```
1. def recur_fibo(n):
   if n <= 1:
       return n
   else:
       return(recur_fibo(n-1) + recur_fibo(n-2))

nterms = (int(input("eNTER your terms")))

# check if the number of terms is valid
if nterms <= 0:
   print("Plese enter a positive integer")
else:
   print("Fibonacci sequence:")
   for i in range(nterms):
       print(recur_fibo(i))
```

•

```
 a=int(input("Enter the first number of the series "))
b=int(input("Enter the second number of the series "))
n=int(input("Enter the number of terms needed "))
print(a,b,end=" ")
while(n-2):
    c=a+b
    a=b
    b=c
    print(c,end=" ")
    n=n-1
```

```
2.import heapq

class node:
     def __init__(self, freq, symbol, left=None, right=None):

          self.freq = freq

          self.symbol = symbol

          self.left = left

          self.right = right

          self.huff = "
```

```
    def __lt__(self, nxt):
        return self.freq < nxt.freq

def printNodes(node, val=''):

    newVal = val + str(node.huff)

    if(node.left):
        printNodes(node.left, newVal)
    if(node.right):
        printNodes(node.right, newVal)

    if(not node.left and not node.right):
        print(f"{node.symbol} -> {newVal}")

chars = ['a', 'b', 'c', 'd', 'e', 'f']

freq = [ 5, 9, 12, 13, 16, 45]

nodes = []

for x in range(len(chars)):
    heapq.heappush(nodes, node(freq[x], chars[x]))

while len(nodes) > 1:

    left = heapq.heappop(nodes)
    right = heapq.heappop(nodes)


    left.huff = 0
    right.huff = 1
    newNode = node(left.freq+right.freq, left.symbol+right.symbol, left, right)

    heapq.heappush(nodes, newNode)

printNodes(nodes[0])
```

```
3/4.# Online Python compiler (interpreter) to run Python online.
# Write Python 3 code in this online editor and run it.
class Item:
    def __init__(self, value, weight):
        self.value = value
        self.weight = weight

def fractionalKnapsack(W, arr):
    arr.sort(key=lambda x: (x.value/x.weight), reverse=True)
    finalvalue = 0.0
    for item in arr:
        if item.weight <= W:
            W -= item.weight
            finalvalue += item.value
        else:
            finalvalue += item.value * W / item.weight
            break

    return finalvalue

if __name__ == "__main__":

    W = 50
    arr = [Item(60, 10), Item(100, 20), Item(120, 30)]

    max_val = fractionalKnapsack(W, arr)
    print(max_val)
```

5.

```
global N
N = 4

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

def isSafe(board, row, col):



    for i in range(col):
        if board[row][i] == 1:
            return False



    for i, j in zip(range(row, -1, -1),
                    range(col, -1, -1)):
        if board[i][j] == 1:
            return False


    for i, j in zip(range(row, N, 1),
                    range(col, -1, -1)):
        if board[i][j] == 1:
            return False

    return True

def solveNQUtil(board, col):


    if col >= N:
        return True


    for i in range(N):

        if isSafe(board, i, col):


            board[i][col] = 1


            if solveNQUtil(board, col + 1) == True:
                return True


            board[i][col] = 0


    return False

def solveNQ():
    board = [ [0, 0, 0, 0],
              [0, 0, 0, 0],
              [0, 0, 0, 0],
              [0, 0, 0, 0] ]

    if solveNQUtil(board, 0) == False:
        print ("Solution does not exist")
        return False
    printSolution(board)
    return True

solveNQ()
```

```
6. def mulMat(mat1, mat2, R1, R2, C1, C2):
    # List to store matrix multiplication result
    rslt = [[0, 0, 0, 0],
            [0, 0, 0, 0],
            [0, 0, 0, 0],
            [0, 0, 0, 0]]

    for i in range(0, R1):
        for j in range(0, C2):
            for k in range(0, R2):
                rslt[i][j] += mat1[i][k] * mat2[k][j]

    print("Multiplication of given two matrices is:")
    for i in range(0, R1):
        for j in range(0, C2):
            print(rslt[i][j], end=" ")
        print("\n", end="")

if __name__ == '__main__':
    R1 = 2
    R2 = 2
    C1 = 2
    C2 = 2

    mat1 = [[1, 1],
           [2, 2]]

    mat2 = [[1, 1],
           [2, 2]]

    if C1 != R2:
        print("The number of columns in Matrix-1  must be equal to the number of rows in " + "Matrix-2", end='')
        print("\n", end='')
        print("Please update MACROs according to your array dimension in #define section", end='')
        print("\n", end='')
    else:

        mulMat(mat1, mat2, R1, R2, C1, C2)
```