1_Factorial_Number.py

```
def factorial(x):
    # res = 1 # (0!=1)
    # for i in range(1,(x+1)):
    #     res *= i
    # return res
    if x == 0:
        return 0
    if x == 1:
        return 1
    return x * factorial(x-1)

def main():
    num = int(input("Enter a Number \n"))
    print(factorial(num))

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

2_Count_Number_Of_Digits.py

```
#write a program to count the number of digits
#input 2597

''' steps
1. check if the digit exists n>0
2. remove the last digit  n/=10
3. increase the value of count by 1 count+=1
4. Repeat step 1 to step 3 till digits remains
'''

def num_of_digits(num):
    count = 0
    while(num > 0):
        count += 1
        num = num // 10 #floor division for hole number
        print(num)
    return count

def main():
    num = int(input("Enter a Number \n"))
    print(num_of_digits(num))

if __name__ == '__main__':
    main()


# def num_of_digits(num):
#     count = 0
#     for i in num:
#         if i.isdigit:
#             count += 1
#     return count

# def main():
#     num = input("Enter a Number \n")
#     print(num_of_digits(num))

# if __name__ == '__main__':
#     main()
```

3_Trailing_Zeros.py

```
# Given a number find the number of trailing zeros of its factorial
#input 10

def trailing_zero(num):
    res = 0
    power_of_5 = 5

    while (num >= power_of_5):
        res = res + (num//power_of_5)
        power_of_5 = power_of_5 * 5
    return res

def main():
    num = int(input("Enter a Number \n"))
    print(trailing_zero(num))

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

4_Find_GCD.py

```
# brute force apporach GCD or HCF
# min(a,b)
def find_gcd (a,b):
    min = 0
    if (a > b):
        min = b
    else:
        min = a

    for i in range(min ,0 ,-1):
        if (a % i == 0) and (b % i == 0):
            return i

def main():
    lst= input("Enter a number \n").split()
    print(find_gcd(int(lst[0]),int(lst[1])))

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

5_Euclid_GCD.py

```
# brute force apporach
#0(max(a,b))
def find_gcd (a,b):
    while(a!=b):
        if (a>b):
            a = a-b
        else:
            b = b-a
    return a

def main():
    lst= input("Enter a number \n").split()
    print(find_gcd(int(lst[0]),int(lst[1])))

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

6_Gabriel_Lame_GCD.py

```
# brute force apporach

def find_gcd (a,b):
    while(a!=0 and b!=0):
        if (a>b):
            a = a%b
        else:
            b = b%a
    return a if a!=0 else b

    # if (a!=0):
    #     return a
    # else:
    #     return b

def main():
    lst= input("Enter a number \n").split()
    print(find_gcd(int(lst[0]),int(lst[1])))

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

7_Find_LCM.py

```
#Least common multiple

def find_lcm (a,b):
    res = max(a,b)
    while(True):
        if (res%a == 0 and res%b == 0):
            break
        res = res + 1
    return res

def main():
    lst= input("Enter a number \n").split()
    print(find_lcm(int(lst[0]),int(lst[1])))

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

8_Euclid_LCM.py

```
#Least common multiple

def find_lcm (a,b):
    return (a*b)//(find_gcd(a,b))

def find_gcd (a,b):
    while (a!=0 and b!=0):
        if a>b:
            a = a%b
        else:
            b = b%a

    if a!=0:
        return a
    else:
        return b

def main():
    lst= input("Enter a number \n").split()
    print(find_lcm(int(lst[0]),int(lst[1])))

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

9_Find_Prime.py

```
#first approach o(n)

def is_prime(num):
    for i in range (2,(num//2)+1):
        if num%i==0:
           return False
    return True

def main():
    num = int(input("Enter a Number \n"))
    print(is_prime(num))

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

10_Find_Prime_2.py

```
#second apporach o(math.square n)

import math


def is_prime(num):
    for i in range (2,int(math.sqrt(num))+1):
        if num%i==0:
           return False
    return True

def main():
    num = int(input("Enter a Number \n"))
    print(is_prime(num))

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

11_Find_Prime_3.py

```
import math

#o(root n ) only but elliminating the more values

def is_prime(num):
    if num==1:
       return False
    if num==2 or num==3:
        return True
    if num%2==0 or num%3==0:
        return False
    for i in range (5,int(math.sqrt(num))+1,6):
        if num%i==0 or num%(i+2)==0:
           return False
    return True

def main():
    num = int(input("Enter a Number \n"))
    print(is_prime(num))

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

12_Find_Prime.py

```
# Find all the numbers from 1 to n

import math


def find_prime_all(n):
    for i in range (2,n+1):
        if (is_prime(i)):
            print(i)

def is_prime(num):
    if num==1:
       return False
    if num==2 or num==3:
        return True
    if num%2==0 or num%3==0:
        return False
    for i in range (5,int(math.sqrt(num))+1,6):
        if num%i==0 or num%(i+1)==0:
           return False
    return True

def main():
    num = int(input("Enter a Number \n"))
    find_prime_all(num)

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

13_Find_Prime_Sieve_Of_Eratosthenes.py

```
# Find all the numbers from 1 to n

import math


def find_prime_all(n):
    prime = [False] * (n+1)
    for i in range (2,int(math.sqrt(n))+1):
        if prime[i] == False:
            for j in range(i*i,n+1,i):
                prime[j] = True
    for i in range(2,n+1):
        if(prime[i]==False):
            print(i)


def main():
    num = int(input("Enter a Number \n"))
    find_prime_all(num)

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

14_Find_Factors.py

```
#0(n)

def find_factors(num):
    for i in range(1,num+1):
        if(num%i==0):
            print(i)

def main():
    num = int(input("Enter a Number \n"))
    find_factors(num)

main()
```

15_Find_Factors_Effective.py

```
#0(root n)
import math

def find_factors(num):
    for i in range(1,int(math.sqrt(num))+1):
        if (num%i==0):
            print(i)
            if i!=(num//i):
                print(num//i)

def main():
    num = int(input("Enter a Number \n"))
    find_factors(num)

main()
```

16_Find_Factors_Acending_Order.py

```
#0(root n)
import math

def find_factors(num):
    for i in range(1,int(math.sqrt(num))+1):
        if (num%i==0):
            print(i)
            '''if i!=(num//i):
                print(num//i)'''
    for i in range(int(math.sqrt(num)),0,-1):
        if (num%i==0 and i!=(num//i)):
            print(num//i)

def main():
    num = int(input("Enter a Number \n"))
    find_factors(num)

main()
```

17_Prime_Factors.py

```
#Find all the prime factors of a given number(o(n))

def prime_factor(n):
    i = 2
    while (n>1):
        while(n%i==0):
            print(i)
            n = n//i
        i +=1

def main():
    num = int(input("Enter a Number\n"))
    prime_factor(num)

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

18_Prime_Factors_Effective.py

```
#Find all the prime factors of a given number(o(root n * log(n)))
import math

def prime_factor(n):
    i = 2
    while(i<=math.sqrt(n)):
        while(n%i==0):
            print(i)
            n = n//i
        i +=1
    if n>1:
        print(n)

def main():
    num = int(input("Enter a Number\n"))
    prime_factor(num)

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

19_Demical_to_Binary.py

```
def decimal_to_binary(n):
    b = ' '
    while(n>=1):
        x = n%2
        n = n//2
        b = str(x) + b
    return b
```

```
def main():
    num = int(input("Enter a Number \n"))
    print(decimal_to_binary(num))

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

20_Binary_to_Decimal.py

```
def decimal_to_binary(n):
    b = ''
    while(n>=1):
        x = n%2
        n = n//2
        b = str(x) + b
    return b

def binary_to_decimal(b):
    result = 0
    pow_of_2 = 1

    for i in range(len(b)-1, -1 , -1):
        if (b[i]=='1'):
            result = result + pow_of_2
        pow_of_2 = pow_of_2 * 2

    return result

def main():
    num = int(input("Enter a Number \n"))
    b = decimal_to_binary(num)
    print(b)
    print(binary_to_decimal(b))

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

21_Bitwise_Operator.py

```
def bit_wise_operator():
    print(5 | 9)  # any one 1 means result 1
    print(5 & 9)  # both should be 1 result 1 otherwise 0
    print(5 ^ 9)  # both same means 0 otherwise 1
    print(5 << 1) # left shift 5 * 2
    print(5 << 2) # left shifted by 2 times (5 * (2**2))
    print(5 >> 1) # right shifted by 1 time (5//2)
    print(5 >> 2) # right shifted by 2 times (5//(2**2))
    print(-5 >> 1) #
    print(-5 >> 2) #
    print(-8 >> 1) #


bit_wise_operator()
```

22_Swith_On_Bit.py

```
def main():
    #n = 36
    n = int(input("Enter a Number \n"))
    i = 3
    on_mask = 1 << i
    print(n|on_mask)

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

23_Swith_Off_Bit.py

```
def main():
    n = int(input("Enter a Number \n"))
    i = 5
    off_mask = ~(1<<i)
    print(n & off_mask)

main()
```

24_Toggle.py

```
def main():
    n = int(input("Enter a Number \n"))
    i = 5
    t_mask = 1 << i
    print(n ^ t_mask)

main()
```

25_Check_On_Off.py

```
def main():
    n = int(input("Enter a Number \n"))
    i = 5
    c_mask = 1 << i

    if ((n&c_mask)==0):
        print("OFF")
    else:
        print("ON")

main()
```

26_Right_Most_Bit.py

```
def right_most_bit(n):
    m = 1
    pos = 0

    while((n&m)==0):
        m = m+1
        pos = pos+1
    return pos+1

def main():
    n = int(input("Enter a Number \n"))
    print(right_most_bit(n))

main()
```

27_Right_Most_Bit_Effective.py

```
import math

def right_most_bit(n):
    # a = n^(n&(n-1))
    # return  a // 8 (we want 2^3 power + 1 should return)
    return  math.log(n^(n&(n-1)),2) + 1

def main():
    n = int(input("Enter a Number \n"))
    print(right_most_bit(n))

main()
```

28_Count_Set_Bit.py

```
def count_set_bit(n):
    count = 0
    while (n >0 ):
        n = n & (n-1)
        count +=1
    return count

def main():
    num = int(input("Enter a Number \n"))
    print(count_set_bit(num))

main()
```

29_Check_Power_Of_2.py

```
def check_power_of_2(num):
    if num == 0:
        return False

    return num&(num-1) == 0

def main():
    num = int(input("Enter a Number \n"))
    print(check_power_of_2(num))

main()
```

30_Find_Lonely_Integer.py

```
# o(N log N) // brute force apporach

def lonely_integer(arr):
    arr.sort()
    for i in range(0,len(arr)-2,2):
        if (arr[i] != arr[i+1]):
            return arr[i]
    return arr[len(arr)-1]
```

```
def main():
    arr = list(map(int, input("Enter an array: ").split()))
    print(lonely_integer(arr))

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

31_Find_Lonely_Integer2.py

```
# o(N) // but space is increased

def lonely_integer(arr):
    s = set()
    for i in arr:
        if i not in s:
            s.add(i)
        else:
            s.remove(i)
    return s.pop()

def main():
    arr = list(map(int, input("Enter an array: ").split()))
    print(lonely_integer(arr))

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

32_Find_Lonely_Integer3.py

```
# o(N) // but space reduced

def lonely_integer(arr):
    result = 0

    for i in arr:
        result = result ^ i

    return result

def main():
    arr = list(map(int, input("Enter an array: ").split()))
    print(lonely_integer(arr))

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

33_Is_Consecutive_Set.py

```
def is_consecutive(n):
    return (n & (n<<1) != 0)

def main():
    num = int(input("Enter a Number \n"))
    print(is_consecutive(num))

main()
```

34_Longest_Consecutive.py

```
#time o (log N)

def max_consecutive(n):
    count = 0

    while(n>0):
        n = n & (n<<1)
        count += 1

    return count

def main():
    num = int(input("Enter a Number \n"))
    print(max_consecutive(num))

main()
```

35_Swap_Odd_Even.py

```
# o(1)

def swap_ood_even_places(n):
    return ( (n & 0Xaaaaaaaa ) >> 1| ( n & 0X55555555 ) << 1)

def main():
    n = int(input("Enter a Number \n"))
    print(swap_ood_even_places(n))

main()
```

36_Trailing_Zero_Count.py

```
import math

def tailing_zero(n):
    # a = n^(n&(n-1))
    # return  a // 8 (we want 2^3 power + 1 should return)
    return  int(math.log(n^(n&(n-1)),2))

def main():
    n = int(input("Enter a Number \n"))
    print(tailing_zero(n))

main()
```

37_Reversing_32_Bit.py

```
#time o(log N)

def reverse_binary(n):
    f = 31
    l = 0
    rev = 0

    while (f>l):
        if ((n & (1<<f)) != 0):
            rev = rev | (1<<l)
        if ((n & (1<<l)) != 0):
            rev = rev | (1<<f)
        f = f - 1
        l = l + 1

    return rev

def main():
    n = int(input("Enter a Number \n"))
    print(reverse_binary(n))

main()
```

38_nth_number_palindrome.py

```
# write a program to find the nth number , whose binary representation is a palindrome

import math

def reverse_binary(n,length):
    f = length - 1
    l = 0
    rev = 0

    while (f>l):
        if ((n & (1<<f)) != 0):
            rev = rev | (1<<l)
        if ((n & (1<<l)) != 0):
            rev = rev | (1<<f)
        f = f - 1
        l = l + 1

    return rev

def nth_palin_binary(n):
    length = 0
    count = 0

    while (count<n):
        length += 1
        count += int(math.pow(2, (length-1)//2))

    count -= int(math.pow(2, (length-1)//2))
    elem = n - count - 1
    ans = ( (1<< (length-1)) | (elem<<(length//2)))
    ans = ans | reverse_binary(ans,length)
    return ans

n = int(input("Enter a Number \n"))
print(bin(nth_palin_binary(n)))
```

39_Factorial_Number.py

```
def fact(n):
    if n == 1 or n == 0:
        return 1
    return n * fact(n-1)

def main():
    n = int(input("Enter a Number \n"))
    print(fact(n))

main()
```

40_Fibonacci_Series.py

```
def fibonacci_series(n):
    if n == 1 or n == 2:
        return 1

    return fibonacci_series(n-1) + fibonacci_series(n-2)

def main():
    n = int(input("Enter a Number \n"))
    print(fibonacci_series(n))

main()
```

41_First_N_Natural_Number.py

```
def n_natural_number(n):
    if n == 0:
        return
    print(n)
    n_natural_number(n-1)
    #print(n)

n = int(input("Enter a Number \n"))
n_natural_number(n)
```

42_Count_Digits.py

```
def count_digits(n):
    if n == 0:
        return n

    return count_digits(n//10) + 1

n = int(input("Enter a Number \n"))
print(count_digits(n))
```

43_Sum_Of_Digits.py

```
def count_digits(n):
    if n == 0:
        return n

    return count_digits(n//10) + n%10

# def count_digits(n):
#     res = 0
#     while n > 0:
#         res = res + n % 10
#         n = n // 10
#     return res
```

```
n = int(input("Enter a Number \n"))
print(count_digits(n))
```

44_Reverse_String_Recurstion.py

```
def reverse_string(s,r,i):
    if i < 0:
        return r

    return reverse_string(s, r + s[i] , i-1)

s = input("Enter a String \n")
print(reverse_string(s,"",len(s)-1))
```

45_Palindrome_Recursion.py

```
def is_palindrome(s,i,j):
    if (s[i] != s[j]):
        return False
    if (j<=i):
        return True

    return is_palindrome(s, i+1, j-1)

s = input("Enter a String \n")
print(is_palindrome(s,0,len(s)-1))
```

46_Sum_Of_Array.py

```
def sum_of_array(arr,i):
    if (len(arr) == i):
        return 0

    return sum_of_array(arr,i+1) + arr[i]

s = list(map(int , input("Enter an array \n").split()))
print(sum_of_array(s,0))
```

48_Balanced_Parenthesis.py

```
def balanced_parenthesis(arr,n,i,o,c):
    if i == len(arr):
       print("".join(arr))

    if (o<n):
        arr[i] = '('
        balanced_parenthesis(arr,n,i+1,o+1,c)

    if (c<o):
        arr[i] = ')'
        balanced_parenthesis(arr,n,i+1,o,c+1)

n = int(input("Enter A Number \n"))

list = [""] * (n*2)

balanced_parenthesis(list,n,0,0,0)
```

49_Letter_Compination_Phone_Number.py

```
keypad = ["","","abc","def","ghi","jkl","mno","pqrs","tuv","wxyz"]

def possible_words(s, ans):

    if len(s) == 0:
        print(ans)
        return

    key = keypad[ int(s[0]) ]

    for i in key:
        possible_words(s[1:] , ans+i)

def main():
    s = input("Enter a value \n")
    possible_words(s,'')

main()
```

50_Possible_Combinations.py

```
lst = []

def possible_combinations(s, ans):
    if len(s) == 0:
        #print(ans)
        lst.append(ans)
        return

    possible_combinations(s[1:] , ans+s[0])
    possible_combinations(s[1:], ans)

def main():
    s = input()
    possible_combinations(s,'')
    print(lst)
main()
```

51_Permutations.py

```
def permutations(ar , fi):

    if (fi == len(ar)-1):
        print("".join(ar))
        return

    for i in range(fi , len(ar)):
        ar[fi] , ar[i]= ar[i] , ar[fi]
        permutations(ar , fi+1)
        ar[fi] , ar[i]= ar[i] , ar[fi]

def main():
    s = input()
    permutations (list(s) , 0)

main()
```

52_Rope_Cutting_Problem.py

```
def max_pieces(n, a, b, c):

    if (n == 0):
        return 0
    elif (n < 0) :
        return -1

    # temp1 = max_pieces(n-a , a, b, c)
    # temp2 = max_pieces(n-b , a, b, c)
    # temp3 = max_pieces(n-c , a, b, c)

    # max(temp1 , temp2 , temp3)

    pieces = max(max_pieces(n-a , a, b, c) , max_pieces(n-b , a, b, c) , max_pieces(n-c , a

    if pieces == -1:
        return -1

    return pieces + 1


def main():
    print(max_pieces(15,1,2,2))

main()


# def max_pieces(n, a, b, c):
#     if n == 0:
#         return 0, []  # base case: 0 pieces, empty path
#     elif n < 0:
#         return -1, None  # invalid path

#     # Recursive calls
#     res_a, path_a = max_pieces(n - a, a, b, c)
#     res_b, path_b = max_pieces(n - b, a, b, c)
#     res_c, path_c = max_pieces(n - c, a, b, c)

#     # Find the max among valid results
#     max_val = max(res_a, res_b, res_c)

#     if max_val == -1:
#         return -1, None  # no valid cut

#     # Pick the corresponding path
#     if max_val == res_a:
#         return res_a + 1, path_a + [a]
#     elif max_val == res_b:
#         return res_b + 1, path_b + [b]
#     else:
#         return res_c + 1, path_c + [c]

# def main():
#     count, cuts = max_pieces(15, 5, 8, 7)
#     print("Max pieces:", count)
#     print("Cuts used:", cuts)

# main()
```

53_Sub_Set.py

```
def count_subsets(arr, sum, i):

    if sum == 0:
        return 1
    if sum < 0 :
        return 0
    if i == len(arr):
        return 0

    return count_subsets(arr , sum - arr[i] , i+1) + count_subsets(arr, sum , i+1)

def main():
    ar = [10,15,20,5]
    print(count_subsets(ar,25,0))

main()
```

54_Lucky_Number.py

```
def is_lucky_number(n,counter):

    if n < counter:
       return True

    if n % counter == 0:
        return False

    return is_lucky_number( n-(n//counter) , counter+1)

def main():
    print(is_lucky_number(9,2))

main()
```

55_Tower_Of_Honai.py

```
def tower_of_honai(n , src , aux , dest):
    if n == 1:
       print(src,'-->',dest)
       return
    tower_of_honai(n-1,src,dest,aux)
    tower_of_honai(1,src,aux,dest)
    tower_of_honai(n-1,aux,src,dest)

def main():
   tower_of_honai(4,'A','B','C')

main()
```

56_Power_Of.py

```
def power_of(x,y):

    if y == 0:
       return 1

    if y % 2 == 0:
       res = power_of(x,y//2)
       return res * res
    else:
       return power_of(x,y-1) * x

def main():
   print(power_of(5,2))

main()
```

57_Linear_Search.py

```
def liner_search(a,key):
    for i in range(0,len(a)):
        if key == a[i] :
           return i

    return -1

def main():
    a = [10,20,50,77,90]
    key = 99
    print(liner_search(a,key))

main()
```

58_Binary_Search.py

```
def binary_search(arr,key):
    low , high , mid = 0 , len(arr)-1 , 0

    while(low<=high):
        #mid = (low+high)//2
        mid = low + (high - low) // 2
        if arr[mid] == key:
           return mid
        elif arr[mid] > key:
            high = mid - 1
            #low = low
        else:
            low = mid + 1
            #high = high
    return -1

def main():
    arr = [14,5,67,89,2,3,0]
    arr.sort()
    print("Sorted array:", arr)
    print(binary_search(arr,89))
main()
```

59_span_of_List.py

```
def span_of_list(a):
    max = a[0]
    min = a[0]

    for i in range(0,len(a)):
        if a[i] > max:
           max = a[i]
        if a[i] < min:
            min = a[i]
    return max - min

def main():
    a = [10,20,40,99,6]
    print(span_of_list(a))

main()
```

60_Second_Largest_Element.py

```
def second_largest(arr):
    max1 , max2 = 0 , 0

    if arr[0] > arr[1]:
        max1 , max2 = arr[0], arr[1]
    else:
        max1 , max2 = arr[1], arr[0]

    for i in range(2,len(arr)):
        if max1 < arr[i]:
           max2 , max1 = max1 , arr[i]
        elif max2 < arr[i]:
            max2 = arr[i]

    return max2

def main():
    arr = [20,42,6,25,30,88]
    print(second_largest(arr))

main()
```

62_Second_Smallest_Element.py

```
def second_smallest(arr):
    max1 , max2 = 0 , 0

    if arr[0] < arr[1]:
        max1 , max2 = arr[0], arr[1]
    else:
        max1 , max2 = arr[1], arr[0]

    for i in range(2,len(arr)):
        if max1 > arr[i]:
           max2 , max1 = max1 , arr[i]
        elif max2 > arr[i]:
            max2 = arr[i]

    return max2

def main():
    arr = [20,42,6,25,30,88]
    print(second_smallest(arr))

main()
```

63_Ceil_And_Floor.py

```
def ceil(arr,key):
    low , high , mid = 0 , len(arr)-1 , 0

    while(low<=high):
        #mid = (low+high)//2
        mid = low + (high - low) // 2
        if arr[mid] == key:
           return arr[mid]
        elif arr[mid] > key:
            high = mid - 1
            #low = low
        else:
            low = mid + 1
            #high = high
    if low < len(arr):
        return arr[low]
    else:
        return -1

def floor(arr,key):
    low , high , mid = 0 , len(arr)-1 , 0

    while(low<=high):
        #mid = (low+high)//2
        mid = low + (high - low) // 2
        if arr[mid] == key:
           return arr[mid]
        elif arr[mid] > key:
            high = mid - 1
            #low = low
        else:
            low = mid + 1
            #high = high
    if high >= 0:
        return arr[high]
    else:
        return -1

def main():
    arr = [19,23,56,61,72,88,92]
    print(ceil(arr,68))
    print(floor(arr,70))

main()
```

64_Bitonic_Array.py

```
def ascending_binary_search(arr,key):
    low , high , mid = 0 , len(arr)-1 , 0

    while(low<=high):
        #mid = (low+high)//2
        mid = low + (high - low) // 2
        if arr[mid] == key:
           return mid
        elif arr[mid] > key:
            high = mid - 1
            #low = low
        else:
            low = mid + 1
            #high = high
    return -1

def decending_binary_search(arr,key):
    low , high , mid = 0 , len(arr)-1 , 0

    while(low<=high):
        #mid = (low+high)//2
        mid = low + (high - low) // 2
        if arr[mid] == key:
           return mid
        elif arr[mid] > key:
            low = mid + 1
            #low = low
        else:
            high = mid - 1
            #high = high
    return -1

def bitonic_element(a):
    l,r,m = 0, len(a)-1 , 0

    while(l<=r):
        m = l + (r-l)//2
        if a[m]>a[m+1] and a[m]>a[m-1]:
            return m
        elif a[m]>a[m-1] and a[m]<a[m+1]:
            l = m
        else:
            r=m
    return -1

def main():
    a = [5,6,7,8,9,10,3,2,1]
    key = 1
    bitonic_index = bitonic_element(a)

    if a[bitonic_index] == key:
        print(bitonic_index)
    else:
        index = ascending_binary_search(a[:bitonic_index+1], key)
        if index != -1:
            print(index)
        else:
            index = decending_binary_search(a[bitonic_index+1:], key)
            if index != -1:
                print(bitonic_index + 1 + index)
            else:
                print(-1)

main()
```

65_Count_Smaller_Or_Equal_Element.py

```
def count_smaller_equal(arr, key):
    low, high = 0, len(arr) - 1
    result = -1

    while low <= high:
        mid = low + (high - low) // 2
        if arr[mid] <= key:
            result = mid
            low = mid + 1
        else:
            high = mid - 1
    return result + 1

def count_greater_equal(arr, key):
    low, high = 0, len(arr) - 1
    result = -1

    while low <= high:
        mid = low + (high - low) // 2
        if arr[mid] >= key:
            result = mid
            high = mid - 1
        else:
            low = mid + 1

    if result == -1:
        return 0
    return len(arr) - result


def main():
    arr = [1, 2, 2, 2, 3, 5, 6]
    print(count_smaller_equal(arr,6))
    print(count_greater_equal(arr,6))

main()
```

66_Wood_Cutting_Problem.py

```
def find_wood_count(ht,m):
    wc = 0
    for i in ht:
        if i > m:
            wc = wc + (i-m)

    return wc

def find_maxHeight(ht,b):
    max = 0
    for i in ht:
        if max < i:
            max = i

    l , h, m = 0 , max  , 0

    while(l<=h):
        m = l + (h-l)//2
        wc = find_wood_count(ht,m)

        if wc == b or l== m:
            return m
        elif wc > b:
            l=m
        else:
            h=m
    return -1

def main():
    ht = [20,15,10,17]
    b = 7
    print(find_maxHeight(ht,b))

main()
```

67_Find_Median.py

```
def find_median(ar1,ar2):
    i,j,k = 0,0,0
    m=[]

    while i<len(ar1) and j<len(ar2):
        if ar1[i] < ar2[j]:
            m.append(ar1[i])
            i = i+1
            k = k+1
        else:
            m.append(ar2[j])
            j = j+1
            k = k+1


    while i<len(ar1):
        m.append(ar1[i])
        i = i+1
        k = k+1

    while j<len(ar2):
        m.append(ar2[j])
        j = j+1
        k = k+1

    mid = len(m) // 2
    if len(m) % 2 == 0:
        return (m[mid] + m[mid-1])/2
    else:
        return m[mid]

def main():
    ar1 = [1,3,8,17]
    ar2 = [5,6,7,19,21,25]
    print(find_median(ar1,ar2))

main()
```

68_Find_Median_LOG.py

```
import sys


def find_median(ar1,ar2):
    if len(ar1) > len(ar2):
        return find_median(ar2,ar1)

    l = 0
    h = len(ar1)

    while l<=h:
        m1 = l + (h-l)//2
        m2 = (len(ar1) + len(ar2) + 1)//2 - m1

        l1 = (sys.maxsize * -1) if (m1==0) else (ar1[m1-1])
        r1 = (sys.maxsize) if (m1==len(ar1)) else (ar1[m1])

        l2 = (sys.maxsize * -1) if (m2==0) else (ar2[m2-1])
        r2 = (sys.maxsize) if (m2==len(ar1)) else (ar2[m2])

        if l1 <= r2 and l2 <= r1:
            if (len(ar1) + len(ar2) ) % 2 == 0:
                return ((max(l1,l2)) + min(r1,r2))/2
            else:
                return (max(l1,l2))

        elif l2 > r1:
            l = m1 + 1
        else:
            h = m1 - 1

def main():
    ar1 = [1,3,8,17]
    ar2 = [5,6,7,19,21,25]
    print(find_median(ar1,ar2))

main()
```

69_Allocate_Books.py

```
def max_page(ar,b):
    if b > len(ar):
        return -1

    l = ar[0]
    h = 0

    # for i in ar:
    #     if l > i:
    #         l = i
    #     h = h + i

    l = max(ar)
    h = sum(ar)

    res = -1
    while l <= h:
        m = (l+h)//2

        if is_possible_sol(ar,b,m) == True:
            res = m
            h = m-1
        else:
            l = m+1

    return res

def is_possible_sol(ar,b,m):
    students = 1
    spc = 0

    for i in ar:
        if i > m:
            return False
        if spc + i <= m:
            spc = spc + i
        else:
            students = students + 1
            if students > b:
                return False
            spc = i

    return True

def main():
    ar = [2,3,4,1]
    b = 2
    print(max_page(ar,b))

main()
```

70_Painters_Partition.py

```
def is_possible_soln(ar,a,m):
    painters = 1
    pbc = 0

    for i in ar:
        if m < i:
            return False
        if pbc + i <= m:
            pbc = pbc + i
        else:
            painters = painters + 1

            if painters > a:
                return False
            pbc = i

    return True

def max_time(ar,a,b):
    l = 0
    h = 0

    res = -1

    # for i in ar:
    #     h = h + i

    h = sum(ar)

    while l <= h:
        m = l + (h-l)//2

        if is_possible_soln(ar,a,m) == True:
            res = m
            h = m - 1
        else:
            l = m + 1
    return res * b

def main():
    ar = [10,20,30,40]
    a = 2
    b = 2
    print(max_time(ar,a,b))

main()
```

71_Minimum_Days_To_Make_Bouquets.py

```
def is_possible_soln(ar,boq,flowers,m):
    adj , bc = 0 , 0

    for i in ar:
        if i <= m:
           adj = adj + 1
           if adj == flowers:
               bc = bc + 1
               if bc == boq:
                   return True
               adj = 0
        else:
            adj = 0
    return False

def min_day_to_make_bouquets(ar,boq,flowers):
    if boq * flowers > len(ar):
        return -1

    l = ar[0]
    h = ar[0]
    # for i in ar:
    #     if i > h:
    #         h = i
    #     if i < l:
    #         l = i
    l = min(ar)
    h = max(ar)
    res = -1

    while l <= h:
        m = l + (h-l)//2

        if is_possible_soln(ar,boq,flowers,m) == True:
            res = m
            h = m - 1
        else:
            l = m + 1

    return res

def main():
    ar = [2,5,2,9,3,10,4,6,5,6]
    boq = 4
    flowers = 2
    print(min_day_to_make_bouquets(ar,boq,flowers))

main()
```

72_Is_Array_Sorted.py

```
def is_sorted(ar):

    for i in range(1, len(ar)):
        if ar[i] < ar[i-1]:
           return False

    return True

def main():
    ar = [2,4,6,8,10,12,14]
    print(is_sorted(ar))

main()
```

73_Sqaure_Root_Number_Floor.py

```
def sqrt(n):
    if n==0 or n==1:
        return n
    l = 2
    h = n//2

    res = 0

    while l <= h:
        m = l + (h-l)//2

        if m * m == n:
            return m
        elif m * m < n:
            res = m
            l = m + 1
        else:
            h = m - 1
            #res = m // ceil
    return res

def main():
    n = 24
    print(sqrt(n))

main()
```

74_Remove_Duplicates_From_Sorted_Array.py

```
def remove_duplicates(ar):
    rd = 0

    for i in range(1,len(ar)):
        if ar[rd] != ar[i]:
            rd = rd + 1
            ar[rd] = ar[i]
    return rd + 1

def main():
    ar = [2,2,3,3,4,5,5,6]
    print(ar)
    rd = remove_duplicates(ar)
    print(ar[:rd])

main()
```

75_Rotated_Array_Brute.py

```
def rotate_one(ar):
    temp = ar[0]

    for i in range(1,len(ar)):
        ar[i-1] = ar[i]

    ar[len(ar) - 1] = temp

def rotate(ar,k):
    if k < 0:
        k = k + len(ar)

    k = k % len(ar)

    for i in range(k):
        rotate_one(ar)

def main():
    ar = [1,2,3,4,5]
    print(ar)
    rotate(ar,-1)
    print(ar)

main()
```

76_Reverse_Array.py

```
def reverse(ar):
    i , j = 0 , len(ar) - 1

    while i < j:
          temp = ar[i]
          ar[i] = ar[j]
          ar[j] = temp
          i = i + 1
          j = j - 1

def main():
    ar = [2,4,6,8,10,12,14]
    print(ar)
    reverse(ar)
    print(ar)

main()
```

77_Rotated_Array_Effective.py

```
def reverse(a,start,end):

    while start < end:
        # temp = a[start]
        # a[start] = a[end]
        # a[end] = temp
        a[start] , a[end] = a[end] , a[start]
        start = start + 1
        end = end -1

def rotate(ar,k):
    if k < 0:
        k = k + len(ar)

    k = k % len(ar)

    reverse(ar, 0 , k-1)
    reverse(ar, k , len(ar)-1)
    reverse(ar, 0 , len(ar)-1)

def main():
    ar = [1,2,3,4,5]
    print(ar)
    rotate(ar,-1)
    print(ar)

main()
```

78_Move_Zeros_To_End.py

```
def move_zeros(ar):
    if len(ar) == 0 or len(ar) == 1:
        return

    z , nz = 0 , 0

    while nz < len(ar):
        if ar[nz] != 0:
            ar[z] , ar[nz] = ar[nz] , ar[z]
            nz = nz + 1
            z = z + 1
        else:
            nz = nz + 1

def main():
    #nums = [0,1,0,3,12]
    nums = [5,10,22,8,0,5,0]
    move_zeros(nums)
    print(nums)
main()
```

79_SubArray_Of_Array.py

```
def sub_array(ar):
    for i in range(len(ar)):
        for j in range(i,len(ar)):
            print(ar[i:j+1])

def sub_array(ar):
    n = len(ar)
    for i in range(n):
        temp = []
        for j in range(i, n):
            temp.append(ar[j])
            print(temp)  # or yield tuple(temp) if you want to return them
```

```
def main():
    a = [1,2,3,4,5]
    sub_array(a)
main()
```

80_Inverse_Of_Array.py

```
def inverse(a):
    # b = [0] * len(a)

    # for i in range(len(a)):
    #     v = a[i]
    #     if v < len(a):
    #         b[v] = i
    # return b

    n = len(a)
    if sorted(a) != list(range(n)):
        raise ValueError("Input must be a permutation of 0 to n-1.")

    b = [0] * n
    for i in range(n):
        v = a[i]
        b[v] = i
    return b

def main():
    a = [2,3,1,0,7]
    b = inverse(a)
    print(a)
    print(b)
main()
```

81_Leaders_In_Array.py

```
def leaders_in_array(a):
    # Nave Apporach
    # for i in range(len(a)):
    #     isLeader = True
    #     for j in range(i+1, len(a)):
    #         if a[j] >= a[i] :
    #             isLeader = False
    #             break
    #     if isLeader == True:
    #         print(a[i])

    current_leader = a[len(a)-1]
    print(current_leader)

    for i in range(len(a)-2, -1, -1):
        if a[i] > current_leader:
            current_leader = a[i]
            print(current_leader)

def main():
    a = [8,11,5,11,7,6,3]
    leaders_in_array(a)
main()
```

82_Frequency_Of_Elements_Sorted_Array.py

```
def frequency(a):
    if not a:
        return

    freq = 1
    for i in range(1, len(a)):
        if a[i] == a[i - 1]:
            freq += 1
        else:
            print(f'{a[i - 1]} {freq}')
            freq = 1

    # Always print the last group
    print(f'{a[-1]} {freq}')

def main():
    a = [20, 20, 30, 30, 30, 30]
    #a = [10]
    frequency(a)
main()
```

83_Trapping_Rain_Water.py

```
# def trap(a):
#     res = 0

#     for i in range(1,len(a)-1):
#         lb = a[i]
#         for j in range(i):
#             if lb < a[j]:
#                 lb = a[j]
#         rb = a[i]
#         for j in range(i+1 , len(a)):
#             if rb < a[j]:
#                 rb = a[j]

#         wl = min(lb,rb)
#         tw = wl - a[i]
#         res = res + tw
#     return res
# this o(n*n)
def trap(a):
    n = len(a)
    if n <= 2:
        return 0  # Not enough bars to trap water

    # Check if array is strictly increasing
    if all(a[i] <= a[i + 1] for i in range(n - 1)):
        return 0

    # Check if array is strictly decreasing
    if all(a[i] >= a[i + 1] for i in range(n - 1)):
        return 0

    res = 0
    for i in range(1, n - 1):
        lb = max(a[:i])          # Left boundary
        rb = max(a[i + 1:])      # Right boundary
        wl = min(lb, rb)         # Water level
        if wl > a[i]:
            res += wl - a[i]
    return res

def main():
    a = [4,2,0,3,2,5]
    print(trap(a))
main()
```

84_Trapping_Water_Time_Complexity.py

```
#0(n)
def trap(a):
    n = len(a)
    if n <= 2:
        return 0

    if all(a[i] <= a[i + 1] for i in range(n - 1)):
        return 0
    if all(a[i] >= a[i + 1] for i in range(n - 1)):
        return 0

    left_max = [0] * n
    right_max = [0] * n

    left_max[0] = a[0]
    for i in range(1, n):
        left_max[i] = max(left_max[i - 1], a[i])

    right_max[-1] = a[-1]
    for i in range(n - 2, -1, -1):
        right_max[i] = max(right_max[i + 1], a[i])

    res = 0
    for i in range(1, n - 1):
        wl = min(left_max[i - 1], right_max[i + 1])
        if wl > a[i]:
            res += wl - a[i]
    return res
def main():
    a = [4,2,0,3,2,5]
    print(trap(a))
main()
```

85_Trapping_Rain_Water_Both.py

```
#trapping rain water both time complexity 0(n) and space complexity 0(1)
def trap(height):
    n = len(height)
    if n <= 2:
        return 0

    # Early exit: strictly increasing or decreasing
    if all(height[i] <= height[i + 1] for i in range(n - 1)):
        return 0
    if all(height[i] >= height[i + 1] for i in range(n - 1)):
        return 0

    left = 0
    right = n - 1
    left_max = 0
    right_max = 0
    res = 0

    while left < right:
        if height[left] < height[right]:
            if height[left] >= left_max:
                left_max = height[left]
            else:
                res += left_max - height[left]
            left += 1
        else:
            if height[right] >= right_max:
                right_max = height[right]
            else:
                res += right_max - height[right]
            right -= 1

    return res

def main():
    a = [4, 2, 0, 3, 2, 5]
    print(trap(a))  # Output: 9

main()
```

86_Max_Consecutive_Ones.py

```
# navie apporach
# def max_count(a):
#     max_count = 0
#     for i in range(len(a)):
#         count = 0
#         for j in range(i,len(a)):
#             if a[j] == 1:
#                 count = count + 1
#             else:
#                 break
#         max_count = max(max_count,count)
#     return max_count

# effective apporach
def max_count(a):
    max_count = 0
    current_count = 0

    for i in a:
        if i == 1:
            current_count += 1
            max_count = max(current_count , max_count)
        else:
            current_count = 0
    return max_count

def main():
    a = [0,1,1,1,0,0,1,0]
    print(max_count(a))
main()
```

87_Maximum_SubArray.py

```
#nave apporach
# def max_sub_array(ar):
#     max_sum = 0
#     for i in range(len(ar)):
#         sum = 0
#         for j in range(i,len(ar)):
#             sum = sum + ar[j]
#             max_sum = max(sum,max_sum)
#     return max_sum

#effective apporach
def max_sub_array(a):
    max_sum = a[0]
    sum = a[0]

    for i in range(1,len(a)):
        if (sum >= 0):
            sum = sum + a[i]
        else:
            sum = a[i]

        max_sum = max(sum,max_sum)
    return max_sum

def main():
    ar = [5,6,-3,7,-13,8,-2,5,-6,7,-11,3,10,-10,-6,-10,7,2]
    print(max_sub_array(ar))
main()
```

88_Majority_Element_Array.py

```
#nave apporach
# def majorityelement(nums):
#     for i in range(len(nums)):
#         count = 1
#         for j in range(i+1 , len(nums)):
#             if nums[i] == nums[j]:
#                 count = count + 1
#         if (count > len(nums)//2):
#             return nums[i]

def majorityelement(a):
    maj = a[0]
    count = 1

    for i in range(1,len(a)):
        if a[i] == maj:
            count = count + 1
        else:
            count = count - 1

        if count == 0:
            maj = a[i]
            count = 1
    return maj


def main():
    a = [3,2,3]
    print(majorityelement(a))
```

```
main()
```

89_Longest_Alternative_Even_Odd_Subarray.py

```
# nave apporach
# def longest_even_odd_subarray(a):
#     max_count = 1

#     for i in range(len(a)):
#         count = 1
#         for j in range(i+1,len(a)):
#             if (a[j] % 2 == 0 and a[j-1] % 2 !=0) or (a[j] % 2 != 0 and a[j-1] % 2 ==0) :
#                 count = count + 1
#             else:
#                 break

#         max_count = max(count , max_count)
#     return max_count

#effective apporach
def longest_even_odd_subarray(a):
    count = 1
    max_count = 1
    for i in range(1,len(a)):
        if (a[i] % 2 == 0 and a[i-1] % 2 !=0) or (a[i] % 2 != 0 and a[i-1] % 2 ==0) :
            count = count + 1
            max_count = max(count , max_count)
        else:
            count = 1

    return max_count

def main():
    a = [8,10,13,14,9,5]
    print(longest_even_odd_subarray(a))
main()
```

90_Maximum_Sum_Sub_Array_Given_Length.py

```
#finding the maximum sum subarray finding the givien length k
#nave apporach

# def max_sum_sub_array(a, k):
#     max_sum = float('-inf')

#     for i in range(0, len(a) - k + 1):
#         current_sum = 0
#         for j in range(i, i + k):
#             current_sum += a[j]
#         max_sum = max(current_sum, max_sum)

#     return max_sum

#sliding window approach
def max_sum_sub_array(a, k):

    if len(a) < k:
        return "Invalid: window size k is larger than array"

    window_sum = sum(a[:k])
    max_sum = window_sum

    for i in range(k,len(a)):
        window_sum = window_sum - a[i-k] + a[i]
        max_sum = max(window_sum , max_sum)
    return max_sum

def main():
    a = [2, 9, 31, -4, 21, 7]
    k = 3
    print(max_sum_sub_array(a, k))

main()
```

91_Minimum_Consecutive_Flips.py

```
def min_flips(ar):
    for i in range(1,len(ar)):
        if ar[i] != ar[i-1]:
            if ar[i] != ar[0]:
                print(i , " ", end=" ")
            else:
                print(i-1)

    if (ar[0] != ar[len(ar)-1]): # filps will be same , so it will not print the last index
        print(len(ar)-1)

def main():
    ar = [1,1,0,1,1,0,1,0,0,0]
    min_flips(ar)

main()
```

92_Matrix_Zig_Zag.py

```
#print matrix in zigzag format
# def matrix(a): #normally it will print
#     for i in range(0,len(a)):
#         for j in range(0,len(a[i])):
#             print(a[i][j],end=" ")
#         print()

def matrix(a): #normally it will print
    for i in range(0,len(a)):
        if i % 2 == 0:
            for j in range(0,len(a[i])):
                print(a[i][j],end=" ")
        else:
            for j in range(len(a[i])-1,-1,-1):
                print(a[i][j],end=" ")
        print()

def main():
    a = [[1,2,3,4],[5,6,7,8],[9,10,11,12],[13,14,15,16]]
    matrix(a)
main()
```

93_Matrix_Boundaries.py

```
#n*n matrix
def matrix_boundaries(a):
    n = len(a)  # since it's a square matrix

    # Top row
    for j in range(n):
        print(a[0][j], end=" ")

    # Right column (excluding top element)
    for i in range(1, n):
        print(a[i][n - 1], end=" ")

    # Bottom row (excluding last element of right column)
    for j in range(n - 2, -1, -1):
        print(a[n - 1][j], end=" ")

    # Left column (excluding first and last elements)
    for i in range(n - 2, 0, -1):
        print(a[i][0], end=" ")

    print()  # For newline

def print_boundary_box(a):
    n = len(a)  # For n x n matrix

    for i in range(n):
        for j in range(n):
            # Top row
            if i == 0:
                print(a[i][j], end=" ")

            # Bottom row
            elif i == n - 1:
                print(a[i][j], end=" ")

            # Left and right columns
            elif j == 0 or j == n - 1:
                print(a[i][j], end=" ")

            # Inner elements
            else:
                print("  ", end=" ")  # spacing to keep format
        print()  # Move to next line

def main():
    a = [[1,2,3,4],[5,6,7,8],[9,10,11,12],[13,14,15,16]]
    matrix_boundaries(a)
    print_boundary_box(a)
main()
```

94_Matrix_Boundaries_2.py

```
def matrix_boundaries(a):
    rows = len(a)
    cols = len(a[0])

    # Top row
    for j in range(cols):
        print(a[0][j], end=" ")

    # Right column (excluding top)
    for i in range(1, rows):
        print(a[i][cols - 1], end=" ")

    # Bottom row (excluding rightmost, only if more than 1 row)
    if rows > 1:
        for j in range(cols - 2, -1, -1):
            print(a[rows - 1][j], end=" ")

    # Left column (excluding top and bottom, only if more than 1 col)
    if cols > 1:
        for i in range(rows - 2, 0, -1):
            print(a[i][0], end=" ")

    print()  # Newline

def print_boundary_box(a):
    rows = len(a)
    cols = len(a[0])

    for i in range(rows):
        for j in range(cols):
            # Top row
            if i == 0:
                print(f"{a[i][j]:<3}", end=" ")

            # Bottom row
            elif i == rows - 1:
                print(f"{a[i][j]:<3}", end=" ")

            # Left and right columns
            elif j == 0 or j == cols - 1:
                print(f"{a[i][j]:<3}", end=" ")

            # Inner elements (blank)
            else:
                print("    ", end=" ")
        print()

def main():
    a = [
        [1,  2,  3,  4],
        [5,  6,  7,  8],
        [9, 10, 11, 12],
        [13,14, 15,16],
        [17,18, 19,20]
    ]

    print("Flat boundary traversal:")
    matrix_boundaries(a)

    print("\nVisual boundary box:")
    print_boundary_box(a)

main()
```

95_Search_In_Matrix.py

```
def search_matrix(a,target):
    i = 0
    j = len(a[0])-1
    while (i < len(a) and j >= 0):
        if a[i][j] == target:
            return True
        elif target < a[i][j]:
            j = j - 1
        else:
            i = i + 1
    return False

def main():
    a = [
        [1,  2,  3,  4],
        [5,  6,  7,  8],
        [9, 10, 11, 12],
        [13,14, 15,16],
        [17,18, 19,20]
    ]
    print(search_matrix(a,0))

main()
```

96_Spiral_Traversing_Matrix.py

```
def spiralOrder(matrix):
    top = 0
    bottom = len(matrix) - 1
    left = 0
    right = len(matrix[0]) - 1

    result = []
    while top <= bottom and left <= right:
        # Traverse from Left to Right
        for i in range(left, right + 1):
            result.append(matrix[top][i])
        top += 1

        # Traverse from Top to Bottom
        for i in range(top, bottom + 1):
            result.append(matrix[i][right])
        right -= 1

        # Check if bounds are still valid
        if not (top <= bottom and left <= right):
            break

        # Traverse from Right to Left
        for i in range(right, left - 1, -1):
            result.append(matrix[bottom][i])
        bottom -= 1

        # Traverse from Bottom to Top
        for i in range(bottom, top - 1, -1):
            result.append(matrix[i][left])
        left += 1

    return result

def main():
    matrix = [
        [1, 2, 3, 4, 5, 6, 7, 8],
        [9, 10, 11, 12, 13, 14, 15, 16],
        [17, 18, 19, 20, 21, 22, 23, 24],
        [25, 26, 27, 28, 29, 30, 31, 32],  # Fixed this row (removed duplicate 31)
        [33, 34, 35, 36, 37, 38, 38, 40],
        [41, 42, 43, 44, 45, 46, 47, 48]
    ]
    print(spiralOrder(matrix))

main()
```

97_Transpose_Matrix_1.py

```
def transpose(ar):
    res = []
    for i in range(0,len(ar)):
        res.append([0]*len(ar))

    for i in range(0,len(ar)):
        for j in range(0,len(ar)):
            res[j][i] = ar[i][j]
    return res

def print_matrix(ar):
     for i in range(0,len(ar)):
        for j in range(0,len(ar)):
            print(ar[i][j] , end = " ")
        print()

def main():
    ar = [[1,6,11,16,21],[2,7,12,17,22],[3,8,13,18,23],[4,9,14,19,24],[5,10,15,20,25]]
    print_matrix(ar)
    res = transpose(ar)
    print()
    print_matrix(res)

main()
```

98_Transpose_Matrix_1.py

```
def transpose(ar):
    for i in range(0,len(ar)-1):
        for j in range(i+1,len(ar)):
            ar[i][j],ar[j][i] = ar[j][i],ar[i][j]

def print_matrix(ar):
     for i in range(0,len(ar)):
        for j in range(0,len(ar)):
            print(ar[i][j] , end = " ")
        print()

def main():
    ar = [[1,6,11,16,21],[2,7,12,17,22],[3,8,13,18,23],[4,9,14,19,24],[5,10,15,20,25]]
    print_matrix(ar)
    transpose(ar)
    print()
    print_matrix(ar)

main()
```

99_Transpose_Matrix_nxm.py

```
def transpose_nonsquare(matrix):
    rows = len(matrix)
    cols = len(matrix[0])

    # Sanity check
    for row in matrix:
        if len(row) != cols:
            raise ValueError("All rows must have the same number of columns.")

    return [[matrix[i][j] for i in range(rows)] for j in range(cols)]


def print_matrix(matrix):
    for row in matrix:
        print(" ".join(map(str, row)))

def main():
    ar = [
        [1, 6, 11, 16, 21, 61],
        [2, 7, 12, 17, 22, 60],
        [3, 8, 13, 18, 23, 62],
        [4, 9, 14, 19, 24, 64],
        [5, 10, 15, 20, 25, 70]
    ]
    print("Original:")
    print_matrix(ar)

    res = transpose_nonsquare(ar)

    print("\nTransposed:")
    print_matrix(res)

main()
```

100_Reversing_The_Coloum_Matrix.py

```
def reverse_matrix(ar):
    for i in range(len(ar)):
        left = 0
        right = len(ar[i]) - 1
        while left < right:
            ar[i][left], ar[i][right] = ar[i][right], ar[i][left]
            left += 1
            right -= 1

def print_matrix(ar):
    for row in ar:
        print(" ".join(map(str, row)))

def main():
    ar = [
        [1, 6, 11, 16, 21],
        [2, 7, 12, 17, 22],
        [3, 8, 13, 18, 23],
        [4, 9, 14, 19, 24],
        [5, 10, 15, 20, 25]
    ]
    print("Original:")
    print_matrix(ar)

    reverse_matrix(ar)
    print("\nAfter reversing rows:")
    print_matrix(ar)

main()
```

101_Rotate_Image.py

```
# You are given an n x n 2D matrix representing an image, rotate the image by 90 degrees (c

# You have to rotate the image in-place, which means you have to modify the input 2D matrix



# Example 1:


# Input: matrix = [[1,2,3],[4,5,6],[7,8,9]]
# Output: [[7,4,1],[8,5,2],[9,6,3]]
# Example 2:


# Input: matrix = [[5,1,9,11],[2,4,8,10],[13,3,6,7],[15,14,12,16]]
# Output: [[15,13,2,5],[14,3,4,1],[12,6,8,9],[16,7,10,11]]

def rotate(ar):
    #transpose matrix
    for i in range(0,len(ar)-1):
        for j in range(i+1,len(ar)):
            ar[i][j],ar[j][i] = ar[j][i],ar[i][j]

    #Reverse matrix
    for i in range(len(ar)):
        left = 0
        right = len(ar[i]) - 1
        while left < right:
            ar[i][left], ar[i][right] = ar[i][right], ar[i][left]
            left += 1
            right -= 1

def print_matrix(ar):
    for row in ar:
        print(" ".join(map(str, row)))

def main():
    ar = [[5,1,9,11],[2,4,8,10],[13,3,6,7],[15,14,12,16]]
    ar = [[1,2,3],[4,5,6],[7,8,9]]
    print_matrix(ar)
    rotate(ar)
    print()
    print_matrix(ar)
main()
```

102_Linked_List.py

```
'''
LinkedList Properties

1.Every Linked List will have a reference called as head and head will always point to the
   if the listed list is empty .
   2.List List is a collection of nodes & every nodes has two parts
   3.The next part of the last node will always NULL
'''

class Node:
    def __init__(self,data):
        self.data = data
        self.next = None

class LinkedList:
    def __init__(self):
        self.head = None

    def print_linkedlist(self):
        curr = self.head

        while (curr != None):
            print(curr.data , end = ' ')
            curr = curr.next
        print()

    def add(self,e):
        temp = Node(e)
        if (self.head == None):
            self.head = temp
        else:
            curr = self.head
            while(curr.next != None):
                curr = curr.next
            curr.next = temp

    def add_first(self,e):
        temp = Node(e)
        if (self.head == None):
            self.head = temp
        else:
            temp.next = self.head
            self.head = temp

    def add_element_at(self,index,element):
        try:
            if(index == 0):
                self.add_first(element)
            else:
                temp = Node(element)
                count = 0
                curr = self.head

                while (count < (index - 1)):
                    curr = curr.next
                    count = count + 1

                temp.next = curr.next
                curr.next = temp
        except AttributeError:
            raise IndexError('Index ' + str(index) + ' does not exists')

    def add_all(self,elements):
        for element in elements:
            self.add(element)

    def remove_first(self):
        if (self.head == None):
            print("No element in Linked List")
        elif self.head.next == None:
              self.head = None
        elif self.head != None:
            curr = self.head
            self.head = self.head.next
            curr.next = None

    def remove_last(self):
        if(self.head == None):
            print('No Elements in Linked List')
        elif self.head.next == None:
            self.head = None
        elif self.head.next != None:
            curr = self.head
            while(curr.next.next != None):
                curr = curr.next
            curr.next = None

    def index_of(self,element):
        curr = self.head
        count = 0

        while(curr != None):
            if(curr.data == element):
                return count
            curr = curr.next
            count = count + 1

        return -1

    def last_index_of(self,element):
        curr = self.head
        count = 0
        index = -1

        while(curr != None):
            if(curr.data == element):
                index = count
            curr = curr.next
            count = count + 1

        return index

    def size_of(self):
        curr = self.head
        count = 0

        while(curr != None):
            curr = curr.next
            count += 1

        return count

def main():
    ll = LinkedList()

    # p1 = Node(10)
    # p2 = Node(20)
    # p3 = Node(30)
    # p4 = Node(40)
    # ll.head = p1
    # p1.next = p2
    # p2.next = p3
    # p3.next = p4

    ll.add(10)
    ll.add(20)
    ll.add(30)
    ll.add(40)

    ll.add_first(0)

    ll.add_element_at(3,25)
    ll.add_element_at(1,5)

    #ll.add_element_at(20,5) #index error
    elements = [11,22,33,33]
    ll.add_all(elements)
    #ll.add_all([11,22,33,44])
    ll.print_linkedlist()

    #ll.remove_first()
    #ll.remove_last()

    print(ll.index_of(33))

    print(ll.last_index_of(33))

    ll.print_linkedlist()

    print(ll.size_of())

if __name__ == '__main__':
    main()

# #disadvanges

# add(e) - 0(n)
# removelast - 0(n)
# Reverse Transveral is not possible
```

103_Double_Linked_List.py

```
class Node:
    def __init__(self,data):
        self.data = data
        self.prev = None
        self.next = None

class DoublyLinkedList:
    def __init__(self):
        self.head = None
        self.tail = None

    def print(self):
        curr = self.head
        while(curr != None):
            print(curr.data , end = ' ')
            curr = curr.next
        print()
    def print_reverse(self):
        curr = self.tail
        while(curr != None):
            print(curr.data , end = ' ')
            curr = curr.prev
        print()

    def add(self,e):
        temp = Node(e)
        if(self.head == None):
            self.head = temp
            self.tail = temp
        else:
            self.tail.next = temp
            temp.prev = self.tail
            self.tail = temp

    def add_first(self,e):
        temp = Node(e)
        if(self.head == None):
            self.head = temp
            self.tail = temp
        else:
            temp.next = self.head
            self.head.prev = temp
            self.head = temp

    def add_at(self, index, element):
        if index < 0:
            raise IndexError("Index cannot be negative")

        temp = Node(element)

        if index == 0:
            self.add_first(element)
            return

        curr = self.head
        count = 0

        while curr is not None and count < index - 1:
            curr = curr.next
            count += 1

        if curr is None:
            raise IndexError("Index out of bounds")

        if curr.next is None:
            # Inserting at the end
            curr.next = temp
            temp.prev = curr
            self.tail = temp
        else:
            # Inserting in the middle
            temp.next = curr.next
            curr.next.prev = temp
            curr.next = temp
            temp.prev = curr

    def add_all(self,elements):
        for element in elements:
            self.add(element)
    def remove_first(self):
        if self.head is None:
            print("List is empty. Nothing to remove.")
            return

        if self.head == self.tail:
            # Only one element in the list
            self.head = None
            self.tail = None
        else:
            # More than one element
            self.head = self.head.next
            self.head.prev = None

    def remove_last(self):
        if self.tail is None:
            print("List is empty. Nothing to remove.")
            return

        if self.head == self.tail:
            # Only one element in the list
            self.head = None
            self.tail = None
        else:
            # More than one element
            self.tail = self.tail.prev
            self.tail.next = None

    def index_of(self, element):
        curr = self.head
        index = 0

        while curr:
            if curr.data == element:
                return index
            curr = curr.next
            index += 1

        return -1  # Element not found

    def last_index_of(self, element):
        curr = self.tail
        index = self.size() - 1

        while curr:
            if curr.data == element:
                return index
            curr = curr.prev
            index -= 1

        return -1  # Element not found

    def size(self):
        count = 0
        curr = self.head
        while curr:
            count += 1
            curr = curr.next
        return count

def main():
    # temp = Node(10)
    # print(temp.data)
    # print(temp.prev)
    # print(temp.next)

    dll = DoublyLinkedList()

    dll.add(10)
    dll.add(20)
    dll.add(30)
    dll.add(40)
    dll.add(20)

    dll.add_first(75)

    dll.add_at(2,9)

    dll.add_all([1,2,3,4])

    dll.print()

    print(dll.index_of(20))      # Output: 1
    print(dll.last_index_of(20)) # Output: 3
    print(dll.size())            # Output: 4

    dll.remove_first()
    dll.remove_last()
    # print(dll.head.data)
    # print(dll.tail.data)


    dll.print()
    #dll.print_reverse()

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

104_Circular_Single_Linked_List.py

