1.Exploring the complex data type and their operations, eg: finding the modulus and phase angle of a complex number and print values. [20.09.23]

2.Ask the user to enter two numbers, and output the sum, product, difference, and the GCD. [22.09.23]

3.Ask the user for two strings, print a new string where the first string is reversed, and the second string is converted to upper case. Sample strings: “Pets“, “party”, output: “steP PARTY”. Only use string slicing and + operators. [29.09.23]

4 . From a list of words, join all the words in the odd and even indices to form two strings. Use list slicing and join methods. .[04.10.23]

5. Simulate a stack and a queue using lists. Note that the queue deletion operation won’t run in O(1) time. [11.10.23]

6.Explore the ‘re’ module, especially re.split, re.join, re.search and re.match methods. . [13.10.23]

6. Use list comprehension to find all the odd numbers and numbers divisible by 3 from a list of numbers.[22.11.23]

7. Using while loops to do Gaussian addition on a list having an even number of numbers. Print each partial sum. Eg: if the list is [1, 2, 3, 4, 5, 6], the program should output “1 + 6”, “2 + 5”, and “3+4” in separate lines, and the result of the addition “21”. Extend it to handle lists of odd length. . [29..11.23]

8. Implement popular sorting algorithms like quick sort and merge sort to sort lists of numbers. . [01.12.23]

9Write two functions that simulate the toss of a fair coin, and the roll of an unbiased ‘n’ sided die using the random module. [06.12.23]

10.Invert a dictionary such the previous keys become values and values keys. Eg: if the initial and inverted dictionaries are d1 and d2, where d1 = {1: ‘a’, 2: ‘b’, 3: 120}, then d2 = {‘a’: 1, 2: ‘b’, 120: 3}. [13.12.23]

11. Create a ‘Graph’ class to store and manipulate graphs. It should have the following functions:
i. Read an edge list file, where each edge (u, v) appears exactly once in the file as space separated values.
ii. Add and remove nodes and edges
iii. Print nodes, and edges in a user readable format
Iv. Finding all the neighbors of a node
v. Finding all the connected components and storing them as individual Graph objects inside the class
vi. Finding single source shortest paths using Breadth First Search . [20.12.23]

12. Make a ‘DiGraph’ class to handle directed graphs which inherits from the ‘Graph’ class. In addition to all of the functionalities of (a), it should support the following operations
i. Finding the predecessors and successors of a node
ii. Creating a new ‘DiGraph’ object where all the edges are reversed
.iii. Finding the strongly connected components . [22.12.23]