In [2]:
```
# Q1. Write a program to find all pairs of an integer array whose sum is equal to a given number?
def getPairsCount(arr, n, sum):

    count = 0  # Initialize result

    # Consider all possible pairs
    # and check their sums
for i in range(0, n):
        for j in range(i + 1, n):
            if arr[i] + arr[j] == sum:
                count += 1

    return count



arr = [1, 5, 7, -1, 5]
n = len(arr)
sum = 6
print("Count of pairs is",
      getPairsCount(arr, n, sum))
```

```
Count of pairs is 3
```

In [3]:
```
# Q2. Write a program to reverse an array in place? In place means you cannot create a new array.
# You have to update the original array.
def reverseList(A, start, end):
    while start < end:
        A[start], A[end] = A[end], A[start]
        start += 1
        end -= 1


A = [1, 2, 3, 4, 5, 6]
print(A)
reverseList(A ,0, 5)
print("Reversed list is")
print(A)
```

```
[1, 2, 3, 4, 5, 6]
Reversed list is
[6, 5, 4, 3, 2, 1]
```

In [16]:
```
# Q3. Write a program to check if two strings are a rotation of each other?
def areRotations(string1, string2):
    size1 = len(string1)
    size2 = len(string2)
    temp = ''


    if size1 != size2:
        return 0


    temp = string1 + string1


    if (temp.count(string2)> 0):
        return 1
    else:
        return 0


string1 = "AACD"
string2 = "ACDA"

if areRotations(string1, string2):
    print ("Strings are rotations of each other")
else:
    print ("Strings are not rotations of each other")
```

```
Strings are rotations of each other
```

In [30]:
```
s = "Iam a billionaire"
while s != "":
    slen0 = len(s)
    ch = s[0]
    s = s.replace(ch, "")
    slen1 = len(s)
    if slen1 == slen0-1:
        print ("First non-repeating character is: ",ch)
        break;
    else:
        print ("No Unique Character Found!")
```

```
First non-repeating character is:  I
```

In [9]:
```
# Q5.Read about the Tower of Hanoi algorithm. Write a program to implement it.
def TowerOfHanoi(n , from_rod, to_rod, aux_rod):
    if n == 1:
        print("Move disk 1 from rod",from_rod,"to rod",to_rod)
        return
    TowerOfHanoi(n-1, from_rod, aux_rod, to_rod)
    print("Move disk",n,"from rod",from_rod,"to rod",to_rod)
    TowerOfHanoi(n-1, aux_rod, to_rod, from_rod)


n = 4
TowerOfHanoi(n, 'A', 'C', 'B')
```

```
Move disk 1 from rod A to rod B
Move disk 2 from rod A to rod C
Move disk 1 from rod B to rod C
Move disk 3 from rod A to rod B
Move disk 1 from rod C to rod A
Move disk 2 from rod C to rod B
Move disk 1 from rod A to rod B
Move disk 4 from rod A to rod C
Move disk 1 from rod B to rod C
Move disk 2 from rod B to rod A
Move disk 1 from rod C to rod A
Move disk 3 from rod B to rod C
Move disk 1 from rod A to rod B
Move disk 2 from rod A to rod C
Move disk 1 from rod B to rod C
```

In [10]:
```
# Q6. Read about infix, prefix, and postfix expressions. Write a program to convert postfix to prefix expression.
def isOperator(x):

    if x == "+":
        return True

    if x == "-":
        return True

    if x == "/":
        return True

    if x == "*":
        return True

    return False


def postToPre(post_exp):

    s = []

    length = len(post_exp)

    for i in range(length):

        if (isOperator(post_exp[i])):

            op1 = s[-1]
            s.pop()
            op2 = s[-1]
            s.pop()

            temp = post_exp[i] + op2 + op1

            s.append(temp)

        else:

            s.append(post_exp[i])


    ans = ""
    for i in s:
        ans += i
    return ans


if __name__ == "__main__":

    post_exp = "AB+CD-"

    print("Prefix : ", postToPre(post_exp))
```

```
Prefix :  +AB-CD
```

In [11]:
```
# Q7. Write a program to convert prefix expression to infix expression.
def prefixToInfix(prefix):
    stack = []

    i = len(prefix) - 1
    while i >= 0:
        if not isOperator(prefix[i]):

            stack.append(prefix[i])
            i -= 1
        else:

            str = "(" + stack.pop() + prefix[i] + stack.pop() + ")"
            stack.append(str)
            i -= 1

    return stack.pop()

def isOperator(c):
    if c == "+" or c == "-" or c == "*" or c == "/" or c == "^" or c == "(" or c == ")":
        return True
    else:
        return False

if __name__=="__main__":
    str = "-A/BC-/AKL"
    print(prefixToInfix(str))
```

```
((A-(B/C))*((A/K)-L))
```

In [13]:
```
#Q8. Write a program to check if all the brackets are closed in a given code snippet.
def areBracketsBalanced(expr):
    stack = []

    for char in expr:
        if char in ["(", "{", "["]:


            stack.append(char)
        else:


            if not stack:
                return False
            current_char = stack.pop()
            if current_char == '(':
                if char != ")":
                    return False
            if current_char == '{':
                if char != "}":
                    return False
            if current_char == '[':
                if char != "]":
                    return False


    if stack:
        return False
    return True


if __name__ == "__main__":
    expr = "{()}[]"

    if areBracketsBalanced(expr):
        print("Balanced")
    else:
        print("Not Balanced")
```

```
Balanced
```

In [14]:
```
# Q9. Write a program to reverse a stack.
class Stack:

    def __init__(self):
        self.Elements = []


    def push(self, value):
        self.Elements.append(value)

    def pop(self):
        return self.Elements.pop()

    def empty(self):
        return self.Elements == []

    def show(self):
        for value in reversed(self.Elements):
            print(value)

def BottomInsert(s, value):

    if s.empty():

        s.push(value)

    else:
        popped = s.pop()
        BottomInsert(s, value)
        s.push(popped)

def Reverse(s):
    if s.empty():
        pass
    else:
        popped = s.pop()
        Reverse(s)
        BottomInsert(s, popped)


stk = Stack()

stk.push(1)
stk.push(2)
stk.push(3)
stk.push(4)
stk.push(5)

print("Original Stack")
stk.show()

print("\nStack after Reversing")
Reverse(stk)
stk.show()
```

```
Original Stack
5
4
3
2
1

Stack after Reversing
1
2
3
4
5
```

In [15]:
```
# Q10. Write a program to find the smallest number using a stack.
import sys

def print2Smallest(arr):


    arr_size = len(arr)
    if arr_size < 2:
        print ("Invalid Input")
        return


    first = second = sys.maxsize
    for i in range(0, arr_size):


        if arr[i] < first:
            second = first
            first = arr[i]

        elif (arr[i] < second and arr[i] != first):
            second = arr[i];

    if (second == sys.maxsize):
        print("No second smallest element")
    else:
        print ('The smallest element is',first,'and' \
               ' second smallest element is',second)

arr = [12, 13, 1, 10, 34, 1]
print2Smallest(arr)
```

```
The smallest element is 1 and second smallest element is 10
```

In [ ]:
```

```