In [1]:
```
#Q1. Write a program to find all pairs of an integer array whose sum is equal to a
def find_pairs(arr, target):
    # Create an empty dictionary to store pairs
    pairs = {}

    # Iterate through the array
    for i in range(len(arr)):
        # Check if the complement of the current element exists in the dictionary
        complement = target - arr[i]
        if complement in pairs:
            # If the complement exists, print the pair
            print(arr[i], complement)
        # Add the current element to the dictionary
        pairs[arr[i]] = i

# Example usage
arr = [1, 2, 3, 4, 5, 6, 7, 8, 9]
target = 10
find_pairs(arr, target)
```

```
6 4
7 3
8 2
9 1
```

In [2]:
```
#Q2. Write a program to reverse an array in place? In place means you cannot create
def reverse_array(arr):
    # Get the length of the array
    n = len(arr)

    # Iterate through half of the array and swap elements
    for i in range(n//2):
        arr[i], arr[n-1-i] = arr[n-1-i], arr[i]

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

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

In [3]:
```
#Q3. Write a program to check if two strings are a rotation of each other?
def are_rotations(str1, str2):
    # Check if the length of the strings are the same
    if len(str1) != len(str2):
        return False

    # Concatenate the first string to itself
    temp = str1 + str1

    # Check if the second string is a substring of the concatenated string
    if str2 in temp:
        return True
    else:
        return False

# Example usage
str1 = "abcde"
str2 = "cdeab"
if are_rotations(str1, str2):
    print("The strings are rotations of each other")
else:
    print("The strings are not rotations of each other")
```

```
The strings are rotations of each other
```

In [4]:
```python
#Q4. Write a program to print the first non-repeated character from a string?
def first_non_repeated_char(string):
    # Create a dictionary to store the count of each character in the string
    char_count = {}
    for char in string:
        if char in char_count:
            char_count[char] += 1
        else:
            char_count[char] = 1

    # Iterate through the string again and return the first non-repeated character
    for char in string:
        if char_count[char] == 1:
            return char

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

# Example usage
string = "hello world"
result = first_non_repeated_char(string)
if result:
    print("The first non-repeated character in the string is", result)
else:
    print("No non-repeated character found in the string")
```

```
The first non-repeated character in the string is h
```

In [5]:
```python
#Q5. Read about the Tower of Hanoi algorithm. Write a program to implement it.
def tower_of_hanoi(n, source, auxiliary, destination):
    if n == 1:
        print(f"Move disk 1 from {source} to {destination}")
        return
    tower_of_hanoi(n-1, source, destination, auxiliary)
    print(f"Move disk {n} from {source} to {destination}")
    tower_of_hanoi(n-1, auxiliary, source, destination)

n = 3
tower_of_hanoi(n, 'A', 'B', 'C')
```

```
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 [6]:
```python
#Q6. Read about infix, prefix, and postfix expressions. Write a program to convert
def postfix_to_prefix(postfix):
    stack = []
    operators = set(['+', '-', '*', '/', '^'])
    for char in postfix:
        if char not in operators:
            stack.append(char)
        else:
            op1 = stack.pop()
            op2 = stack.pop()
            stack.append(char + op2 + op1)
    return stack.pop()

postfix = "ab+c*d/"
```

```
prefix = postfix_to_prefix(postfix)
print("Prefix expression:", prefix)
```

```
Prefix expression: /*+abcd
```

In [7]:
```
#Q7. Write a program to convert prefix expression to infix expression.
def prefix_to_infix(prefix):
    stack = []
    operators = set(['+', '-', '*', '/', '^'])
    for char in reversed(prefix):
        if char not in operators:
            stack.append(char)
        else:
            op1 = stack.pop()
            op2 = stack.pop()
            exp = '(' + op1 + char + op2 + ')'
            stack.append(exp)
    return stack.pop()

prefix = "*+ab-cd"
infix = prefix_to_infix(prefix)
print("Infix expression:", infix)
```

```
Infix expression: ((a+b)*(c-d))
```

In [8]:
```
#Q8. Write a program to check if all the brackets are closed in a given code snippe
def check_brackets(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] != '(') or (char == ']' and stack[-1] !=
                return False
            stack.pop()
    return len(stack) == 0

code = "{[(a+b)*c]-{d/e}}"
if check_brackets(code):
    print("All brackets are closed properly")
else:
    print("Brackets are not closed properly")
```

```
All brackets are closed properly
```

In [9]:
```
#Q9. Write a program to reverse a stack.
def reverse_stack(stack):
    temp_stack = []
    while stack:
        temp_stack.append(stack.pop())
    while temp_stack:
        stack.append(temp_stack.pop())
    return stack

stack = [1, 2, 3, 4, 5]
print("Original stack:", stack)
reversed_stack = reverse_stack(stack)
print("Reversed stack:", reversed_stack)
```

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

In [10]:
```
#Q10. Write a program to find the smallest number using a stack.
def find_smallest_number(stack):
    if not stack:
        return None
    smallest = stack.pop()
    while stack:
        current = stack.pop()
        if current < smallest:
            smallest = current
    return smallest

stack = [5, 3, 8, 1, 6, 2]
print("Stack:", stack)
smallest_number = find_smallest_number(stack)
print("Smallest number in stack:", smallest_number)
```

```
Stack: [5, 3, 8, 1, 6, 2]
Smallest number in stack: 1
```

In [ ]:

In [ ]: