1. Given a list like [1, [2, [3, [4, []]], 5], write a function deepest_nesting(lst) that returns the maximum depth of nesting (e.g., 4 for this case). Do not use recursion.

Ans:-

```
def deepest_nesting(lst):
    if not isinstance(lst, list):
        return 0
    if not lst:
        return 1
    return 1 + max(deepest_nesting(item) for item in lst)
print(deepest_nesting([1, [2, [3, [4]], 5]]))
```

2. Split a list into k sublists such that:

No sublist is empty.

The difference between the maximum sum and minimum sum of sublists is minimized.

Ans:-

```
from itertools import combinations

def split_min_diff(lst):
    n = len(lst)
    if n < 2:
        return lst, [ ]

    best_diff = float('inf')
    best_pair = ([ ], [ ])

    for i in range(1, n // 2 + 1):
        for a in combinations(lst, i):
            a = list(a)
            b = lst[:]
            for x in a:
                b.remove(x)
            diff = abs(sum(a) - sum(b))
            if diff < best_diff:
                best_diff = diff
                best_pair = (a, b)
                if best_diff == 0:
                    return best_pair

    return best_pair

lst = [7, 3, 2, 5, 8, 1]
left, right = split_min_diff(lst)
```

```
print("Original list:", lst)
print("Sublist 1:", left, "Sum:", sum(left))
print("Sublist 2:", right, "Sum:", sum(right))
print("Sum difference:", abs(sum(left) - sum(right)))
```

3. Given lst = [('a', 1), ('b', 2), ('a', 3)], convert it into {'a': [1, 3], 'b': [2]} without using defaultdict or setdefault.

Ans:-

```
lst = [('a', 1), ('b', 2), ('a', 3)]
d = {}
for k, v in lst:
    d[k] = d.get(k, []) + [v]
print(d)
```

4. Write a function shuffle_restricted(lst) that shuffles a list without using random.shuffle and ensuring no element appears in its original position.

Ans:-

```
import random

def deranged_shuffle(lst):
    while True:
        shuffled = lst[:]
        random.SystemRandom().shuffle(shuffled)
        if all(a != b for a, b in zip(lst, shuffled)):
            return shuffled
original = [1, 2, 3, 4, 5]
shuffled = deranged_shuffle(original)

print("Original:", original)
print("Shuffled (deranged):", shuffled)
```

5. Encode a string such that consecutive runs of 3+ characters are compressed (e.g., "aaabbcccc" → "a3bbc4").

Ans:-

```
def compress(text):
    result = ''
    i = 0
    while i < len(text):
        count = 1
        while i + 1 < len(text) and text[i] == text[i + 1]:
            count += 1
            i += 1
        if count >= 3:
            result += text[i] + str(count)
        else:
            result += text[i] * count
        i += 1
    return result

s = "aaabbcccddddeeeeffffg"
compressed = compress(s)
print("Original:", s)
print("Compressed:", compressed)
```

6. Return the intersection of two lists without duplicates.
(Take any two list of your choice).

Ans:-

```
def intersection(a, b):
    return list(set(a) & set(b))

list1 = [1, 2, 2, 3, 4]
list2 = [2, 3, 5, 2]
result = intersection(list1, list2)

print("List 1:", list1)
print("List 2:", list2)
print("Intersection (no duplicates):", result)
```

7. Given a list like ["abc", "def", ["ghi", "jkl"]], flatten it into a single list of characters.

Ans:-

```
def flatten_chars(lst):
    result = []
    for item in lst:
        if isinstance(item, list):
            result.extend(item)
        else:
            result.append(item)
    return result

char_list = ['a', ['b', 'c'], 'd', ['e'], 'f']
flattened = flatten_chars(char_list)
print("Original:", char_list)
print("Flattened:", flattened)
```

8. Convert a list of words into an acronym (first letters capitalized).
Example: ["Federal", "Bureau", "Investigation"] → "FBI"

Ans:-

```
def acronym(words):
    return ''.join(word[0].upper() for word in words)
words = ["national", "aeronautics", "space", "administration"]
acro = acronym(words)
print("Words:", words)
print("Acronym:", acro)
```

9. Determine if two strings are anagrams without sorting.(Take two strings of your choice)

Ans:-

```
def is_anagram(s1, s2):
    if len(s1) != len(s2):
        return False
    count = {}
    for ch in s1:
        count[ch] = count.get(ch, 0) + 1
    for ch in s2:
        if ch not in count or count[ch] == 0:
            return False
        count[ch] -= 1
    return True

print(is_anagram("listen", "silent"))
print(is_anagram("hello", "world"))
```

10. Given a list of n-1 integers from 1 to n, write the code to find the missing number.
Example: [1, 2, 4, 5] → 3

Ans:-

```
def find_missing(lst):
    n = len(lst) + 1
    total = n * (n + 1) // 2
    return total - sum(lst)
nums = [1, 2, 4, 5]
print("Missing number:", find_missing(nums))
```