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

In [20]:
```
def find(array, len, summ):
    print("Pairs whose sum is : ", summ)
    for i in range(len):
        for j in range(i, len):
            if (array[i] + array[j]) == summ:
                print(array[i], array[j])

array = []
n = int(input("Enter number of elements : "))
for i in range(0, n):
    ele = int(input())
    array.append(ele)
summ = int(input())

print("Array= ", array)

find(array, len(array), summ)
```

```
Enter number of elements : 7
5
2
3
4
1
6
7
7
Array=  [5, 2, 3, 4, 1, 6, 7]
Pairs whose sum is :  7
5 2
3 4
1 6
```

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 [22]:
```
def reverseList(A):
    print( A[::-1])

A = []
n = int(input("Enter number of elements : "))
for i in range(0, n):
    ele = int(input())
    A.append(ele)

print("Array= ", A)
reverseList(A)
```

```
Enter number of elements : 5
10
20
30
40
50
Array=  [10, 20, 30, 40, 50]
[50, 40, 30, 20, 10]
```

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

In [23]:
```
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 =input()
string2 = input()

if areRotations(string1, string2):

    print("Strings are rotations of each other")
else:
    print("Strings are not rotations of each other")
```

```
hello
llohe
Strings are rotations of each other
```

In [ ]:
```
Q4. Write a program to print the first non-repeated character from a string?
```

In [24]:
```
string = input()
index = -1
fnc = ""
for i in string:
    if string.count(i) == 1:
        fnc += i
        break
    else:
        index += 1
if index == 1:
    print("Either all characters are repeating or string is empty")
else:
    print("First non-repeating character is", fnc)
```

```
jupyter
First non-repeating character is j
```

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

In [1]:
```
def tower_of_hanoi(disks, source, auxiliary, target):
    if(disks == 1):
        print('Move disk 1 from rod {} to rod {}.'.format(source, target))
        return

    tower_of_hanoi(disks - 1, source, target, auxiliary)
    print('Move disk {} from rod {} to rod {}.'.format(disks, source, target))
    tower_of_hanoi(disks - 1, auxiliary, source, target)


disks = int(input('Enter the number of disks: '))
tower_of_hanoi(disks, 'A', 'B', 'C')
```

```
Enter the number of disks: 3
Move disk 1 from rod A to rod C.
Move disk 2 from rod A to rod B.
Move disk 1 from rod C to rod B.
Move disk 3 from rod A to rod C.
Move disk 1 from rod B to rod A.
Move disk 2 from rod B to rod C.
Move disk 1 from rod A to rod C.
```

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

In [8]:
```
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 = input()
    print("Prefix : ", postToPre(post_exp))
```

```
ABC/-AK/L-*
Prefix :  *-A/BC-/AKL
```

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

In [10]:
```
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 = input()
    print(prefixToInfix(str))
```

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

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

In [14]:
```
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 = input()
    if areBracketsBalanced(expr):
        print("Balanced")
    else:
        print("Not Balanced")
```

```
[(){}]
Balanced
```

Q9. Write a program to reverse a stack.

In [17]:
```
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 bottom_insert(s, value):
    if s.empty():
        s.push(value)
    else:
        popped = s.pop()
        bottom_insert(s, value)
        s.push(popped)

def reverse(s):
        if s.empty():
            pass
        else:
            popped = s.pop()
            reverse(s)
            bottom_insert(s, popped)

stack = Stack()
stack.push(1)
stack.push(2)
stack.push(3)
stack.push(4)
stack.push(5)

print("original stack:")
stack.show()

print("reverse order:")
reverse(stack)
stack.show()
```

```
original stack:
5
4
3
2
1
reverse order:
1
2
3
4
5
```

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

In [19]:
```
stack = []
n=int(input())
for i in range(n):
    ele=int(input())
    stack.append(ele)

stack.sort()

print("Smallest element is:", stack[0])
```

```
5
10
2
6
9
20
Smallest element is: 2
```

In [ ]: