Q1. Write a program to find all pairs of an integer array whose sum is equal to a given number?

In [1]:
```
def getPairsCount(arr, n, sum):


        count = 0
        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 [ ]:

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.

In [2]:
```
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 [ ]:

Q3. Write a program to check if two strings are a rotation of each other?

In [3]:

```
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 [ ]:

Q4. Write a program to print the first non-repeated character from a string?

In [4]:
```
NO_OF_CHARS = 256

def getCharCountArray(string):
    count = [0] * NO_OF_CHARS
    for i in string:
        count[ord(i)]+= 1
    return count

def firstNonRepeating(string):
    count = getCharCountArray(string)
    index = -1
    k = 0

    for i in string:
        if count[ord(i)] == 1:
            index = k
            break
        k += 1

    return index

string = "geeksforgeeks"
index = firstNonRepeating(string)
if index == 1:
    print ("Either all characters are repeating or string is empty")
else:
    print ("First non-repeating character is " + string[index])
```

```
First non-repeating character is f
```

In [ ]:
```
```

Q5. Read about the Tower of Hanoi algorithm. Write a program to implement it.

In [5]:
```
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 [ ]:

Q6. Read about infix, prefix, and postfix expressions. Write a program to convert postfix to prefix expression.

In [6]:
```
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 [ ]:

Q7. Write a program to convert prefix expression to infix expression.

In [7]:

```
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 [ ]:

Q8. Write a program to check if all the brackets are closed in a given code snippet.

In [8]:

```
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 [ ]:

Q10. Write a program to find the smallest number using a stack.

In [9]:

```
class MinStack(object):
   min=float('inf')
   def __init__(self):
      self.min=float('inf')
      self.stack = []
   def push(self, x):
      if x<=self.min:
         self.stack.append(self.min)
         self.min = x
      self.stack.append(x)
   def pop(self):
      t = self.stack[-1]
      self.stack.pop()
      if self.min == t:
         self.min = self.stack[-1]
         self.stack.pop()
   def top(self):
      return self.stack[-1]
   def getMin(self):
      return self.min
m = MinStack()
m.push(-2)
m.push(0)
m.push(-3)
print(m.getMin())
m.pop()
```

```
print(m.top())
print(m.getMin())
```

```
-3
0
-2
```

In [ ]:

```
```