In [ ]:

```
def fib(n):
    global arr, cnt
    if n <= (len(arr)-1):
        return arr[n]
    else:
        arr.append(fib(n-1) + fib(n-2))
        return arr[n]
```

In [ ]:

```
arr=[]
arr.append(0)
arr.append(1)
cnt = 1
n = 3
print(n, 'Nth of fibonacci is',fib(n))
```

In [ ]:

```
arr
```

In [ ]:

```
def fib(n):
    if n == 0:
        return 0
    elif n == 1:
        return 1
    else:
        return fib(n-1) + fib(n-2)
```

In [ ]:

```
fib(6)
```

In [ ]:

```
fib(4)
```

In [ ]:

```
class S_Node:
    def __init__(self,data):
        self.data = data
        self.link = None

class Link_list:
    def __init__(self):
        self.head = None
    def print_list(self):
        t = self.head
        while t:
            print(t.data)
            t = t.link
l = Link_list()
l.head = S_Node(12)
l.head.link = S_Node(34)
l.head.link.link = S_Node(26)
l.head.link.link.link = S_Node(67)
l.head.link.link.link.link = S_Node(98)


l.print_list()
```

In [ ]:

```
def fun(head, n):
    t = head
    if (head == None):
        return
    else:
        i = 1
        while(i != n):
            t = t.link
            i += 1
        t1 = head.data
        head.data = t.data
        t.data = t1
```

In [ ]:

```
fun(l.head,3)
```

In [ ]:

```
l.print_list()
```

In [ ]:

```
class S_Node:
    def __init__(self,data):
        self.data = data
        self.link = None

class Link_list:
    def __init__(self):
        self.head = None
    def print_list(self):
        t = self.head
        while t:
            print(t.data)
            t = t.link
l = Link_list()
l.head = S_Node(12)
l.head.link = S_Node(121)
l.head.link.link = S_Node(112)
l.head.link.link.link = S_Node(111)
l.head.link.link.link.link = S_Node(222)


l.print_list()
```

In [ ]:

```
def add_l(head):
    t = head
    if head == None:
        return
    else:
        while t.link != None:
            t.data = t.data + t.link.data
            t = t.link
```

In [ ]:

```
add_l(l.head)
l.print_list()
```

In [ ]:

```
class stack:
    def __init__(self):
        self.item = []
    def display(self):
        return self.item
    def length_item(self):
        return len(self.item)
    def is_empty(self):
        return self.item == []
    def push(self,item):
        self.item.append(item)
    def pop(self):
        return self.item.pop()
    def peek(self):
        return self.item[len(self.item)-1]
    def size(self):
        return len(self.item)
```

In [ ]:

```
s = stack()
s.push(2)
s.push(4)
s.push(5)
s.push(1)
s.push(12)
s.push(7)
```

In [ ]:

```
s.display()
```

In [ ]:

```
def s1(s):
    if s.is_empty():
        return
    else:
        s2 = stack()
        while not s.is_empty():
            s2.push(s.pop())
            s2.push(s.pop())
            s2.push(s2.pop()*s2.pop())
        return s2.display()
```

In [ ]:

```
s1(s)
```

In [ ]:

```

class stack:
    def __init__(self):
        self.item = []
    def display(self):
        return self.item
    def length_item(self):
        return len(self.item)
    def is_empty(self):
        return self.item == []
    def push(self,item):
        self.item.append(item)
    def pop(self):
        return self.item.pop()
    def peek(self):
        return self.item[len(self.item)-1]
    def size(self):
        return len(self.item)
```

In [ ]:

```
s = stack()
s.push(7)
s.push(12)
s.push(1)
s.push(5)
s.push(4)
s.push(2)
```

In [ ]:

```
s.display()
```

In [ ]:

```
def s1(s):
    if s.is_empty():
        return
    else:
        s2 = stack()
        while not s.is_empty():
            s2.push(s.pop())
            s2.push(5)
            s2.push(s2.pop()+s2.pop())
        while not s2.is_empty():
            s.push(s2.pop())
        return s.display()

```

In [ ]:

```
s1(s)
```

In [ ]:

```
class Queue:
    def __init__(self):
        self.item = []
    def display(self):
        return self.item
    def peek(self):
        return self.item[-1]
    def is_empty(self):
        return self.item == []
    def enqueue(self,item):
        self.item.insert(0,item)
    def dequeue(self):
        return self.item.pop()
    def size(self):
        return len(self.item)
```

In [ ]:

```
q = Queue()
for i in range(1,6):
    q.enqueue(i*i)
print(q.display())
q.dequeue()
print(q.display())
```

In [ ]:

```
def fun(n):
    q = Queue()
    if n == 0:
        return 1
    else:
        for i in range(1,n+1):
            q.enqueue(i)
        q.display()
        t = 1
        while not q.is_empty():
            t = t * q.dequeue()
        return t
```

In [ ]:

```
fun(5)
```

In [ ]:

```
def fun1(n):
    q = Queue()
    if n == 0:
        return 1
    else:
        for i in range(1,n+1):
            q.enqueue(i*i)
        q.display()
        t = 1
        while not q.is_empty():
            t = t + q.dequeue()
        return t
```

In [ ]:

```
fun1(4)
```

In [ ]:

```
class S_Node:
    def __init__(self,data):
        self.data = data
        self.link = None

    def get_data(self):
        return self.data

    def set_data(self, data):
        self.data = data

    def get_next(self):
        return self.link

    def set_next(self, node):
        self.link = node
class Link_list:
    def __init__(self):
        self.head = None
    def print_list(self):
        t = self.head
        while t:
            print(t.data)
            t = t.link
l = Link_list()
l.head = S_Node(1)
l.head.link = S_Node(4)
l.head.link.link = S_Node(6)
l.head.link.link.link = S_Node(7)
l.head.link.link.link.link = S_Node(9)

l.print_list()
```

In [ ]:

```
def fun2(head, n):
    t = head
    if head == None:
        return
    else:
        i = 1
        while (i<n):
            val = t.get_data()
            print("val",val)
            t = t.get_next()
            head.set_data(val)
            print('set_data',head.data)
            i = i+1
        t1 = head.get_data()
        t.set_data(t1)
        print('t',t.get_data())
print("__________________________________")
```

In [ ]:

```
fun2(l.head, 4)
l.print_list()
```

In [ ]:

```
fun2(l.head, 5)
l.print_list()
```

In [ ]:

```
fun2(l.head, 3)
l.print_list()
```

In [ ]:

```
class stack:
    def __init__(self):
        self.item = []
    def display(self):
        return self.item
    def length_item(self):
        return len(self.item)
    def is_empty(self):
        return self.item == []
    def push(self,item):
        self.item.append(item)
    def pop(self):
        return self.item.pop()
    def peek(self):
        return self.item[len(self.item)-1]
    def size(self):
        return len(self.item)
```

In [ ]:

```python
def proce(ip_s):
    s2 = stack()
    cnt = 0
    while not ip_s.is_empty():
        t = ip_s.pop()
        for i in t:
            cnt += 1
        print(cnt)
        s2.push(str(cnt)+t)
    return s2

```

In [ ]:

```python
ip_s = stack()
ip_s.push("India")
ip_s.push("Australia")
ip_s.push("England")
ip_s.push("SouthAfrica")
s2 = proce(ip_s)
```

In [ ]:

```python
s2.display()
```

In [ ]:

```python
#front : last item of a list
#rear : rear is the first index
class Deque:
    def __init__(self):
        self.item = []
    def is_empty(self):
        return self.item == []
    def display(self):
        return self.item
    def peek(self):
        return self.item[-1]
    def add_front(self,item):
        self.item.append(item)
    def add_rear(self,item):
        self.item.insert(0,item)
    def del_front(self):
        self.item.pop()
    def del_rear(self):
        self.item.pop(0)
    def size(self):
        return len(self.item)
    def front(self):
        return self.item[-1]
    def rear(self):
        return self.item[0]
```

In [ ]:

```
dq = Deque()
dq.add_front(10)
dq.add_front(20)
dq.add_front(30)
dq.add_front(40)
dq.add_front(50)

dq.add_rear(60)
dq.add_rear(70)
dq.add_rear(80)
dq.add_rear(90)
dq.add_rear(100)
```

In [ ]:

```
dq.display()
```

In [ ]:

```
dq.front()
```

In [ ]:

```
dq.rear()
```

In [1]:

```
class Queue:
    def __init__(self):
        self.item = []
    def display(self):
        return self.item
    def peek(self):
        return self.item[-1]
    def is_empty(self):
        return self.item == []
    def enqueue(self,item):
        self.item.insert(0,item)
    def dequeue(self):
        return self.item.pop()
    def size(self):
        return len(self.item)
```

In [2]:

```
def fun(Q):
    q1 = Queue()
    q2 = Queue()
    for i in range(len(Q.display())):
        t = Q.dequeue()
        if t>10:
            q1.enqueue(t)
        else:
            q2.enqueue(t)
    return q1,q2
```

In [3]:

```
Q = Queue()
Q.enqueue(12)
Q.enqueue(10)
Q.enqueue(15)
Q.enqueue(1)
Q.enqueue(20)
print(Q.display())
q1,q2 = fun(Q)
```

```
[20, 1, 15, 10, 12]
```

In [4]:

```
q1.display()
```

Out[4]:

```
[20, 15, 12]
```

In [5]:

```
q2.display()
```

Out[5]:

```
[1, 10]
```

In [ ]:

```
```