**Queue:**

from queue import Queue  
  
q=Queue(maxsize=3)  
print(q.qsize())  
q.put('a')  
q.put('b')  
q.put('c')  
print("\nFull: ",q.full())  
print("\n’Elements dequeued from queue’)  
print(q.get())  
print(q.get())  
q.put(1)  
print("\nEmpty: ",q.empty())  
print("\nFull: ",q.full())

**Stack:**

stack=[]  
def push():  
 element=input("enter the number")  
stack=[]  
def push():  
 element=input("enter the number")  
 print(stack)  
  
def pop():  
 if not stack:  
 print("the list is empty")  
 else:  
 e=stack.pop()  
  
while True:  
 print("choose 1 to push , 2 to pop and 3 to quit")  
 choose = int(input())  
 if choose==1:  
 push()  
 elif choose ==2:  
 pop()  
 elif choose==3:  
 quit()  
 else:  
 print("choose the correct number")

**Fibonacci:**

def fib(n):  
 if n<0:  
 return "!not defined"  
 elif n==0 or n==1:  
 return n  
 else:  
 return fib(n-1) + fib(n-2)  
  
n=int(input("enter the number:"))  
print("fibonacci ,end=" ")  
for i in range(0, n):  
 print(fib(i), end=" ")

**Factorial:**

import sys  
sys.recursionlimit(10\*\*6)  
def factorial(n):  
 if n<0 or int(n)!=n:  
 return "!not defined"  
 if (n==0 or n==1):  
 return n  
 else:  
 return n\*factorial(n-1)  
n=int[input("enter the number:")]  
print("factorial of given number";, factorial(n))

**Tower of honai**

def tower\_of\_honai(disks, target, source, auxiliary):  
 if (disks==1):  
 print('move disk 1 from rod{} to rod{},'.format(source,target))  
 return  
 tower\_of\_honai(disks-1, target, source, auxiliary)  
 print('move disks{} from rod{} to rod{},'.format(disks, source, target))  
 tower\_of\_honai(disks-1, target, source. auxiliary)  
  
disks=int(input("enter the number:"))  
tower\_of\_honai(disks,'A','B','C')

**DLL:**

**class** Node:  
 **def** \_\_init\_\_(self, data):  
 self.data = data  
self.prev = **None  
  
class** DoublyLinkedList:  
 **def** \_\_init\_\_(self):  
 self.first=**None** self.last=**None  
  
 def** get\_node(self,index):  
 **for** i **in** range(index):  
 **return None** current=current.next  
 **return** current  
  
 **def** insert\_after(self,ref\_node,new\_node):  
 new\_node.prev=ref\_node  
 **if** ref\_node.next **is None**:  
 self.last=new\_node  
 **else**:  
 new\_node.next=ref\_node.next  
 ref\_node.next=new\_node  
  
 **def** insert\_before(self,ref\_node,new\_node):  
 new\_node.next=ref\_node  
 self.first=new\_node  
 **else**:  
 new\_node.prev=ref\_node.prev  
 new\_node.prev.next=new\_node  
 ref\_node.prev=new\_node  
  
 **def** insert\_at\_beg(self,new\_node):  
 **if** self.first **is None**:  
 self.last=new\_node  
 **else**:  
 self.insert\_before(self.first, new\_node)  
  
 **def** insert\_at\_end(self,new\_node):  
 **if** self.last **is None**:  
 self.first=new\_node  
 **else**:  
 self.insert\_after(self.last,new\_node)  
  
 **def** remove(self,node):  
 self.first=node.next  
 **else**:  
 node.prev.next=node.next  
  
 **if** node.next **is None**:  
 **else**:  
 node.next.prev=node.prev  
  
 **def** display(self):  
 current=self.first  
 **while** current:  
 print(current.data,end=**' '**)  
 current=current.next  
  
  
a\_list=DoublyLinkedList()  
print(**"Menu"**)  
print(**"insert <data> after <index>"**)  
print(**"insert <data> before <index>"**)  
print(**"insert <data> at beg"**)  
print(**"insert <data> at end"**)  
print(**"remove <index>"**)  
print(**"quit"**)  
**while True**:  
 print(**"the list: "**,end=**''**)  
 a\_list.display()  
 print()  
 do=input(**"which operation do you like to perform? "**).split()  
 operation=do[0].strip().lower()  
 position=do[3].strip().lower()  
 new\_node=Node(data)  
 suboperation=do[2].strip().lower()  
 **if** position ==**"beg"**:  
 a\_list.insert\_at\_beg(new\_node)  
 **elif** position==**"end"**:  
 a\_list.insert\_at\_end(new\_node)  
  
 **else**:  
 index=int(position)  
 ref\_node=a\_list.get\_node(index)  
 **if** ref\_node **is None**:  
 print(**"No such index."**)  
 **continue**  a\_list.insert\_after(ref\_node,new\_node)  
 **elif** suboperation==**"before"**:  
 a\_list.insert\_before(ref\_node,new\_node)  
 **elif** operation==**"remove"**:  
 index=int(do[1])  
 **if** node **is None**:  
 print(**"no such index."**)  
 **continue**  
 **elif** operation==**"quit"**:  
 **break**