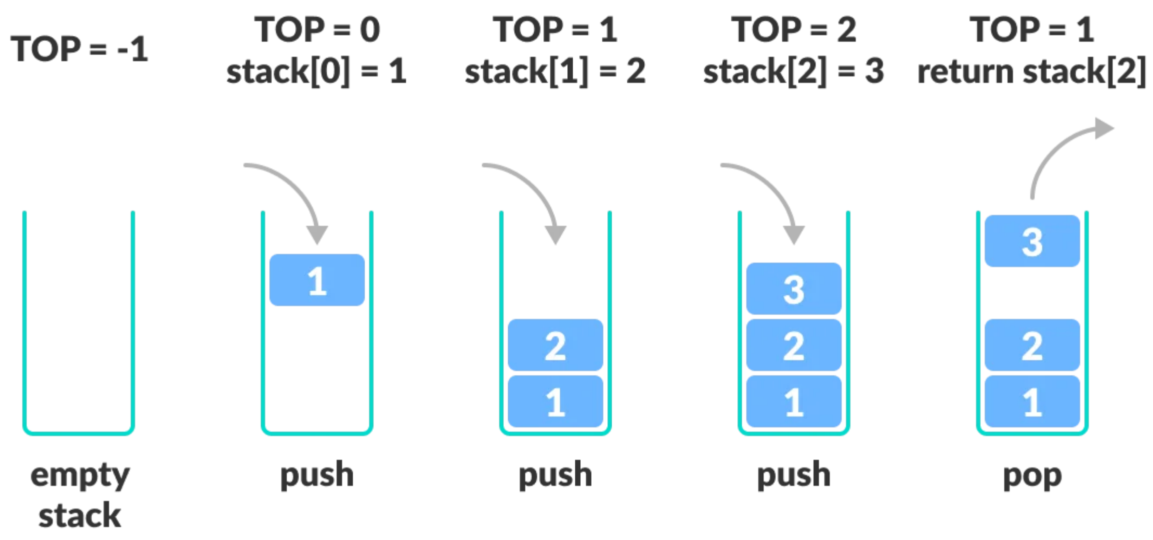
1. Describe Stack and Queue with illustrations.

1a) A stack is a data structure that is a FILO (first in last out) / LIFO (last in first out). This means that each element appended onto the stack [push() function] will then be the first element in line to be removed from the structure [pop() function]. In other words pop() operates on the tail of the linked list, while the head of the linked list remains in place and untouched unless all other elements in the stack are untouched.

A stack differs from a queue in that the only element a pointer is keeping track of is the "top" element since that is the element which will be popped as well as the element that will have any new element pushed onto the stack on top of it.

