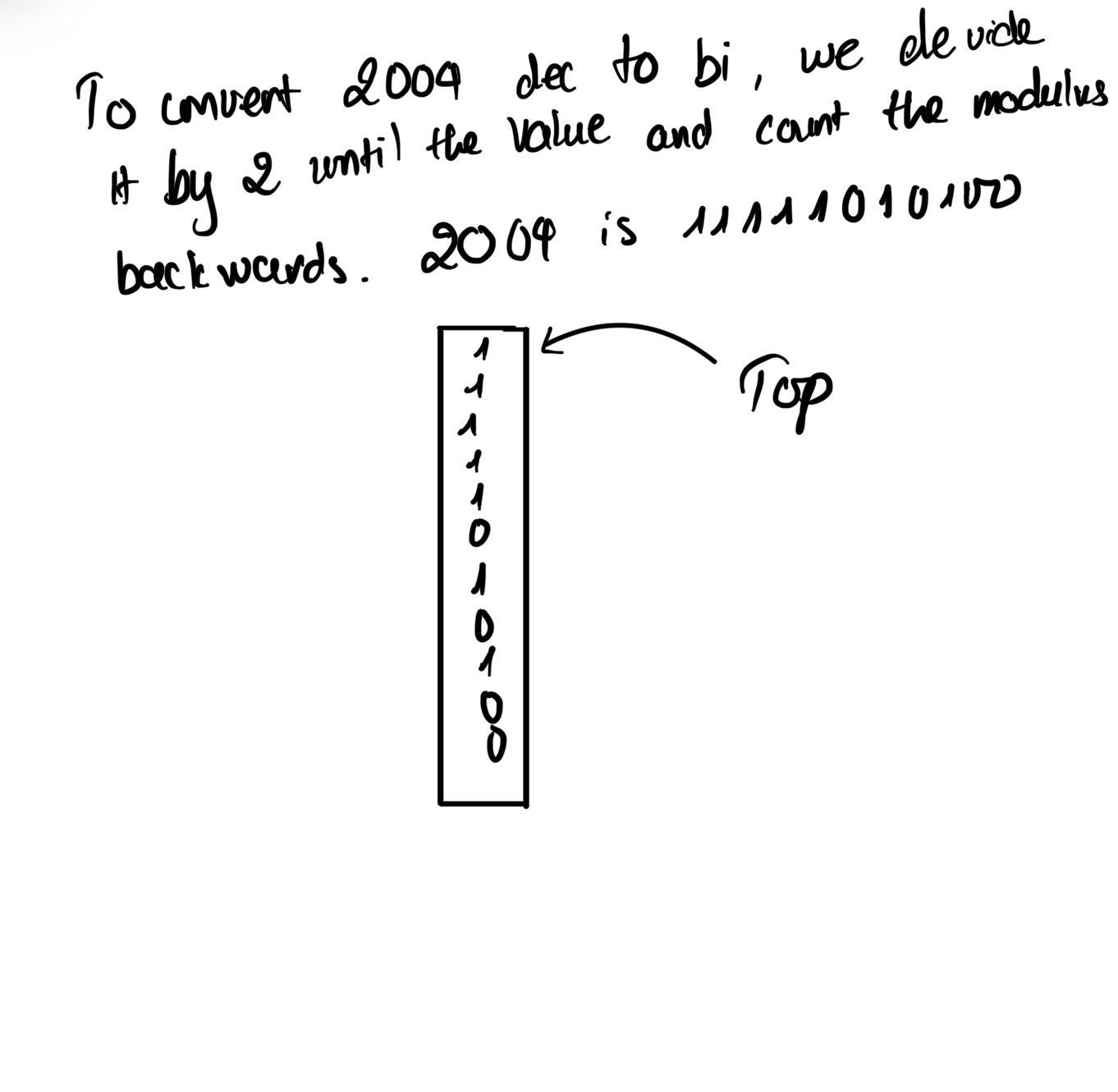
1.

A, 3 2 5 6 4 1= I I I 0 0 I I 0 I 0 0 0

B, 1 5 4 6 2 3= I 0 I I 0 0 I I I I 0 0 I 0 0 0

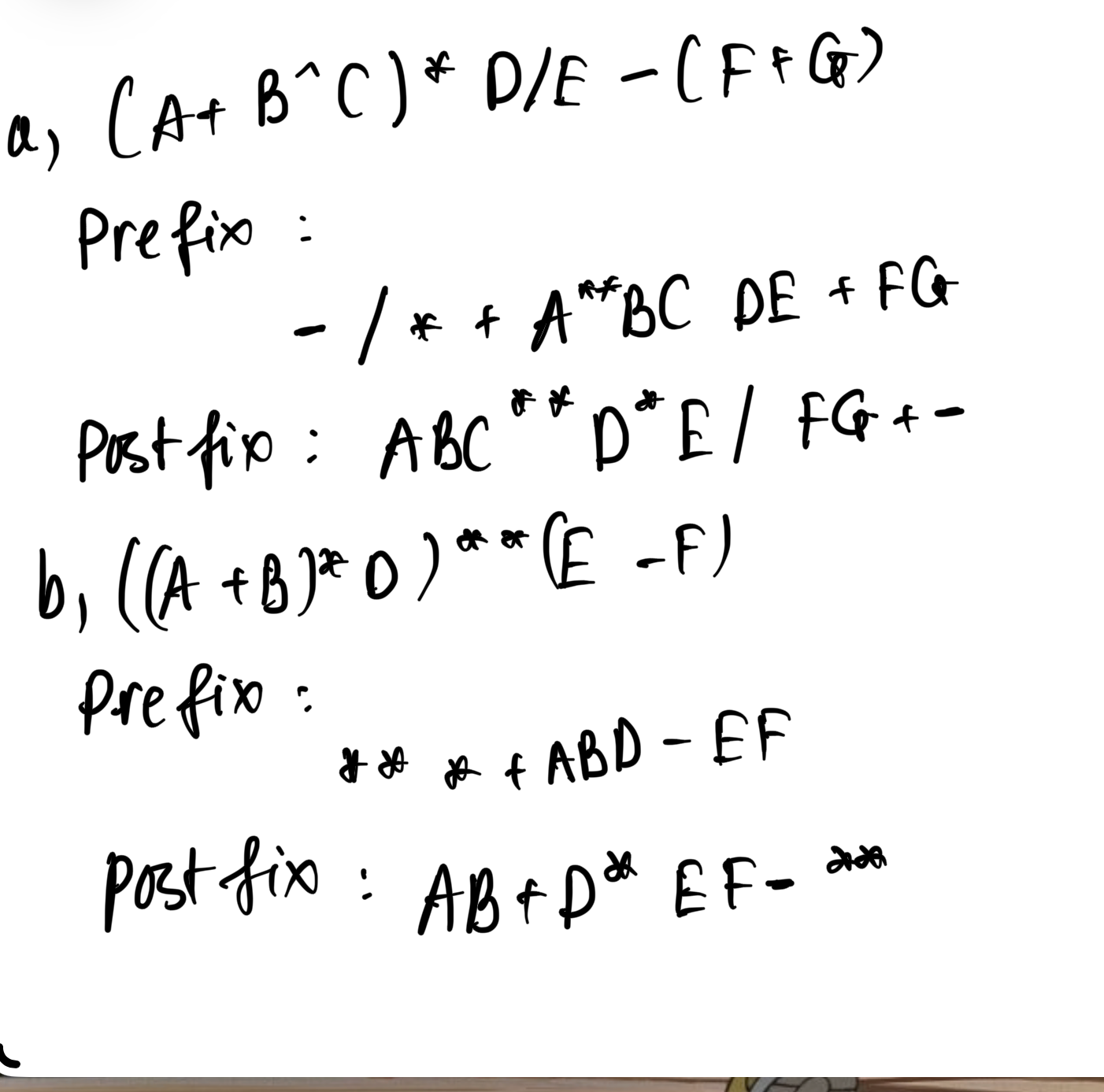
C, 2 4 5 3 1 6= I I 0 I I 0 I 0 0 0 I 0

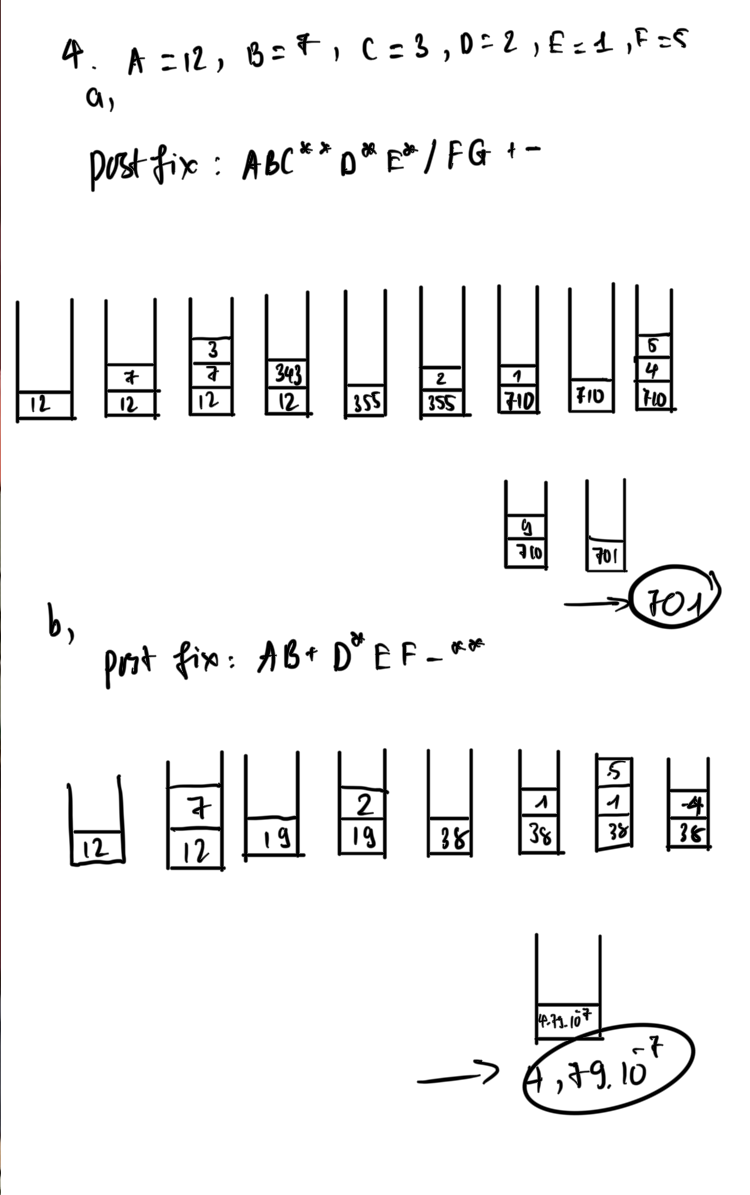
2.



We divide it then take the modulus into the stack, when we can’t divide anymore, we start taking values from the stack and put it into a full number.

3.





**Exercise 5:**

Algorithm:

Set an empty stack to store characters.

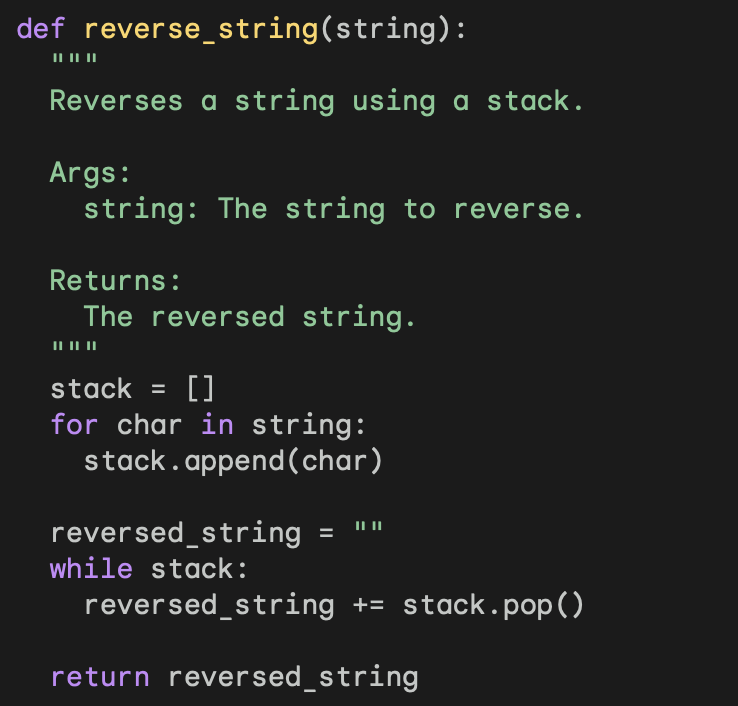
Browse through each character in the initial string:

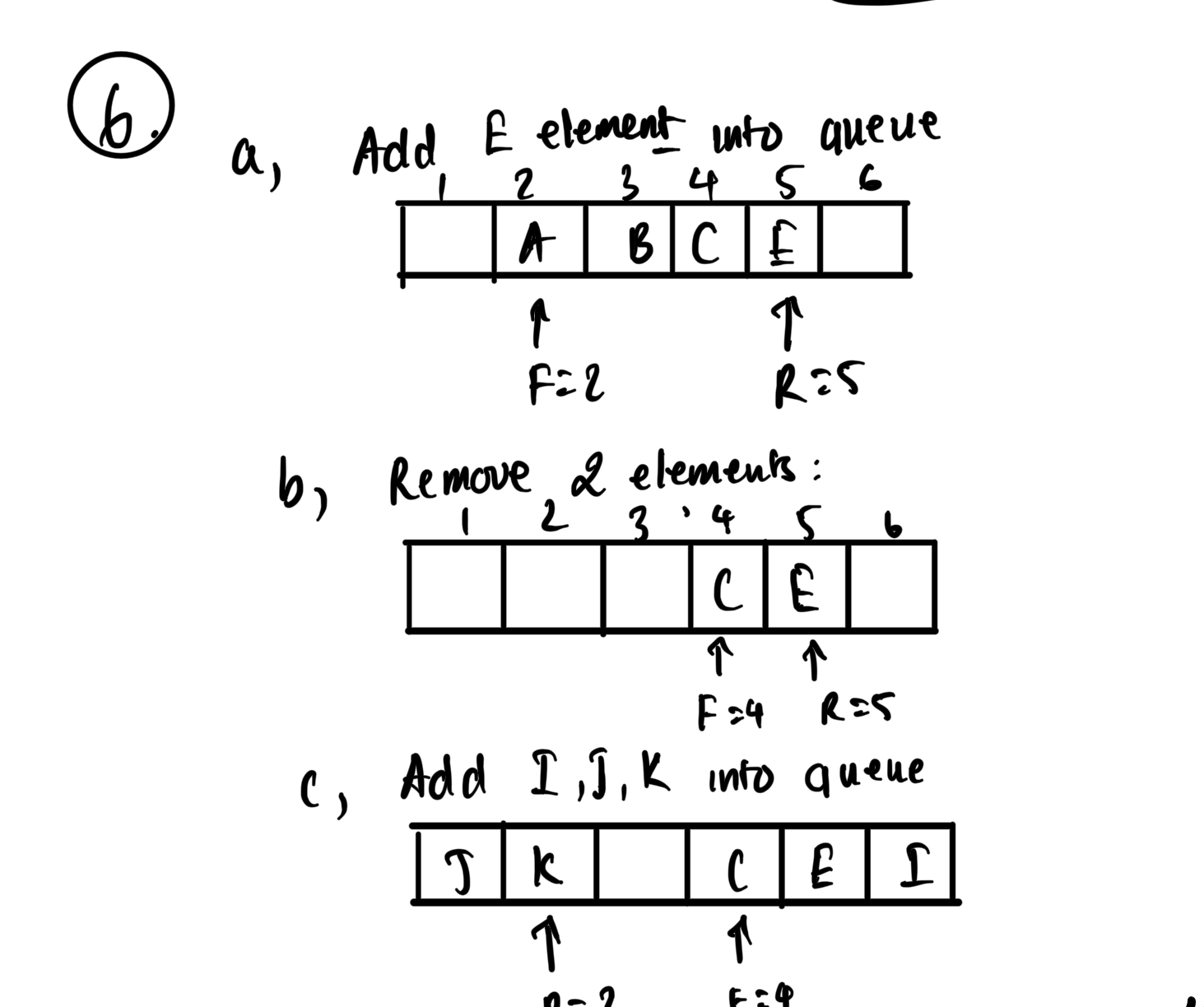
Push the character into the stack.

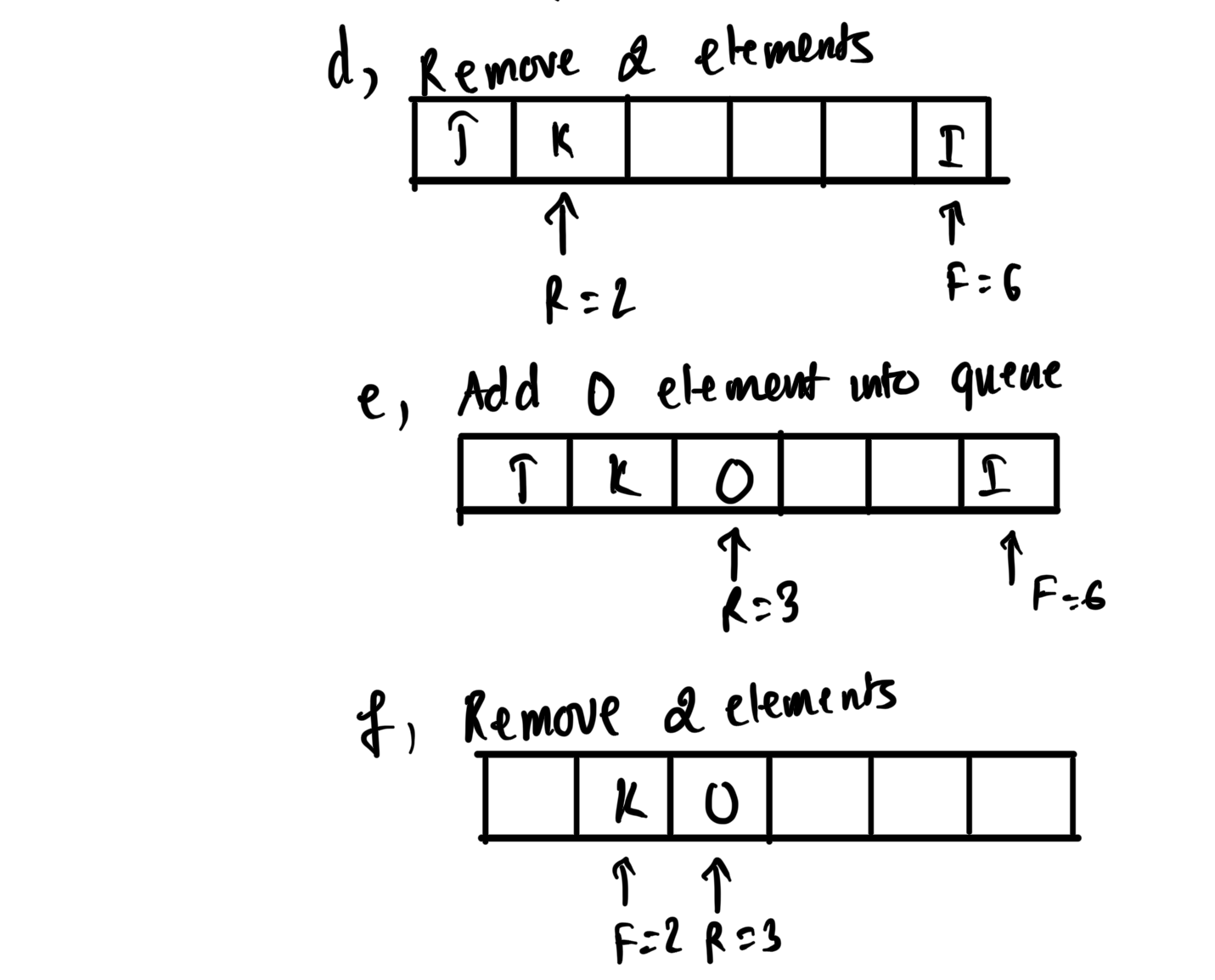
Initialize an empty result string.

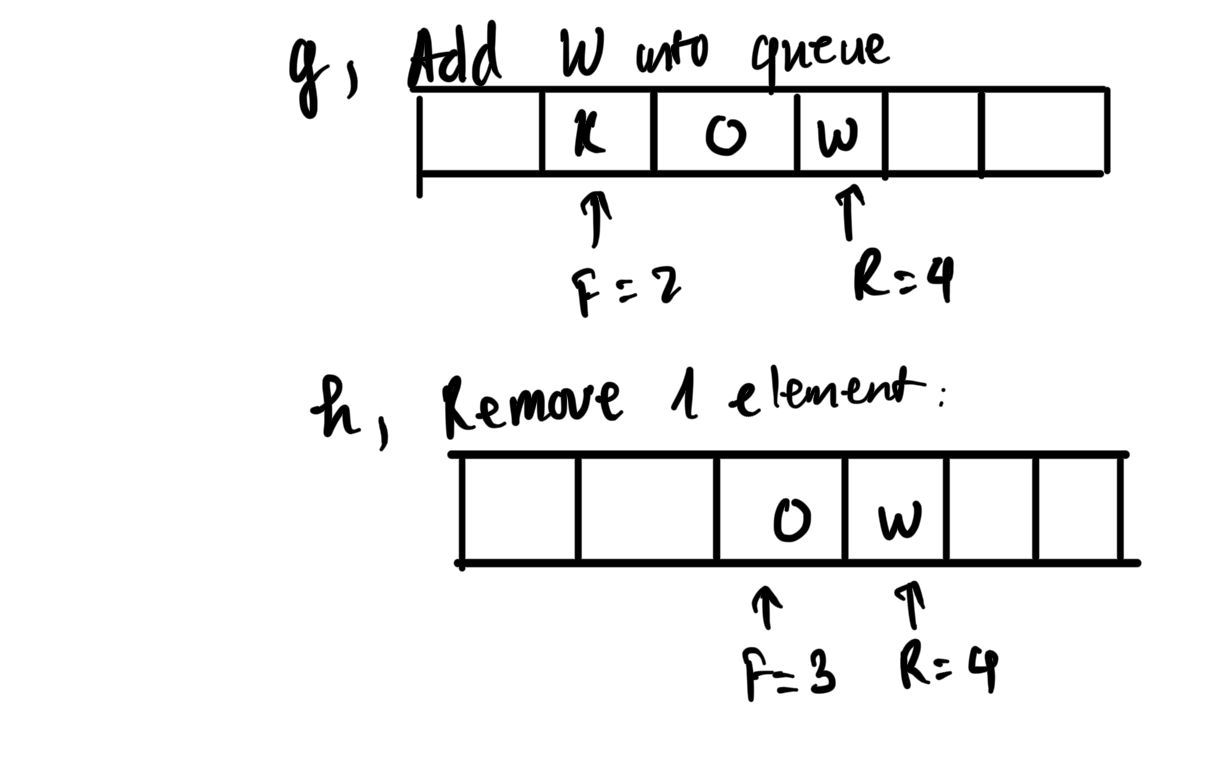
Take the characters from the stack and add them to the result string.

