In [1]:
```python
# #   Q1.Write a program to find all pairs of an integer array whose sum is equal to a g

def find_pairs(arr, target_sum):
    pairs = []
    seen = set()

    for num in arr:
        complement = target_sum - num
        if complement in seen:
            pairs.append((num, complement))
        seen.add(num)

    return pairs


arr = [1, 2, 3, 4, 5, 6]
target = 7
result = find_pairs(arr, target)
print(result)
```

```
[(4, 3), (5, 2), (6, 1)]
```

In [3]:
```python
 #Q2. Write a program to reverse an array in place? In place means you cannot create a n




def reverse_array(arr):
    start = 0
    end = len(arr) - 1

    while start < end:
        arr[start], arr[end] = arr[end], arr[start]
        start += 1
        end -= 1


arr = [1, 2, 3, 4, 5]
reverse_array(arr)
print(arr)
```

```
[5, 4, 3, 2, 1]
```

In [4]:
```python
#Q3. Write a program to check if two strings are a rotation of each other?


def are_rotations(string1, string2):
    if len(string1) != len(string2):
        return False

    concatenated = string1 + string1
    if string2 in concatenated:
        return True
    else:
        return False

# Example usage:
str1 = "abcd"
str2 = "cdab"
result = are_rotations(str1, str2)
print(result)
```

```
True
```

In [6]:
```python
#Q4. Write a program to print the first non-repeated character from a string?



def find_first_non_repeated_char(string):
    char_count = {}

    # Count the occurrence of each character
    for char in string:
        char_count[char] = char_count.get(char, 0) + 1

    # Find the first non-repeated character
    for char in string:
        if char_count[char] == 1:
            return char

    # If no non-repeated character found, return None
    return None

# Example usage:
input_string = "Kavi Yarasan"
result = find_first_non_repeated_char(input_string)
print(result)
```

```
K
```

In [7]:
```python
#Q5. Read about the Tower of Hanoi algorithm. Write a program to implement it.




def tower_of_hanoi(n, source, destination, auxiliary):
    if n == 1:
        print(f"Move disk 1 from {source} to {destination}")
        return

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


# Example usage:
n = 3
tower_of_hanoi(n, 'A', 'C', 'B')
```

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

In [8]:
```python
#Q6. Read about infix, prefix, and postfix expressions. Write a program to convert postf



def postfix_to_prefix(expression):
    stack = []
    operators = set(['+', '-', '*', '/', '^'])

    for char in expression:
        if char not in operators:
```

```
            stack.append(char)
        else:
            operand2 = stack.pop()
            operand1 = stack.pop()
            prefix_expression = char + operand1 + operand2
            stack.append(prefix_expression)

    return stack.pop()


# Example usage:
postfix_expression = "23*4+"
prefix_expression = postfix_to_prefix(postfix_expression)
print("Prefix Expression:", prefix_expression)
```

```
Prefix Expression: +*234
```

In [9]:

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



def prefix_to_infix(expression):
    stack = []
    operators = set(['+', '-', '*', '/', '^'])

    for char in reversed(expression):
        if char not in operators:
            stack.append(char)
        else:
            operand1 = stack.pop()
            operand2 = stack.pop()
            infix_expression = '(' + operand1 + char + operand2 + ')'
            stack.append(infix_expression)

    return stack.pop()


# Example usage:
prefix_expression = "+*23*456"
infix_expression = prefix_to_infix(prefix_expression)
print("Infix Expression:", infix_expression)
```

```
Infix Expression: ((2*3)+(4*5))
```

In [11]:

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


def check_brackets(code):
    stack = []
    opening_brackets = set(['(', '[', '{'])
    closing_brackets = set([')', ']', '}'])
    bracket_pairs = {')': '(', ']': '[', '}': '{'}

    for char in code:
        if char in opening_brackets:
            stack.append(char)
        elif char in closing_brackets:
            if len(stack) == 0 or stack[-1] != bracket_pairs[char]:
                return False
            stack.pop()

    return len(stack) == 0
```

```
# Example usage:
code_snippet = "{ (a + b) * [c - d] }"
brackets_closed = check_brackets(code_snippet)
print("Brackets Closed:", brackets_closed)
```

```
Brackets Closed: True
```

In [13]:

```
#Q9. Write a program to reverse a stack.



def reverse_stack(stack):
    if not stack:
        return

    bottom = pop_bottom(stack)
    reverse_stack(stack)
    stack.append(bottom)


def pop_bottom(stack):
    item = stack.pop()
    if not stack:
        return item
    else:
        bottom = pop_bottom(stack)
        stack.append(item)
        return bottom


# Example usage:
stack = [1, 2, 3, 4, 5]
print("Original Stack:", stack)

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

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

In [14]:

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


def find_smallest_number(stack):
    if not stack:
        return None

    minimum = stack[-1]  # Assume the topmost element as the minimum

    for num in stack:
        if num < minimum:
            minimum = num

    return minimum


# Example usage:
stack = [5, 3, 9, 1, 7]
smallest_number = find_smallest_number(stack)
print("Smallest Number:", smallest_number)
```

```
Smallest Number: 1
```

In [ ]: