NSHM INSTITUTE OF ENGINEERING & TECHNOLOGY

LABORATORY MANUAL

CSE 3rd semester

Subject Name: IT Workshop

Subject Code: PCC CS-393

Prepared by:	**Checked by:**	**Approved by:**
Mr.Saurabh Banerjee	Mr. Arup Dey	Dr. Bijoy Kumar Mandal Principal, NIET
Sign.: …………………….	**Sign.: ………………….**	**Sign.: ………………….**

DEPARTMENT OF CSE
NSHM KNOWLEDGE CAMPUS, DURGAPUR
SHIBTALA RD, MUCHIPARA, ARRAH KALINAGAR, ARRA, DURGAPUR, WEST BENGAL 713212

TABLE OF CONTENTS

Vision and Mission of the Institute

Vision:

"To be a knowledge hub of global excellence"

Mission:

M1: Bringing prosperity to the society.

M2: Enhancing quality of life by imparting and advancing knowledge and skills, unleashing creative abilities and inculcating responsible and responsive values and attitudes.

Vision and Mission of the Department

Vision:

To achieve academic excellence and to build the leaders with escalating multi – skilled professionals in Computer Science & Engineering with global competence enabled by technical expertise with in depth Knowledge, innovative research and the knowledge of leading edge technologies, constant learning, promoting employability, higher education with socio-ethical, eco-friendly and entrepreneurial values for being a responsible citizen with a positive attitude.

Mission:

M1: To give quality technical education and to develop the learners as leaders in Computer Science & Engineering with fundamental engineering principles with intellectually adept and professionally expert, innovative research capabilities.

M2: To intensify exemplary skills and competence in scholars to transmit them to lead and making them to use technology for the progress of mankind, training and adopting mutating technological environment.

M3: To be a world class pioneer by adopting them to changing technological environment by providing the high quality instruction, infra, faculty, training, modern teaching and learning methods acquiring the socio-ethical and entrepreneurial values as the inner potential.

Programme Educational Objectives (PEOs)

PEO1:

The scholars of computer Science and Engineering can be able to apply the knowledge of Mathematics, Applied Science, Computing, Basic Engineering field to identify, analyze, formulate, design, simulate and develop the practical solutions for hardware and software problems in industry and academia.

PEO2:

To enable the scholars with core curriculum knowledge in theory and practical experiments of computer Science and Engineering to develop the innovative skills in design, simulation, investigation of complex problems, critical reasoning, development and testing knowledge for offering solutions to real life problems related to globally evolving techno corridor requirements of computer Science and Engineering Field.

PEO3:

To provide the scholars with extent knowledge to build the Computer Science and engineering professionals to have the team work and skills for developing communicative abilities, long-lasting learning, and proficiency of project management, finance with entrepreneurial values for rural development.

PEO4:

To give training using a system of multifaceted, multidisciplinary approach to develop R&D skills by MOUs with premier industries and institutions interacting with training sessions and industrial visits for the learners to have awareness on the latest trends, Modern software tools and programming techniques of Computer Science and Engineering to cater the escalating needs of society.

PEO5:

To build the learners with the aptitude of competitive knowledge of real time requirement of cutting edge technologies by promoting employability and higher education with a blend of ethical, social and eco-friendly.

PROGRAM OUTCOMES (POs)

PO1: Engineering knowledge: Apply the knowledge of mathematics, science, engineering fundamentals, and an engineering specialization to the solution of complex engineering problems.

PO2: Problem analysis: Identify, formulate, review research literature, and analyze complex engineering problems reaching substantiated conclusions using first principles of mathematics, natural sciences, and engineering sciences.

PO3: Design/development of solutions: Design solutions for complex engineering problems and design system components or processes that meet the specified needs with appropriate consideration for the public health and safety, and the cultural, societal, and environmental considerations.

PO4: Conduct investigations of complex problems: Use research-based knowledge and research methods including design of experiments, analysis and interpretation of data, and synthesis of the information to provide valid conclusions.

PO5: Modern tool usage: Create, select, and apply appropriate techniques, resources, and modern engineering and IT tools including prediction and modeling to complex engineering activities with an understanding of the limitations.

PO6: The engineer and society: Apply reasoning informed by the contextual knowledge to assess societal, health, safety, legal and cultural issues and the consequent responsibilities relevant to the professional engineering practice.

PO7: Environment and sustainability: Understand the impact of the professional engineering solutions in societal and environmental contexts, and demonstrate the knowledge of, and need for sustainable development.

PO8: Ethics: Apply ethical principles and commit to professional ethics and responsibilities and norms of the engineering practice.

PO9: Individual and teamwork: Function effectively as an individual, and as a member or leader in diverse teams, and in multidisciplinary settings.

PO10: Communication: Communicate effectively on complex engineering activities with the engineering community and with society at large, such as, being able to comprehend and write effective reports and design documentation, make effective presentations, and give and receive clear instructions.

P11: Project management and finance: Demonstrate knowledge and understanding of the engineering and management principles and apply these to one‟s own work, as a member and leader in a team, to manage projects and in multidisciplinary environments.

P12: Life-long learning: Recognize the need for and have the preparation and ability to engage in independent and life-long learning in the broadest context of technological change.

PROGRAM SPECIFIC OUTCOMES (PSOs)

PSO1: The Computer Science and Engineering graduates are able to analyze, design, develop, test and apply skills on the basis of mathematical and programming foundations in the development of computational solutions to design software and hardware.

PSO2: Work with and communicate effectively with professionals in inter-disciplinary fields and pursue lifelong professional development in computing and identify research gaps and hence to provide solutions to new ideas and innovations to satisfy the environmental and social issues.

University Syllabus

Course code	PCC AI-492				
Category	Practical				
Course title	IT Workshop				
Scheme and Credits	L	T	P	Credits	Semester : 3
	-	-	2	2	
Classwork	-				
Exam	100				
Total	100				
Duration of Exam	3 Hours				

Course Outcomes (COs)

Upon successful completion of the course, the students will be able to:

CO1: Interpret the basic syntax of python variables,datatypes and operator in python and

CO2: Make use of conditional and control flow statements in python fluently.

CO3: Define the use of String and list datatype in proficiency level.

CO4: Discover the method to create and manipulation of python data structure like tuple and dictionary.

CO5: Explain the use of python function and uses of different modules in python.

CO6: Discuss the concept of object oriented programming like exception handling.

CO-PO Mapping

CO	PO1	PO2	PO3	PO4	PO5	PO6	PO7	PO8	PO9	PO10	PO11	PO12
CO1	3	3	2	-	1	1	1	-	-	1	-	3
CO2	3	2	1	-	1	1	1	1	-	1	-	2
CO3	3	3	3	1	3	-	2	-	2	1	1	2
CO4	3	3	3	1	2	-	-	1	1	-	1	3
CO5	3	3	3	2	3	-	-	1	2	-	1	3
CO6	3	3	3	2	3	-	2	2	2	3	1	3
AVG	3	2.83	2.5	1.5	2.17	1	1.5	1.25	1.75	1.5	1	2.67

CO-PSO Mapping

CO	PSO1	PSO2
CO1	3	2
CO2	3	1
CO3	3	2
CO4	2	1
CO5	3	2
CO6	3	2

*3-HIGH
*2-MEDIUM
*1-LOW

Course Overview

Course: IT Workshop (Python)

Course Code: PCC-CS393

Semester: 3rd (Second Year)
Type: Practical / Lab course
Contact Hours: 4P per week

Credits: 3 credits

Pre-requisites: None (no prior requirement)

Course Objectives

The objectives of the IT Workshop course generally include:

1. To provide hands-on exposure to scientific / programming tools and environments
2. To enable students to solve engineering / scientific problems using computational tools and scripting.
3. To bridge theoretical knowledge and practical execution by applying programming in domains of data, simulation, or scientific computation. (Implied in lab nature)
4. To build familiarity with multiple tools/environments so students can choose appropriate ones for future courses/projects.

Topics / Modules (Typical Content)

- Introduction to the lab environments (Python)
- Basic programming constructs: variables, operators, control statements, functions, etc. (in Python)
- Scientific computing tasks: numerical methods, plotting, data analysis
- Using libraries or built-in functions of Python for matrix operations, statistics, etc.
- Mini-projects or lab assignments integrating the tools

Assessment & Evaluation

- Practical / Lab sessions form the core evaluation. makautexam.net+1
- End-semester lab examination / viva / project demonstration.
- Continuous assessment via assignments, lab submissions, quizzes.
- Marks distribution likely includes lab performance, submissions, and viva (though exact weight may vary by semester).

Complete Program List

Unit 1 — BASICS (30 Programs)

Program Title
Number Checks & Operations 1. Check Even / Odd 2. Swap Two Numbers (without temporary variable) 3. Simple Interest & Compound Interest calculation 4. Prime Check for a given number 5. List All Primes in a Range 6. Fibonacci Sequence (efficiently using fast doubling or iterative method) 7. Factorial (iterative, recursion, and memoized) 8. GCD & LCM of two numbers 9. Palindrome Check (Number & String) 10. Armstrong Numbers in a Range 11. Strong Number check 12. Perfect Number check 13. Decimal ↔ Binary ↔ Octal ↔ Hexadecimal conversion
Matrix Operations 14. Matrix Multiplication (without NumPy) 15. Transpose of Matrix
Patterns & Combinatorics 16. Pascal's Triangle (print n rows)
Number/String Properties 17. Combined Checks: Armstrong / Automorphic / Palindrome 18. Sum of Digits, Reverse, Digit Count
Bitwise Operations & Math

PATTERN PROGRAMMING QUESTIONS

PATTERN 1. Right Triangle Star

```
*
* *
* * *
* * * *
* * * * *
```

PATTERN 2.Inverted Right Triangle

```
* * * * *
* * * *
* * *
* *
*
```

PATTERN 3.Pyramid

```
    *
   * *
  * * *
 * * * *
* * * * *
```

PATTERN 4.Inverted Pyramid

```
* * * * *
 * * * *
  * * *
   * *
    *
```

PATTERN 5.Diamond Pattern

```
    *
   * *
  * * *
```

```
 * * * *
* * * * *
 * * * *
  * * *
   * *
    *
```

PATTERN 6.Square Pattern

```
* * * * *
* * * * *
* * * * *
* * * * *
* * * * *
```

PATTERN 7.Hollow Square

```
* * * * *
*       *
*       *
*       *
* * * * *
```

PATTERN 8.Number Triangle

```
1
1 2
1 2 3
1 2 3 4
```

PATTERN 9.Floyd's Triangle

```
1
2 3
4 5 6
7 8 9 10
```

PATTERN 10 — Inverted Number Triangle

```
1 2 3 4 5
1 2 3 4
1 2 3
1 2
1
```

PATTERN 11.Repeated Number Triangle

```
1
2 2
3 3 3
4 4 4 4
```

PATTERN 12 — Alphabet Triangle

```
A
A B
A B C
A B C D
```

PATTERN 13 — Inverted Alphabet Triangle

```
A B C D
A B C
A B
A
```

PATTERN 14 — Hollow Pyramid

```
    *
   * *
  *   *
 *     *
* * * * *
```

PATTERN 15 — Hollow Diamond

```
    *
   * *
  *   *
 *     *
*       *
 *     *
  *   *
   * *
    *
```

PATTERN 16 — Hourglass Pattern

```
* * * * *
 * * * *
  * * *
 * * * *
* * * * *
```

PATTERN 17 — Mirrored Right Triangle

```
        *
      * *
    * * *
  * * * *
* * * * *
```

PATTERN 18 — Number Pyramid

```
    1
   2 2
  3 3 3
 4 4 4 4
5 5 5 5 5
```

PATTERN 19 — Reverse Number Pyramid

```
5 5 5 5 5
 4 4 4 4
  3 3 3
   2 2
```

```
    1
```

PATTERN 20 — Palindrome Number Pyramid

```
    1
   121
  12321
 1234321
123454321
```

PATTERN 21 — Continuous Number Pyramid

```
1
2 3
4 5 6
7 8 9 10
```

PATTERN 22 — Binary Triangle

```
1
0 1
1 0 1
0 1 0 1
```

PATTERN 23 — Sandglass Number Pattern

```
12345
 1234
  123
   12
    1
   12
  123
 1234
12345
```

PATTERN 24 — Cross "X" Pattern

```
*       *
 *     *
  *   *
   * *
    *
   * *
  *   *
 *     *
*       *
```

PATTERN 25-Butterfly Pattern

```
*        *
**       **
```

```
***    ***
****  ****
**********
****  ****
***    ***
**      **
*        *
```

UNIT-2 STRING

Beginner String Programs

1. Print length of a string
2. Reverse a string
3. Check if a string is palindrome
4. Count vowels and consonants in a string
5. Count uppercase, lowercase letters
6. Count digits and special characters
7. Convert string to uppercase and lowercase
8. Toggle case of each character
9. Check if a string is numeric
10. Remove whitespaces from string
11. Find first occurrence of a character
12. Find last occurrence of a character
13. Count occurrences of a character
14. Concatenate two strings
15. Compare two strings
16. Check if a substring exists
17. Replace a substring with another
18. Split string into words
19. Join list of strings into a single string
20. Check if string starts with a substring
21. Check if string ends with a substring
22. Remove punctuation from a string
23. Reverse words in a sentence
24. Check if string contains only alphabets
25. Find largest and smallest character in string

Intermediate String Programs

26. Remove duplicate characters
27. Count frequency of each character
28. Find all unique characters
29. Check if two strings are anagrams
30. Check if two strings are isomorphic
31. Find all substrings of a string
32. Count number of words in a string

Case-Based String Problems

1. Text Analysis Tool

- Input: Paragraph of text
- Tasks:
    - Count sentences, words, and characters
    - Find most frequent word
    - Count vowels, consonants, digits, and special characters
    - Identify longest and shortest word

2. Log File Analyzer

- Input: Server logs (multi-line string)
- Tasks:
    - Extract all IP addresses
    - Count occurrences of each HTTP status code
    - Find top 5 requested URLs
    - Detect error messages and line numbers

3. Password Strength Checker

- Input: Password string
- Tasks:
    - Check length, uppercase, lowercase, digit, special character
    - Check against dictionary of common passwords
    - Provide a strength score and suggestion

4. Chat Message Formatter

- Input: Chat messages with timestamps
- Tasks:
    - Extract username, timestamp, and message
    - Convert timestamps to readable format
    - Remove unwanted emojis or special characters
    - Count number of messages per user

5. Spell Checker and Corrector

- Input: Paragraph of text
- Tasks:

- Identify misspelled words using a dictionary
- Suggest top 3 possible corrections for each misspelled word
- Output corrected paragraph

6. Email/URL Extractor

- Input: Text containing multiple emails and URLs
- Tasks:
  - Extract all email addresses
  - Extract all URLs
  - Count unique domains
  - Sort by frequency

7. Substring Search & Highlight

- Input: Large text and search keywords
- Tasks:
  - Find all occurrences of the keyword
  - Highlight or mark them in the text
  - Count frequency of each keyword

8. Text Compression & Decompression Tool

- Input: Large text
- Tasks:
  - Implement run-length encoding
  - Implement simple Huffman coding or dictionary-based compression
  - Decompress text and validate integrity

9. CSV/Text Table Parser

- Input: CSV formatted text or tabular data
- Tasks:
  - Parse rows and columns
  - Allow querying by column or row
  - Detect empty or malformed entries
  - Export modified table as string

10. String-Based Calculator

- Input: Expression string, e.g., `"12 + 34 * (5 - 2)"`
- Tasks:

  - Parse and evaluate arithmetic expression
  - Handle parentheses, operator precedence
  - Output result

Project-Style / Placement-Level String Challenges

11. DNA Sequence Analyzer

- Input: DNA string (A, C, G, T)
- Tasks:
  - Count nucleotides frequency
  - Find longest repeating sequence
  - Complement and reverse complement sequences
  - Search for specific motifs/patterns

12. Log Compression & Aggregation

- Input: Server log file string
- Tasks:
  - Group log messages by type (ERROR, INFO, WARNING)
  - Count frequency per hour
  - Compress repeated consecutive messages

13. Social Media Text Analytics

- Input: Collection of tweets or posts
- Tasks:
  - Extract hashtags and mentions
  - Count sentiment-positive and negative words
  - Identify top trending hashtags
  - Detect and remove spam content

14. Code Parser / Formatter

- Input: Source code string
- Tasks:
  - Remove comments and blank lines
  - Count functions, classes, variables
  - Detect naming conventions violations
  - Output formatted code

15. Chatbot Preprocessing Module

- Input: User input string
- Tasks:
    - Tokenize sentences and words
    - Normalize text (lowercase, remove punctuation)
    - Identify entities or keywords
    - Respond based on pre-defined patterns

16. Palindrome & Anagram Finder in Text

- Input: Large document string
- Tasks:
    - Find all palindromic words and phrases
    - Detect anagrams across the document
    - Count frequency of each unique palindrome/anagram

17. Text-Based Game Engine

- Input: Game commands as string
- Tasks:
    - Parse commands and arguments
    - Validate input and execute actions
    - Keep track of game state using strings
    - Output results and messages

18. Custom Markup Language Parser

- Input: Text in custom markup (like `[b]bold[/b]`)
- Tasks:
    - Parse opening and closing tags
    - Apply formatting rules
    - Validate nesting and generate output HTML or markdown

19. Multi-Language Text Translator (String Simulation)

- Input: Text in English
- Tasks:
    - Use dictionary to map English words → target language
    - Handle punctuation and capitalization
    - Detect unknown words and suggest fallback translation

20. Report Generator from Raw Text

- Input: Raw logs or CSV-like text
- Tasks:
  - Summarize data into sections
  - Count totals, averages
  - Output formatted summary as multi-line string

Tips for approaching project-based string problems:

1. Break the problem into **small tasks** (counting, splitting, searching).
2. Use **Python built-in string methods** and **collections module** for efficiency.
3. For complex tasks, use **functions** or **classes** to organize code.
4. Think of **edge cases**: empty strings, special characters, large input.

UNIT-3(LIST)

Basic List Programs _(List solution)

1. Create a list of 5 elements and print it.
2. Access the 3rd element of a list.
3. Add an element to the end of a list.
4. Insert an element at 2nd position.
5. Remove an element from a list.
6. Find the length of a list.
7. Print all elements using a loop.
8. Sort a list in ascending order.
9. Reverse a list.
10. Sum all elements of a list.

Slightly Advanced List Programs

11. Find the maximum element in a list.
12. Find the minimum element in a list.
13. Sum only even numbers in a list.
14. Count occurrences of an element.
15. Copy a list.
16. Merge two lists.
17. Get a sublist using slicing.
18. Reverse a list using slicing.
19. Remove duplicates from a list.
20. Flatten a nested list.

Problem-Solving List Programs

21. Find the second largest element in a list.
22. Check if a list is a palindrome.
23. Rotate a list to the right by 2 positions.
24. Find all indices of an element in a list.
25. Merge two lists and remove duplicates.
26. Find common elements between two lists.
27. Find elements greater than a given number.
28. Find the product of all elements in a list.
29. Check if a list contains a given element.
30. Find the average of elements in a list.

Challenging/Logic List Programs

31. Find duplicates in a list.
32. Count positive and negative numbers in a list.
33. Separate even and odd numbers into two lists.
34. Merge two sorted lists into a single sorted list.
35. Find the largest even number in a list.
36. Find the smallest odd number in a list.
37. Count elements greater than their previous element.
38. Find the second smallest element in a list.
39. Replace negative numbers with zero in a list.
40. Count frequency of all elements in a list.

Exam-Level Tricky List Programs

41. Move all zeros to the end of a list.
42. Find missing numbers from 1 to n in a list.
43. Cumulative sum of a list.
44. Reverse a list without using reverse() or slicing.
45. Rotate a list to the left by 3 positions.
46. Find the largest sum of consecutive elements of length 3.
47. Count elements that are greater than both neighbors.
48. Find all pairs with a given sum.
49. Flatten a deeply nested list.
50. Find the missing number in a consecutive sequence using formula.

Lists(ADVANCED)

Program Title

1. Bubble Sort – Implement and understand each pass.
2. Insertion Sort – Build a sorted list one element at a time.
3. Selection Sort – Select the minimum/maximum element each pass.
4. Merge Sort – Divide-and-conquer recursive sorting.
5. Quick Sort – Partition-based recursive sorting.
6. **Heap Sort** – Use heap properties to sort a list.
7. **Linear Search** – Search for an element sequentially.
8. **Binary Search (Iterative)** – Search in a sorted list.
9. **Binary Search (Recursive)** – Recursive implementation for sorted list.
10. **Reverse a List** – Without using built-in functions.
11. **Find Union of Two Lists** – Combine all unique elements.
12. **Find Intersection of Two Lists** – Elements common to both lists.
13. **Generate All Permutations of a List** – Using recursion/backtracking.
14. **Pascal's Triangle using Lists** – Store rows in lists for dynamic computation.

15. **Polynomial Addition using Lists** – Represent polynomials as lists and sum them.
16. **Polynomial Multiplication using Lists** – Multiply two polynomials represented as lists.
17. **Filter Prime Numbers from a List** – Generate a new list containing only primes.

Unit 4 — Tuples

Basic Tuple Programs

1. Create a tuple of 5 elements and print it.
2. Access the 3rd element of a tuple.
3. Count occurrences of an element in a tuple.
4. Find the index of an element in a tuple.
5. Slice a tuple to get a sub-tuple.
6. Concatenate two tuples.
7. Repeat a tuple 3 times.
8. Find the length of a tuple.
9. Check if an element exists in a tuple.
10. Convert a list to a tuple.

Advanced Tuple Programs

11. Find the maximum element in a tuple.
12. Find the minimum element in a tuple.
13. Sum all elements in a tuple.
14. Find the product of all elements in a tuple.
15. Reverse a tuple.
16. Count elements greater than a given number in a tuple.
17. Find all even numbers in a tuple.
18. Find all odd numbers in a tuple.
19. Merge two tuples and sort the result.
20. Remove duplicates from a tuple.

Problem-Solving Tuple Programs

21. Find the second largest element in a tuple.
22. Check if a tuple is a palindrome.
23. Find all indices of an element in a tuple.
24. Count elements greater than their previous element.
25. Find common elements between two tuples.
26. Find elements greater than a given number.
27. Find the sum of elements at even indices.
28. Find the product of elements at odd indices.
29. Check if all elements in a tuple are unique.
30. Find the tuple with the maximum sum from a list of tuples.

Challenging/Logic Tuple Programs

31. Find duplicates in a tuple.
32. Count positive and negative numbers in a tuple.

33. Separate even and odd numbers into two tuples.
34. Merge two sorted tuples into a single sorted tuple.
35. Find the largest even number in a tuple.
36. Find the smallest odd number in a tuple.
37. Count elements greater than both neighbors in a tuple.
38. Replace negative numbers with zero in a tuple.
39. Count frequency of all elements in a tuple.
40. Find the second smallest element in a tuple.

Exam-Level Tricky Tuple Programs

41. Move all zeros to the end of a tuple.
42. Find missing numbers from 1 to n in a tuple.
43. Cumulative sum of a tuple.
44. Reverse a tuple without using slicing.
45. Rotate a tuple to the left by 3 positions.
46. Find the largest sum of consecutive elements of length 3 in a tuple.
47. Count elements that are greater than both neighbors in a tuple.
48. Find all pairs with a given sum in a tuple.
49. Flatten a nested tuple.
50. Find the missing number in a consecutive sequence in a tuple using formula.

TUPLES(ADVANCED)

1. Employee Performance Scores:
A company stores employee performance scores as tuples for different quarters:

```
performance = (
    (101, 85, 90),
    (102, 78, 88),
    (103, 95, 92)
)
```

Find the employee with the highest total score.

2. Product Sales:
Sales data for products across months:

```
sales = (
    (1001, 1200, 1300),
    (1002, 1100, 1400),
    (1003, 1500, 1200)
)
```

Find the product with maximum average monthly sales.

3. Stock Price Increase:
Stock prices over days:

```
stocks = (
    ("AAPL", 150, 155, 160),
    ("GOOGL", 2700, 2720, 2710)
)
```

Find the stock with the largest price increase between first and last day.

4. Employee Attendance:
Employee attendance tracked as tuples `(employee_id, days_attended)`:

```
attendance = (
    (101, 22),
    (102, 18),
    (103, 25)
)
```

Print employees with attendance below 20 days.

5. Server Response Times:
Server response times in milliseconds:

```
servers = (
    ("S1", 120, 150, 110),
    ("S2", 100, 130, 140)
)
```

Find the server with minimum average response time.

6. Quarterly KPI Scores:
Employee quarterly KPI scores:

```
projects = (
    (1, 80, 90, 100),
    (2, 60, 70, 80),
    (3, 85, 95, 90)
)
```

Find projects with all months > 80%.

7. Transactions:
Transaction tuples `(transaction_id, amount)`:

```
transactions = (
    (101, 5000),
    (102, 7000),
    (103, 6000),
    (104, 7000)
)
```

Find all transactions with the maximum amount.

8. System Logs:
System logs `(timestamp, status_code)`:

```
logs = (
    ("10:00", 200),
    ("10:05", 500),
    ("10:10", 200)
)
```

Count the number of failed logs (status_code != 200).

9. Order Status:
Orders stored as `(order_id, status)`:

```
orders = (
    (1, "Delivered"),
    (2, "Pending"),
    (3, "Delivered")
)
```

Extract all order IDs which are pending.

10. Salaries with Bonuses:
Employee salaries and bonuses `(employee_id, base_salary, bonus)`:

```
salaries = (
    (101, 50000, 5000),
    (102, 60000, 7000)
)
```

Calculate final salary (base + bonus) for each employee.

11. Delivery Times:
A logistics company tracks delivery times for each driver over a week:

```
delivery_times = (
    ("Driver1", 5, 7, 6, 8, 5, 6, 7),
    ("Driver2", 6, 6, 5, 7, 5, 6, 6),
    ("Driver3", 7, 8, 7, 9, 6, 8, 7)
)
```

Find the driver with least average delivery time.

12. Store Weekly Sales:

```
store_sales = (
    ("Store1", 12000, 13000, 12500, 14000),
    ("Store2", 15000, 14000, 13500, 14500),
    ("Store3", 11000, 12000, 11500, 12500)
)
```

Identify stores consistently above 13000 in all weeks.

13. Server Uptime:

```
servers = (
    ("Server1", *(24 for _ in range(30))),
    ("Server2", *(23 if i%5==0 else 24 for i in range(30))),
    ("Server3", *(22 + i%3 for i in range(30)))
)
```

Determine which server had most number of full uptime days (24 hours).

14. Employee KPI Scores:

```
employees = (
    (101, 80, 85, 90, 95),
    (102, 70, 75, 80, 85),
    (103, 90, 92, 88, 91),
    (104, 85, 87, 89, 90)
)
```

Find all employees whose average KPI score > 90.

15. Product Ratings:

```
products = (
    (201, 5, 4, 5, 3, 4),
    (202, 4, 4, 4, 5, 4),
    (203, 3, 2, 4, 3, 3)
)
```

Identify the product with highest average rating.

16. Shipment Efficiency:

```
shipments = (
    (301, 100, 500, 48),
    (302, 120, 400, 36),
    (303, 90, 450, 40),
    (304, 110, 550, 50)
)
```

Identify the shipment with best efficiency `(weight*distance)/delivery_time`.

17. Daily Work Hours:

```
work_hours = (
    (401, *(8 for _ in range(30))),
    (402, *(7 + i%2 for i in range(30))),
    (403, *(6 + i%3 for i in range(30)))
)
```

Find employees who never worked less than 7 hours on any day.

18. Customer Spending:

```
customers = (
    (501, 10, 5000, 100),
    (502, 15, 8000, 150),
    (503, 5, 3000, 50)
)
```

Find customers who spent above average AND have more than 100 loyalty points.

19. Patient Fever:

```
patients = (
    (601, 98.6, 99.1, 98.9),
    (602, 100.5, 101.0, 100.0),
    (603, 97.5, 97.8, 98.0)
)
```

Find patients with fever (>100°F) on any day.

20. Shift Production Units:

```
shifts = (
    (701, 100, 120, 110),
    (702, 90, 130, 115),
    (703, 105, 125, 120)
)
```

Identify shift with highest total production.

21. Branch Revenue by Region:

```
branches = (
    ("Branch1", ("North", 120, 130, 125, 140)),
    ("Branch2", ("South", 150, 140, 135, 145)),
    ("Branch3", ("East", 110, 120, 115, 125)),
    ("Branch4", ("West", 130, 135, 140, 145))
)
```

Identify region with branch having highest total quarterly revenue.

22. Developer Productivity:

```
developers = (
    (101, ("Sprint1", 15), ("Sprint2", 18), ("Sprint3", 20)),
    (102, ("Sprint1", 12), ("Sprint2", 20), ("Sprint3", 19)),
    (103, ("Sprint1", 20), ("Sprint2", 22), ("Sprint3", 25))
)
```

Find developer with most consistent performance (lowest standard deviation).

23. Patients 7-Day Fever:

```
patients = (
    (201, (98.6, 99.1, 100.2, 98.9, 99.5, 100.5, 98.7)),
    (202, (97.8, 98.2, 97.5, 98.0, 98.5, 98.1, 97.9)),
    (203, (100.1, 101.0, 100.5, 101.2, 100.8, 101.5, 100.9))
)
```

Identify patients with fever (>100°F) on at least 3 days.

24. Machine Production:

```
machines = (
    ("M1", (100, 120, 110, 115, 130, 125, 140)),
    ("M2", (90, 100, 95, 105, 110, 100, 115)),
    ("M3", (120, 130, 125, 135, 140, 150, 145))
)
```

Identify machines that exceeded 130 units production on any day.

25. Student Marks:

```
students = (
    (301, "Alice", (85, 90, 78, 92)),
    (302, "Bob", (88, 76, 85, 80)),
    (303, "Charlie", (95, 92, 90, 96))
)
```

Find students who scored above 90 in at least 2 subjects.

26. Driver Distance:

```
drivers = (
    ("D1", (120, 130, 115, 140, 135, 150, 145)),
    ("D2", (100, 110, 105, 120, 115, 130, 125)),
    ("D3", (150, 160, 155, 165, 170, 160, 175))
)
```

Find driver with maximum total distance.

27. Project Durations:

```
projects = (
    (401, (4, 5, 3)),
    (402, (6, 5, 4)),
    (403, (3, 2, 5))
)
```

Identify projects that took more than 12 weeks in total.

28. Company Profits:

```
companies = (
    (501, (20000, 25000, 22000, 24000)),
    (502, (30000, 28000, 29000, 31000)),
    (503, (15000, 18000, 17000, 16000))
)
```

Find company with maximum profit growth (Q4 - Q1).

29. Vehicle Fuel Consumption:

```
vehicles = (
    ("V1", (15, 16, 15, 14, 17, 16, 15)),
    ("V2", (14, 15, 14, 15, 14, 15, 14)),
    ("V3", (16, 18, 17, 19, 18, 20, 17))
)
```

Find vehicles consuming more than 18 liters on any day.

30. Customer Call Durations:

```
customers = (
    (601, (12, 15, 20, 10)),
    (602, (5, 6, 4, 5)),
    (603, (25, 30, 28, 32))
)
```

Find customers with average call duration above 20 minutes.

UNIT-5 SETS

Beginner Level

1. Set Union, Intersection, and Difference
2. Symmetric Difference of Sets
3. Cartesian Product of Two Sets
4. Frequency of Elements using Set
5. Check Subset and Superset Relationship
6. Frozen Set Demonstration
7. Remove Multiple Elements from a Set
8. Extract Unique Words from a String using Set
9. You have two sets of employee IDs:

```
engineering = {101, 102, 103, 104, 105}
marketing = {104, 105, 106, 107}
```

Task: Find employees who are **only in one department** but not in both.

10. You are tracking inventory in two warehouses:

```
warehouse_A = {"Laptop", "Mouse", "Keyboard", "Monitor"}
warehouse_B = {"Monitor", "Mouse", "Printer", "Scanner"}
```

Task: Find **items common in both warehouses**, **items unique to warehouse A**, and **all unique items across both warehouses**.

Intermediate Level

11. Calculate average marks of each student: `[("Alice", [85, 90, 78]), ("Bob", [75, 80, 85]), ("Charlie", [95, 92, 88])].`
12. Calculate total sales for each store: `[("Store1", [1200, 1300, 1250]), ("Store2", [1500, 1400, 1350]), ("Store3", [1100, 1200, 1150])].`
13. List employees earning more than 55000: `[("E101", 50000), ("E102", 60000), ("E103", 45000), ("E104", 70000)].`
14. Flatten the nested list `[[1,2],[3,4],[5,6]].`
15. Find the student with highest total marks: `[("Alice",(85,90,78)),("Bob",(88,76,85)),("Charlie",(95,92,90))].`
16. Count products with average sales above 125: `[("P1",[120,130,125]),("P2",[150,140,135]),("P3",[110,120,115])].`
17. Find maximum temperature for each city: `[("CityA",[30,32,31,29]),("CityB",[28,27,26,29]),("CityC",[33,34,32,35])].`
18. Identify employees with perfect attendance: `[("E101",[8,8,8,8,8]),("E102",[8,7,8,8,8]),("E103",[8,8,8,8,8])].`

19. List students who scored >= 60 in all subjects:
    `[("Alice",[85,90,78]),("Bob",[55,60,65]),("Charlie",[75,80,85])].`
20. Find average quarterly profit for each company:
    `[("C1",[20000,25000,22000,24000]),("C2",[30000,28000,29000,31000]),("C3",[15000,18000,17000,16000])].`

Advanced / Case-Based

21. Identify the region with the branch that had the highest total quarterly revenue:
    `(("Branch1",("North",120,130,125,140)),("Branch2",("South",150,140,135,145)),("Branch3",("East",110,120,115,125)),("Branch4",("West",130,135,140,145))).`
22. Find the developer with the most consistent performance across sprints:
    `((101,("Sprint1",15),("Sprint2",18),("Sprint3",20)),(102,("Sprint1",12),("Sprint2",20),("Sprint3",19)),(103,("Sprint1",20),("Sprint2",22),("Sprint3",25))).`
23. Identify patients with fever (>100°F) on at least 3 days:
    `((201,(98.6,99.1,100.2,98.9,99.5,100.5,98.7)),(202,(97.8,98.2,97.5,98.0,98.5,98.1,97.9)),(203,(100.1,101.0,100.5,101.2,100.8,101.5,100.9))).`
24. Identify machines that exceeded 130 units production on any day:
    `(("M1",(100,120,110,115,130)),("M2",(90,100,95,105,110)),("M3",(120,130,125,135,140))).`
25. Find students who scored above 90 in at least 2 subjects:
    `((301,"Alice",(85,90,78,92)),(302,"Bob",(88,76,85,80)),(303,"Charlie",(95,92,90,96))).`
26. Find the driver with maximum total distance:
    `(("D1",(120,130,115,140,135,150,145)),("D2",(100,110,105,120,115,130,125)),("D3",(150,160,155,165,170,160,175))).`
27. Identify projects that took more than 12 weeks in total:
    `((401,(4,5,3)),(402,(6,5,4)),(403,(3,2,5))).`
28. Find the company with maximum profit growth (Q4 - Q1):
    `((501,(20000,25000,22000,24000)),(502,(30000,28000,29000,31000)),(503,(15000,18000,17000,16000))).`
29. Find vehicles that consumed more than 18 liters on any day:
    `(("V1",(15,16,15,14,17,16,15)),("V2",(14,15,14,15,14,15,14)),("V3",(16,18,17,19,18,20,17))).`
30. Find customers with average call duration above 20 minutes:
    `((601,(12,15,20,10)),(602,(5,6,4,5)),(603,(25,30,28,32))).`

Expert / Multi-Nested

31. Find employees whose average KPI score is above 90:
    `((101,(80,85,90,95)),(102,(70,75,80,85)),(103,(90,92,88,91)),(104,(85,87,89,90))).`
32. Identify the shipment with the best efficiency `(weight*distance)/delivery_time`:
    `((301,100,500,48),(302,120,400,36),(303,90,450,40),(304,110,550,50)).`

33. Identify students with perfect attendance:
    `((401,(1,1,1,1,1,1,1)),(402,(1,0,1,1,1,1,1)),(403,(1,1,1,1,1,1,1))).`
34. Identify drivers who improved delivery time every consecutive day:
    `(("D1",(60,58,55,53,50)),("D2",(50,52,51,50,49)),("D3",(70,68,66,64,62))).`
35. Find machines with zero errors for the entire week:
    `(("M1",(0,0,0,0,0,0,0)),("M2",(0,1,0,0,1,0,0)),("M3",(0,0,0,0,0,0,0))).`
36. Identify products with consistent growth every week:
    `(("P1",(100,110,120,130)),("P2",(90,95,100,105)),("P3",(150,140,145,150))).`
37. Find flights with at least 3 days of delay >30 minutes:
    `(("F1",(10,20,35,40,25,30,45)),("F2",(5,10,15,20,25,10,15)),("F3",(30,40,50,35,20,45,30))).`
38. Identify patients who received more than 3 doses on any day:
    `((601,(2,3,2,4,3,2,3)),(602,(1,2,1,2,1,2,1)),(603,(3,4,3,5,4,3,4))).`
39. Identify developers who resolved more than 10 bugs in at least 2 projects:
    `((701,(5,12,8)),(702,(15,9,11)),(703,(10,11,12))).`
40. Identify users who used more than 5 GB on more than 3 days:
    `((801,(2,3,5,6,4,7,3)),(802,(1,2,1,2,1,2,1)),(803,(5,6,7,6,5,8,6))).`

Advanced / Tricky / Multi-Level

41. Find the store with the highest total sales across all branches: `( ("North",[("Store1",[1200,1300,1250]),("Store2",[1500,1400,1350])]), ("South",[("Store3",[1100,1200,1150]),("Store4",[1600,1550,1500])]), ("East",[("Store5",[900,950,1000]),("Store6",[1200,1250,1300])]) ).`
42. Find employees with minimal variation across all project scores:
    `((101,("ProjA",85),("ProjB",88),("ProjC",90)),(102,("ProjA",78),("ProjB",82),("ProjC",80)),(103,("ProjA",92),("ProjB",95),("ProjC",91))).`
43. Identify patients with fever >100°F on at least 3 consecutive days:
    `((201,(98,102,101,99,103)),(202,(97,98,96,97,99)),(203,(100,101,102,100,99))).`
44. Identify machines with average production >130 units:
    `(("M1",(100,120,130,125,140)),("M2",(90,110,95,105,100)),("M3",(130,135,140,145,150))).`
45. Find developers who completed >10 tasks in at least 2 projects:
    `((301,("Proj1",10),("Proj2",12),("Proj3",8)),(302,("Proj1",15),("Proj2",16),("Proj3",18)),(303,("Proj1",8),("Proj2",7),("Proj3",9))).`
46. Identify users with >5GB usage on more than 3 days:
    `((401,(2,3,5,6,4,7,3)),(402,(1,2,1,2,1,2,1)),(403,(5,6,7,6,5,8,6))).`
47. Find the branch with maximum growth (last quarter - first quarter):
    `(("North",[20000,22000,24000,26000]),("South",[25000,24000,24500,25500]),("East",[15000,16000,17000,18000])).`
48. Identify flights delayed >30 min on at least 3 days:
    `(("F1",(10,20,35,40,25,30,45)),("F2",(5,10,15,20,25,10,15)),("F3",(30,40,50,35,20,45,30))).`
49. Identify patients receiving >3 doses on any day:
    `((501,(2,3,2,4,3,2,3)),(502,(1,2,1,2,1,2,1)),(503,(3,4,3,5,4,3,4))).`

50. Identify developers resolving >10 bugs in at least 3 days:

```
((601,(5,12,8,14)),(602,(15,9,11,12)),(603,(10,11,12,13))).
```

Unit 6 — Dictionaries

Beginner

1. Create a dictionary with keys: `'name'`, `'age'`, `'city'` and assign values.
2. Access the value of `'city'` in the dictionary.
3. Update `'age'` to a new value.
4. Add a new key `'country'` with a value.
5. Delete the `'city'` key from the dictionary.
6. Check if `'age'` exists in the dictionary.
7. Get all keys from the dictionary.
8. Get all values from the dictionary.
9. Iterate over the dictionary and print key-value pairs.
10. Create a dictionary using the `dict()` function.

Intermediate

11. Merge two dictionaries using Python 3.10+ syntax.
12. Create a dictionary of squares for numbers 1 to 5 using dictionary comprehension.
13. Invert a dictionary (swap keys and values).
14. Sort a dictionary by its values.
15. Count the frequency of each character in a string using dictionary comprehension.
16. Create a nested dictionary for students with their marks.
17. Find all keys common to two dictionaries.
18. Remove duplicates from dictionary values if they are lists.
19. Rename a key in a dictionary.
20. Combine two lists into a dictionary using `zip()`.

Tricky / Conceptual

21. Explain what happens when you assign one dictionary to another and modify it.
22. Create a true copy of a dictionary to avoid reference issues.
23. Why can’t lists be used as dictionary keys?
24. Check if a specific value exists in a dictionary.
25. Merge two dictionaries and sum values for common keys.
26. Demonstrate dictionary key uniqueness with duplicate keys.
27. Demonstrate that dictionary keys must be immutable.
28. Use `setdefault()` to avoid KeyError.
29. Write a dictionary comprehension with a conditional filter.
30. Swap keys and values in a dictionary with duplicate values.

Advanced / Industry-Level

31. Use `defaultdict` to count character frequency in a string.
32. Use `Counter` to find the most common element in a list.
33. Merge a list of dictionaries into one dictionary.
34. Convert a JSON string to a dictionary.
35. Flatten a nested dictionary into a single-level dictionary.
36. Group values by key from a list of tuples.
37. Write a dictionary comprehension with multiple conditions.
38. Reverse map a dictionary (values → keys) with unique values.
39. Sort a nested dictionary by the inner dictionary's value.
40. Merge two dictionaries and keep the maximum value for common keys.

Case-Based / Project-Level

41. Store student records with marks and calculate total marks.
42. Count frequency of words in a text using a dictionary.
43. Track inventory updates in a store.
44. Create a dictionary mapping employees to multiple departments.
45. Find students scoring above a threshold in a nested dictionary.
46. Reverse lookup to find keys by value in a dictionary.
47. Merge multiple logs with timestamps, keeping the latest entry.
48. Build a dictionary of lists where keys are the first letters of words.
49. Sort students by total marks from a nested dictionary.
50. Find the most common value in a nested dictionary.

Advanced Questions with Datasets

1. Student Performance Analyzer
Dataset:

```
Students = {
    'Alice': {'Math': 90, 'Physics': 85, 'Chemistry': 88},
    'Bob': {'Math': 75, 'Physics': 80, 'Chemistry': 70},
    'Charlie': {'Math': 95, 'Physics': 92, 'Chemistry': 90}
}
```

Task: Calculate total marks, average, and grade for each student.

2. Inventory Management System
Dataset:

```
Inventory = {'apple': 50, 'banana': 30, 'orange': 20}
Sales = {'apple': 5, 'orange': 2, 'mango': 10}
Purchase = {'banana': 10, 'mango': 20}
```

Task: Update inventory after sales and purchases.

3. Employee Department Tracker
Dataset:

```
Employee_Departments = [('Alice','HR'), ('Bob','IT'), ('Alice','Finance'),
('Charlie','IT')]
```

Task: Map employees to all their departments using a dictionary of lists.

4. Text Word Frequency Analyzer
Dataset:

```
Text = "apple banana apple orange banana apple mango banana"
```

Task: Count frequency of each word and sort by frequency.

5. Library Book Management System
Dataset:

```
Library = {
    'Book1': {'Author':'Author A','Available':True},
    'Book2': {'Author':'Author B','Available':False},
    'Book3': {'Author':'Author C','Available':True}
}
```

Task: Update availability when books are borrowed or returned.

6. Online Quiz Result Tracker
Dataset:

```
Quiz_Scores = {
    'Alice':[90,85,88],
    'Bob':[75,80,70],
    'Charlie':[95,92,90]
}
```

Task: Calculate average scores for each student and identify the top scorer.

7. E-Commerce Order Summary
Dataset:

```
Orders = {
    'User1':[('Apple',2), ('Banana',3)],
    'User2':[('Apple',1), ('Orange',5)],
    'User3':[('Banana',2), ('Mango',4)]
}
```

Task: Calculate total items purchased per user.

8. Movie Rating Database
Dataset:

```
Movie_Ratings = {
    'Alice': {'Titanic':5, 'Avengers':4},
    'Bob': {'Titanic':4, 'Avengers':5},
    'Charlie': {'Titanic':5, 'Avengers':3}
}
```

Task: Calculate average rating per movie.

9. Sports Tournament Scoreboard
Dataset:

```
Match_Scores = {
    'TeamA': {'Match1':2, 'Match2':3},
    'TeamB': {'Match1':1, 'Match2':4},
    'TeamC': {'Match1':3, 'Match2':2}
}
```

Task: Calculate total scores and rank teams.

10. Nested Dictionary Data Analytics
Dataset:

```
Sales_Data = {
    'Region1': {'ProductA':100, 'ProductB':150, 'ProductC':120},
    'Region2': {'ProductA':200, 'ProductB':120, 'ProductC':180},
    'Region3': {'ProductA':150, 'ProductB':130, 'ProductC':160}
}
```

Task: Calculate total sales per product across all regions.

UNIT-7 PYTHON MODULES

1.Write a Python program to generate a random color hex, a random alphabetical string, random value between two integers (inclusive) and a random multiple of 7 between 0 and 70.
2. Write a Python program to select a random element from a list, set, dictionary-value, and file from a directory.
3. Write a Python program that generates random alphabetical characters, alphabetical strings, and alphabetical strings of a fixed length.
4. Write a Python program to construct a seeded random number generator, also generate a float between 0 and 1, excluding 1.
5. Write a Python program to generate a random integer between 0 and 6 - excluding 6, random integer between 5 and 10 - excluding 10, random integer between 0 and 10, with a step of 3 and random date between two dates.
6. Write a Python program to shuffle the elements of a given list.
7. Write a Python program to generate a float between 0 and 1, inclusive and generate a random float within a specific range.
8. Write a Python program to set a random seed and get a random number between 0 and 1.
9. Write a Python program to check if a function is a user-defined function or not. Use types.FunctionType, types.LambdaType()
10. Write a Python program to check if a given value is a method of a user-defined class. Use types.MethodType()
11. Write a Python program to check if a given function is a generator or not. Use types.GeneratorType()
12. Write a Python program to check if a given value is compiled code or not. Also check if a given value is a module or not. Use types.CodeType, types.ModuleType()
13. Write a Python program to construct a Decimal from a float and a Decimal from a string. Also represent the decimal value as a tuple. Use decimal.Decimal
14. Write a Python program to configure rounding to round up and round down a given decimal value. Use decimal.Decimal
15. Write a Python program to configure rounding to round up and round down a given decimal value. Use decimal.Decimal

ADVANCED

1.Write a Python program to round a decimal value to the nearest multiple of 0.10, unless already an exact multiple of 0.05. Use decimal.Decimal
2. Write a Python program to configure the rounding to round to the floor, ceiling. Use decimal.ROUND_FLOOR, decimal.ROUND_CEILING
3. Write a Python program that can be configured to round to the nearest - with ties going towards 0 and ties going away from 0. Use decimal.ROUND_HALF_DOWN, decimal.ROUND_HALF_UP
4. Write a Python program to configure rounding to round to the nearest integer, with ties going to the nearest even integer. Use decimal.ROUND_HALF_EVEN
5. Write a Python program to configure rounding to round to the nearest integer, with ties going to the nearest even integer. Use decimal.ROUND_HALF_EVEN`
6. Write a Python program to create a shallow copy of a given list. Use copy.copy
7. Write a Python program to create a deep copy of a given list. Use copy.copy
8. Write a Python program to create a shallow copy of a given dictionary. Use copy.copy
9. Write a Python program to create a deep copy of a given dictionary. Use copy.copy
10. Write a Python program to read and display the content of a given CSV file. Use csv.reader
11. Write a Python program to count the number of lines in a given CSV file. Use csv.reader
12. Write a Python program to parse a given CSV string and get a list of lists of string values. Use csv.reader
13. Write a Python program to parse a given CSV string and get a list of lists of string values. Use csv.reader
14. Write a Python program to read the current line from a given CSV file. Use csv.reader
15. Write a Python program to skip the headers of a given CSV file. Use csv.reader

CASE BASED

1.Write a Python program to write (without writing separate lines between rows) and read a CSV file with a specified delimiter. Use csv.reader
2. Converting Python Objects to JSON Strings
3. Converting JSON Strings to Python Objects
4. Writing Python Objects to a File (Serialization to File)
5. Convert a Python object containing all the legal data types

UNIT-8 PYTHON FUNCTIONS

1. Write a Python function to find the maximum of three numbers.

2. Write a Python function to sum all the numbers in a list.

3. Write a Python function to multiply all the numbers in a list.

4. Write a Python program to reverse a string.

5. Write a Python function to calculate the factorial of a number (a non-negative integer).

6. Write a Python function to check whether a number falls within a given range

7. Write a Python function that accepts a string and counts the number of upper- and lower-case letters.

8. Write a Python function that takes a list and returns a new list with distinct elements from the first list.

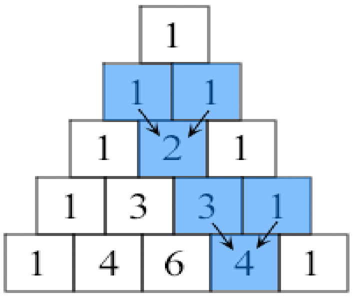
9. Write a Python function that takes a number as a parameter and checks whether the number is prime or not.

10. Write a Python program to print the even numbers from a given list.

11. Write a Python function to check whether a number is "Perfect" or not.

12. Write a Python function that checks whether a passed string is a palindrome or not.

13. Write a Python function that prints out the first n rows of Pascal's triangle.



14. Write a Python function to check whether a string is a pangram or not.

15. Write a Python program that accepts a hyphen-separated sequence of words as input and prints the words in a hyphen-separated sequence after sorting them alphabetically

16. Write a Python function to create and print a list where the values are the squares of numbers between 1 and 30 (both included).

PYTHON RECURSION

Beginner Level

1. Write a recursive function to calculate the factorial of a number.
2. Create a recursive function to print numbers from 1 to N.
3. Write a recursive function to find the sum of first N natural numbers.
4. Write a recursive function to calculate the nth Fibonacci number.
5. Write a recursive function to find the power of a number (x^n).

6. Write a recursive function to reverse a string.
7. Write a recursive function to find the length of a string without using `len()`.
8. Write a recursive function to find the maximum element in a list.

Intermediate Level

9. Write a recursive function to count the number of digits in an integer.
10. Write a recursive function to calculate the sum of digits of a number.
11. Write a recursive function to check if a number is palindrome or not.
12. Write a recursive function to compute the greatest common divisor (GCD) of two numbers.
13. Write a recursive function to print elements of a list in reverse order.
14. Write a recursive function to check whether a string is palindrome.
15. Write a recursive function to convert a decimal number to binary.
16. Write a recursive function to count occurrences of a given element in a list.

Advanced / Case-Based / Trick Questions

17. Write a recursive function to flatten a nested list (e.g., `[1, [2, [3, 4]], 5] → [1, 2, 3, 4, 5]`).
18. Write a recursive function to find all permutations of a string.
19. Write a recursive function to find the sum of all elements in a nested list.
20. Write a recursive function to find the nth term in a custom sequence defined by:

```
f(n) = f(n-1) + 2*f(n-2), with f(1)=1, f(2)=2
```

21. Write a recursive function to perform binary search on a sorted list.
22. Write a recursive function to print all subsets of a given set/list.
23. Write a recursive function to compute the sum of series: S=1+12+13+…+1nS = 1 + \frac{1}{2} + \frac{1}{3} + … + \frac{1}{n}S=1+21+31+…+n1.
24. Write a recursive function to generate all possible combinations of a given list of numbers.
25. Write a recursive function to count the total number of vowels in a string.

Python lambda function
1. Write a lambda function to check if a number is a **palindrome**.
2. Use lambda and filter() to extract all **prime numbers** from a given list.
3. Create a lambda function that returns the **maximum of three numbers**.
4. Write a lambda function that checks if a given **string is a pangram** (contains all letters a–z).
5. Use map() and a lambda to convert a list of temperatures in **Celsius to Fahrenheit**.
6. Sort a list of **tuples by the second element** using a lambda function.
7. Sort a dictionary by **its values** using a lambda.
8. Use reduce() with a lambda to **find the product** of all numbers in a list.
9. Write a lambda that **reverses words in a sentence** while keeping their order.
10. Create a lambda to **count vowels** in a string using filter().
11. Use a lambda inside map() to **capitalize alternate words** in a sentence.
12. Write a lambda that returns 'Even' if a number is even, otherwise 'Odd' — using a **single-line conditional expression**.
13. Write a lambda to **find factorial** of a number using **recursion-like behavior** with reduce().
14. Combine two lists element-wise using a lambda and map() (like [1,2,3] and [4,5,6] → [5,7,9]).
15. Use lambda with filter() to get all strings from a list that **start and end with the same letter**.
16. Create a lambda that checks whether a string is a **valid email address** (very simple rule-based).
17. Use a lambda with sorted() to arrange names based on their **last character**.
18. Write a lambda to compute the **sum of squares of even numbers** from a list.
19. Use nested lambdas to find the **square of sum** of two numbers.
20. Write a lambda to remove **duplicate words** from a sentence.
21. Use lambda and map() to **prefix every word** in a list with "Python_".
22. Create a lambda to **flatten a list of lists** using reduce().

23. Use a lambda to **count frequency** of each character in a string (return a dictionary).
24. Write a lambda to compute the **Nth Fibonacci number** using recursion-like logic.
25. Combine multiple operations: use map(), filter(), and lambda to get the **square of all odd numbers greater than 5** from a list.

PYTHON FUNCTION ADVANCED

1. Create a Chain of Function Decorators (Bold, Italic, Underline, etc.)
2. Access a Function Inside a Function
3. Detect the Number of Local Variables Declared in a Function
4. Invoke a Function After a Specified Period of Time
5. Use a lambda with the filter() function to get all even numbers from a list
6. Use a lambda with the map() function to double each element in a list
7. Use a lambda with the sorted() function to sort a list of tuples based on the second element
8. Create Higher-Order Function
9. Python Function TO Check Before You Append
10. Python Function to Removing Duplicates and Sorting
11. Python Function To Find the Second Occurrence
12. Python Function to Sorting Non-Negative Numbers
13. Python Function to Caesar Cipher

Unit-9 File handling

Basic File Operations

1. Write a Python program to **create a new text file** and write a string into it.
2. Write a program to **read and display the contents** of a text file line by line.
3. Write a program to **count the number of lines, words, and characters** in a file.
4. Write a program to **append user input** to an existing text file.
5. Write a program to **copy contents** from one file to another.
6. Write a program to **read only the first n lines** of a file.
7. Write a program to **read the last n lines** of a file (like a `tail` command).
8. Write a program to **check if a file exists** before reading or writing.
9. Write a program to **rename and delete** a file using the `os` module.
10. Write a program to **find the size** of a file in bytes.

File Data Analysis

11. Write a program to **count the frequency of each word** in a text file.
12. Write a program to **find and replace a word** in a file.
13. Write a program to **find the longest word** in a text file.
14. Write a program to **count how many times each vowel** appears in a file.
15. Write a program to **read numeric data** from a file and compute the **sum and average**.
16. Write a program to **remove all blank lines** from a text file.
17. Write a program to **find all unique words** and store them in a new file.
18. Write a program to **compare two files** and show differences line-by-line.
19. Write a program to **merge the contents of two text files** into one.
20. Write a program to **sort the lines** of a text file alphabetically.

CSV & JSON Handling

21. Write a program to **read a CSV file** and display its contents.
22. Write a program to **write student records** into a CSV file.
23. Write a program to **find the student with the highest marks** from a CSV file.
24. Write a program to **count rows and columns** in a CSV file.
25. Write a program to **update a particular cell** in a CSV file.
26. Write a program to **convert CSV data to JSON** format.
27. Write a program to **read data from a JSON file** and display it.
28. Write a program to **create a JSON file** containing employee details.
29. Write a program to **update a key-value pair** in a JSON file.
30. Write a program to **merge two JSON files** into a single dictionary.

Advanced File Handling

31. Write a program to **encrypt and decrypt file content** (basic Caesar cipher).

32. Write a program to **remove duplicate lines** from a text file.
33. Write a program to **split a large text file** into smaller parts (by lines).
34. Write a program to **combine multiple files** into one.
35. Write a program to **log all read/write actions** into a separate log file.
36. Write a program to **search for a specific keyword** across multiple files in a folder.
37. Write a program to **read a binary file** and print its hexadecimal representation.
38. Write a program to **copy an image (binary file)** using file handling.
39. Write a program to **track changes** made to a file (version log).
40. Write a program to **compress and decompress a text file** using `gzip`.

Real-Life / Case-Based Scenarios

41. Case: You have a log file (`server.log`) — **count total error lines** containing `"ERROR"`.
42. Case: Given a `transactions.txt` file — **calculate total debit and credit amount**.
43. Case: From a CSV of employee salaries — **find department-wise average salary**.
44. Case: Write a script to **generate a report file** summarizing student grades.
45. Case: Given a file of names — **separate names by first letter** into multiple files (A.txt, B.txt…).
46. Case: Write a program to **analyze a chat log** (`chat.txt`) and count how many messages each user sent.
47. Case: Write a program to **parse a configuration file** (`config.txt`) into a dictionary.
48. Case: Write a program to **extract all email addresses** from a text file and store them in another file.
49. Case: Create a **mini text editor** in Python (open, edit, save) using file handling.
50. Case: Create a **backup system** that automatically copies a file into a `backup/` folder whenever it's modified.

UNIT-10 EXCEPTIONAL HANDLING

Beginner Level

1. Write a program to handle division by zero error.
2. Write a program to handle invalid integer input using `try-except`.
3. Handle `IndexError` in a list access.
4. Handle `KeyError` in a dictionary lookup.
5. Handle multiple exceptions (`ZeroDivisionError` and `ValueError`) in a single `try`.
6. Write a program that uses `else` block in exception handling.
7. Demonstrate `finally` block usage with file handling.
8. Handle `TypeError` when adding a string and integer.
9. Handle `AttributeError` when calling a non-existent method.
10. Handle `FileNotFoundError` when reading a file.
11. Handle `ImportError` for a non-existent module.
12. Write a program to catch `NameError` for an undefined variable.
13. Use `pass` in an exception block to ignore an exception.
14. Demonstrate `try-except-else-finally` flow with a simple number input.
15. Raise a `ValueError` if input number is negative.
16. Catch exception for converting invalid string to integer.
17. Handle `ZeroDivisionError` inside a loop.
18. Handle `IndexError` in a loop iterating over a shorter list.
19. Demonstrate `Exception` base class to catch all exceptions.
20. Use `try-except` to validate user input until correct integer is entered.
21. Handle `KeyboardInterrupt` gracefully in a loop.
22. Handle `OSError` when opening a file with incorrect permissions.
23. Handle `UnicodeEncodeError` when writing non-ASCII to ASCII file.
24. Handle `ValueError` and prompt user to re-enter data.
25. Demonstrate nested `try-except` blocks with basic arithmetic operations.

Intermediate / Campus Interview Level

26. Handle multiple exceptions with separate `except` blocks for file operations.
27. Raise and catch `IndexError` manually.
28. Handle `KeyError` in nested dictionaries.
29. Catch exception for dividing a number by a user input value.
30. Use `try-except` to handle list `pop()` on empty list.
31. Handle exception when opening multiple files simultaneously.
32. Catch `TypeError` in a function with mixed argument types.
33. Handle `ZeroDivisionError` in a function and return `None`.
34. Handle exceptions in reading JSON data from a file.
35. Raise `TypeError` if function argument is not string.
36. Handle `AttributeError` in object without the required method.
37. Catch `ValueError` when parsing multiple inputs from a string.
38. Handle exception in a program to compute square roots of numbers (negative input).
39. Handle exceptions in `try-except-finally` while writing to a file.
40. Handle `IOError` for reading/writing non-existent files.

41. Handle exception when converting list of strings to integers.
42. Raise `ZeroDivisionError` manually and catch it.
43. Handle `OverflowError` in exponentiation operation.
44. Handle `FileNotFoundError` and prompt user to re-enter filename.
45. Handle exception when importing a module dynamically.
46. Use `finally` to close a file even if exception occurs.
47. Catch `StopIteration` in iterating over a custom iterator.
48. Handle exception in a function that returns division of two numbers.
49. Handle `KeyError` in a program simulating student database lookup.
50. Handle exceptions when accessing elements of a tuple by index.

Advanced / Case-Based

51. Handle exceptions in a banking system (withdraw exceeding balance).
52. Handle exceptions in an ATM simulation (invalid PIN, insufficient balance).
53. Handle exceptions in a flight booking system (booking unavailable seat).
54. Handle exceptions in a hotel room booking system (no rooms left).
55. Handle exceptions in a library management system (book not available).
56. Handle exceptions in online shopping cart (product out of stock).
57. Handle exceptions in a payroll system (invalid employee data).
58. Handle exceptions in cinema ticket booking (seat already booked).
59. Handle exceptions in quiz system (invalid answer input).
60. Handle exceptions in school grading system (marks out of range).
61. Handle exceptions in gym membership management (invalid member ID).
62. Handle exceptions in smart home devices control (non-existent device).
63. Handle exceptions in digital library (borrowing already borrowed book).
64. Handle exceptions in multi-project task management (task already completed).
65. Handle exceptions in food ordering system (quantity negative).
66. Handle exceptions in flight fare calculation (seat class invalid).
67. Handle exceptions in hotel multi-room booking (not enough rooms).
68. Handle exceptions in vehicle parking system (parking full).
69. Handle exceptions in employee hierarchy system (invalid employee type).
70. Handle exceptions in inventory management system (negative stock).
71. Handle exceptions in e-commerce order (total exceeds limit).
72. Handle exceptions in payroll with overtime calculation (negative hours).
73. Handle exceptions in smart home scenes (scene not defined).
74. Handle exceptions in banking system with transaction history (invalid amount).
75. Handle exceptions in quiz leaderboard (negative score or invalid participant).

Industry Level

76. Implement a function to handle multiple exceptions for file parsing (JSON + CSV).
77. Handle exception when reading corrupted Excel file.
78. Handle exception in multi-threaded application accessing shared data.
79. Handle exceptions in network socket programming (connection errors).
80. Handle exceptions in API response parsing (missing keys, invalid data types).

81. Handle exceptions in multi-level inheritance (missing attribute/method).
82. Handle exceptions in database connection (MySQL/PostgreSQL) and query execution.
83. Handle exceptions in web scraping (invalid URL, timeout, connection error).
84. Handle exception when writing large binary files (disk full).
85. Handle exceptions in image processing library (invalid image format).
86. Handle exceptions in AI/ML dataset preprocessing (missing/NaN values).
87. Handle exceptions in pandas DataFrame operations (key/column missing).
88. Handle exceptions in numpy array operations (shape mismatch).
89. Handle exception in multi-file processing with logging.
90. Handle exceptions in JSON API with retry logic.
91. Handle exceptions in sending emails programmatically (SMTP errors).
92. Handle exceptions in GUI applications (Tkinter button click, missing widgets).
93. Handle exceptions in multiprocessing pool (worker failure).
94. Handle exceptions in data serialization (pickle/dill errors).
95. Handle exceptions in URL request and response parsing (HTTPError, URLError).
96. Handle exceptions in custom iterator implementation.
97. Handle exceptions in recursive function with base case validation.
98. Handle exceptions in threaded producer-consumer queue.
99. Handle exceptions in exception chaining (`raise ... from ...` usage).
100. Implement a robust exception handling system in a mini industry-level project (Banking + Transactions + Logging + User Input Validation).

UNIT 11-OOPS

Beginner Level

(Basic class, object, attributes, methods, constructor/destructor)

1. Define a class `Car` and create an object for it.
2. Add attributes `brand` and `model` to the `Car` class and initialize them via a constructor.
3. Add a method `display_info` to show `brand` and `model` of the car.
4. Create multiple objects of the `Car` class with different attribute values.
5. Add a class variable `count` to track the number of `Car` objects created.
6. Write a destructor method to print a message when an object is destroyed.
7. Create a class `Student` with attributes `name`, `roll_no`, `marks`. Initialize using `__init__`.
8. Add a method `percentage()` to calculate the percentage of marks.
9. Modify the `Student` class to include default values for attributes.
10. Create an object of a class and access its attributes using `getattr()` and modify using `setattr()`.
11. Create a method in `Student` class to update marks.
12. Create a class `Circle` with an attribute `radius` and method `area()` to calculate area.
13. Add another method `circumference()` to the `Circle` class.
14. Create a class `Rectangle` and initialize `length` and `width`. Write a method to calculate area.
15. Add a method to `Rectangle` class to check if it is a square.

16. Create a class `BankAccount` with `account_no` and `balance`. Write `deposit()` and `withdraw()` methods.
17. Modify `withdraw()` method to prevent balance going below zero.
18. Create a class `Person` and a class `Employee` that inherits from `Person`.
19. Use `super()` to call parent class constructor in `Employee`.
20. Create a class `Library` with method `add_book()`, `show_books()`.
21. Write a class `Temperature` to convert Celsius to Fahrenheit and vice versa.
22. Create a class `Time` to store hours and minutes, and add a method to display in 12-hour format.
23. Implement a class `Fraction` and methods for addition, subtraction of fractions.
24. Create a class `Point` with `x` and `y` coordinates and method to calculate distance from origin.
25. Implement a class `Person` with `name` and `age` and a method `is_adult()` to check if age ≥ 18.

Intermediate Level

(Inheritance, encapsulation, polymorphism, class methods, static methods, properties)

26. Create a class `Shape` and inherited classes `Square`, `Circle`.
27. Override a method `area()` in each subclass.
28. Demonstrate single inheritance in Python.
29. Demonstrate multiple inheritance with classes `Person` and `Employee`.
30. Demonstrate multilevel inheritance (`Grandparent → Parent → Child`).
31. Demonstrate hierarchical inheritance (`Parent → Child1`, `Parent → Child2`).
32. Demonstrate method overriding.
33. Demonstrate method overloading using default arguments.
34. Create a class with a private attribute and write getter and setter methods.
35. Use `@property` decorator to access private attributes.
36. Use `@staticmethod` decorator to create a utility method in a class.
37. Use `@classmethod` to create a method that returns class-level information.
38. Create a `Counter` class that keeps track of how many objects are created.
39. Implement `__str__` method to display object info neatly.
40. Implement `__repr__` method and demonstrate difference from `__str__`.
41. Create a class `Employee` and override `__eq__` to compare based on employee ID.
42. Override `__lt__` and `__gt__` to compare employees based on salary.
43. Implement encapsulation: make `salary` private and provide getter/setter with validation.
44. Create a class `Book` with `title` and `author`. Use `__del__` to print deletion message.
45. Create a class `Student` with class variable `college_name` and modify it using class method.
46. Create abstract base class `Vehicle` and implement subclasses `Car` and `Bike`.
47. Demonstrate polymorphism with a function that accepts multiple types of objects.
48. Demonstrate operator overloading for `+` operator in a `Vector` class.
49. Implement operator overloading for `*` operator in a `Point` class.
50. Implement `__len__` method in a `Book` class to return length of title.

Case-Based / Campus Interview Level

51. Create a `Bank` class to handle multiple accounts using OOP.
52. Implement a `Library Management System` with classes `Book`, `Member`, `Library`.
53. Implement a `Student Management System` with class `Student`, `Course`, `Grades`.
54. Implement a `Car Rental System` with class `Car`, `Customer`, `Rental`.
55. Implement a `Shopping Cart` with `Product`, `Cart`, `Order` classes.
56. Implement a `Zoo Management` system with `Animal`, `Cage`, `Zoo` classes.
57. Implement a `Hospital Management System` with `Patient`, `Doctor`, `Appointment` classes.
58. Implement a `Flight Reservation System` with `Flight`, `Passenger`, `Reservation` classes.
59. Implement a `Cinema Booking System` with `Movie`, `Theater`, `Seat` classes.
60. Create a `Game` class hierarchy: `Game → VideoGame → BoardGame`.
61. Implement `Employee Payroll System` with salary calculation and bonus.
62. Create a `School Management System` with teacher and student interactions.
63. Implement a `Text Editor` with classes `Document`, `File`, `Editor`.
64. Create a `Weather System` class with polymorphic methods for different regions.
65. Implement a `Restaurant Ordering System` with `Menu`, `Order`, `Bill`.
66. Implement `E-commerce Order Tracking System` with `Order`, `Shipment`, `Customer`.
67. Implement `Inventory Management` with class `Item` and `Inventory`.
68. Create `Bank ATM Simulation` using OOP with `ATM`, `Account`, `Transaction`.
69. Implement `School Library Fine System` using classes.
70. Implement `Hotel Reservation System` using OOP.
71. Implement `Online Quiz System` using classes for `Quiz`, `Question`, `Participant`.
72. Create a `Banking Transaction` class that supports deposit, withdraw, transfer.
73. Create a `Vehicle Rental` system with inheritance for `Bike`, `Car`, `Truck`.
74. Implement `Smart Home Automation` classes for `Device`, `Switch`, `Sensor`.
75. Create `Employee Attendance System` with check-in/check-out functionality.

DATA STRUCTURE BASED

76. Implement a `Polynomial` class with addition, subtraction, and multiplication.
77. Implement a `Matrix` class with addition, multiplication, and transpose.
78. Implement `Complex Number` class with arithmetic operations.
79. Implement `Rational Number` class with fraction simplification.
80. Implement `Vector` class with dot product and cross product.
81. Implement a `Graph` class with adjacency list representation.
82. Implement a `Stack` and `Queue` class using OOP.
83. Implement a `Priority Queue` class.
84. Implement `Linked List` using classes and methods.
85. Implement a `Doubly Linked List` using classes.
86. Implement `Binary Tree` using classes with traversal methods.
87. Implement `BST` (Binary Search Tree) insertion, search, delete.
88. Implement `Heap` class with heapify and insert operations.

89. Implement a `Sparse Matrix` class using OOP.
90. Implement a `Graph BFS` and `DFS` traversal using classes.

Industry / Project Level

91. Implement `User Authentication System` with `User`, `Login`, `Role` classes.
92. Implement a `Chat Application` with `User`, `Message`, `ChatRoom` classes.
93. Implement `File Management System` with `File`, `Folder`, `User` classes.
94. Implement `Notification System` with `Email`, `SMS`, `PushNotification` classes.
95. Implement a `Bank Loan Processing` system with `Customer`, `Loan`, `Bank` classes.
96. Implement `E-commerce Shopping Cart` with product discounts, tax, and total calculation.
97. Implement `Task Management System` with `Project`, `Task`, `User` classes.
98. Implement `IoT Device Monitoring System` with `Device`, `Sensor`, `DataLogger` classes.
99. Implement `School ERP System` with `Student`, `Teacher`, `Class`, `Exam` classes.
100. Implement `Hospital Management System` with `Patient`, `Doctor`, `Appointment`, `Billing` classes.

UNIT- 12(DECORATOR,GENERATOR,CONSTRUCTOR)

Decorators (1–10)

1. Write a simple decorator that prints “Before” and “After” a function call.
2. Write a decorator to measure execution time of a function.
3. Create a decorator that doubles the return value of a function.
4. Write a decorator that prints the function name before executing it.
5. Write a decorator that repeats a function call 3 times.
6. Write a decorator that logs the arguments passed to a function.
7. Write a decorator that converts the result of a function to uppercase (string function).
8. Write a decorator that counts the number of times a function is called.
9. Write a decorator that prints the current date and time before executing the function.
10. Write a decorator that validates input to ensure it is a positive integer.

Generators (11–18)

11. Write a generator that yields the first 10 natural numbers.
12. Write a generator for squares of numbers from 1 to 10.
13. Write a generator that yields Fibonacci series up to n.
14. Write a generator that yields even numbers from 1 to 20.
15. Write a generator that yields characters from a string one by one.
16. Write a generator to iterate over a list and yield only positive numbers.
17. Write a generator to yield prime numbers up to n.
18. Write a generator to yield factorial of numbers from 1 to n.

Constructors (19–25)

19. Create a class with a constructor to initialize name and age.
20. Write a class with a constructor that prints a welcome message.
21. Create a class with a constructor to initialize an empty list.
22. Write a class constructor with default values for some attributes.
23. Create a class constructor that validates age to be positive.
24. Create a class with constructor chaining using `__init__`.
25. Write a class with a constructor that increments a class-level counter for every instance.

Intermediate / Campus Interview Level

Decorators (26–35)

26. Write a decorator that caches function results (memoization).
27. Write a decorator to log function execution time in milliseconds.
28. Write a decorator to ensure a function accepts only integers as arguments.
29. Write a decorator that restricts access to a function based on a “role” argument.
30. Write a decorator that retries a function if it raises an exception (up to 3 times).
31. Write a decorator that prints function docstring before execution.
32. Write a decorator that applies another decorator to all methods in a class.

33. Write a decorator that multiplies numeric function outputs by a factor of 5.
34. Write a decorator that converts all string arguments of a function to uppercase.
35. Write a decorator that prints a message only when a function executes successfully.

Generators (36–45)

36. Write a generator that reads a large file line by line.
37. Write a generator that yields numbers divisible by 3 from 1 to 100.
38. Write a generator to produce an infinite sequence of natural numbers.
39. Write a generator to iterate over nested lists and yield all elements.
40. Write a generator that merges two sorted lists.
41. Write a generator that produces squares of only odd numbers.
42. Write a generator for cumulative sum of numbers in a list.
43. Write a generator that yields numbers in reverse from n to 1.
44. Write a generator that yields powers of 2 up to n.
45. Write a generator to produce a stream of random numbers (use `random.randint`).

Constructors (46–50)

46. Create a class with constructor that accepts variable number of arguments.
47. Write a class with constructor that reads data from a file to initialize attributes.
48. Create a class with constructor that copies attributes from another object.
49. Write a class constructor that raises exception if a required field is missing.
50. Create a class constructor that initializes attributes based on a dictionary input.

Advanced / Case-Based

Decorators (51–60)

51. Write a decorator that logs exceptions raised by a function.
52. Create a decorator to enforce type hints at runtime.
53. Write a decorator that measures memory usage of a function.
54. Write a decorator that limits the number of times a function can be called.
55. Create a decorator that ensures function execution order in a pipeline.
56. Write a decorator that adds retry with exponential backoff.
57. Write a decorator to validate email or phone number inputs.
58. Create a decorator that formats the return value of a function (e.g., currency formatting).
59. Write a decorator to cache function results with expiration time.
60. Write a decorator that logs both arguments and return value.

Generators (61–70)

61. Write a generator to stream large CSV data and convert to dictionaries.
62. Write a generator that yields sentences from a text file one by one.
63. Create a generator to simulate a live stock price feed.
64. Write a generator that produces prime numbers lazily for large n.
65. Write a generator to flatten nested dictionaries into key-value pairs.

66. Write a generator that reads multiple files and yields lines alternately.
67. Write a generator that produces unique combinations of a list.
68. Create a generator that yields Fibonacci numbers indefinitely but stops on exceeding a limit.
69. Write a generator that merges multiple sorted files into one sorted stream.
70. Write a generator that yields moving averages of a list of numbers.

Constructors (71–75)

71. Write a class constructor that initializes attributes from JSON input.
72. Create a class constructor that connects to a database and stores connection object.
73. Write a class with constructor that raises exception for duplicate entries.
74. Create a constructor that initializes a nested object hierarchy.
75. Write a constructor that validates multiple attributes with complex rules.

Industry Level

Decorators (76–85)

76. Write a decorator to log function execution asynchronously to a file.
77. Write a decorator that converts a blocking function to a non-blocking coroutine.
78. Create a decorator to throttle function calls (e.g., 1 call per second).
79. Write a decorator that validates a JSON payload for a function.
80. Create a decorator to retry API calls with exponential backoff.
81. Write a decorator to monitor performance metrics for a class of functions.
82. Create a decorator that dynamically adds authentication check for functions.
83. Write a decorator that handles multiple exception types and logs to separate files.
84. Create a decorator that converts outputs to a specific format (e.g., XML/JSON).
85. Write a decorator that enables multi-threaded execution of a function.

Generators (86–95)

86. Write a generator to stream live tweets using Twitter API.
87. Write a generator to yield streaming data from sensors.
88. Create a generator that lazily evaluates large mathematical series.
89. Write a generator that produces a batch of data for ML training.
90. Create a generator that streams video frames for processing.
91. Write a generator that produces random UUIDs indefinitely.
92. Write a generator that monitors log files and yields new lines as they appear.
93. Create a generator to yield sliding windows of size k over a large list.
94. Write a generator that produces unique pairs of users for social network matching.
95. Write a generator that yields real-time stock prices with delay simulation.

Constructors (96–100)

96. Create a class constructor that initializes multi-threaded logging system.

97. Write a class constructor that initializes a configuration manager reading from multiple sources.
98. Create a constructor that sets up an ML model pipeline with default preprocessing.
99. Write a constructor that validates complex user input and raises detailed exceptions.
100. Create a constructor that initializes a mini industry-level banking system with accounts, transactions, and logging.

UNIT-13 NUMPY

Beginner to Intermediate

1. Create a 1D NumPy array of integers from 1 to 50.
2. Create a 2D NumPy array of shape (5,5) filled with zeros.
3. Create a 3D NumPy array of ones with shape (3,3,3).
4. Create an identity matrix of size 6×6.
5. Create a NumPy array of numbers from 10 to 100 with step 10.
6. Generate 10 random integers between 0 and 50.
7. Generate a 3×3 matrix of random floats between 0 and 1.
8. Find the shape, dimension, and size of a given array.
9. Reshape a 1D array of size 12 to a 3×4 2D array.
10. Flatten a 2D array into a 1D array.
11. Concatenate two 2D arrays vertically and horizontally.
12. Split a 1D array into 3 equal parts.
13. Find the maximum, minimum, and their indices in an array.
14. Compute the mean, median, and standard deviation of an array.
15. Compute the sum along rows and columns of a 2D array.
16. Multiply two matrices using `np.dot`.
17. Find the element-wise square and square root of an array.
18. Replace all negative numbers in an array with 0.
19. Select all even numbers from an array.
20. Create a boolean mask to select elements greater than a threshold.
21. Sort a 1D array in ascending and descending order.
22. Reverse a 1D array.
23. Round an array of floats to 2 decimal places.
24. Generate 20 equally spaced numbers between 0 and 5 using `linspace`.
25. Create a 2D array and swap its rows and columns.

Advanced / Case-Based

26. Find the unique elements and their counts in a NumPy array.
27. Compute cumulative sum and cumulative product of an array.
28. Compute the dot product of two vectors.
29. Compute the cross product of two 3D vectors.
30. Compute the determinant of a 3×3 matrix.
31. Compute the eigenvalues and eigenvectors of a square matrix.
32. Solve a system of linear equations using NumPy.
33. Compute the inverse of a matrix.
34. Create a diagonal matrix from a given 1D array.
35. Extract the diagonal of a square matrix.
36. Repeat elements of an array multiple times.
37. Tile an array to form a larger array.
38. Compute element-wise logarithm and exponential of an array.
39. Compute element-wise sine, cosine, and tangent of an array.
40. Create a structured array with fields `name`, `age`, `salary`.

41. Extract all rows where age > 30 in a structured array.
42. Use `np.where` to replace all negative numbers with the mean of positive numbers.
43. Generate a random 5×5 matrix and normalize it to range [0,1].
44. Create a checkerboard matrix of size 8×8.
45. Generate a random permutation of numbers from 0 to 19.
46. Find all prime numbers in a 1D NumPy array.
47. Compute the rank of a matrix.
48. Compute the trace of a square matrix.
49. Broadcast a 1D array to add to each row of a 2D matrix.
50. Compute pairwise distances between rows of a 2D array.

Critical / Industry Level

51. Implement element-wise conditional operations using `np.where` (e.g., apply tax if salary > 50000).
52. Vectorize a loop that computes `y = x^2 + 3x + 2` for a large 1D array.
53. Create a large 2D random array (10^6 × 10) and compute column-wise mean efficiently.
54. Compute covariance matrix of a dataset using NumPy.
55. Perform PCA on a dataset using NumPy linear algebra functions.
56. Implement Min-Max scaling manually using NumPy.
57. Implement Z-score normalization manually using NumPy.
58. Find the top 5 largest elements in a large array without sorting the entire array.
59. Compute matrix exponentiation (A^n) for a square matrix.
60. Simulate 1000 random walks using NumPy arrays.
61. Generate a random symmetric matrix.
62. Find the indices of the top 3 maximum values in a 2D array.
63. Compute moving average of a 1D time-series array using convolution.
64. Generate a 2D Gaussian kernel matrix.
65. Implement batch matrix multiplication efficiently using NumPy broadcasting.
66. Compute the Moore-Penrose pseudo-inverse of a non-square matrix.
67. Implement one-hot encoding for a 1D array of categorical labels.
68. Compute correlation coefficients between all pairs of columns in a 2D dataset.
69. Identify outliers in a dataset using the 1.5*IQR rule.
70. Compute the cumulative distribution function (CDF) of a 1D array.
71. Rotate a 2D matrix by 90, 180, 270 degrees without using loops.
72. Flip a 2D array along vertical and horizontal axes.
73. Implement a 2D convolution operation manually using NumPy arrays.
74. Perform eigen decomposition and reconstruct the original matrix.
75. Simulate and vectorize Monte Carlo estimation of Pi using NumPy.

UNIT-14 PANDAS

Beginner

1. Create a Pandas Series from a Python list of numbers.
2. Create a Pandas DataFrame from a Python dictionary.
3. Load a CSV file into a Pandas DataFrame.
4. Display the first 5 and last 5 rows of a DataFrame.
5. Get column names, index, and basic info of a DataFrame.
6. Select a single column as Series and as DataFrame.
7. Select multiple columns from a DataFrame.
8. Filter rows based on a condition (e.g., age > 30).
9. Filter rows based on multiple conditions (age > 30 & salary > 50000).
10. Add a new column to a DataFrame (e.g., tax = 10% of salary).
11. Delete a column and a row from a DataFrame.
12. Rename columns of a DataFrame.
13. Sort a DataFrame by one column and multiple columns.
14. Get basic statistics (mean, median, std) of numeric columns.
15. Count unique values and value counts of a column.
16. Check for missing values and count them.
17. Fill missing values with mean, median, or mode.
18. Drop rows or columns with missing values.
19. Replace values in a column (e.g., 'M' → 'Male').
20. Select rows by position using `iloc`.
21. Select rows by label using `loc`.
22. Reset index and set a new index.
23. Apply a function to a column using `apply()`.
24. Map a function or dictionary to a column using `map()`.
25. Filter rows using `isin()`.

Intermediate / Case-Based

26. Group by a column and compute aggregate statistics.
27. Group by multiple columns and aggregate using sum, mean, count.
28. Pivot a DataFrame using `pivot_table()`.
29. Melt a DataFrame from wide to long format.
30. Merge two DataFrames on a common column (inner join).
31. Merge two DataFrames with outer, left, and right joins.
32. Concatenate two DataFrames vertically and horizontally.
33. Perform an outer join and handle missing values.
34. Create a categorical column from a numeric column using `cut()`.
35. Create a rank column based on another column.
36. Find correlation between numeric columns.
37. Compute rolling mean and rolling sum for time-series data.
38. Shift and lag columns in a DataFrame.
39. Compute cumulative sum and cumulative product of a column.
40. Drop duplicate rows based on a subset of columns.

41. Extract year, month, day, weekday from a datetime column.
42. Filter rows by date range.
43. Resample time-series data by month or week.
44. Compute percentage change in a numeric column.
45. Create a flag column based on conditions (e.g., bonus if salary > 50000).
46. Apply multiple functions to a column using `agg()`.
47. Sort by index and column values together.
48. Use `query()` to filter rows with a string expression.
49. Sample random rows from a DataFrame.
50. Convert a column to categorical and perform operations.

Advanced / Critical Industry-Level

51. Handle large CSV files efficiently using chunksize.
52. Read multiple CSV files and concatenate into a single DataFrame.
53. Pivot multi-index DataFrame and perform aggregation.
54. Perform cross-tabulation of two categorical columns.
55. Handle missing data using interpolation.
56. Forward-fill and backward-fill missing values.
57. Detect and remove outliers using IQR method.
58. Apply a custom function to multiple columns simultaneously.
59. Vectorize operations on multiple columns for performance.
60. Merge multiple DataFrames iteratively using a loop.
61. Perform groupby-apply operations with a custom function.
62. Detect duplicate rows and retain first/last occurrences.
63. Perform one-hot encoding for categorical columns.
64. Compute weighted average for grouped data.
65. Join DataFrames on multiple keys.
66. Perform time-series analysis using rolling window and expanding window.
67. Reshape a dataset from long to wide using pivot.
68. Create hierarchical indexing (multi-index) and select subsets.
69. Stack and unstack a multi-index DataFrame.
70. Apply lambda functions with multiple arguments across columns.
71. Perform conditional updates for multiple columns using `np.where`.
72. Merge datasets with fuzzy matching (e.g., approximate string match).
73. Optimize memory usage by converting dtypes in large DataFrames.
74. Simulate a small ETL pipeline: load, clean, transform, aggregate.
75. Perform exploratory data analysis (EDA) on a real dataset, including plots with Pandas and Seaborn integration.

UNIT-15

Scikit-learn, Matplotlib, Seaborn

Part A: Scikit-learn (1–100)

Beginner (1–25)

1. Import `scikit-learn` and check its version.
2. Load the `iris` dataset and display its features.
3. Split a dataset into train and test sets.
4. Standardize features using `StandardScaler`.
5. Apply `MinMaxScaler` to scale features between 0–1.
6. Encode categorical variables using `LabelEncoder`.
7. Apply `OneHotEncoder` to a categorical column.
8. Fit a `LinearRegression` model on a dataset.
9. Predict outputs using a trained `LinearRegression` model.
10. Calculate mean squared error (MSE) of predictions.
11. Fit a `LogisticRegression` model for classification.
12. Compute accuracy score for classification predictions.
13. Split data with stratified sampling for classification.
14. Use `KNeighborsClassifier` for classification.
15. Apply `KMeans` clustering on a dataset.
16. Visualize clusters using PCA for dimensionality reduction.
17. Fit a `DecisionTreeClassifier` and plot its tree.
18. Apply `RandomForestClassifier` on a dataset.
19. Check feature importances from a Random Forest model.
20. Fit a `SupportVectorMachine` classifier with linear kernel.
21. Apply `PolynomialFeatures` to transform input features.
22. Fit `Ridge` regression and check coefficients.
23. Fit `Lasso` regression and check coefficients.
24. Use `train_test_split` with different `random_state` values.
25. Handle missing data using `SimpleImputer` and fit a model.

Intermediate / Case-Based (26–50)

26. Apply cross-validation using `cross_val_score`.
27. Tune hyperparameters using `GridSearchCV`.
28. Use `RandomizedSearchCV` for hyperparameter tuning.
29. Fit a `GradientBoostingClassifier` and evaluate accuracy.
30. Fit `AdaBoostClassifier` and compare with Random Forest.
31. Perform PCA to reduce dimensions of the dataset.
32. Visualize explained variance of PCA components.
33. Use `StandardScaler` + `PCA` + classifier in a pipeline.
34. Use `PolynomialFeatures` + `Ridge` in a pipeline.

35. Fit `LogisticRegression` with L1 and L2 regularization.
36. Compute confusion matrix for classification predictions.
37. Plot ROC curve for a binary classifier.
38. Calculate AUC score for ROC curve.
39. Apply `SMOTE` to handle imbalanced dataset.
40. Fit `DecisionTreeRegressor` and compute R² score.
41. Fit `RandomForestRegressor` and compute RMSE.
42. Use `BaggingClassifier` on a small dataset.
43. Use `StackingClassifier` combining multiple models.
44. Implement `Pipeline` to chain preprocessing and model.
45. Apply `StandardScaler` inside a pipeline and fit model.
46. Fit `SGDClassifier` for large datasets.
47. Evaluate model using `cross_val_predict`.
48. Apply `MinMaxScaler` and visualize feature distributions.
49. Fit `ElasticNet` regression and compare with Lasso/Ridge.
50. Split dataset into train/validation/test sets manually.

Advanced / Industry-Level (51–75)

51. Apply feature selection using `SelectKBest`.
52. Apply recursive feature elimination (RFE) with estimator.
53. Apply `VarianceThreshold` to remove low variance features.
54. Handle categorical features with `ColumnTransformer`.
55. Encode categorical variables with `OneHotEncoder` inside pipeline.
56. Perform nested cross-validation.
57. Fit `XGBoostClassifier` and compute feature importance.
58. Apply `LightGBM` for classification task.
59. Perform hyperparameter tuning with Bayesian optimization (skopt).
60. Implement custom scoring function in `cross_val_score`.
61. Apply `StandardScaler` + `PCA` + `LogisticRegression` for pipeline.
62. Fit `KMeans` and compute silhouette score.
63. Use `DBSCAN` clustering on noisy dataset.
64. Fit `IsolationForest` for anomaly detection.
65. Fit `OneClassSVM` for novelty detection.
66. Apply ensemble stacking with multiple classifiers.
67. Fit `HistGradientBoostingClassifier` and evaluate performance.
68. Perform multi-output regression using `MultiOutputRegressor`.
69. Use `GridSearchCV` with multiple scoring metrics.
70. Fit `Pipeline` with imputer, scaler, PCA, and classifier.
71. Handle missing values using `IterativeImputer`.
72. Fit `BaggingRegressor` on a regression dataset.
73. Fit `VotingClassifier` using hard and soft voting.
74. Apply time-series split for cross-validation.
75. Evaluate a regression model using `mean_absolute_percentage_error`.

Critical / Project-Level (76–100)

76. Implement custom transformer for preprocessing in pipeline.
77. Implement custom scoring function for model evaluation.
78. Apply stacking regressor with multiple base models.
79. Apply feature importance analysis on Random Forest and plot top 10 features.
80. Fit `CatBoostClassifier` and handle categorical features automatically.
81. Fit `LGBMRegressor` for regression task and evaluate RMSE.
82. Apply `FeatureUnion` to combine multiple feature sets.
83. Handle high-cardinality categorical variables in a dataset.
84. Apply `Pipeline` with OneHotEncoder and KNNClassifier.
85. Apply PCA on high-dimensional text embeddings (TF-IDF).
86. Fit `LogisticRegression` on imbalanced dataset using class_weight.
87. Apply iterative hyperparameter tuning with `RandomizedSearchCV`.
88. Apply nested cross-validation for unbiased model evaluation.
89. Evaluate models with `precision`, `recall`, and `f1-score`.
90. Fit `SGDRegressor` on large datasets.
91. Apply multi-class ROC curves.
92. Perform clustering evaluation with Davies-Bouldin score.
93. Fit `ExtraTreesClassifier` and extract feature importance.
94. Apply `Pipeline` with scaling, feature selection, and classifier.
95. Fit `MLPClassifier` (neural network) for classification.
96. Apply dimensionality reduction using `TruncatedSVD` for sparse data.
97. Fit `RidgeClassifierCV` and compare with Logistic Regression.
98. Apply out-of-fold predictions for stacking ensemble.
99. Fit multiple regression models and select best using cross-validation.
100. Perform end-to-end ML workflow: load data, preprocess, feature engineer, train model, evaluate.

Part B: Matplotlib (101–150)

Beginner (101–120)

101. Import Matplotlib and plot a simple line graph.
102. Plot multiple lines in a single graph.
103. Label x-axis, y-axis, and add a title.
104. Customize line styles and colors.
105. Plot a scatter plot.
106. Plot a bar chart.
107. Plot a horizontal bar chart.
108. Plot a histogram of data.
109. Customize histogram bins.
110. Add legend to a plot.
111. Save a plot to PNG or PDF.
112. Set x and y limits.
113. Plot multiple subplots using `subplot()`.
114. Use `figure()` to set figure size.

115. Plot error bars on a line chart.
116. Customize marker styles.
117. Add text annotation to a plot.
118. Plot stacked bar chart.
119. Plot pie chart.
120. Add gridlines to a plot.

Intermediate / Case-Based (121–135)

121. Plot time-series data using Matplotlib.
122. Customize ticks and tick labels.
123. Plot multiple lines with different markers.
124. Plot multiple subplots with shared x-axis.
125. Create dual-axis plot (two y-axes).
126. Plot grouped bar chart.
127. Plot cumulative sum line chart.
128. Plot density plot using Matplotlib.
129. Plot boxplot for multiple groups.
130. Customize boxplot appearance.
131. Plot violin plot using Matplotlib patches.
132. Plot horizontal error bars.
133. Customize color maps for scatter plots.
134. Annotate points on scatter plots.
135. Use `tight_layout()` to adjust spacing.

Advanced / Industry-Level (136–150)

136. Plot correlation heatmap using Matplotlib imshow.
137. Plot stacked area chart.
138. Plot polar plots.
139. Plot 3D surface plot using `Axes3D`.
140. Plot 3D scatter plot.
141. Animate a plot using FuncAnimation.
142. Create interactive plots using Matplotlib widgets.
143. Plot multiple histograms in one figure with transparency.
144. Plot logarithmic scale axes.
145. Customize font styles globally.
146. Plot financial candlestick chart.
147. Plot network graph using Matplotlib.
148. Plot geographical data using Matplotlib basemap.
149. Overlay multiple chart types (line + scatter).
150. Visualize regression line with confidence interval.

Part C: Seaborn (151–200)

Beginner (151–170)

151. Import Seaborn and check version.
152. Load `tips` dataset and inspect it.
153. Plot a simple scatterplot using Seaborn.
154. Plot lineplot with confidence interval.
155. Plot histogram using `histplot()`.
156. Plot KDE plot using `kdeplot()`.
157. Plot boxplot for a categorical variable.
158. Plot violin plot.
159. Plot countplot for categorical column.
160. Plot barplot for grouped data.
161. Customize color palette.
162. Add hue to scatterplot.
163. Plot jointplot for two variables.
164. Plot pairplot for entire dataset.
165. Plot heatmap for correlation matrix.
166. Plot categorical stripplot.
167. Plot swarmplot.
168. Plot regression line using `regplot()`.
169. Plot residuals using `residplot()`.
170. Customize figure size and style.

Intermediate / Case-Based (171–185)

171. Plot facet grid for multiple subsets.
172. Plot multiple lineplots in FacetGrid.
173. Plot categorical boxplot with hue.
174. Plot violinplot with split option.
175. Plot scatterplot with multiple markers.
176. Plot jointplot with hex bins.
177. Plot pairplot with different hues.
178. Plot heatmap with annotations.
179. Plot correlation matrix with mask.
180. Plot categorical barplot with confidence interval.
181. Plot lineplot with multiple categories.
182. Plot regression with multiple predictors.
183. Plot residuals for multiple groups.
184. Use Seaborn style context (`darkgrid`, `whitegrid`).
185. Save Seaborn plots to files.

Advanced / Industry-Level (186–200)

186. Plot multi-dimensional data with FacetGrid and hue + style.

187. Overlay KDE plots for multiple categories.
188. Plot violin + swarm plot overlay.
189. Plot heatmap for missing data visualization.
190. Customize Seaborn themes globally.
191. Plot time-series with rolling averages.
192. Plot scatterplot with size encoding for third variable.
193. Plot pairplot with regression line overlay.
194. Plot clustered heatmap using hierarchical clustering.
195. Plot categorical boxplot with split by another variable.
196. Plot multi-panel regression plots for multiple features.
197. Plot jointplot with regression + residuals.
198. Visualize multi-class classification results with Seaborn.
199. Plot model predictions vs actual using scatter and regression line.
200. Combine multiple Seaborn plots for dashboard-style visualization.

UNIT-16 Python Database

Basic Database Operations

1. Write a Python program to connect to a SQLite database and create a new database file.
2. Write a Python program to create a table with columns: `id`, `name`, `age`, `email`.
3. Insert single row data into a table using Python.
4. Insert multiple rows at once into a table using `executemany()`.
5. Write a Python program to **fetch all rows** from a table.
6. Fetch **a single row** based on a condition (e.g., `id=2`).
7. Update a record in a table (e.g., change the `name` for a given `id`).
8. Delete a record from a table using Python.
9. Drop a table if it exists using Python DB API.
10. Count the **total number of records** in a table.

Intermediate Database Operations

11. Select **rows where age > 25** from the table.
12. Sort table records by **name in ascending order**.
13. Fetch records with **LIKE operator** (e.g., name starting with 'A').
14. Fetch the **maximum, minimum, and average age** from the table.
15. Use **parameterized queries** to prevent SQL injection.
16. Write a Python program to **join two tables** and fetch combined data.
17. Fetch **distinct values** from a column.
18. Write a program to **search for a record dynamically** using user input.

Advanced / Case-Based Questions

19. Create a Python program to **implement transactions** (commit and rollback).
20. Fetch **records in chunks** (pagination) using Python.
21. Export table data into a **CSV file** using Python.
22. Import CSV data into a database table using Python.
23. Write a program to **create a view** and fetch data from it.
24. Use Python to handle database exceptions gracefully.
25. Write a program to connect to a MySQL/PostgreSQL database, execute queries, and close the connection properly.

PROJECT Topic 1: Tkinter : TUTORIAL(WITH BASIC APPLICATIONS SOLVED)

PROJECT Topic 2: KIVY TUTORIAL(WITH BASIC APPLICATIONS SOLVED)

DOs and DON'Ts

DOs

1. Conform to the academic discipline of the department.
2. Enter your credentials in the laboratory attendance register.
3. Read and understand how to carry out an activity thoroughly before coming to the laboratory.
4. Ensure the uniqueness with respect to the methodology adopted for carrying out the experiments.
5. Shutdown the machine once you are done using it.

DON'Ts

1. Eatables are not allowed in the laboratory.
2. Usage of mobile phones is strictly prohibited.
3. Do not open the system unit casing.
4. Do not remove anything from the computer laboratory without permission.
5. Do not touch, connect or disconnect any plug or cable without your faculty/laboratory technician'spermission.

General Safety Precautions

1. Electrical Safety

- Always check that power cords and plugs are in good condition—no frays or exposed wires.
- Avoid overloading power outlets or using multiple extension cords.
- Do not touch electrical equipment with wet hands.
- Turn off computers and peripherals before cleaning or servicing them.

---

2. Ergonomic Safety

- Adjust the chair and monitor height to maintain a comfortable posture.
- Keep feet flat on the floor and maintain a straight back while sitting.
- Position the monitor at eye level to avoid neck strain.
- Take short breaks every 30–45 minutes to reduce eye strain and muscle fatigue.

---

3. Fire Safety

- Keep flammable materials away from computers and electrical outlets.
- Know the location of fire extinguishers and emergency exits.
- Do not attempt to fix electrical fires on your own—inform the lab supervisor immediately.

---

4. Equipment Handling

- Handle computers, keyboards, and peripherals gently.
- Avoid eating or drinking near computers to prevent spills and damage.
- Do not attempt to open or repair hardware unless trained.
- Keep cables organized to prevent tripping hazards.

---

5. Network and Data Safety

- Do not install unauthorized software or hardware.
- Follow proper login and password protocols.
- Save your work frequently to avoid data loss.
- Avoid visiting suspicious websites or downloading unsafe files.

---

6. Personal Conduct

- Maintain silence or low noise levels to avoid disturbing others.
- Report any malfunctioning equipment immediately.

- Follow the lab rules and instructions of the lab supervisor.
- Avoid running or horseplay in the lab to prevent accidents.

7. Hygiene and Health

- Keep the workspace clean and organized.
- Wipe keyboards, mouse, and screens regularly.
- Wash hands after using shared equipment to prevent germs.
- Ensure proper ventilation in the lab to avoid overheating.

Emergency Contact :

Security Contact :

Guidelines to students for report preparation

All students are required to maintain a record of the experiments conducted by them. Guidelines for its preparation are as follows: -

1) All files must contain a title page followed by an index page. ***The files will not be signed by the faculty without an entry in the index page.***
2) Student's Name, roll number and date of conduction of experiment must be written on all pages.
3) For each experiment, the record must contain the following:
   (i) Aim/Objective of the experiment
   (ii) Pre-experiment work (as given by the faculty, if any)
   (iii) Lab assignment questions and their solutions
   (iv) Test Cases (if applicable to the course)
   (v) Results/ output

Note:

1. Students must bring their lab record along with them whenever they come for the lab.
2. Students must ensure that their lab record is regularly evaluated.

Lab Assessment Criteria

An estimated 10 lab classes are conducted in a semester for each lab course. These lab classes are assessed continuously. Each lab experiment is evaluated based on 5 assessment criteria as shown in following table. Assessed performance in each experiment is used to compute CO attainment as well as internal marks in the lab course.

Grading Criteria	**Exemplary (4)**	**Competent (3)**	**Needs Improvement (2)**	**Poor (1)**
AC1: Pre-Lab written work (this may be assessed through viva)	Complete procedure with underlined concept is properly written	Underlined concept is written but procedure is incomplete	Not able to write concept and procedure	Underlined concept is not clearly understood
AC2: Program Writing/ Modeling	Unable to understand the reason for errors/ bugs even after they are explicitly pointed out	Assigned problem is properly analyzed, correct solution designed, appropriate language constructs/ tools are applied	Assigned problem is properly analyzed & correct solution designed	Assigned problem is properly analyzed
AC3: Identification &	Able to identify errors/ bugs	Able to identify errors/	Is dependent totally on	Unable to understand the

Removal of errors/ bugs	and remove them	bugs and remove them with little bit of guidance	someone for identification of errors/ bugs and their removal	reason for errors/ bugs even after they are explicitly pointed out
<u>AC4:</u> Execution & Demonstration	All variants of input /output are tested, Solution is well demonstrated and implemented concept is clearly explained	All variants of input /output are not tested, However, solution is well demonstrated and implemented concept is clearly explained	Only few variants of input /output are tested, Solution is well demonstrated but implemented concept is not clearly explained	Solution is not well demonstrated and implemented concept is not clearly explained
<u>AC5:</u> Lab Record Assessment	All assigned problems are well recorded with objective, design constructs and solution along with Performance analysis using all variants of input and output	More than 70 % of the assigned problems are well recorded with objective, design contracts and solution along with Performance analysis is done with all variants of input and output	Less than 70 % of the assigned problems are well recorded with objective, design contracts and solution along with Performance analysis is done with all variants of input and output	

LAB EXPERIMENTS SOLUTION

UNIT-1 Solution

1. Check Even / Odd

```
import java.util.*;

class EvenOdd {
    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        System.out.print("Enter a number: ");
        int n = sc.nextInt();

        if (n % 2 == 0)
            System.out.println(n + " is Even");
        else
            System.out.println(n + " is Odd");
    }
}
```

Output Example:

```
Enter a number: 7
7 is Odd
```

---

2. Swap Two Numbers (without temporary variable)

```
class SwapNumbers {
    public static void main(String[] args) {
        int a = 10, b = 20;
        System.out.println("Before Swap: a = " + a + ", b = " + b);

        a = a + b;  // a = 30
        b = a - b;  // b = 10
        a = a - b;  // a = 20

        System.out.println("After Swap: a = " + a + ", b = " + b);
    }
}
```

Output Example:

```
Before Swap: a = 10, b = 20
After Swap: a = 20, b = 10
```

---

3. Simple Interest & Compound Interest

```
import java.util.*;

class InterestCalc {
    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        System.out.print("Enter Principal: ");
        double p = sc.nextDouble();
        System.out.print("Enter Rate of Interest: ");
        double r = sc.nextDouble();
        System.out.print("Enter Time (in years): ");
        double t = sc.nextDouble();

        double si = (p * r * t) / 100;
        double ci = p * Math.pow((1 + r / 100), t) - p;
        System.out.println("Simple Interest = " + si);
        System.out.println("Compound Interest = " + ci);
    }
}
```

Output Example:

```
Enter Principal: 10000
Enter Rate of Interest: 5
Enter Time (in years): 2
Simple Interest = 1000.0
Compound Interest = 1025.0
```

4. Prime Number Check

```
import java.util.*;

class PrimeCheck {
    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        System.out.print("Enter a number: ");
        int n = sc.nextInt();

        boolean isPrime = true;
        if (n <= 1)
            isPrime = false;
        else {
            for (int i = 2; i <= Math.sqrt(n); i++) {
                if (n % i == 0) {
                    isPrime = false;
                    break;
                }
            }
        }

        if (isPrime)
            System.out.println(n + " is a Prime number");
        else
            System.out.println(n + " is not a Prime number");
    }
}
```

Output Example:

```
Enter a number: 17
17 is a Prime number
```

5. List All Primes in Range

```
class PrimesInRange {
    public static void main(String[] args) {
        int start = 10, end = 50;
        for (int n = start; n <= end; n++) {
            boolean prime = true;
            if (n < 2) prime = false;
            for (int i = 2; i * i <= n; i++)
                if (n % i == 0) prime = false;
            if (prime) System.out.print(n + " ");
        }
    }
}
```

6. Fibonacci (Iterative)

```
class Fibonacci {
    public static void main(String[] args) {
        int n = 10, a = 0, b = 1;
        System.out.print(a + " " + b + " ");
        for (int i = 2; i < n; i++) {
            int c = a + b;
            System.out.print(c + " ");
            a = b; b = c;
        }
    }
}
```

7. Factorial (Recursion)

```
class Factorial {
    static long fact(int n) {
        return (n <= 1) ? 1 : n * fact(n - 1);
    }
    public static void main(String[] args) {
        System.out.println("Factorial of 5 = " + fact(5));
    }
}
```

8. GCD & LCM

```
class GcdLcm {
    static int gcd(int a, int b) {
        return b == 0 ? a : gcd(b, a % b);
    }
    public static void main(String[] args) {
        int a = 12, b = 18;
        int g = gcd(a, b);
        int l = (a * b) / g;
        System.out.println("GCD=" + g + ", LCM=" + l);
    }
}
```

9. Palindrome (Number)

```
class Palindrome {
    public static void main(String[] args) {
        int n = 121, temp = n, rev = 0;
        while (n > 0) {
            rev = rev * 10 + n % 10;
            n /= 10;
        }
        System.out.println(temp == rev ? "Palindrome" : "Not Palindrome");
    }
}
```

10. Armstrong Numbers in Range

```
class ArmstrongRange {
    public static void main(String[] args) {
        for (int n = 100; n <= 999; n++) {
            int sum = 0, temp = n;
            while (temp > 0) {
                int d = temp % 10;
                sum += d * d * d;
                temp /= 10;
            }
            if (sum == n) System.out.println(n);
        }
    }
}
```

11. Strong Number

```
class StrongNumber {
    static int fact(int n){ return n <= 1 ? 1 : n * fact(n - 1); }
    public static void main(String[] args){
        int n = 145, sum = 0, temp = n;
        while(n > 0){
            sum += fact(n % 10);
            n /= 10;
        }
        System.out.println(temp == sum ? "Strong Number" : "Not Strong");
    }
}
```

12. Perfect Number

```
class PerfectNumber {
    public static void main(String[] args) {
        int n = 28, sum = 0;
        for (int i = 1; i <= n / 2; i++)
            if (n % i == 0) sum += i;
        System.out.println(n == sum ? "Perfect Number" : "Not Perfect");
    }
}
```

13. Number Base Conversion

```
class Conversion {
    public static void main(String[] args) {
        int n = 25;
        System.out.println("Binary: " + Integer.toBinaryString(n));
        System.out.println("Octal: " + Integer.toOctalString(n));
        System.out.println("Hex: " + Integer.toHexString(n));
    }
}
```

Matrix Operations

14. Matrix Multiplication

```
class MatrixMultiplication {
    public static void main(String[] args) {
        int[][] a = {{1,2,3},{4,5,6}};
        int[][] b = {{7,8},{9,10},{11,12}};
        int[][] c = new int[2][2];
        for (int i=0;i<2;i++)
            for (int j=0;j<2;j++)
                for (int k=0;k<3;k++)
                    c[i][j]+=a[i][k]*b[k][j];
        for (int[] row : c) {
            for (int val : row) System.out.print(val+" ");
            System.out.println();
        }
    }
}
```

15. Transpose of Matrix

```
class MatrixTranspose {
    public static void main(String[] args) {
        int[][] a = {{1,2,3},{4,5,6}};
        int[][] t = new int[3][2];
        for(int i=0;i<2;i++)
            for(int j=0;j<3;j++)
                t[j][i]=a[i][j];
        for(int[] row:t){
            for(int val:row) System.out.print(val+" ");
            System.out.println();
        }
    }
}
```

Patterns & Combinatorics

16. Pascal’s Triangle

```
class PascalsTriangle {
    public static void main(String[] args) {
        int n = 5;
        for (int i=0;i<n;i++) {
            int num = 1;
            for (int j=0;j<=i;j++) {
                System.out.print(num + " ");
                num = num * (i - j) / (j + 1);
            }
            System.out.println();
        }
    }
}
```

Number/String Properties

17. Combined Check (Armstrong, Automorphic, Palindrome)

```
class NumberChecks {
    static boolean isArmstrong(int n){
        int sum=0,temp=n,d; int len=(n+"").length();
        while(n>0){d=n%10; sum+=Math.pow(d,len); n/=10;}
        return sum==temp;
    }
    static boolean isAutomorphic(int n){
        int sq=n*n;
        return (sq+"").endsWith(n+"");
    }
    static boolean isPalindrome(int n){
        String s=n+"";
        return new StringBuilder(s).reverse().toString().equals(s);
    }
    public static void main(String[] args){
        int n=121;
        System.out.println("Armstrong: "+isArmstrong(n));
        System.out.println("Automorphic: "+isAutomorphic(n));
        System.out.println("Palindrome: "+isPalindrome(n));
    }
}
```

Digit Operations

18. Sum of Digits, Reverse, Count

```
class DigitOps {
    public static void main(String[] args){
        int n=12345, sum=0, rev=0, count=0;
        int temp=n;
        while(n>0){
            int d=n%10;
            sum+=d; rev=rev*10+d; count++;
            n/=10;
        }
        System.out.println("Sum="+sum+", Reverse="+rev+", Digits="+count);
    }
}
```

Bitwise & Math

19. Count Set Bits

```
class CountBits {
    public static void main(String[] args){
        int n=29;
        System.out.println("Set bits: "+Integer.bitCount(n));
    }
}
```

20. Check Power of Two & Next Power

```
class PowerOfTwo {
    static boolean isPowerOfTwo(int n){ return (n>0)&&(n&(n-1))==0; }
    static int nextPowerOfTwo(int n){
        int p=1;
        while(p<n) p<<=1;
        return p;
    }
    public static void main(String[] args){
        int n=20;
        System.out.println("Is Power of 2: "+isPowerOfTwo(n));
        System.out.println("Next Power of 2: "+nextPowerOfTwo(n));
    }
}
```

Strings & Anagrams

21. Check Anagram

```
import java.util.*;
class AnagramCheck {
    public static void main(String[] args){
        String a="listen", b="silent";
        char[] x=a.toCharArray(), y=b.toCharArray();
        Arrays.sort(x); Arrays.sort(y);
        System.out.println(Arrays.equals(x,y) ? "Anagram" : "Not
Anagram");
    }
}
```

22. Most Frequent K Elements

```
import java.util.*;
class MostFrequent {
    public static void main(String[] args){
        String s="banana";
        Map<Character,Integer> map=new HashMap<>();
        for(char c:s.toCharArray())
            map.put(c,map.getOrDefault(c,0)+1);
        map.entrySet().stream()
            .sorted((a,b)->b.getValue()-a.getValue())
            .forEach(e->System.out.println(e.getKey()+"="+e.getValue()));
    }
}
```

Advanced Math

23. Evaluate Infix Expression (using Stack)

```
import java.util.*;
class InfixEvaluation {
    static int precedence(char c){
        if(c=='+'||c=='-') return 1;
        if(c=='*'||c=='/') return 2;
        return -1;
    }
    static int apply(int a,int b,char op){
        switch(op){
            case '+': return a+b;
            case '-': return a-b;
            case '*': return a*b;
            case '/': return a/b;
        }
        return 0;
    }
    public static void main(String[] args){
        String exp="3+(2*5)-1";
        Stack<Integer> val=new Stack<>();
        Stack<Character> ops=new Stack<>();
        for(char c:exp.toCharArray()){
            if(Character.isDigit(c)) val.push(c-'0');
            else if(c=='(') ops.push(c);
            else if(c==')'){
                while(ops.peek()!='(')
                    val.push(apply(val.remove(val.size()-2), val.pop(),
ops.pop()));
                ops.pop();
            } else {
                while(!ops.isEmpty() &&
precedence(ops.peek())>=precedence(c))
                    val.push(apply(val.remove(val.size()-2), val.pop(),
ops.pop()));
                ops.push(c);
            }
        }
        while(!ops.isEmpty())
            val.push(apply(val.remove(val.size()-2), val.pop(),
ops.pop()));
        System.out.println("Result: "+val.pop());
    }
}
```

24. Solve Quadratic Equation

```
class Quadratic {
    public static void main(String[] args){
        double a=1,b=-3,c=2;
        double d=b*b-4*a*c;
        if(d>0){
            double r1=(-b+Math.sqrt(d))/(2*a);
            double r2=(-b-Math.sqrt(d))/(2*a);
            System.out.println("Roots: "+r1+", "+r2);
        } else if(d==0){
            System.out.println("Equal roots: "+(-b/(2*a)));
        } else {
            double real=-b/(2*a);
            double imag=Math.sqrt(-d)/(2*a);
            System.out.println("Complex roots: "+real+" ± "+imag+"i");
        }
    }
}
```

PATTERN 1 — Right Triangle Star

```
*
* *
* * *
* * * *
* * * * *
```

Logic:

- Use two loops:
  - Outer loop → for each row.
  - Inner loop → print stars equal to current row number.

```
n = 5
for row in range(1, n + 1):
    print("* " * row)
```

Simple, readable, uses Python string repetition.

PATTERN 2 — Inverted Right Triangle

```
* * * * *
* * * *
* * *
* *
*
```

Logic:

- Outer loop runs backward.
- In each iteration, print fewer stars.

```
n = 5
for row in range(n, 0, -1):
    print("* " * row)
```

Avoids nested loops for simplicity.

PATTERN 3 — Pyramid

```
    *
   * *
  * * *
 * * * *
* * * * *
```

Logic:

- Print spaces then stars.
- Number of spaces = (n - row)
- Number of stars = row

```
n = 5
```

```
for row in range(1, n + 1):
    print(" " * (n - row) + "* " * row)
```

Uses spacing calculation instead of nested loops.

PATTERN 4 — Inverted Pyramid

```
* * * * *
 * * * *
  * * *
   * *
    *
```

Logic:

- Start with full stars and reduce each row.
- Increase spaces each row.

```
n = 5
for row in range(n, 0, -1):
    print(" " * (n - row) + "* " * row)
```

Space increases while star count decreases.

PATTERN 5 — Diamond Pattern

```
    *
   * *
  * * *
 * * * *
* * * * *
 * * * *
  * * *
   * *
    *
```

Logic:

- Combine pyramid + inverted pyramid.
- First half → increasing stars.
- Second half → decreasing stars.

```
n = 5
for row in range(1, n + 1):
    print(" " * (n - row) + "* " * row)
for row in range(n - 1, 0, -1):
    print(" " * (n - row) + "* " * row)
```

Reuses pyramid logic for clarity.

PATTERN 6 — Square Pattern

```
* * * * *
* * * * *
* * * * *
* * * * *
* * * * *
```

Logic:

- Each row prints same number of stars.

```
n = 5
for _ in range(n):
    print("* " * n)
```

_ used as throwaway variable.

PATTERN 7 — Hollow Square

```
* * * * *
*       *
*       *
*       *
* * * * *
```

Logic:

- Print full stars on first & last rows.
- Else → star, spaces, star.

```
n = 5
for row in range(1, n + 1):
    if row == 1 or row == n:
        print("* " * n)
    else:
        print("* " + "  " * (n - 2) + "*")
```

Efficient conditional control.

PATTERN 8 — Number Triangle

```
1
1 2
1 2 3
1 2 3 4
```

Logic:

- Each row prints numbers from 1 to row number.

```
n = 4
for row in range(1, n + 1):
    print(" ".join(str(col) for col in range(1, row + 1)))
```

Uses `join()` → no trailing spaces, faster and clean.

PATTERN 9 — Floyd's Triangle

```
1
2 3
4 5 6
7 8 9 10
```

Logic:

- Keep a counter variable that increments every time you print a number.

```
n = 4
num = 1
for row in range(1, n + 1):
    print(" ".join(str(num + i) for i in range(row)))
    num += row
```

Efficient numeric control with `join()`.

PATTERN 10 — Inverted Number Triangle

```
1 2 3 4 5
1 2 3 4
1 2 3
1 2
1
```

Logic:

- Decrease upper limit each row.

```
n = 5
for row in range(n, 0, -1):
    print(" ".join(str(col) for col in range(1, row + 1)))
```

Clean loop with reversed control.

PATTERN 11 — Repeated Number Triangle

```
1
2 2
3 3 3
4 4 4 4
```

Logic:

- Each row prints its own row number multiple times.

```
n = 4
for row in range(1, n + 1):
    print(" ".join(str(row) for _ in range(row)))
```

No nested printing logic — clean and concise.

PATTERN 12 — Alphabet Triangle

```
A
A B
A B C
A B C D
```

Logic:

- Print characters from 'A' to current row letter.
- Use ASCII value (`chr(65)` = 'A').

```
n = 4
for row in range(1, n + 1):
    print(" ".join(chr(65 + col) for col in range(row)))
```

Direct ASCII manipulation keeps logic simple.

PATTERN 13 — Inverted Alphabet Triangle

```
A B C D
A B C
A B
A
```

Logic:

- Reduce number of letters each row.

```
n = 4
for row in range(n, 0, -1):
    print(" ".join(chr(65 + col) for col in range(row)))
```

Just reverse the outer loop.

PATTERN 14 — Hollow Pyramid

```
    *
   * *
  *   *
 *     *
* * * * *
```

Logic:

- First & last rows full of stars.
- Middle rows → star + spaces + star.

```
n = 5
for row in range(1, n + 1):
    if row == 1:
        print(" " * (n - 1) + "*")
    elif row == n:
        print("* " * n)
    else:
        print(" " * (n - row) + "*" + " " * (2 * row - 3) + "*")
```

Handles spacing and borders efficiently.

PATTERN 15 — Hollow Diamond

```
    *
   * *
  *   *
 *     *
*       *
 *     *
  *   *
   * *
    *
```

Logic:

- Combine hollow pyramid + inverted hollow pyramid.

```
n = 5
for row in range(1, n + 1):
    print(" " * (n - row) + "*" + (" " * (2 * row - 3) + "*" if row > 1 else ""))
for row in range(n - 1, 0, -1):
    print(" " * (n - row) + "*" + (" " * (2 * row - 3) + "*" if row > 1 else ""))
```

Compact two-phase logic with ternary for clarity.

PATTERN 16 — Hourglass Pattern

```
* * * * *
 * * * *
  * * *
 * * * *
* * * * *
```

Logic:

- Two mirrored pyramids joined together.

```
n = 5
for row in range(n, 0, -1):
    print(" " * (n - row) + "* " * row)
for row in range(2, n + 1):
    print(" " * (n - row) + "* " * row)
```

Reuses pyramid structure effectively.

PATTERN 17 — Mirrored Right Triangle

```
        *
```

```
      * *
    * * *
  * * * *
* * * * *
```

Logic:

- Print leading spaces before each row.

```
n = 5
for row in range(1, n + 1):
    print("  " * (n - row) + "* " * row)
```

Minimalist and symmetric.

PATTERN 18 — Number Pyramid

```
    1
   2 2
  3 3 3
 4 4 4 4
5 5 5 5 5
```

Logic:

- Space before row, repeated number pattern.

```
n = 5
for row in range(1, n + 1):
    print(" " * (n - row) + (" ".join(str(row) for _ in range(row))))
```

Uses `join()` for spacing precision.

PATTERN 19 — Reverse Number Pyramid

```
5 5 5 5 5
 4 4 4 4
  3 3 3
   2 2
    1
```

Logic:

- Space increases; repeated number decreases.

```
n = 5
for row in range(n, 0, -1):
    print(" " * (n - row) + " ".join(str(row) for _ in range(row)))
```

Simple reverse of pattern 18.

PATTERN 20 — Palindrome Number Pyramid

```
    1
   121
  12321
 1234321
123454321
```

Logic:

- Left ascending numbers, right descending.
- Space before each row.

```
n = 5
for row in range(1, n + 1):
    left = "".join(str(i) for i in range(1, row + 1))
    right = "".join(str(i) for i in range(row - 1, 0, -1))
    print(" " * (n - row) + left + right)
```

Split left-right halves for readability.

PATTERN 21 — Continuous Number Pyramid

```
1
2 3
4 5 6
7 8 9 10
```

Logic:

- Maintain a counter variable across rows.

```
n = 4
num = 1
for row in range(1, n + 1):
    current_row = []
    for _ in range(row):
        current_row.append(str(num))
        num += 1
    print(" ".join(current_row))
```

Clean and flexible — easily scalable.

PATTERN 22 — Binary Triangle

```
1
0 1
1 0 1
0 1 0 1
```

Logic:

- Print `(i + j) % 2` for alternating pattern.

```
n = 4
```

```
for i in range(1, n + 1):
    print(" ".join(str((i + j) % 2) for j in range(1, i + 1)))
```

Elegant one-line logic.

PATTERN 23 — Sandglass Number Pattern

```
12345
 1234
  123
   12
    1
   12
  123
 1234
12345
```

Logic:

- Upper decreasing half + lower increasing half.

```
n = 5
for row in range(n, 0, -1):
    print(" " * (n - row) + "".join(str(i) for i in range(1, row + 1)))
for row in range(2, n + 1):
    print(" " * (n - row) + "".join(str(i) for i in range(1, row + 1)))
```

Balanced symmetry with dual loops.

PATTERN 24 — Cross "X" Pattern

```
*       *
 *     *
  *   *
   * *
    *
   * *
  *   *
 *     *
*       *
```

Logic:

- Print star at both diagonal positions.

```
n = 5
for i in range(1, n + 1):
    print(" " * (i - 1) + "*" + " " * (2 * (n - i) - 1) + ("*" if i != n else ""))
for i in range(n - 1, 0, -1):
    print(" " * (i - 1) + "*" + " " * (2 * (n - i) - 1) + "*")
```

Clear geometric relation between `i` and spaces.

PATTERN 25 — Butterfly Pattern

```
*        *
**      **
***    ***
****  ****
**********
****  ****
***    ***
**      **
*        *
```

Logic:

- Two mirrored right triangles separated by spaces.
- First half increasing, second half decreasing.

```
n = 5
for i in range(1, n + 1):
    print("*" * i + " " * (2 * (n - i)) + "*" * i)
for i in range(n - 1, 0, -1):
    print("*" * i + " " * (2 * (n - i)) + "*" * i)
```

Purely arithmetic — easy and efficient.

UNIT-2- STRING SOLUTION

Print length of a string

```
s = "Hello, World!"
print(len(s))
```

2. Reverse a string

```
s = "Hello"
print(s[::-1])
```

3. Check if a string is palindrome

```
s = "madam"
if s == s[::-1]:
    print("Palindrome")
else:
    print("Not Palindrome")
```

4. Count vowels and consonants

```
s = "Hello World"
vowels = consonants = 0
for ch in s.lower():
    if ch.isalpha():
        if ch in "aeiou":
            vowels += 1
        else:
            consonants += 1
print("Vowels:", vowels)
print("Consonants:", consonants)
```

5. Count uppercase and lowercase letters

```
s = "Hello World"
upper = sum(1 for c in s if c.isupper())
lower = sum(1 for c in s if c.islower())
print("Uppercase:", upper)
print("Lowercase:", lower)
```

6. Count digits and special characterss = "Hello123!@#"

```
digits = sum(1 for c in s if c.isdigit())
special = sum(1 for c in s if not c.isalnum())
print("Digits:", digits)
print("Special characters:", special)
```

7. Convert string to uppercase and lowercase

```
s = "Hello World"
print(s.upper())
print(s.lower())
```

8. Toggle case of each character

```
s = "Hello World"
toggled = "".join(c.upper() if c.islower() else c.lower() for c in s)
print(toggled)
```

9. Check if string is numeric

```
s = "12345"
print(s.isdigit())
```

10. Remove whitespaces from string

```
s = " H e l l o "
print(s.replace(" ", ""))
```

11. Find first occurrence of a character

```
s = "hello"
char = 'l'
print(s.find(char))
```

12. Find last occurrence of a characters = "hello"

```
char = 'l'
print(s.rfind(char))
```

13. Count occurrences of a character

```
s = "hello"
char = 'l'
print(s.count(char))
```

14. Concatenate two strings

```
s1 = "Hello"
s2 = "World"
print(s1 + " " + s2)
```

15. Compare two strings

s1 = "apple"

```
s2 = "banana"
if s1 == s2:
    print("Equal")
elif s1 > s2:
    print("s1 is greater")
else:
    print("s2 is greater")
```

16. Check if a substring exists

```
s = "Hello World"
sub = "World"
print(sub in s)
```

17. Replace a substring with another

```
s = "Hello World"
print(s.replace("World", "Python"))
```

18. Split string into words

```
s = "Hello World from Python"
words = s.split()
print(words)
```

19. Join list of strings into a single string

```
words = ["Hello", "World", "Python"]
print(" ".join(words))
```

20. Check if string starts with a substring
```
s = "Hello World"
print(s.startswith("Hello"))
```

21. Check if string ends with a substring
```
s = "Hello World"
print(s.endswith("World"))
```

22. Remove punctuation from a string
```
import string
s = "Hello, World!"
s = "".join(c for c in s if c not in string.punctuation)
print(s)
```

23. Reverse words in a sentence
```
s = "Hello World"
reversed_words = " ".join(word[::-1] for word in s.split())
print(reversed_words)
```

24. Check if string contains only alphabets
```
s = "Hello"
print(s.isalpha())
```

25. Find largest and smallest character in string
```
s = "python"
print("Largest:", max(s))
print("Smallest:", min(s))
```

Remove duplicate characters
```
s = "programming"
result = "".join(sorted(set(s), key=s.index))
print(result)
```

27. Count frequency of each character
```
s = "programming"
freq = {}
for c in s:
    freq[c] = freq.get(c, 0) + 1
print(freq)
```

28. Find all unique characters
```
s = "programming"
unique = [c for c in s if s.count(c) == 1]
print("".join(unique))
```

29. Check if two strings are anagrams
```
s1 = "listen"
s2 = "silent"
print(sorted(s1) == sorted(s2))
```

30. Check if two strings are isomorphic
```
def is_isomorphic(s1, s2):
    if len(s1) != len(s2):
        return False
    mapping = {}
    used = set()
    for c1, c2 in zip(s1, s2):
        if c1 in mapping:
            if mapping[c1] != c2:
                return False
        else:
            if c2 in used:
                return False
            mapping[c1] = c2
            used.add(c2)
    return True

print(is_isomorphic("egg", "add"))  # True
print(is_isomorphic("foo", "bar"))  # False
```

31. Find all substrings of a string

```
s = "abc"
substrings = [s[i:j] for i in range(len(s)) for j in range(i+1, len(s)+1)]
print(substrings)
```

32. Count number of words in a string

```
s = "Hello World from Python"
words = s.split()
print(len(words))
```

33. Count number of sentences in text

```
text = "Hello. How are you? I am fine!"
sentences = [c for c in text if c in ".!?"]
print(len(sentences))
```

34. Remove all vowels from a string

```
s = "Hello World"
result = "".join(c for c in s if c.lower() not in "aeiou")
print(result)
```

35. Remove all consonants from a string

```
s = "Hello World"
result = "".join(c for c in s if c.lower() in "aeiou")
print(result)
```

36. Reverse words individually in a sentence

```
s = "Hello World"
result = " ".join(word[::-1] for word in s.split())
print(result)
```

37. Reverse sentence word order

```
s = "Hello World from Python"
words = s.split()
print(" ".join(words[::-1]))
```

38. Find longest word in a string

```
s = "Find the longest word in this sentence"
words = s.split()
longest = max(words, key=len)
print(longest)
```

