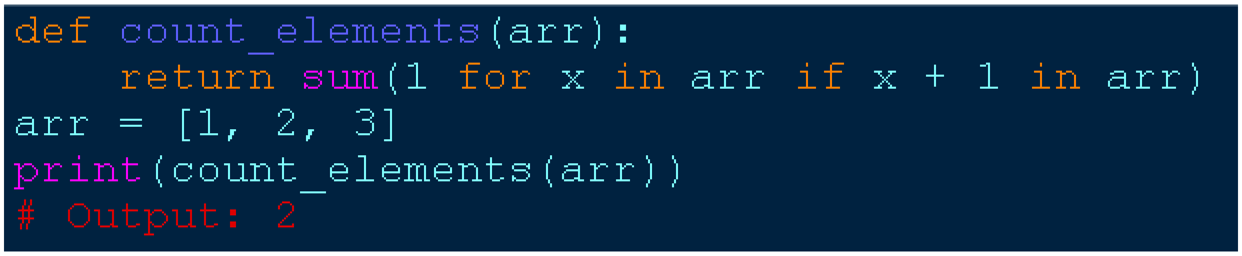
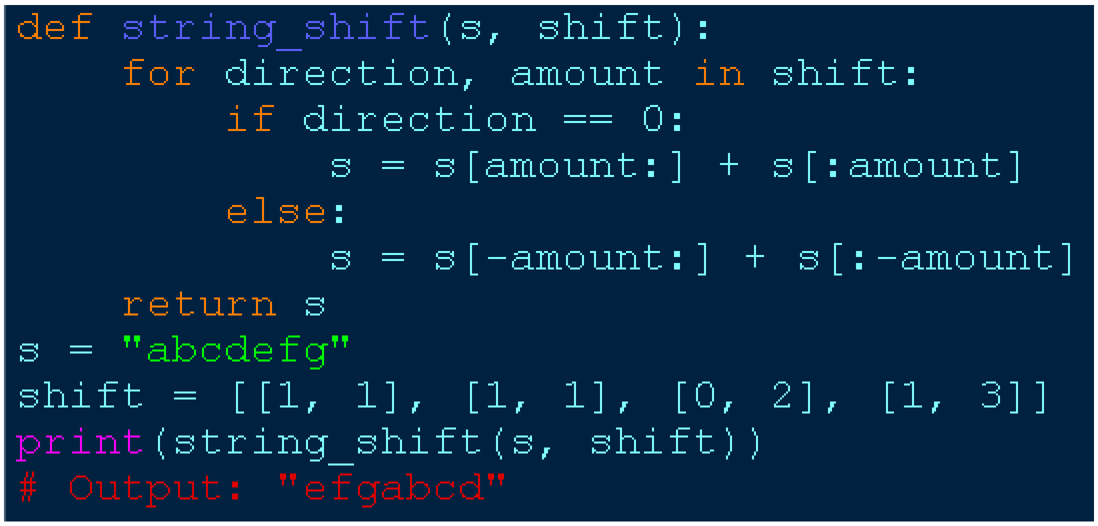
CSA0674 - DAA ASSIGNMENT 3

1) COUNT ELEMENTS :



```
def count_elements(arr):
    return sum(1 for x in arr if x + 1 in arr)
arr = [1, 2, 3]
print(count_elements(arr))
# Output: 2
```

2) PERFORM STRING SHIFTS :



```
def string_shift(s, shift):
    for direction, amount in shift:
        if direction == 0:
            s = s[amount:] + s[:amount]
        else:
            s = s[-amount:] + s[:-amount]
    return s
s = "abcdefg"
shift = [[1, 1], [1, 1], [0, 2], [1, 3]]
print(string_shift(s, shift))
# Output: "efgabcd"
```

3) LEFTMOST COLUMN WITH AT LEAST A ONE :

```python
class BinaryMatrix:
    def __init__(self, matrix):
        self.matrix = matrix
    def get(self, x, y):
        return self.matrix[x][y]
    def dimensions(self):
        return len(self.matrix), len(self.matrix[0])
def leftmost_column_with_one(binaryMatrix):
    rows, cols = binaryMatrix.dimensions()
    col = cols - 1
    for row in range(rows):
        while col >= 0 and binaryMatrix.get(row, col) == 1:
            col -= 1
    return col + 1 if col != cols - 1 else -1
matrix = BinaryMatrix([[0, 0], [1, 1]])
print(leftmost_column_with_one(matrix))
# Output: 0
```

4) FIRST UNIQUE NUMBER :

```python
from collections import Counter, deque
class FirstUnique:
    def __init__(self, nums):
        self.queue = deque(nums)
        self.count = Counter(nums)
    def showFirstUnique(self):
        while self.queue and self.count[self.queue[0]] > 1:
            self.queue.popleft()
        return self.queue[0] if self.queue else -1
    def add(self, value):
        self.queue.append(value)
        self.count[value] += 1
nums = [2, 3, 5]
firstUnique = FirstUnique(nums)
print(firstUnique.showFirstUnique())  # Output: 2
firstUnique.add(5)
print(firstUnique.showFirstUnique())  # Output: 2
firstUnique.add(2)
print(firstUnique.showFirstUnique())  # Output: 3
firstUnique.add(3)
print(firstUnique.showFirstUnique())  # Output: -1
```

