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

In [1]:
```
def find_pairs(arr, target):
    pairs = []
    for i in range(len(arr)):
        for j in range(i+1, len(arr)):
            if arr[i] + arr[j] == target:
                pairs.append((arr[i], arr[j]))
    return pairs
```

In [5]:
```
arr = [2, 5, 6, 3, 9, 1, 8]
target = 11
pairs = find_pairs(arr,target)
print(pairs)
```

```
[(2, 9), (5, 6), (3, 8)]
```

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 [26]:
```
def reverse_array(arr):
    n = len(arr)
    for i in range(n // 2):
        arr[i], arr[n - i - 1] = arr[n - i - 1], arr[i]


arr = [1, 2, 3, 4, 5]
print("Original array:", arr)
reverse_array(arr)
print("Reversed array:", arr)
```

```
Original array: [1, 2, 3, 4, 5]
Reversed array: [5, 4, 3, 2, 1]
```

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

In [27]:
```
def is_rotation(s1, s2):
    if len(s1) != len(s2):
        return False
    s1s1 = s1 + s1
    return s2 in s1s1


print(is_rotation('abcd', 'dabc'))
print(is_rotation('hello', 'loleh'))
print(is_rotation('foo', 'bar'))
```

```
True
False
False
```

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

In [30]:
```
def find_first_non_repeated_char(string):
    char_freq = {}
    for char in string:
        if char in char_freq:
            char_freq[char] += 1
        else:
            char_freq[char] = 1
    for char in string:
        if char_freq[char] == 1:
            return char
    return None


print(find_first_non_repeated_char("hello"))
print(find_first_non_repeated_char("leetcode"))
print(find_first_non_repeated_char("aaaaa"))
```

```
h
l
None
```

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

In [32]:
```
def tower_of_hanoi(n, source, destination, auxiliary):

    if n == 1:
        print(f"Move disc 1 from peg {source} to peg {destination}")
        return

    tower_of_hanoi(n-1, source, auxiliary, destination)
    print(f"Move disc {n} from peg {source} to peg {destination}")
    tower_of_hanoi(n-1, auxiliary, destination, source)


tower_of_hanoi(3, 1, 3, 2)
```

```
Move disc 1 from peg 1 to peg 3
Move disc 2 from peg 1 to peg 2
Move disc 1 from peg 3 to peg 2
Move disc 3 from peg 1 to peg 3
Move disc 1 from peg 2 to peg 1
Move disc 2 from peg 2 to peg 3
Move disc 1 from peg 1 to peg 3
```

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

In [33]:
```
def postfix_to_prefix(postfix):
    stack = []
    for token in postfix:
        if token.isalnum():
            stack.append(token)
        else:
            operand2 = stack.pop()
            operand1 = stack.pop()
            new_expr = token + operand1 + operand2
            stack.append(new_expr)
    return stack[-1]


postfix_expr = '34+2*'
prefix_expr = postfix_to_prefix(postfix_expr)
print(prefix_expr) # expected output: '*+342'
```

```
*+342
```

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

In [35]:
```
def prefix_to_infix(prefix_expr):
    stack = []
    prefix_expr = prefix_expr[::-1]
    for token in prefix_expr:
        if token.isalnum():

            stack.append(token)
        else:
            operand1 = stack.pop()
            operand2 = stack.pop()
            infix_expr = f"({operand1} {token} {operand2})"
            stack.append(infix_expr)
    return stack.pop()


prefix_expr = "*+AB-CD"
infix_expr = prefix_to_infix(prefix_expr)
print(infix_expr) # Output: ((A+B)*(C-D))
```

```
((A + B) * (C - D))
```

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

In [1]:
```
def is_balanced(code):
    stack = []
    for char in code:
        if char in ["(", "{", "["]:
            stack.append(char)
        elif char in [")", "}", "]"]:
            if not stack:
                return False
            if char == ")" and stack[-1] == "(":
                stack.pop()
            elif char == "}" and stack[-1] == "{":
                stack.pop()
            elif char == "]" and stack[-1] == "[":
                stack.pop()
            else:
                return False
    return len(stack) == 0

code_snippet = input("Enter code snippet: ")
if is_balanced(code_snippet):
    print("All brackets are closed.")
else:
    print("Some brackets are not closed.")
```

```
Enter code snippet: ( ) { } [ ]
All brackets are closed.
```

Q9. Write a program to reverse a stack.

In [2]:
```
class Stack:
    def __init__(self):
        self.items = []

    def push(self, item):
        self.items.append(item)

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

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

    def peek(self):
        return self.items[-1]

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

    def __str__(self):
        return str(self.items)


def reverse_stack(stack):
    if stack.is_empty():
        return

    temp = stack.pop()
    reverse_stack(stack)
    insert_at_bottom(stack, temp)


def insert_at_bottom(stack, item):
    if stack.is_empty():
        stack.push(item)
        return

    temp = stack.pop()
    insert_at_bottom(stack, item)
    stack.push(temp)


# Example
stack = Stack()
stack.push(1)
stack.push(2)
stack.push(3)
stack.push(4)

print("Original stack:", stack)
reverse_stack(stack)
print("Reversed stack:", stack)
```

```
Original stack: [1, 2, 3, 4]
Reversed stack: [4, 3, 2, 1]
```

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

In [4]:
```
class Stack:
    def __init__(self):
        self.stack = []

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

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

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

    def peek(self):
        return self.stack[-1]

def smallest_number(stack):
    if stack.is_empty():
        return None

    min_num = stack.pop()
    while not stack.is_empty():
        num = stack.pop()
        if num < min_num:
            min_num = num

    return min_num

# Example
s = Stack()
s.push(4)
s.push(2)
s.push(7)
s.push(10)
s.push(8)
s.push(3)

print(smallest_number(s))
```

```
2
```

In [ ]: