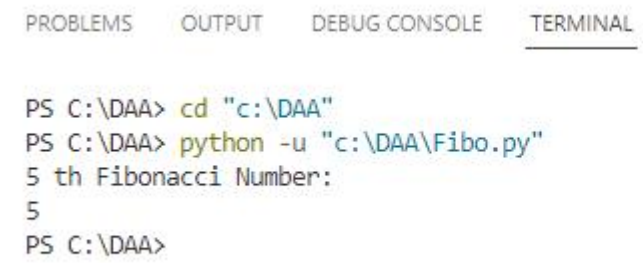
ASSIGNMENT NO 1

CODE

```
def fibonacci(n):
        a = 0
        b = 1
        if n < 0:
                print("Incorrect input")
        elif n == 0:
                return a
        elif n == 1:
                return b
        else:
                for i in range(2, n+1):
                        c = a + b
                        a = b
                        b = c
                return b


print(fibonacci(9))
```

OUTPUT



ASSIGNMENT NO 3

CODE

```
class Item:
    def __init__(self, profit, weight):
        self.profit = profit
        self.weight = weight
def fractionalKnapsack(W, arr):
    arr.sort(key=lambda x: (x.profit/x.weight), reverse=True)
    finalvalue = 0.0
    for item in arr:
        if item.weight <= W:
            W -= item.weight
            finalvalue += item.profit
        else:
            finalvalue += item.profit * 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)
```

OUTPUT

```
PROBLEMS    OUTPUT    DEBUG CONSOLE    TERMINAL

PS C:\DAA> cd "c:\DAA"
PS C:\DAA> python -u "c:\DAA\FractionalKnapsack.py"
240.0
PS C:\DAA>
```

ASSIGNMENT NO 4

CODE

```
def knapsack(wt, val, W, n):

    if n == 0 or W == 0:
        return 0
    if t[n][W] != -1:
        return t[n][W]

    if wt[n-1] <= W:
        t[n][W] = max(
            val[n-1] + knapsack(
                wt, val, W-wt[n-1], n-1),
            knapsack(wt, val, W, n-1))
        return t[n][W]
    elif wt[n-1] > W:
        t[n][W] = knapsack(wt, val, W, n-1)
        return t[n][W]

if __name__ == '__main__':
    profit = [60, 100, 120]
    weight = [10, 20, 30]
    W = 50
    n = len(profit)
    t = [[-1 for i in range(W + 1)] for j in range(n + 1)]
    print(knapsack(weight, profit, W, n))
```

OUTPUT

ASSIGNMENT NO 5

CODE

```
global N
N = 4


def printSolution(board):
    for i in range(N):
        for j in range(N):
            if board[i][j] == 1:
                print("Q",end=" ")
            else:
                print(".",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


if __name__ == '__main__':
    solveNQ()
```

OUTPUT

ASSIGNMENT NO 2

CODE

```
import heapq

# Creating Huffman tree node
class node:
    def __init__(self,freq,symbol,left=None,right=None):
        self.freq=freq # frequency of symbol
        self.symbol=symbol # symbol name (character)
        self.left=left # node left of current node
        self.right=right # node right of current node
        self.huff= '' # # tree direction (0/1)

    def __lt__(self,nxt): # Check if curr frequency less than next nodes freq
        return self.freq<nxt.freq

def printnodes(node,val=''):
    newval=val+str(node.huff)
    # if node is not an edge node then traverse inside it
    if node.left:
        printnodes(node.left,newval)
    if node.right:
        printnodes(node.right,newval)

    # if node is edge node then display its huffman code
    if not node.left and not node.right:
        print("{} -> {}".format(node.symbol,newval))

if __name__=="__main__":
    chars = ['a', 'b', 'c', 'd', 'e', 'f']
    freq = [ 5, 9, 12, 13, 16, 45]
    nodes=[]

    for i in range(len(chars)): # converting characters and frequencies into huffman tree nodes
        heapq.heappush(nodes, node(freq[i],chars[i]))

    while len(nodes)>1:
        left=heapq.heappop(nodes)
        right=heapq.heappop(nodes)

        left.huff = 0
        right.huff = 1
        # Combining the 2 smallest nodes to create new node as their parent
        newnode = node(left.freq + right.freq , left.symbol + right.symbol , left , right)
        # node(freq,symbol,left,right)
        heapq.heappush(nodes, newnode)

    printnodes(nodes[0]) # Passing root of Huffman Tree
```

OUTPUT


PROBLEMS
OUTPUT
DEBUG CONSOLE
TERMINAL
PS C:\DAA> cd "c:\DAA"
PS C:\DAA> python -u "c:\DAA\Huffman.py"
f -> 0
c -> 100
d -> 101
a -> 1100
b -> 1101
e -> 111
PS C:\DAA>