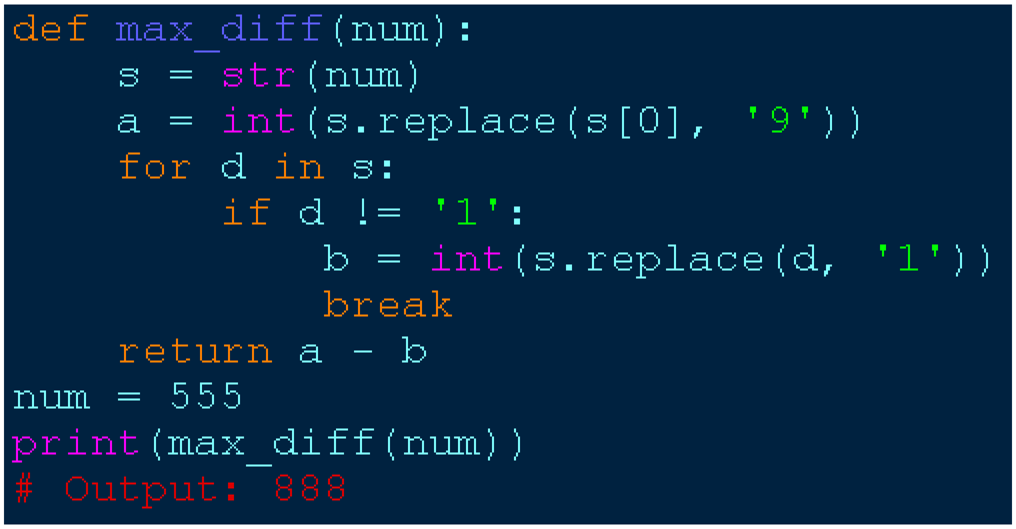
5) CHECK IF A STRING IS VALID FROM ROOT :

```python
class TreeNode:
    def __init__(self, x):
        self.val = x
        self.left = None
        self.right = None
def is_valid_sequence(root, arr):
    def dfs(node, idx):
        if not node or idx >= len(arr) or node.val != arr[idx]:
            return False
        if not node.left and not node.right and idx == len(arr) - 1:
            return True
        return dfs(node.left, idx + 1) or dfs(node.right, idx + 1)
    return dfs(root, 0)
root = TreeNode(0)
root.left = TreeNode(1)
root.right = TreeNode(0)
root.left.left = TreeNode(0)
root.left.right = TreeNode(1)
root.right.left = TreeNode(0)
root.left.left.right = TreeNode(1)
root.left.right.left = TreeNode(0)
root.left.right.right = TreeNode(0)
arr = [0, 1, 0, 1]
print(is_valid_sequence(root, arr))
# Output: True
```

6) KIDS WITH THE GREATEST NUMBER OF CANDIES :

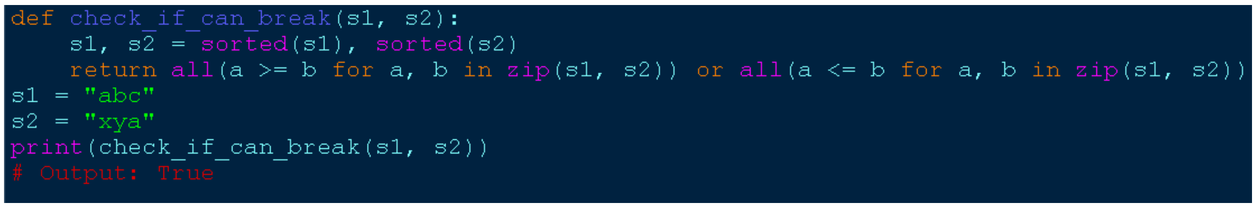
```python
def kids_with_candies(candies, extraCandies):
    max_candies = max(candies)
    return [candy + extraCandies >= max_candies for candy in candies]
candies = [2, 3, 5, 1, 3]
extraCandies = 3
print(kids_with_candies(candies, extraCandies))
# Output: [True, True, True, False, True]
```

7) MAX DIFFERENCE YOU CAN GET FROM CHANGING AN INTEGER :



```python
def max_diff(num):
    s = str(num)
    a = int(s.replace(s[0], '9'))
    for d in s:
        if d != '1':
            b = int(s.replace(d, '1'))
            break
    return a - b
num = 555
print(max_diff(num))
# Output: 888
```

8) CHECK IF A STRING CAN BREAK ANOTHER STRING :



```python
def check_if_can_break(s1, s2):
    s1, s2 = sorted(s1), sorted(s2)
    return all(a >= b for a, b in zip(s1, s2)) or all(a <= b for a, b in zip(s1, s2))
s1 = "abc"
s2 = "xya"
print(check_if_can_break(s1, s2))
# Output: True
```

9) NUMBER OF WAYS TO WEAR DIFFERENT HATS :

```python
def number_of_ways(hats):
    from functools import lru_cache
    n = len(hats)
    all_hats = list(range(1, 41))
    hat_to_people = {i: [] for i in all_hats}
    for person, hat_list in enumerate(hats):
        for hat in hat_list:
            hat_to_people[hat].append(person)
    @lru_cache(None)
    def dp(i, mask):
        if mask == (1 << n) - 1:
            return 1
        if i == 41:
            return 0
        res = dp(i + 1, mask)
        for person in hat_to_people[i]:
            if mask & (1 << person) == 0:
                res += dp(i + 1, mask | (1 << person))
        return res % (10**9 + 7)
    return dp(1, 0)
hats = [[3, 4], [4, 5], [5]]
print(number_of_ways(hats))
# Output: 1
```

10) DESTINATION CITY :

```python
def next_permutation(nums):
    i = len(nums) - 2
    while i >= 0 and nums[i] >= nums[i + 1]:
        i -= 1
    if i >= 0:
        j = len(nums) - 1
        while nums[j] <= nums[i]:
            j -= 1
        nums[i], nums[j] = nums[j], nums[i]
    nums[i + 1:] = reversed(nums[i + 1:])
    return nums
nums = [1, 2, 3]
print(next_permutation(nums))
# Output: [1, 3, 2]
```