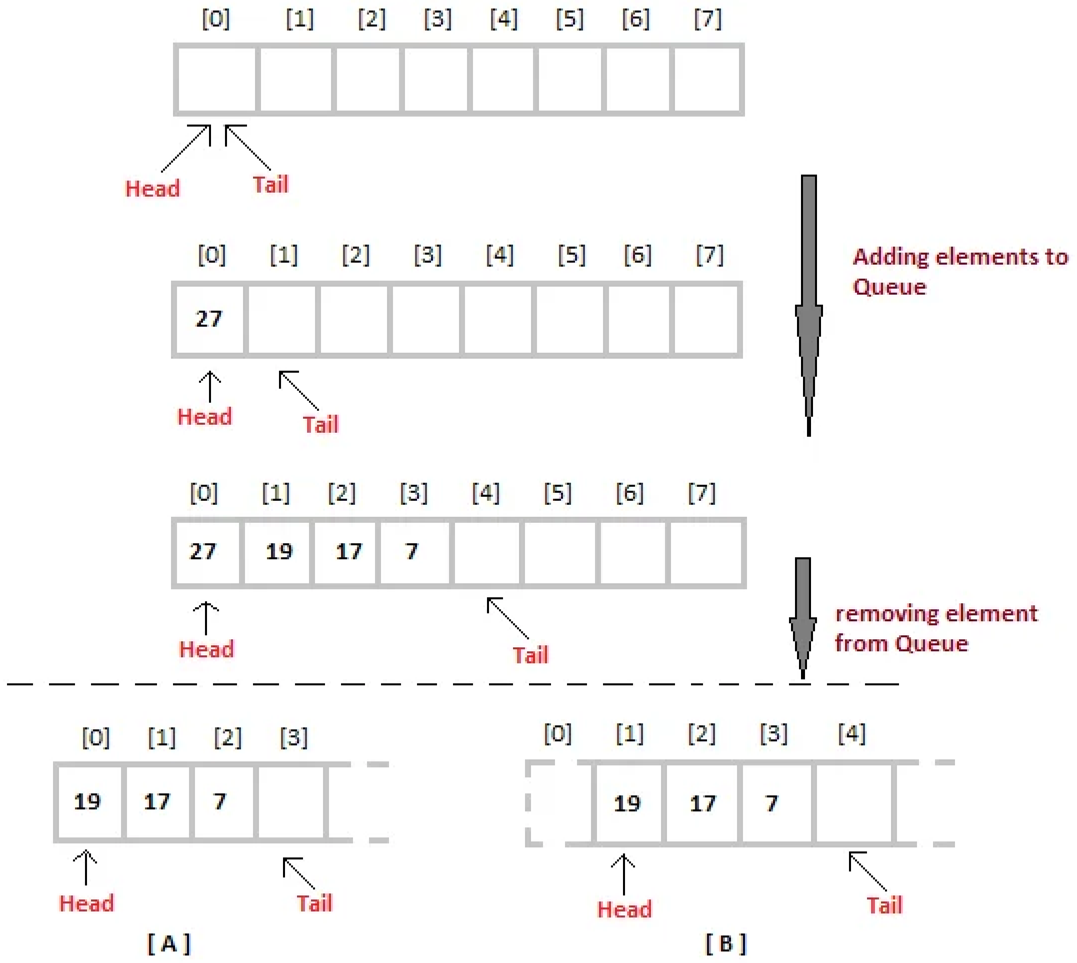
A stack differs from several other types of data structures in that it has a predefined capacity that cannot be altered once the stack is created. This condition must be checked before each individual element is pushed to the stack. This condition is referred to as the "stack overflow" requirement.



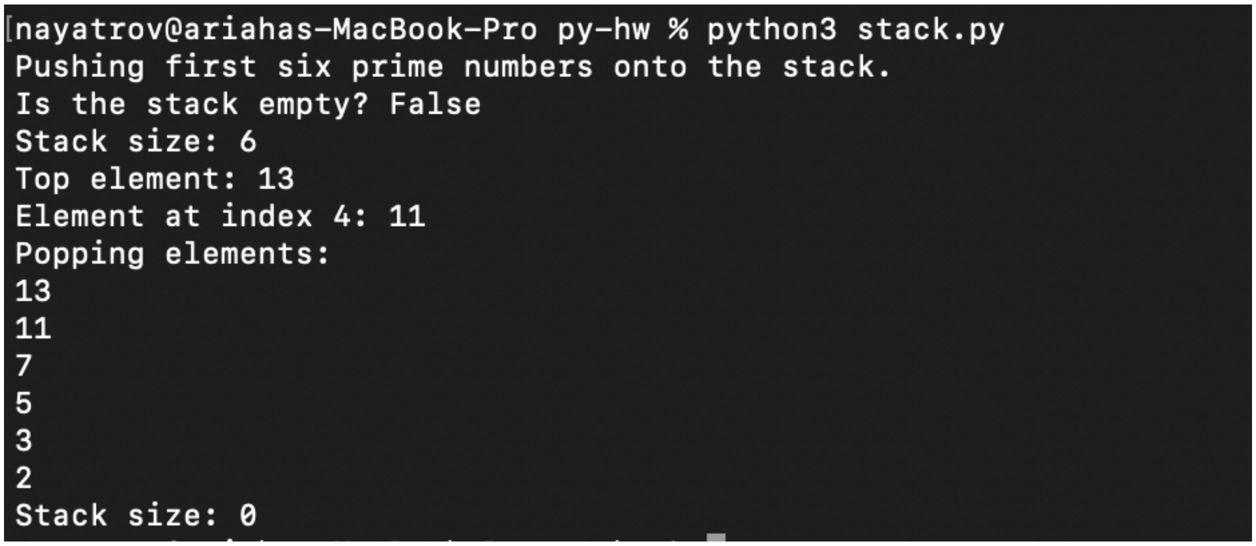
1b) A queue is a data structure that is a FIFO (first in first out) / LILO (last in last out). This means that each element appended onto the queue [push() function] will then be the element last in line to be removed from the queue [via pop() function]. In other words pop() operates on the head of the linked list, removing it and setting the next element which the current head points to as the new head. It is the opposite of a stack in which the head remains unchanged unless the head is the only element in the structure.

One unique thing about the queue data structure is that the linked list data structure is created using the queue interface, and has all the functionality that a queue has, plus more. It is important to note that a queue is not a linked list, but a linked list is a queue.

A queue differs from a stack in the method pointers are used and which elements are kept track of. Where a stack only points to the “top” element of the stack data structure, a queue uses pointers to track the head (front) of the queue, as well as a pointer attached to the end of the queue which points to “null” to signify the end of the queue.

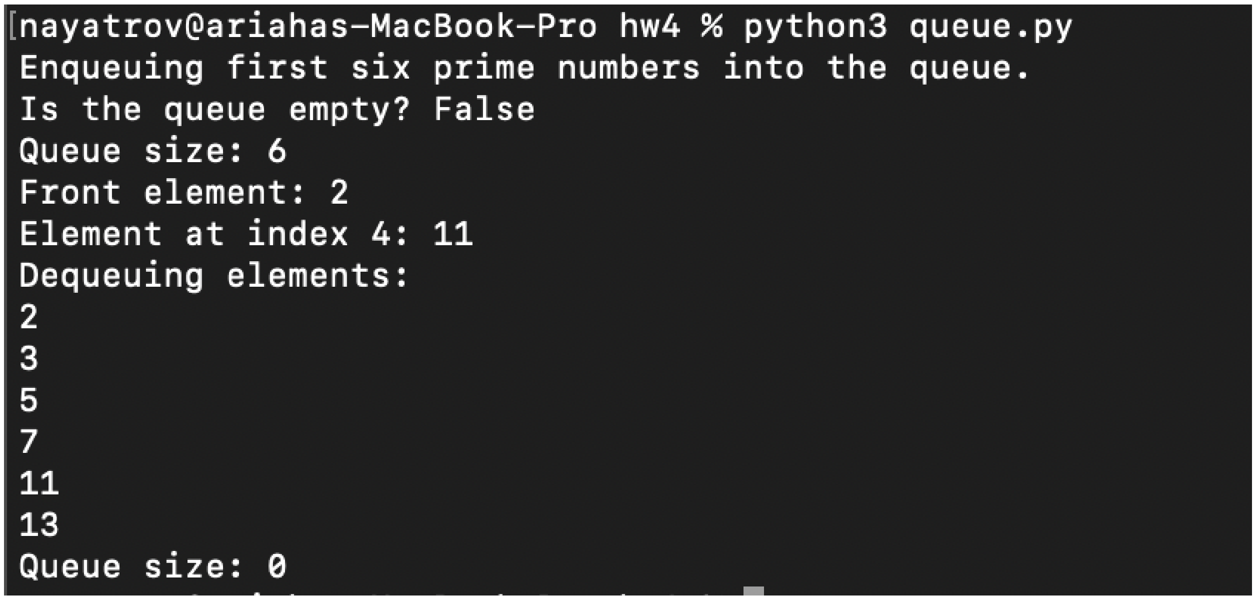


2. Write code to implement a stack and its basic operations. (Code included in stack.py)
   [output for stack.py below]



```
[nayatrov@ariahas-MacBook-Pro py-hw % python3 stack.py
Pushing first six prime numbers onto the stack.
Is the stack empty? False
Stack size: 6
Top element: 13
Element at index 4: 11
Popping elements:
13
11
7
5
3
2
Stack size: 0
```

3. Write code to implement a queue and its basic operations. (Code included in queue.py)
   [output for queue.py below]



```
[nayatrov@ariahas-MacBook-Pro hw4 % python3 queue.py
Enqueuing first six prime numbers into the queue.
Is the queue empty? False
Queue size: 6
Front element: 2
Element at index 4: 11
Dequeuing elements:
2
3
5
7
11
13
Queue size: 0
```