① a) what are the stacks, and what are the main charecteristics of a stack.

⇒ STACK (abstract datatype that serves a collection of a data)

→ LIFO Rule

→ Top is used to keep or remove the elements

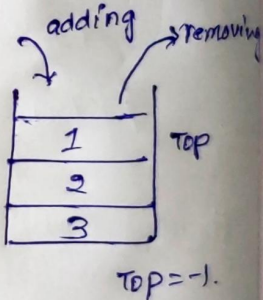
→ main operations

Push, pop → user defined

↓ adding ↓ Removing

Is empty
Is full
} → conditions to be checked.

Peek (or) Top (to know the top/last element)



⇒ WORKING OF A STACK

Stack is empty then top value = -1.

CHARECTERISTICS :-

→ Implemented through array / linked list.
→ Towers of Hanoi, tree traversals
→ over flow - stack full
→ under flow - stack empty.

Applications:-

→ Traversing
→ Browsing.

b) what are the key operations that can be performed on the stack are.

→ List
→ Modules → collections
→ queue.

List → stacks implementing lists.

ex:-
```
stack = [ ]
stack.append(1)
stack.pop
print(len(stack)==0)
print(len(stack))
```

collections module

↳ dequeue - double ended queue.
- class
- append()
- pop()

ex:-

~~import collections~~
~~stack = collections.dequeue()~~
~~print(len(stack)-1)~~

```
from collections import *
s = dequeue(maxlen=3)
print(s)
# adding elements
s.append(12)
s.append(13)
s.append(14)
if len(s) == s.maxlen:
        print("stack is full")
print("The popped element is", s.pop())
print("The stack is", s)
print("The size of the stack is", len(s))
print("The Top element is:", s[-1])
```

Output :-

dequeue (maxlen = 3)
stack is full
The popped element is 14
The stack is dequeue [[12,13], maxlen = 3)
The size of the stack is: 2.
The top element of the stack is 13.

2) Last In first out principle. (LIFO):

The element which was inserted at first will go to the bottom of the stack and the last In elements will go to first. and the deletion and adding should be done from the top position.

example program.

```
stack = [ ]
stack.append [1]
stack.append (2)
stack.append (3)
#print (stack[-1])
print ("The Top element is", stack[-1])
stack.pop()
if not stack:
    print ("The stack is empty")
print("The length of the stack is" (len(stack))
```

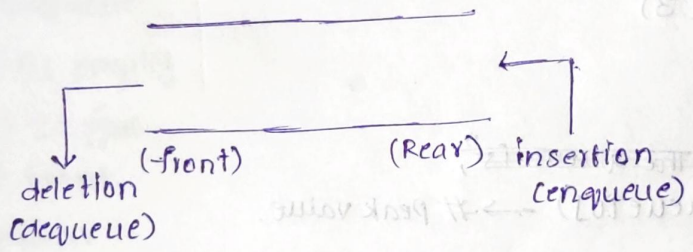
Output :-

The Top element is: 3
The length of the stack is 2.

3) Discuss the principle of queue datastructure? Implement queue datastructure by using python program.

A) Queue :-

It performs the First in First out principle. (FIFO) The deletion and insertion can be done from the each end so that the First inserted element can be removed from the first. or starting of the queue. Queues are the linear datastructures which are dynamic. The front and Rear positions will be the insertion and deletion ~~parts~~ points in queues.



Basic operations :-

* Enqueue (insertion)
* Dequeue (Deletion)
* Is Full
* Is Empty
* Front (start element)
* Rear (End element)

a) enqueue (insertion)

→ check weather the queue is full or not.
→ for the starting element, set the value of front to zero (0).
→ Rear pointer is incremented by 1
→ always insert the value at rear position.

b> Dequeue (deletion)

→ check weather the queue is empty (or) not.

→ Remove the value pointed by front.

→ The front pointer is incremented by 1 for every dequeue operation.

→ For the last element, the front and rear pointer reset to -1.

Example program:-

```
queue = [ ]
queue.append(1)
queue.append(2)
queue.append(3)
queue.pop(0)
if queue:
    print("The queue is", queue[0]) → # peak value.
if not queue:
    print("Queue is empty")
# print(len(queue))
```

Output:-

```
print("The length of the queue is:", len(queue))
```

Output:-

```
2
The length of the queue is: 2.
```

5> What are the circular queues, how do they differ from regular queues?

A> The queues are the linear datastructures which are used to do the combination operations. It performs the first in first out principle and comming to circular queues these are the extended versions of the linear queue. Ring buffering is possible in this circular queues. Once Rear reaches the last index the insertion is not possible in the normal queue. To overcome this situation we use circular queue.

Basic Operations :-

1> enqueue

2> Dequeue

3> Is empty

4> Is full

5> Front

6> Rear.

working structure :

- front & rear = -1.

Enqueue :-

① check queue is full or not.

② For start element set front value to Zero 0.

③ circularly increment the rear pointer by 1.

④ Assign the value at rear position.

Dequeue :-

① check the queue is empty or not.

② Remove the value pointed by front.

③ Front value is circularly incremented by 1.

④ The front & Rear pointer reset to -1.