- Python program for reversing a string with stack:







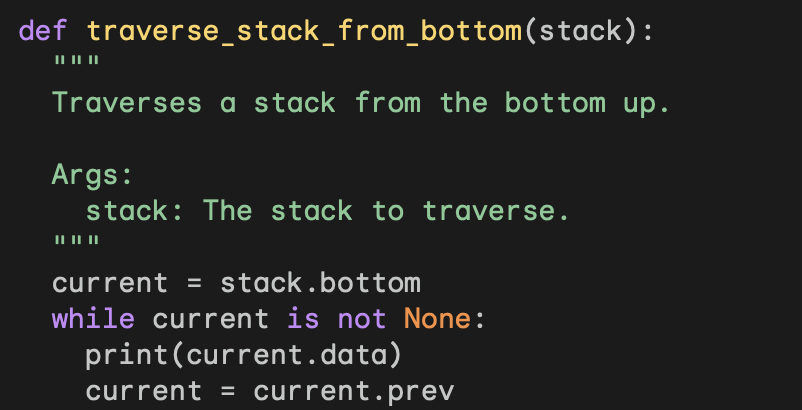
**Exercise 7:**

A) Browse the stack from the bottom:

* Set the current pointer at the BT pointer (bottom of the stack).

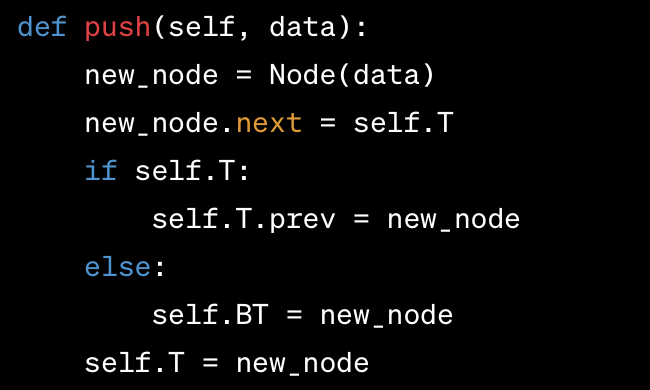
Loop through each element of the stack:

* Handle the current element (e.g., print the value of the current node).
* Adjust the current pointer to move to the previous element (if any).



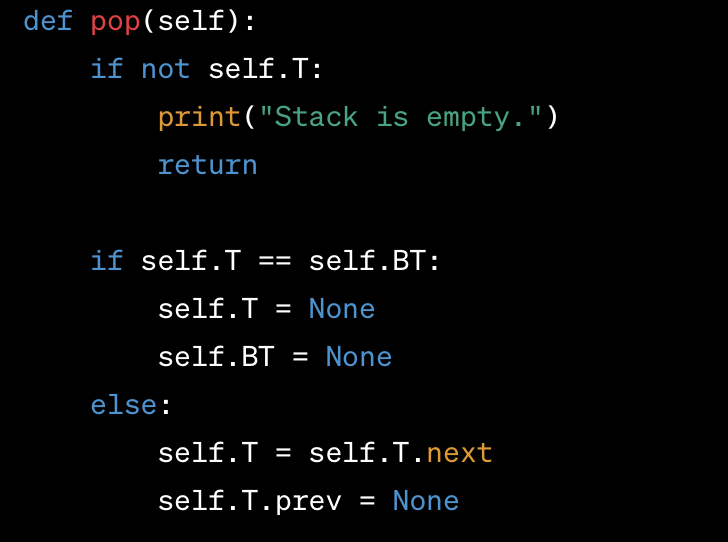
B) Add an element to the stack:

* Create a new node with the new value.
* Place the next cursor of the new node pointing to the current vertex element (pointer T).
* Place the prev cursor of the current vertex element (cursor T) pointing to the new node.
* Set the T pointer to the new button.



C) Remove an element from the stack:

* Handle empty stack case (BT pointer points to None).
* Handle the case where the stack has one element (the BT pointer and the T pointer point to the same node).
* Handle the case where the stack has many elements (adjust the next and prev cursors of related buttons).



To remove an element from the stack, we can use the pop() function of the join list. stack.pop()