In [ ]:

```
n = int(input("Enter the no: "))
n1, n2 = 0, 1

if(n <= 0):
    print("Enter valid number")
elif(n == 1):
    print(n1)
else:
    for i in range(n):
        print(n1);
        a = n1 + n2
        n1 = n2
        n2 = a
```

In [ ]:

```
def febonacci(n):
    if(n <= 1):
        return n
    return febonacci(n-1) + febonacci(n-2)

n = int(input("Enter the number: "))

for i in range(n):
    print(febonacci(i))
```

In [ ]:

```
class item:
    def __init__(self, value, weight):
        self.value = value
        self.weight = weight

    def FKanpsack(w, arr):
        ans = 0
        arr.sort(key = lambda x: (x.value/x.weight), reverse = True)

        for a in arr:

            if w >= a.weight:
                w -= a.weight
                ans += a.value
            else:
                ans += (a.value*(w/a.weight))
                break

        return ans

if __name__ == "__main__":
    w = 50
    arr = [item(60, 10), item(100, 20), item(120, 30)]
    print(item.FKanpsack(w, arr))
```

In [ ]:

```
def BKnapsack(wt, w, v, n):

    d = [[0 for x in range(wt+1)] for x in range(n+1)]

    for i in range(n+1):
        for j in range(wt+1):
            if i == 0 or j == 0:
                d[i][j] = 0
            elif w[i-1] <= j:
                d[i][j] = max(v[i-1] + d[i-1][j-w[i-1]], d[i-1][j])
            else:
                d[i][j] = d[i-1][j]

    return d[n][wt]

v = [60, 100, 120]
w = [10, 20, 30]
wt = 50
print(BKnapsack(wt, w, v, len(w)))
```

In [ ]:

```
import heapq

class node:
    def __init__(self, c, f, left = None, right = None):
        self.c = c
        self.f = f
        self.left = left
        self.right = right
        self.v = ''

    def __lt__(self, next):
        return self.f < next.f

    def disp(self, n, val = ''):
        nv = val + str(n.v)
        if(n.left):
            self.disp(self, n.left, nv)
        if(n.right):
            self.disp(self, n.right, nv)
        if(not n.left and not n.right):
            print(n.c, " -> ", nv)

c = ['a', 'b', 'c', 'd', 'e', 'f']
f = [5, 9, 12, 13, 16, 45]
n = []

for i in range(len(c)):
    heapq.heappush(n, node(c[i], f[i]))

while len(n) > 1:
    left = heapq.heappop(n)
    right = heapq.heappop(n)
    left.v = 0
    right.v = 1
    new = node(left.c+right.c, left.f+right.f, left, right)
    heapq.heappush(n, new)

node.disp(node, n[0])
```

In [ ]:

```
global N
N = 4

def printSolution(b):
    for i in range(N):
        for j in range(N):
            print(b[i][j], end = " ")
        print()

def isSafe(b, r, c):
    for i in range(c):
        if(b[r][i] == 1):
            return False

    for i, j in zip(range(r, -1, -1), range(c, -1, -1)):
        if(b[i][j] == 1):
            return False

    for i, j in zip(range(r, N, -1), range(c, -1, -1)):
        if(b[i][j] == 1):
            return False

    return True

def NQueenS(b, c):
    if c >= N:
        return True

    for i in range(N):
        if(isSafe(b, i, c)):
            b[i][c] = 1
            if(NQueenS(b, c+1)):
                return True
            b[i][c] = 0

    return False

def NQueen():
    b = [[0, 0, 0, 0],
        [0, 0, 0, 0],
        [0, 0, 0, 0],
        [0, 0, 0, 0]]

    if NQueenS(b, 0) == False:
        print("Solution does not exist")
        return False

    printSolution(b)
    return True

NQueen()
```