**MATPLOTLIB ACTIVITES IN PYTHON**

**1. Basic Plotting**

* **Activity:** Plot the function y=x2y = x^2y=x2 for xxx values ranging from -10 to 10.

**2. Bar Chart**

* **Activity:** Plot the number of students in different grades.

**3. Histogram**

* **Activity:** Plot the distribution of heights of a group of people.

**4. Scatter Plot**

* **Activity:** Plot a scatter plot of random data points to visualize the relationship between two variables.

**5. Pie Chart**

* **Activity:** Create a pie chart to show the market share of different smartphone brands.

**6. Box Plot**

* **Activity:** Create a box plot to show the distribution of test scores in a class.

**7. Heatmap**

* **Activity:** Generate a heatmap to visualize the correlation between different features in a dataset.

**8. Subplots**

* **Activity:** Create multiple subplots in a single figure to compare different types of data visualizations.

**9. 3D Plot**

* **Activity:** Plot a 3D surface plot of the function z=sin⁡(x)⋅cos⁡(y)z = \sin(x) \cdot \cos(y)z=sin(x)⋅cos(y).

**10. Animation**

* **Activity:** Create an animated plot to visualize changes in data over time, such as a sine wave animation.

**11. Stacked Bar Chart**

* **Activity:** Create a stacked bar chart to show the breakdown of sales by region and product type.

**12. Bubble Chart**

* **Activity:** Create a bubble chart to visualize data points with three variables, such as GDP, population, and area of countries.

**13. Polar Plot**

* **Activity:** Create a polar plot to visualize data in a circular format, such as wind directions and speeds.

**14. Contour Plot**

* **Activity:** Create a contour plot to visualize the contour lines of a 3D surface on a 2D plane.

**15. Error Bars**

* **Activity:** Plot a line graph with error bars to show the uncertainty in data measurements.

**DSA IN PYTHON**

**1. Linked List Implementation**

Implement a singly linked list and perform basic operations such as insertion, deletion, and traversal.

**Features:**

* Insert elements at the beginning, end, or at a given position.
* Delete elements by value or position.
* Traverse and display elements in the list.

**2. Stack Implementation**

Create a stack using a list and implement standard stack operations.

**Features:**

* Push elements onto the stack.
* Pop elements from the stack.
* Peek at the top element.
* Check if the stack is empty.

**3. Queue Implementation**

Implement a queue using a list and perform basic queue operations.

**Features:**

* Enqueue elements to the rear.
* Dequeue elements from the front.
* Peek at the front element.
* Check if the queue is empty.

**4. Binary Search Tree (BST)**

Implement a binary search tree with basic operations such as insertion, deletion, and traversal.

**Features:**

* Insert elements into the BST.
* Delete elements from the BST.
* Perform in-order, pre-order, and post-order traversal.
* Search for an element in the BST.

**5. Graph Representation and Traversal**

Implement a graph using an adjacency list and perform graph traversal algorithms.

**Features:**

* Add and remove vertices and edges.
* Perform Depth-First Search (DFS).
* Perform Breadth-First Search (BFS).

**6. Sorting Algorithms**

Implement and compare different sorting algorithms.

**Features:**

* Implement Bubble Sort, Insertion Sort, Selection Sort, Merge Sort, and Quick Sort.
* Compare their time complexity and performance on different datasets.

**7. Hash Table**

Create a hash table and handle collisions using chaining or open addressing.

**Features:**

* Insert key-value pairs.
* Delete key-value pairs.
* Search for a value by key.
* Handle collisions.

**8. Priority Queue**

Implement a priority queue using a heap.

**Features:**

* Insert elements with a priority.
* Extract the element with the highest priority.
* Peek at the element with the highest priority.
* Check if the priority queue is empty.

**9. Dijkstra's Algorithm**

Implement Dijkstra's algorithm to find the shortest path in a weighted graph.

**Features:**

* Represent the graph using an adjacency list.
* Implement the algorithm to find the shortest path from a source vertex to all other vertices.
* Display the shortest path and distances.

**10. Knapsack Problem**

Solve the 0/1 Knapsack problem using dynamic programming.

**Features:**

* Implement the dynamic programming solution to the knapsack problem.
* Display the maximum value that can be obtained for a given capacity.
* Display the items included in the optimal solution.

**11. Trie Implementation**

Implement a Trie (prefix tree) for efficient string search operations.

**Features:**

* Insert words into the Trie.
* Search for words in the Trie.
* Implement auto-complete suggestions based on a prefix.

**12. LRU Cache**

Implement a Least Recently Used (LRU) cache using a combination of a hash table and a doubly linked list.

**Features:**

* Add key-value pairs to the cache.
* Retrieve values by key.
* Ensure the cache does not exceed a specified capacity.
* Evict the least recently used items when the capacity is exceeded.

**13. Merging Sorted Linked Lists**

Merge two sorted linked lists into one sorted linked list.

**Features:**

* Implement the merge function.
* Display the merged linked list.

**14. Cycle Detection in a Linked List**

Detect if a linked list contains a cycle using Floyd's Tortoise and Hare algorithm.

**Features:**

* Implement the cycle detection function.
* Return the starting node of the cycle if a cycle is detected.

**15. Graph Cycle Detection**

Detect cycles in a directed graph using Depth-First Search (DFS).

**Features:**

* Represent the graph using an adjacency list.
* Implement the cycle detection algorithm.
* Display if a cycle is detected.