```
class Node:
    def __init__(self, data):
        self.data = data
        self.next = None

class CircularLinkedList:
    def __init__(self):
        self.head = None

    def print_linkedlist(self):
        if self.head is None:
            print("List is empty")
            return
        curr = self.head
        while True:
            print(curr.data, end=' ')
            curr = curr.next
            if curr == self.head:
                break
        print()

    def add(self, data):
        new_node = Node(data)
        if self.head is None:
            self.head = new_node
            new_node.next = self.head
        else:
            curr = self.head
            while curr.next != self.head:
                curr = curr.next
            curr.next = new_node
            new_node.next = self.head

    def add_first(self, data):
        new_node = Node(data)
        if self.head is None:
            self.head = new_node
            new_node.next = self.head
        else:
            curr = self.head
            while curr.next != self.head:
                curr = curr.next
            new_node.next = self.head
            self.head = new_node
            curr.next = self.head

    def add_element_at(self, index, data):
        if index == 0:
            self.add_first(data)
        else:
            new_node = Node(data)
            curr = self.head
            count = 0
            while count < index - 1:
                curr = curr.next
                if curr == self.head:
                    raise IndexError("Index out of bounds")
                count += 1
            new_node.next = curr.next
            curr.next = new_node

    def add_all(self, elements):
        for element in elements:
            self.add(element)

    def remove_first(self):
        if self.head is None:
            print("List is empty")
            return
        if self.head.next == self.head:
            self.head = None
        else:
            last = self.head
            while last.next != self.head:
                last = last.next
            self.head = self.head.next
            last.next = self.head

    def remove_last(self):
        if self.head is None:
            print("List is empty")
            return
        if self.head.next == self.head:
            self.head = None
        else:
            curr = self.head
            prev = None
            while curr.next != self.head:
                prev = curr
                curr = curr.next
            prev.next = self.head

    def index_of(self, value):
        curr = self.head
        index = 0
        if self.head is None:
            return -1
        while True:
            if curr.data == value:
                return index
            curr = curr.next
            index += 1
            if curr == self.head:
                break
        return -1

    def last_index_of(self, value):
        curr = self.head
        index = 0
        last_index = -1
        if self.head is None:
            return -1
        while True:
            if curr.data == value:
                last_index = index
            curr = curr.next
            index += 1
            if curr == self.head:
                break
        return last_index

    def size_of(self):
        if self.head is None:
            return 0
        count = 0
        curr = self.head
        while True:
            count += 1
            curr = curr.next
            if curr == self.head:
                break
        return count

def main():
    cll = CircularLinkedList()

    cll.add(10)
    cll.add(20)
    cll.add(30)
    cll.add_first(5)
    cll.add_element_at(2, 15)
    cll.add_all([40, 50, 50])

    cll.print_linkedlist()

    print("Index of 50:", cll.index_of(50))
    print("Last index of 50:", cll.last_index_of(50))
    print("Size:", cll.size_of())

    cll.remove_first()
    cll.remove_last()
    cll.print_linkedlist()

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

105_Circular_Doublely_Linked_List.py

```
class Node:
    def __init__(self, data):
        self.data = data
        self.next = None
        self.prev = None

class CircularDoublyLinkedList:
    def __init__(self):
        self.head = None

    def print_forward(self):
        if self.head is None:
            print("List is empty")
            return
        curr = self.head
        while True:
            print(curr.data, end=' ')
            curr = curr.next
            if curr == self.head:
                break
        print()

    def print_backward(self):
        if self.head is None:
            print("List is empty")
            return
        curr = self.head.prev  # last node
        while True:
            print(curr.data, end=' ')
            curr = curr.prev
            if curr == self.head.prev:
                break
        print()

    def add(self, data):
        new_node = Node(data)
        if self.head is None:
            new_node.next = new_node.prev = new_node
            self.head = new_node
        else:
            tail = self.head.prev
            tail.next = new_node
            new_node.prev = tail
            new_node.next = self.head
            self.head.prev = new_node

    def add_first(self, data):
        self.add(data)
        self.head = self.head.prev
    def remove_first(self):
        if self.head is None:
            print("List is empty")
            return
        if self.head.next == self.head:
            self.head = None
        else:
            tail = self.head.prev
            self.head = self.head.next
            self.head.prev = tail
            tail.next = self.head

    def remove_last(self):
        if self.head is None:
            print("List is empty")
            return
        if self.head.next == self.head:
            self.head = None
        else:
            tail = self.head.prev
            new_tail = tail.prev
            new_tail.next = self.head
            self.head.prev = new_tail

    def size(self):
        if self.head is None:
            return 0
        count = 0
        curr = self.head
        while True:
            count += 1
            curr = curr.next
            if curr == self.head:
                break
        return count

def main():
    cdll = CircularDoublyLinkedList()

    cdll.add(10)
    cdll.add(20)
    cdll.add_first(5)
    cdll.add(30)

    print("Forward:")
    cdll.print_forward()

    print("Backward:")
    cdll.print_backward()

    print("Size:", cdll.size())

    cdll.remove_first()
    cdll.remove_last()

    print("After Deletions:")
    cdll.print_forward()

if __name__ == "__main__":
    main()
```

106_Stack(List).py

```
# A stack is a linear data structure that follows the LIFO (Last In, First Out) principle —
# the last element added is the first one to be removed.

# ■ Key Stack Operations
# Operation■Description
# push(x)■Add element x to the top
# pop()■Remove and return the top element
# peek() / top()■Return the top element without removing it
# is_empty()■Check if the stack is empty
# size()■Return the number of elements

class Stack:
    def __init__(self):
        self.stack = []

    def push(self, data):
        self.stack.append(data)

    def pop(self):
        if self.is_empty():
            raise IndexError("Pop from empty stack")
        return self.stack.pop()

    def peek(self):
        if self.is_empty():
            raise IndexError("Peek from empty stack")
        return self.stack[-1]

    def is_empty(self):
        return len(self.stack) == 0

    def size(self):
        return len(self.stack)

    def display(self):
        print("Stack (top -> bottom):", self.stack[::-1])

def main():
    s = Stack()
    s.push(10)
    s.push(20)
    s.push(30)
    s.display()

    print("Top:", s.peek())
    print("Pop:", s.pop())
    s.display()

    print("Is Empty:", s.is_empty())
    print("Size:", s.size())

if __name__ == "__main__":
    main()
```

107_Stack(Linked_List).py

```
class Node:
    def __init__(self, data):
        self.data = data
        self.next = None

class Stack:
    def __init__(self):
        self.top = None  # top points to the head node
        self._size = 0

    def push(self, data):
        new_node = Node(data)
        new_node.next = self.top
        self.top = new_node
        self._size += 1

    def pop(self):
        if self.is_empty():
            raise IndexError("Pop from empty stack")
        popped = self.top.data
        self.top = self.top.next
        self._size -= 1
        return popped

    def peek(self):
        if self.is_empty():
            raise IndexError("Peek from empty stack")
        return self.top.data

    def is_empty(self):
        return self.top is None

    def size(self):
        return self._size

    def display(self):
        curr = self.top
        print("Stack (top -> bottom):", end=" ")
        while curr:
            print(curr.data, end=" ")
            curr = curr.next
        print()


def main():
    s = Stack()
    s.push(10)
    s.push(20)
    s.push(30)
    s.display()

    print("Top:", s.peek())
    print("Pop:", s.pop())
    s.display()

    print("Is Empty:", s.is_empty())
    print("Size:", s.size())

if __name__ == "__main__":
    main()
```

108_Closest_Smallest.py

```
#Find the closet smallest element towards the left of the given array
```

```
def nearest_smaller_to_left(arr):
    result = []
    stack = []

    for num in arr:
        while stack and stack[-1] >= num:
            stack.pop()
        if not stack:
            result.append(-1)
        else:
            result.append(stack[-1])
        stack.append(num)

    return result

# Example usage
arr = [4, 5, 2, 10, 8]
print("Input:", arr)
print("Output:", nearest_smaller_to_left(arr))
```

109_Stock_Span_Problem.py

```
def calculate_stock_span(prices):
    n = len(prices)
    span = [0] * n
    stack = []  # stores index of prices

    for i in range(n):
        # Pop elements from stack while stack is not empty and price[stack top] <= price[i]
        while stack and prices[stack[-1]] <= prices[i]:
            stack.pop()

        # If stack is empty, span = i + 1 (no greater element to the left)
        span[i] = i + 1 if not stack else i - stack[-1]

        # Push this element's index to stack
        stack.append(i)

    return span

def main():
    prices = [100, 80, 60, 70, 60, 75, 85]
    print("Prices:", prices)
    print("Spans: ", calculate_stock_span(prices))

if __name__ == "__main__":
    main()
```

110_infix_prefix_postfix.py

```
# ■ 1. What Are These Notations?
# Notation■Format■Example for A + B
# Infix■Operator is between operands■A + B
# Prefix■Operator is before operands■+ A B
# Postfix■Operator is after operands■A B +

# ■ Why Use Prefix/Postfix?
# Infix is natural for humans but needs parentheses and operator precedence.
# Prefix/Postfix is easier for computers — no need for parentheses or operator precedence r

# ■ Conversion Techniques
# All conversions involve using stacks effectively.


# ■ 2. Infix → Postfix (Shunting Yard Algorithm)
# Rules:
# Operands go directly to output.

# Operators go to a stack (pop based on precedence).

# Parentheses handled specially:

# Push '(',

# On ')', pop until '('.

def infix_to_postfix(expression):
    precedence = {'+':1, '-':1, '*':2, '/':2, '^':3}
    stack = []
    output = []

    for token in expression:
        if token.isalnum():  # Operand
            output.append(token)
        elif token == '(':
            stack.append(token)
        elif token == ')':
            while stack and stack[-1] != '(':
                output.append(stack.pop())
            stack.pop()  # Remove '('
        else:  # Operator
            while (stack and stack[-1] != '(' and
                   precedence.get(stack[-1], 0) >= precedence.get(token, 0)):
                output.append(stack.pop())
            stack.append(token)

    while stack:
        output.append(stack.pop())

    return ''.join(output)

# Example:
exp = "(A+B)*C"
print(infix_to_postfix(exp))  # Output: AB+C*


# ■ 3. Infix → Prefix
# Trick: Reverse the infix expression, swap ( and ), convert to postfix, then reverse the r

def infix_to_prefix(expression):
    def reverse_expr(expr):
        expr = expr[::-1]
        expr = ['(' if ch == ')' else ')' if ch == '(' else ch for ch in expr]
        return expr

    reversed_expr = reverse_expr(expression)
    postfix = infix_to_postfix(reversed_expr)
    return postfix[::-1]

# Example:
```

```
exp = "(A+B)*C"
print(infix_to_prefix(exp))  # Output: *+ABC

# ■ 4. Postfix → Infix
# Use a stack:

# Push operands

# On operator: pop two operands, combine "(a op b)", push result back.

def postfix_to_infix(expression):
    stack = []
    for token in expression:
        if token.isalnum():
            stack.append(token)
        else:
            b = stack.pop()
            a = stack.pop()
            stack.append(f"({a}{token}{b})")
    return stack[0]

# ■ 5. Prefix → Infix
# Reverse process of Postfix → Infix.

def prefix_to_infix(expression):
    stack = []
    for token in reversed(expression):
        if token.isalnum():
            stack.append(token)
        else:
            a = stack.pop()
            b = stack.pop()
            stack.append(f"({a}{token}{b})")
    return stack[0]

# Conversion■Approach
# Infix → Postfix■Shunting Yard (stack)
# Infix → Prefix■Reverse + postfix + reverse
# Postfix → Infix■Stack, combine a op b
# Prefix → Infix■Stack, reversed, combine a op b
```

111_Queue_With_Array.py

```
# Here’s a complete explanation of the Queue data structure along with an implementation us

# Operation■Description
# enqueue■Add element at the rear
# dequeue■Remove element from the front
# get_front■Return front element
# get_rear■Return rear element
# is_empty■Check if queue is empty
# is_full■Check if queue is full (in fixed size)

# ■ What is a Queue?
# A Queue is a linear data structure that follows the FIFO principle:

# First In First Out – The first element inserted is the first to be removed.

#Linear Queue

class LinearQueue:
    def __init__(self, capacity):
        self.capacity = capacity
        self.queue = [None] * capacity
        self.front = 0
        self.rear = -1

    def is_empty(self):
        return self.front > self.rear

    def is_full(self):
        return self.rear == self.capacity - 1

    def enqueue(self, data):
        if self.is_full():
            print("Queue is full")
            return
        self.rear += 1
        self.queue[self.rear] = data

    def dequeue(self):
        if self.is_empty():
            print("Queue is empty")
            return None
        removed = self.queue[self.front]
        self.queue[self.front] = None  # Optional clear
        self.front += 1
        return removed

    def get_front(self):
        if self.is_empty():
            return "Queue is empty"
        return self.queue[self.front]

    def get_rear(self):
        if self.is_empty():
            return "Queue is empty"
        return self.queue[self.rear]

    def display(self):
        if self.is_empty():
            print("Queue is empty")
        else:
            print("Queue elements:", end=' ')
            for i in range(self.front, self.rear + 1):
                print(self.queue[i], end=' ')
            print()

def main():
    q = LinearQueue(5)

    q.enqueue(10)
    q.enqueue(20)
    q.enqueue(30)
    q.enqueue(40)
    q.enqueue(50)

    q.display()

    print("Front:", q.get_front())
    print("Rear:", q.get_rear())

    q.dequeue()
    q.dequeue()

    q.display()

    print("Is Empty:", q.is_empty())
    print("Is Full:", q.is_full())

main()
```

112_Circular_Queue.py

```
class Queue:
    def __init__(self, capacity):
        self.capacity = capacity
        self.queue = [None] * capacity
        self.front = 0
        self.rear = -1
        self.size = 0

    def is_empty(self):
        return self.size == 0

    def is_full(self):
        return self.size == self.capacity

    def enqueue(self, data):
        if self.is_full():
            print("Queue is full")
            return
        self.rear = (self.rear + 1) % self.capacity
        self.queue[self.rear] = data
        self.size += 1

    def dequeue(self):
        if self.is_empty():
            print("Queue is empty")
            return
        removed = self.queue[self.front]
        self.queue[self.front] = None  # Optional: Clear the slot
        self.front = (self.front + 1) % self.capacity
        self.size -= 1
        return removed

    def get_front(self):
        if self.is_empty():
            return "Queue is empty"
        return self.queue[self.front]

    def get_rear(self):
        if self.is_empty():
            return "Queue is empty"
        return self.queue[self.rear]

    def display(self):
        if self.is_empty():
            print("Queue is empty")
            return
        print("Queue elements:", end=' ')
        count = 0
        i = self.front
        while count < self.size:
            print(self.queue[i], end=' ')
            i = (i + 1) % self.capacity
            count += 1
        print()

def main():
    q = Queue(5)

    q.enqueue(10)
    q.enqueue(20)
    q.enqueue(30)
    q.enqueue(40)
    q.enqueue(50)

    q.display()

    print("Front:", q.get_front())
    print("Rear:", q.get_rear())

    q.dequeue()
    q.dequeue()

    q.display()

    print("Is Empty:", q.is_empty())
    print("Is Full:", q.is_full())

main()
```

113_Queue_Linked_List.py

```
class Node:
    def __init__(self, data):
        self.data = data
        self.next = None

class LinkedListQueue:
    def __init__(self):
        self.front = None
        self.rear = None

    def is_empty(self):
        return self.front is None

    def enqueue(self, data):
        new_node = Node(data)
        if self.rear is None:
            self.front = self.rear = new_node
            return
        self.rear.next = new_node
        self.rear = new_node

    def dequeue(self):
        if self.is_empty():
            print("Queue is empty")
            return None
        removed = self.front.data
        self.front = self.front.next
        if self.front is None:
            self.rear = None
        return removed

    def get_front(self):
        if self.is_empty():
            return "Queue is empty"
        return self.front.data

    def get_rear(self):
        if self.is_empty():
            return "Queue is empty"
        return self.rear.data

    def display(self):
        if self.is_empty():
            print("Queue is empty")
            return
        print("Queue elements:", end=' ')
        current = self.front
        while current:
            print(current.data, end=' ')
            current = current.next
        print()

def main():
    q = LinkedListQueue()

    q.enqueue(10)
    q.enqueue(20)
    q.enqueue(30)
    q.enqueue(40)

    q.display()

    print("Front:", q.get_front())
    print("Rear:", q.get_rear())

    q.dequeue()
    q.dequeue()

    q.display()

    print("Is Empty:", q.is_empty())

main()
```

114_Reversing_Queue.py

```
from queue import Queue

def reverse_queue(q):
    stack = []

    # Dequeue all elements and push into stack
    while not q.empty():
        stack.append(q.get())

    # Pop from stack and enqueue back to queue
    while stack:
        q.put(stack.pop())

# Example usage
```

```
def main():
    q = Queue()
    for i in [10, 20, 30, 40, 50]:
        q.put(i)

    print("Original queue:")
    print(list(q.queue))

    reverse_queue(q)

    print("Reversed queue:")
    print(list(q.queue))

main()
```

115_56_Number_Series.py

```
# from queue import Queue

# def generate_5_6_series(n):
#     q = Queue()
#     q.put("5")
#     q.put("6")

#     result = []

#     while len(result) < n:
#         curr = q.get()
#         result.append(curr)

#         q.put(curr + "5")
#         q.put(curr + "6")

#     return result

# # Example
# n = 12
# output = generate_5_6_series(n)
# print(" ".join(output))

from collections import deque

def generate_5_6_series(n):
    result = []
    q = deque()
    q.append("5")
    q.append("6")

    while len(result) < n:
        curr = q.popleft()
        result.append(curr)
        q.append(curr + "5")
        q.append(curr + "6")

    return result

# Example
n = 12
output = generate_5_6_series(n)
print(" ".join(output))
```

116_Deque.py

```python
# Here’s a complete implementation of a Deque (Double-Ended Queue) using a Doubly Linked Li

# ■ Features Implemented:
# offer_first(data) – Add at front

# offer_last(data) – Add at rear

# poll_first() – Remove from front

# poll_last() – Remove from rear

# peek_first() – View front

# peek_last() – View rear

# is_empty() – Check if empty

# size() – Return current size

class Node:
    def __init__(self, data):
        self.data = data
        self.prev = None
        self.next = None

class Deque:
    def __init__(self):
        self.front = None
        self.rear = None
        self._size = 0

    def is_empty(self):
        return self._size == 0

    def size(self):
        return self._size

    def offer_first(self, data):
        new_node = Node(data)
        if self.is_empty():
            self.front = self.rear = new_node
        else:
            new_node.next = self.front
            self.front.prev = new_node
            self.front = new_node
        self._size += 1

    def offer_last(self, data):
        new_node = Node(data)
        if self.is_empty():
            self.front = self.rear = new_node
        else:
            new_node.prev = self.rear
            self.rear.next = new_node
            self.rear = new_node
        self._size += 1
    def poll_first(self):
        if self.is_empty():
            print("Deque is empty")
            return None
        removed_data = self.front.data
        self.front = self.front.next
        if self.front:
            self.front.prev = None
        else:
            self.rear = None  # Deque becomes empty
        self._size -= 1
        return removed_data

    def poll_last(self):
        if self.is_empty():
            print("Deque is empty")
            return None
        removed_data = self.rear.data
        self.rear = self.rear.prev
        if self.rear:
            self.rear.next = None
        else:
            self.front = None  # Deque becomes empty
        self._size -= 1
        return removed_data

    def peek_first(self):
        if self.is_empty():
            return None
        return self.front.data

    def peek_last(self):
        if self.is_empty():
            return None
        return self.rear.data

# Example usage
def main():
    dq = Deque()
    dq.offer_last(10)
    dq.offer_first(20)
    dq.offer_last(30)
    dq.offer_first(40)

    print("Front:", dq.peek_first())  # 40
    print("Rear:", dq.peek_last())    # 30
    print("Size:", dq.size())         # 4

    print("Poll First:", dq.poll_first())  # 40
    print("Poll Last:", dq.poll_last())    # 30
    print("Size after polling:", dq.size())  # 2

    print("Is Empty?", dq.is_empty())  # False

main()
```

117_Maximum_Sum_Of_All_Sub_Array.py

```
#Find the maximum number of all sub arrays of size k
from collections import deque
```

```
def max_subarrays(arr, k):
    if k > len(arr):
        return []

    dq = deque()  # stores indices
    result = []

    for i in range(len(arr)):
        # Remove elements not in current window
        while dq and dq[0] <= i - k:
            dq.popleft()

        # Remove elements smaller than current from the rear
        while dq and arr[dq[-1]] < arr[i]:
            dq.pop()

        dq.append(i)

        # Start adding to result after first k elements
        if i >= k - 1:
            result.append(arr[dq[0]])

    return result

# Example usage
arr = [10, 5, 2, 7, 8, 7]
k = 3
print(max_subarrays(arr, k))
```

118.py

```
def zoho(n):
    odd = 1
    even = 2
    for i in range(1,n+1):
        for j in range(i):
            if i % 2 == 0:
                print(even, end=' ')
                even += 2
            else:
                print(odd, end=' ')
                odd += 2
        print()
zoho(7)
```