39. Find shortest word in a string

```
words = s.split()
shortest = min(words, key=len)
print(shortest)
```

40. Check if string is pangram

```
import string
s = "The quick brown fox jumps over the lazy dog"
alphabet = set(string.ascii_lowercase)
print(set(s.lower()) >= alphabet)
```

41. Find first non-repeating character

```
s = "swiss"
for c in s:
    if s.count(c) == 1:
        print(c)
        break
```

42. Find first repeating character

```
s = "swiss"
seen = set()
for c in s:
    if c in seen:
        print(c)
        break
    seen.add(c)
```

43. Left rotate a string by d positions

```
s = "abcdef"
d = 2
rotated = s[d:] + s[:d]
print(rotated)
```

44. Right rotate a string by d positions

```
s = "abcdef"
d = 2
rotated = s[-d:] + s[:-d]
print(rotated)
```

45. Check if two strings are rotations of each other

```
s1 = "abcd"
s2 = "cdab"
print(len(s1) == len(s2) and s2 in s1*2)
```

46. Count number of substrings containing only vowels

```
s = "aeiouxyz"
count = sum(1 for i in range(len(s)) for j in range(i+1, len(s)+1)
 if all(c in "aeiou" for c in s[i:j]))
print(count)
```

47. Count number of substrings containing only consonants

```
s = "aeiouxyz"
count = sum(1 for i in range(len(s)) for j in range(i+1, len(s)+1)
 if all(c not in "aeiou" for c in s[i:j]))
```

```
print(count)
```

48. Print all palindromic substrings
```
s = "ababa"
palindromes = [s[i:j] for i in range(len(s)) for j in range(i+1, len(s)+1)
if s[i:j] == s[i:j][::-1]]
print(palindromes)
```

49. Remove duplicate words from sentence
```
s = "this is is a test test"
words = s.split()
result = " ".join(sorted(set(words), key=words.index))
print(result)
```

50. Count number of times a word occurs
```
s = "this is a test this is only a test"
word = "test"
words = s.split()
print(words.count(word))
```

51.Longest common prefix among words
```
def longest_common_prefix(words):
    if not words:
        return ""
    prefix = words[0]
    for word in words[1:]:
        while not word.startswith(prefix):
            prefix = prefix[:-1]
            if not prefix:
                return ""
    return prefix

words = ["flower","flow","flight"]
print(longest_common_prefix(words))
```

52. Longest palindromic substring
```
def longest_palindrome(s):
    n = len(s)
    longest = ""
    for i in range(n):
        for j in range(i+1, n+1):
            substr = s[i:j]
            if substr == substr[::-1] and len(substr) > len(longest):
                longest = substr
    return longest

print(longest_palindrome("babad"))
```

53. Minimum window substring containing all characters of another string
```
from collections import Counter

def min_window(s, t):
    if not s or not t:
        return ""
    dict_t = Counter(t)
    required = len(dict_t)
    l = r = formed = 0
    window_counts = {}
    ans = float("inf"), None, None
    while r < len(s):
        c = s[r]
        window_counts[c] = window_counts.get(c,0)+1
        if c in dict_t and window_counts[c] == dict_t[c]:
            formed += 1
        while l <= r and formed == required:
            c = s[l]
            if r-l+1 < ans[0]:
                ans = (r-l+1, l, r)
            window_counts[c] -= 1
            if c in dict_t and window_counts[c] < dict_t[c]:
                formed -= 1
            l +=1
        r +=1
    return "" if ans[0] == float("inf") else s[ans[1]:ans[2]+1]

print(min_window("ADOBECODEBANC", "ABC"))
```

54. Count number of anagram pairs in string

```
from collections import Counter
s = "abba"
sub_count = Counter()
for i in range(len(s)):
    for j in range(i+1, len(s)+1):
        sub = ''.join(sorted(s[i:j]))
        sub_count[sub] += 1
pairs = sum(v*(v-1)//2 for v in sub_count.values())
print(pairs)
```

55. Group anagrams together from list

```
from collections import defaultdict
words = ["eat","tea","tan","ate","nat","bat"]
anagrams = defaultdict(list)
for word in words:
    anagrams[tuple(sorted(word))].append(word)
print(list(anagrams.values()))
```

56. Find all permutations of a string

```
from itertools import permutations
s = "abc"
perms = [''.join(p) for p in permutations(s)]
print(perms)
```

57. Generate all subsequences of a string

```
def subsequences(s):
    res = []
    def dfs(index, path):
        if index == len(s):
            res.append(path)
            return
        dfs(index+1, path)
        dfs(index+1, path + s[index])
    dfs(0, "")
    return res

print(subsequences("abc"))
```

58. Longest substring without repeating characters

```
s = "abcabcbb"
start = max_len = 0
seen = {}
for i, c in enumerate(s):
    if c in seen and seen[c] >= start:
        start = seen[c] + 1
    seen[c] = i
    max_len = max(max_len, i - start + 1)
print(max_len)
```

59. Longest substring with at most k distinct characters

```
s = "eceba"
k = 2
start = 0
max_len = 0
count = {}
for end, c in enumerate(s):
    count[c] = count.get(c, 0)+1
    while len(count) > k:
        count[s[start]] -=1
        if count[s[start]] ==0:
            del count[s[start]]
        start +=1
    max_len = max(max_len, end-start+1)
print(max_len)
```

60. Check if string can be rearranged to form palindrome

```
from collections import Counter
s = "carrace"
counts = Counter(s)
odd_count = sum(1 for v in counts.values() if v % 2)
print(odd_count <= 1)
```

61. Count number of distinct substrings

```
s = "abc"
substrings = set()
for i in range(len(s)):
    for j in range(i+1, len(s)+1):
        substrings.add(s[i:j])
print(len(substrings))
```

62. Count number of distinct palindromic substrings

```
def distinct_palindromes(s):
    pal_set = set()
    for i in range(len(s)):
        # odd length
        l, r = i, i
        while l>=0 and r<len(s) and s[l]==s[r]:
            pal_set.add(s[l:r+1])
            l-=1; r+=1
        # even length
        l, r = i, i+1
        while l>=0 and r<len(s) and s[l]==s[r]:
            pal_set.add(s[l:r+1])
            l-=1; r+=1
    return pal_set

s = "ababa"
```

```
print(distinct_palindromes(s))
```

63. Find lexicographically smallest rotation

```
s = "baca"
rotations = [s[i:] + s[:i] for i in range(len(s))]
print(min(rotations))
```

64. Check if string matches a pattern (simple regex)

```
import re
s = "abc123"
pattern = r"^[a-z]+[0-9]+$"
print(bool(re.match(pattern, s)))
```

65. Implement strstr / indexOf functionality

```
def strstr(haystack, needle):
    for i in range(len(haystack)-len(needle)+1):
        if haystack[i:i+len(needle)] == needle:
            return i
    return -1

print(strstr("hello", "ll"))
```

66. Compress string ("aaabbc" → "a3b2c1")

```
from itertools import groupby
s = "aaabbc"
compressed = ''.join([k+str(len(list(g))) for k,g in groupby(s)])
print(compressed)
```

67. Decompress string

```
import re
s = "a3b2c1"
result = ''.join([c*int(n) for c,n in re.findall(r'([a-zA-Z])(\d+)', s)])
print(result)
```

68. Encode string using run-length encoding

```
from itertools import groupby
s = "aaabccddd"
encoded = ''.join([f"{k}{len(list(g))}" for k,g in groupby(s)])
print(encoded)
```

69. Decode run-length encoded string

```
import re
s = "a3b1c2"
decoded = ''.join([c*int(n) for c,n in re.findall(r'([a-zA-Z])(\d+)', s)])
print(decoded)
```

70. Check if string is valid shuffle of two strings

```
def is_valid_shuffle(s1,s2,result):
    from collections import Counter
    return Counter(s1+ s2) == Counter(result)

print(is_valid_shuffle("abc","def","adbcef"))
```

71. Count subsequences matching a given string

```
def count_subseq(s, t):
    n, m = len(s), len(t)
    dp = [0]*(m+1)
    dp[0] = 1
    for c in s:
        for j in range(m,0,-1):
            if c == t[j-1]:
                dp[j] += dp[j-1]
    return dp[m]

print(count_subseq("rabbbit","rabbit"))
```

72. Check if one string is subsequence of another

```
def is_subsequence(s, t):
    it = iter(t)
    return all(c in it for c in s)

print(is_subsequence("abc","ahbgdc"))
```

73. Edit distance between two strings (Levenshtein distance)

```
def edit_distance(s1,s2):
    n,m = len(s1),len(s2)
    dp = [[0]*(m+1) for _ in range(n+1)]
    for i in range(n+1):
        for j in range(m+1):
            if i==0: dp[i][j]=j
            elif j==0: dp[i][j]=i
            elif s1[i-1]==s2[j-1]: dp[i][j]=dp[i-1][j-1]
            else: dp[i][j]=1+min(dp[i-1][j], dp[i][j-1], dp[i-1][j-1])
    return dp[n][m]

print(edit_distance("kitten","sitting"))
```

74. Minimum insertions to make string palindrome

```
def min_insertions_palindrome(s):
    n = len(s)
    dp = [[0]*n for _ in range(n)]
    for length in range(2,n+1):
        for i in range(n-length+1):
            j=i+length-1
            if s[i]==s[j]: dp[i][j]=dp[i+1][j-1]
            else: dp[i][j]=1+min(dp[i+1][j], dp[i][j-1])
    return dp[0][n-1]

print(min_insertions_palindrome("ab"))
```

75. Minimum deletions to make string palindrome

```
def min_deletions_palindrome(s):
    n = len(s)
    rev = s[::-1]
    dp = [[0]*(n+1) for _ in range(n+1)]
    for i in range(1,n+1):
        for j in range(1,n+1):
            if s[i-1]==rev[j-1]: dp[i][j]=1+dp[i-1][j-1]
            else: dp[i][j]=max(dp[i-1][j],dp[i][j-1])
    return n - dp[n][n]

print(min_deletions_palindrome("abca"))
```

SOLUTION TO MORE COMPLEX PROBLEM

1. Text Analysis Tool

Problem: Given a paragraph of text, perform multiple analyses:

- Count sentences, words, characters
- Find most frequent word
- Count vowels, consonants, digits, special characters
- Identify longest and shortest word

```
from collections import Counter
import string

text = """Hello World! This is a sample text.
It contains multiple sentences, words, and characters."""

# 1. Count characters
total_chars = len(text)
# 2. Count words
words = text.split()
total_words = len(words)
# 3. Count sentences
sentences = sum(text.count(c) for c in ".!?")
# 4. Count vowels, consonants, digits, special characters
vowels = consonants = digits = special = 0
for c in text:
    if c.isalpha():
        if c.lower() in "aeiou":
            vowels +=1
        else:
            consonants +=1
    elif c.isdigit():
        digits +=1
    elif not c.isspace():
        special +=1
# 5. Most frequent word
freq = Counter(words)
most_common_word, count = freq.most_common(1)[0]
# 6. Longest and shortest words
longest_word = max(words, key=len)
shortest_word = min(words, key=len)

# Output
print(f"Characters: {total_chars}")
print(f"Words: {total_words}")
print(f"Sentences: {sentences}")
print(f"Vowels: {vowels}, Consonants: {consonants}, Digits: {digits},
 Special: {special}")
print(f"Most frequent word: '{most_common_word}' appears {count} times")
print(f"Longest word: {longest_word}")
print(f"Shortest word: {shortest_word}")
```

Explanation:

- `split()` → splits words
- `Counter` → counts frequency
- Iterated characters → to classify vowels, consonants, digits, special characters

- `max` and `min` → find longest/shortest word

2. Log File Analyzer

Problem: Analyze server logs:

- Extract IP addresses
- Count HTTP status codes
- Find top 5 requested URLs
- Detect error messages

```
import re
from collections import Counter

logs = """
192.168.1.1 GET /home 200
192.168.1.2 POST /login 401
192.168.1.1 GET /home 200
192.168.1.3 GET /dashboard 500
"""

# Extract IP addresses
ips = re.findall(r'\d+\.\d+\.\d+\.\d+', logs)
ip_count = Counter(ips)

# Count HTTP status codes
status_codes = re.findall(r'\s(\d{3})', logs)
status_count = Counter(status_codes)

# Find top 5 requested URLs
urls = re.findall(r'GET\s(/[\w]*)|POST\s(/[\w]*)', logs)
urls = [u[0] or u[1] for u in urls]
top_urls = Counter(urls).most_common(5)

# Detect errors (HTTP 4xx and 5xx)
errors = [line for line in logs.splitlines() if re.search(r'\s[45]\d{2}', line)]

print("IP Counts:", ip_count)
print("Status Code Counts:", status_count)
print("Top URLs:", top_urls)
print("Errors:", errors)
```

Explanation:

- `re.findall` → extract IPs, URLs, and status codes
- `Counter` → frequency counts
- Errors filtered using regex for 4xx or 5xx codes

3.Password Strength Checker

Problem: Validate password strength:

```
import string
```

```
password = "Passw0rd!"

# Criteria
length_ok = len(password) >= 8
upper_ok = any(c.isupper() for c in password)
lower_ok = any(c.islower() for c in password)
digit_ok = any(c.isdigit() for c in password)
special_ok = any(c in string.punctuation for c in password)

score = sum([length_ok, upper_ok, lower_ok, digit_ok, special_ok])

strength = ["Very Weak", "Weak", "Moderate", "Strong", "Very Strong"]
print(f"Password Strength: {strength[score-1]}")
```

Explanation:

- `any()` → checks if at least one character meets criteria
- `score` → sum of criteria matched
- Maps score to strength label

4.Chat Message Formatter

```
messages = """
[10:00] Alice: Hello!
[10:01] Bob: Hi Alice 🙂
[10:02] Alice: How are you?
"""

import re

pattern = r'\[(.*?)\]\s(.*?):\s(.*)'
for line in messages.splitlines():
    if line.strip():
        time, user, msg = re.match(pattern, line).groups()
        # Remove emojis
        msg_clean = re.sub(r'[^\w\s.,!?]', '', msg)
        print(f"{time} | {user} | {msg_clean}")
```

Explanation:

- Regex extracts timestamp, username, message
- `re.sub` removes emojis and special chars

5.Spell Checker & Corrector (Basic)

```
text = "Ths is a smple sentnce"
dictionary = {"this","is","a","sample","sentence"}

words = text.split()
corrected = [w if w.lower() in dictionary else "??" for w in words]
print(" ".join(corrected))
```

Explanation:

- Checks each word against dictionary
- Replaces unknown words with `"??"` (can be enhanced with suggestions)

6.Email/URL Extractor

```
import re

text = "Contact us at info@example.com or visit https://example.com"

emails = re.findall(r'\b[\w.-]+@[\w.-]+\.\w+\b', text)
urls = re.findall(r'https?://[^\s]+', text)

print("Emails:", emails)
print("URLs:", urls)
```

Explanation:

- Regex patterns extract emails and URLs efficiently

7.Substring Search & Highlight

```
text = "Python is amazing. I love Python programming."
keywords = ["Python", "programming"]

for kw in keywords:
    text = text.replace(kw, f"[{kw}]")  # highlight
print(text)
```

Explanation:

- Replaces keyword occurrences with `[keyword]`

8.Text Compression & Decompression (RLE)

```
from itertools import groupby

text = "aaabbccddd"

# Compression
compressed = "".join([k+str(len(list(g))) for k,g in groupby(text)])
print("Compressed:", compressed)

# Decompression
import re
decompressed = "".join([c*int(n) for c,n in re.findall(r'([a-zA-Z])(\d+)', compressed)
print("Decompressed:", decompressed)
```

Explanation:

- Run-Length Encoding compresses repeated characters
- Regex extracts character and count for decompression

9.CSV/Text Table Parser

```
csv_text = """Name,Age,City
Alice,25,NY
Bob,30,LA"""

rows = [line.split(",") for line in csv_text.splitlines()]
for row in rows:
    print(f"Name: {row[0]}, Age: {row[1]}, City: {row[2]}")
```

Explanation:

- Splits rows and columns using `split()`

10.String-Based Calculator

```
expr = "12 + 34 * (5 - 2)"
result = eval(expr)
print(result)
```

Explanation:

- `eval()` evaluates arithmetic string expressions (for safe practice, parse manually for production)

11.DNA Sequence Analyzer

```
dna = "AGCTTAGCTA"
# Nucleotide frequency
freq = {n:dna.count(n) for n in "ACGT"}
# Longest repeating sequence
max_len = 0; max_seq = ""
for n in "ACGT":
    current_len = 0
    for c in dna:
        if c==n:
            current_len+=1
            if current_len>max_len:
                max_len=current_len
                max_seq=n*max_len
        else:
            current_len=0
print("Frequency:", freq)
print("Longest repeat:", max_seq)
```

Explanation:

- Counts nucleotides and finds longest consecutive repetition

12.Log Compression & Aggregation

```
logs = ["ERROR Disk full","ERROR Disk full","INFO System started","WARNING Low memory"

from itertools import groupby
compressed = [(k,len(list(g))) for k,g in groupby(logs)]
```

```
print(compressed)
```

Explanation:

- Groups repeated consecutive logs using `groupby`

13.Social Media Text Analytics

```
text = "Love #python #coding @user1. Python is awesome! #python"

hashtags = re.findall(r"#\w+", text)
mentions = re.findall(r"@\w+", text)
top_hashtags = Counter(hashtags).most_common(3)

print("Hashtags:", hashtags)
print("Mentions:", mentions)
print("Top Hashtags:", top_hashtags)
```

Explanation:

- Regex extracts hashtags/mentions
- Counter finds trending hashtags

14.Code Parser / Formatter (Basic)

```
code = """
# This is a comment
def add(a,b):
    return a+b
"""

lines = [line for line in code.splitlines() if line.strip() and not line.strip().start
print("\n".join(lines))
```

Explanation:

- Removes comments and blank lines

15.Chatbot Preprocessing Module

```
user_input = "Hello! How are you? 🙂"
# Tokenize
tokens = user_input.split()
# Normalize
tokens = [t.lower().strip(string.punctuation) for t in tokens]
print(tokens)
```

Explanation:

- Tokenizes and normalizes user input for chatbot processing

16.Palindrome & Anagram Finder in Text

```
text = "madam racecar level refer"
words = text.split()
palindromes = [w for w in words if w==w[::-1]]

from collections import defaultdict
anagrams = defaultdict(list)
for w in words:
    anagrams[tuple(sorted(w))].append(w)
print("Palindromes:", palindromes)
print("Anagram groups:", dict(anagrams))
```

Explanation:

- Detects palindromes and groups anagrams

17.Text-Based Game Engine (Command Parsing)

```
commands = ["MOVE NORTH","ATTACK GOBLIN","PICKUP SWORD"]
for cmd in commands:
    action, *args = cmd.split()
    print(f"Action: {action}, Args: {args}")
```

Explanation:

- Parses action and arguments for game commands

18.Custom Markup Language Parser

```
text = "[b]Bold[/b] and [i]Italic[/i]"
formatted = text.replace("[b]", "<b>").replace("[/b]", "</b>").replace("[i]", "<i>").
print(formatted)
```

Explanation:

- Replaces custom tags with HTML tags

19.Multi-Language Text Translator (Simulation)

```
text = "hello world"
dictionary = {"hello":"hola","world":"mundo"}
translated = " ".join([dictionary.get(w,w) for w in text.split()])
print(translated)
```

Explanation:

- Simple word-by-word translation

20.Report Generator from Raw Text

```
raw = "Alice,25,NY\nBob,30,LA\nAlice,25,NY"
rows = [line.split(",") for line in raw.splitlines()]
report = {}
for r in rows:
    name = r[0]
    report[name] = report.get(name,0)+1
print("Report:", report)
```

Explanation:

- Aggregates data by name and counts occurrences

UNIT-3(LIST) (Back to List Question)

```
# 1. Create a list of 5 elements and print it

my_list = [10, 20, 30, 40, 50]

print("Original list:", my_list)


# 2. Access the 3rd element of a list

print("3rd element:", my_list[2])


# 3. Add an element to the end of a list

my_list.append(60)

print("After appending 60:", my_list)


# 4. Insert an element at 2nd position

my_list.insert(1, 15)   # index starts from 0

print("After inserting 15 at 2nd position:", my_list)


# 5. Remove an element from a list

my_list.remove(30)

print("After removing 30:", my_list)


# 6. Find the length of a list

print("Length of the list:", len(my_list))


# 7. Print all elements using a loop

print("All elements using loop:")
```

for item in my_list:

print(item)

# 8. Sort a list in ascending order

my_list.sort()

print("Sorted list:", my_list)

# 9. Reverse a list

my_list.reverse()

print("Reversed list:", my_list)

# 10. Sum all elements of a list

print("Sum of all elements:", sum(my_list))

Program 11: Find the maximum element in a list (Back to question)

```
my_list = [4, 7, 1, 9, 2]
print("Maximum element:", max(my_list))
```

Program 12: Find the minimum element in a list

```
my_list = [4, 7, 1, 9, 2]
print("Minimum element:", min(my_list))
```

Program 13: Sum only even numbers in a list

```
my_list = [1, 2, 3, 4, 5, 6]
sum_even = sum(x for x in my_list if x % 2 == 0)
print("Sum of even numbers:", sum_even)
```

Program 14: Count occurrences of an element

```
my_list = [1, 2, 2, 3, 2, 4]
count_2 = my_list.count(2)
print("Number of times 2 occurs:", count_2)
```

Program 15: Copy a list

```
my_list = [1, 2, 3]
new_list = my_list.copy()
print("Copied list:", new_list)
```

Program 16: Merge two lists

```
list1 = [1, 2, 3]
list2 = [4, 5, 6]
merged = list1 + list2
print("Merged list:", merged)
```

Program 17: Get a sublist using slicing

```
my_list = [1, 2, 3, 4, 5]
sublist = my_list[1:4]
print("Sublist:", sublist)
```

Program 18: Reverse a list using slicing

```
my_list = [1, 2, 3, 4]
reversed_list = my_list[::-1]
print("Reversed list:", reversed_list)
```

Program 19: Remove duplicates from a list

```
my_list = [1, 2, 2, 3, 3, 4]
unique_list = list(set(my_list))
print("List without duplicates:", unique_list)
```

Program 20: Flatten a nested list

```
nested_list = [[1, 2], [3, 4], [5]]
flat_list = [item for sublist in nested_list for item in sublist]
print("Flattened list:", flat_list)
```

Program 21: Find the second largest element in a list

```
my_list = [4, 7, 1, 9, 2]
unique_list = list(set(my_list))  # Remove duplicates
unique_list.sort()
print("Second largest element:", unique_list[-2])
```

Program 22: Check if a list is a palindrome

```
my_list = [1, 2, 3, 2, 1]
if my_list == my_list[::-1]:
    print("The list is a palindrome")
```

```
else:
    print("The list is not a palindrome")
```

Program 23: Rotate a list to the right by 2 positions

```
my_list = [1, 2, 3, 4, 5]
n = 2
rotated_list = my_list[-n:] + my_list[:-n]
print("Rotated list:", rotated_list)
```

Program 24: Find all indices of an element in a list

```
my_list = [1, 2, 3, 2, 4, 2]
element = 2
indices = [i for i, x in enumerate(my_list) if x == element]
print("Indices of 2:", indices)
```

Program 25: Merge two lists and remove duplicates

```
list1 = [1, 2, 3]
list2 = [2, 3, 4]
merged_unique = list(set(list1 + list2))
print("Merged list without duplicates:", merged_unique)
```

Program 26: Find common elements between two lists

```
list1 = [1, 2, 3, 4]
list2 = [3, 4, 5, 6]
common = [x for x in list1 if x in list2]
print("Common elements:", common)
```

Program 27: Find elements greater than a given number

```
my_list = [1, 5, 8, 2, 10]
num = 5
greater = [x for x in my_list if x > num]
print("Elements greater than 5:", greater)
```

Program 28: Find the product of all elements in a list

```
my_list = [1, 2, 3, 4]
product = 1
for x in my_list:
    product *= x
print("Product of elements:", product)
```

Program 29: Check if a list contains a given element

```
my_list = [1, 2, 3, 4]
```

```
element = 3
if element in my_list:
    print("List contains", element)
else:
    print("List does not contain", element)
```

Program 30: Find the average of elements in a list

```
my_list = [1, 2, 3, 4, 5]
average = sum(my_list) / len(my_list)
print("Average of elements:", average)
```

Program 31: Find duplicates in a list

```
my_list = [1, 2, 3, 2, 4, 3, 5]
duplicates = []
for item in my_list:
    if my_list.count(item) > 1 and item not in duplicates:
        duplicates.append(item)
print("Duplicate elements:", duplicates)
```

Program 32: Count positive and negative numbers in a list

```
my_list = [1, -2, 3, -4, 5]
positives = sum(1 for x in my_list if x > 0)
negatives = sum(1 for x in my_list if x < 0)
print("Positive numbers:", positives)
print("Negative numbers:", negatives)
```

Program 33: Separate even and odd numbers into two lists

```
my_list = [1, 2, 3, 4, 5, 6]
even = [x for x in my_list if x % 2 == 0]
odd = [x for x in my_list if x % 2 != 0]
print("Even numbers:", even)
print("Odd numbers:", odd)
```

Program 34: Merge two sorted lists into a single sorted list

```
list1 = [1, 3, 5]
list2 = [2, 4, 6]
merged_sorted = sorted(list1 + list2)
print("Merged and sorted list:", merged_sorted)
```

Program 35: Find the largest even number in a list

```
my_list = [1, 4, 7, 8, 10]
even_numbers = [x for x in my_list if x % 2 == 0]
if even_numbers:
    print("Largest even number:", max(even_numbers))
else:
    print("No even numbers found")
```

Program 36: Find the smallest odd number in a list

```
my_list = [2, 4, 7, 9, 10]
odd_numbers = [x for x in my_list if x % 2 != 0]
if odd_numbers:
    print("Smallest odd number:", min(odd_numbers))
else:
    print("No odd numbers found")
```

Program 37: Count elements greater than their previous element

```
my_list = [1, 3, 2, 4, 5]
count = sum(1 for i in range(1, len(my_list)) if my_list[i] > my_list[i-1])
print("Count of elements greater than previous:", count)
```

Program 38: Find second smallest element in a list

```
my_list = [4, 7, 1, 9, 2]
unique_list = list(set(my_list))
unique_list.sort()
print("Second smallest element:", unique_list[1])
```

Program 39: Replace negative numbers with zero in a list

```
my_list = [1, -2, 3, -4, 5]
new_list = [x if x >= 0 else 0 for x in my_list]
print("List after replacing negatives with 0:", new_list)
```

Program 40: Count frequency of all elements in a list

```
my_list = [1, 2, 2, 3, 3, 3, 4]
frequency = {x: my_list.count(x) for x in set(my_list)}
print("Frequency of elements:", frequency)
```

Program 41: Move all zeros to the end of a list

```
my_list = [1, 0, 2, 0, 3, 0, 4]
non_zeros = [x for x in my_list if x != 0]
zeros = [x for x in my_list if x == 0]
result = non_zeros + zeros
print("List after moving zeros to end:", result)
```

Program 42: Find missing numbers from 1 to n in a list

```
my_list = [1, 2, 4, 6, 5]
n = 6
missing = [x for x in range(1, n+1) if x not in my_list]
print("Missing numbers:", missing)
```

Program 43: Cumulative sum of a list

```
my_list = [1, 2, 3, 4]
cumulative = []
total = 0
for x in my_list:
    total += x
    cumulative.append(total)
print("Cumulative sum list:", cumulative)
```

Program 44: Reverse a list without using reverse() or slicing

```
my_list = [1, 2, 3, 4]
reversed_list = []
for i in range(len(my_list)-1, -1, -1):
    reversed_list.append(my_list[i])
print("Reversed list:", reversed_list)
```

Program 45: Rotate a list to the left by 3 positions

```
my_list = [1, 2, 3, 4, 5, 6]
n = 3
rotated_list = my_list[n:] + my_list[:n]
print("Left rotated list:", rotated_list)
```

Program 46: Find the largest sum of consecutive elements of length 3

```
my_list = [1, 2, 3, 4, 5]
k = 3
max_sum = max(sum(my_list[i:i+k]) for i in range(len(my_list)-k+1))
print("Largest sum of 3 consecutive elements:", max_sum)
```

Program 47: Count elements that are greater than both neighbors

```
my_list = [1, 3, 2, 4, 1]
count = sum(1 for i in range(1, len(my_list)-1)
 if my_list[i] > my_list[i-1]
 and my_list[i] > my_list[i+1])
print("Elements greater than neighbors:", count)
```

Program 48: Find all pairs with a given sum

```
my_list = [1, 2, 3, 4, 5]
target = 5
pairs = [(my_list[i], my_list[j])
for i in range(len(my_list))
 for j in range(i+1, len(my_list))
 if my_list[i]+my_list[j] == target]
print("Pairs with sum 5:", pairs)
```

Program 49: Flatten a deeply nested list

```
nested_list = [1, [2, [3, 4], 5], 6]
def flatten(lst):
    flat = []
    for i in lst:
        if isinstance(i, list):
            flat.extend(flatten(i))
        else:
            flat.append(i)
    return flat
print("Flattened list:", flatten(nested_list))
```

Program 50: Find the missing number in a consecutive sequence using formula

```
my_list = [1, 2, 4, 5, 6]
n = len(my_list) + 1  # total numbers including missing
expected_sum = n * (n + 1) // 2
actual_sum = sum(my_list)
missing_number = expected_sum - actual_sum
print("Missing number:", missing_number)
```

Lists(ADVANCED)

Program 1: Bubble Sort

Aim:
To implement bubble sort on a list.

Algorithm:

1. Start.
2. Compare adjacent elements in the list.
3. Swap them if they are out of order.
4. Repeat until the list is sorted.
5. Print the sorted list.

Code:

```
def bubble_sort(arr):
    n = len(arr)
    for i in range(n-1):
        for j in range(n-i-1):
            if arr[j] > arr[j+1]:
                arr[j], arr[j+1] = arr[j+1], arr[j]

arr = [64, 25, 12, 22, 11]
bubble_sort(arr)
print("Sorted List:", arr)
```

Sample Output:

```
Sorted List: [11, 12, 22, 25, 64]
```

Result:
Bubble Sort executed successfully.

Program 2: Insertion Sort

Aim:
To implement insertion sort on a list.

Algorithm:

1. Start.
2. Assume first element is sorted.
3. Insert each subsequent element into its correct position.
4. Continue until all elements are sorted.

Code:

```
def insertion_sort(arr):
    for i in range(1, len(arr)):
        key = arr[i]
        j = i - 1
        while j >= 0 and key < arr[j]:
            arr[j+1] = arr[j]
            j -= 1
        arr[j+1] = key

arr = [12, 11, 13, 5, 6]
insertion_sort(arr)
print("Sorted List:", arr)
```

Sample Output:

```
Sorted List: [5, 6, 11, 12, 13]
```

Result:
Insertion Sort executed successfully.

Program 3: Selection Sort

Aim:
To implement selection sort on a list.

Algorithm:

1. Start.
2. Find the smallest element in the list.
3. Swap it with the first element.
4. Repeat for the remaining unsorted portion.
5. Print the sorted list.

Code:

```
def selection_sort(arr):
    for i in range(len(arr)):
        min_idx = i
        for j in range(i+1, len(arr)):
            if arr[j] < arr[min_idx]:
                min_idx = j
        arr[i], arr[min_idx] = arr[min_idx], arr[i]

arr = [64, 25, 12, 22, 11]
selection_sort(arr)
print("Sorted List:", arr)
```

Sample Output:

```
Sorted List: [11, 12, 22, 25, 64]
```

Result:
Selection Sort executed successfully.

Program 4: Merge Sort

Aim:
To implement merge sort on a list.

Algorithm:

1. Divide the list into two halves.
2. Recursively sort each half.
3. Merge the two sorted halves.

Code:

```
def merge_sort(arr):
    if len(arr) > 1:
        mid = len(arr)//2
        L = arr[:mid]
        R = arr[mid:]

        merge_sort(L)
        merge_sort(R)

        i = j = k = 0
        while i < len(L) and j < len(R):
            if L[i] < R[j]:
                arr[k] = L[i]
                i += 1
            else:
                arr[k] = R[j]
                j += 1
            k += 1

        while i < len(L):
            arr[k] = L[i]
            i += 1
            k += 1

        while j < len(R):
            arr[k] = R[j]
            j += 1
            k += 1

arr = [38, 27, 43, 3, 9, 82, 10]
merge_sort(arr)
print("Sorted List:", arr)
```

Sample Output:

```
Sorted List: [3, 9, 10, 27, 38, 43, 82]
```

Result:
Merge Sort executed successfully.

Program 5: Quick Sort

Aim:
To implement quick sort on a list.

Algorithm:

1. Select a pivot element.
2. Partition list into two halves (less than pivot and greater than pivot).
3. Recursively sort both halves.
4. Combine the results.

Code:

```
def quick_sort(arr):
    if len(arr) <= 1:
        return arr
    pivot = arr[len(arr)//2]
    left = [x for x in arr if x < pivot]
    middle = [x for x in arr if x == pivot]
    right = [x for x in arr if x > pivot]
    return quick_sort(left) + middle + quick_sort(right)

arr = [3, 6, 8, 10, 1, 2, 1]
print("Sorted List:", quick_sort(arr))
```

Sample Output:

```
Sorted List: [1, 1, 2, 3, 6, 8, 10]
```

Result:
Quick Sort executed successfully.

Program 6: Heap Sort

Aim:
To implement heap sort using lists.

Algorithm:

1. Build a max-heap from the list.
2. Swap the root with the last element.
3. Reduce the heap size by one and heapify.
4. Repeat until all elements are sorted.

Code:

```
def heapify(arr, n, i):
    largest = i
    l = 2 * i + 1
    r = 2 * i + 2
    if l < n and arr[i] < arr[l]:
        largest = l

    if r < n and arr[largest] < arr[r]:
        largest = r

    if largest != i:
        arr[i], arr[largest] = arr[largest], arr[i]
        heapify(arr, n, largest)

def heap_sort(arr):
    n = len(arr)

    for i in range(n//2 - 1, -1, -1):
        heapify(arr, n, i)

    for i in range(n-1, 0, -1):
        arr[i], arr[0] = arr[0], arr[i]
        heapify(arr, i, 0)

arr = [12, 11, 13, 5, 6, 7]
heap_sort(arr)
print("Sorted List:", arr)
```

Sample Output:

```
Sorted List: [5, 6, 7, 11, 12, 13]
```

Result:
Heap Sort executed successfully.

Program 7: Linear Search

Aim:
To implement linear search on a list.

Algorithm:

1. Start.
2. Traverse each element of the list.
3. If the element matches the target, return its index.
4. If not found, return -1.

Code:

```
def linear_search(arr, x):
    for i in range(len(arr)):
        if arr[i] == x:
            return i
    return -1

arr = [10, 20, 30, 40, 50]
x = 30
result = linear_search(arr, x)

if result != -1:
    print(f"Element found at index {result}")
else:
    print("Element not found")
```

Sample Output:

```
Element found at index 2
```

Result:
Linear Search executed successfully.

Program 8: Binary Search (Iterative + Recursive)

Aim:
To implement binary search in both iterative and recursive approaches.

Algorithm:

1. Start with low and high indices.
2. Find the mid index.
3. Compare target with middle element.
4. If equal, return mid.
5. If smaller, search left half; else, search right half.

Code (Iterative + Recursive):

```
def binary_search_iterative(arr, x):
    low, high = 0, len(arr)-1
    while low <= high:
        mid = (low + high) // 2
        if arr[mid] == x:
            return mid
        elif arr[mid] < x:
            low = mid + 1
        else:
            high = mid - 1
    return -1

def binary_search_recursive(arr, low, high, x):
    if high >= low:
        mid = (low + high) // 2
        if arr[mid] == x:
            return mid
        elif arr[mid] > x:
            return binary_search_recursive(arr, low, mid-1, x)
        else:
            return binary_search_recursive(arr, mid+1, high, x)
    return -1

arr = [10, 20, 30, 40, 50]
x = 40
print("Iterative:", binary_search_iterative(arr, x))
print("Recursive:", binary_search_recursive(arr, 0, len(arr)-1, x))
```

Sample Output:

```
Iterative: 3
Recursive: 3
```

Result:
Binary Search executed successfully.

Program 9: Reverse a List without Built-in Functions

Aim:
To reverse a list without using built-in functions like `reverse()` or slicing.

Algorithm:

1. Start.
2. Swap the first and last element.
3. Move inward and repeat until middle is reached.

Code:

```
def reverse_list(arr):
    start, end = 0, len(arr)-1
    while start < end:
        arr[start], arr[end] = arr[end], arr[start]
        start += 1
        end -= 1
    return arr

arr = [1, 2, 3, 4, 5]
print("Reversed List:", reverse_list(arr))
```

Sample Output:

```
Reversed List: [5, 4, 3, 2, 1]
```

Result:
List reversed successfully.

Program 10: Find Union and Intersection of Two Lists

Aim:
To find the union and intersection of two lists without using sets.

Algorithm:

1. Start.
2. Union → merge both lists and remove duplicates.
3. Intersection → check common elements between lists.

Code:

```
def union_intersection(list1, list2):
    union = list1[:]
    for item in list2:
        if item not in union:
            union.append(item)
    intersection = [item for item in list1 if item in list2]
    return union, intersection

list1 = [1, 2, 3, 4]
list2 = [3, 4, 5, 6]
u, i = union_intersection(list1, list2)
print("Union:", u)
print("Intersection:", i)
```

Sample Output:

```
Union: [1, 2, 3, 4, 5, 6]
Intersection: [3, 4]
```

Result:
Union and intersection found successfully.

Program 11: Generate All Permutations of a List

Aim:
To generate all permutations of a list using recursion.

Algorithm:

1. Start.
2. If list length is 1, return it.
3. For each element, fix it and recursively find permutations of remaining list.
4. Append results.

Code:

```
def permutations(arr):
    if len(arr) == 0:
        return [[]]
    result = []
    for i in range(len(arr)):
        rest = arr[:i] + arr[i+1:]
        for p in permutations(rest):
            result.append([arr[i]] + p)
    return result

arr = [1, 2, 3]
print("Permutations:", permutations(arr))
```

Sample Output:

```
Permutations: [[1, 2, 3], [1, 3, 2], [2, 1, 3], [2, 3, 1], [3, 1, 2], [3, 2, 1]]
```

Result:
All permutations generated successfully.

Program 12: Pascal’s Triangle using Lists

Aim:
To generate Pascal’s Triangle using lists.

Algorithm:

1. Start.
2. First row → `[1]`.
3. Each next row is formed by adding adjacent elements of the previous row.
4. Continue till `n` rows.

Code:def pascal_triangle(n):

```
    triangle = [[1]]
    for i in range(1, n):
        row = [1]
        for j in range(1, i):
            row.append(triangle[i-1][j-1] + triangle[i-1][j])
        row.append(1)
        triangle.append(row)
    return triangle

n = 5
for row in pascal_triangle(n):
    print(row)
```

Sample Output:

```
[1]
[1, 1]
[1, 2, 1]
[1, 3, 3, 1]
[1, 4, 6, 4, 1]
```

Result:
Pascal’s Triangle generated successfully.

Program 13: Polynomial Addition and Multiplication using Lists

Aim:
To perform addition and multiplication of polynomials represented as coefficient lists.

Algorithm:

1. Represent polynomial as list of coefficients (index = power).
2. For addition → add corresponding coefficients.

3. For multiplication → multiply terms and add to proper index.

Code:

```
def poly_add(p1, p2):
    n = max(len(p1), len(p2))
    result = [0] * n
    for i in range(len(p1)):
        result[i] += p1[i]
    for i in range(len(p2)):
        result[i] += p2[i]
    return result

def poly_mul(p1, p2):
    result = [0] * (len(p1) + len(p2) - 1)
    for i in range(len(p1)):
        for j in range(len(p2)):
            result[i+j] += p1[i] * p2[j]
    return result

p1 = [3, 2, 5]   # 3 + 2x + 5x^2
p2 = [5, 1]      # 5 + x
print("Addition:", poly_add(p1, p2))
print("Multiplication:", poly_mul(p1, p2))
```

Sample Output:

```
Addition: [8, 3, 5]
Multiplication: [15, 13, 27, 5]
```

Result:
Polynomial operations executed successfully.

Program 14: Filter Prime Numbers from a List

Aim:
To filter prime numbers from a list using function and filtering method.

Algorithm:

1. Write function `is_prime()`.
2. Traverse list, check primality.
3. Collect primes into new list.

Code:

```
def is_prime(n):
    if n < 2:
        return False
    for i in range(2, int(n**0.5)+1):
        if n % i == 0:
            return False
    return True

arr = [10, 15, 3, 7, 19, 20]
primes = [x for x in arr if is_prime(x)]
print("Prime Numbers:", primes)
```

Sample Output:

```
Prime Numbers: [3, 7, 19]
```

Result:
Prime numbers filtered successfully.

UNIT-4 TUPLES

Question 1: Create a tuple of 5 elements and print it.

```
my_tuple = (1, 2, 3, 4, 5)
print(my_tuple)
```

Question 2: Access the 3rd element of a tuple.

```
my_tuple = ('a', 'b', 'c', 'd')
print(my_tuple[2])
```

Question 3: Count occurrences of an element in a tuple.

```
my_tuple = (1, 2, 2, 3, 2)
print(my_tuple.count(2))
```

Question 4: Find the index of an element in a tuple.

```
my_tuple = (10, 20, 30, 40)
print(my_tuple.index(30))
```

Question 5: Slice a tuple to get a sub-tuple.

```
my_tuple = (1, 2, 3, 4, 5)
print(my_tuple[1:4])
```

Question 6: Concatenate two tuples.

```
tuple1 = (1, 2, 3)
tuple2 = (4, 5, 6)
result = tuple1 + tuple2
print(result)
```

Question 7: Repeat a tuple 3 times.

```
my_tuple = (1, 2)
print(my_tuple * 3)
```

Question 8: Find the length of a tuple.

```
my_tuple = (1, 2, 3, 4, 5)
print(len(my_tuple))
```

Question 9: Check if an element exists in a tuple.

```
my_tuple = (1, 2, 3, 4)
print(3 in my_tuple)
```

Question 10: Convert a list to a tuple.

```
my_list = [1, 2, 3, 4]
my_tuple = tuple(my_list)
print(my_tuple)
```

Question 11: Find the maximum element in a tuple.

```
my_tuple = (4, 7, 1, 9, 2)
print("Maximum element:", max(my_tuple))
```

Question 12: Find the minimum element in a tuple.

```
my_tuple = (4, 7, 1, 9, 2)
print("Minimum element:", min(my_tuple))
```

Question 13: Sum all elements in a tuple.

```
my_tuple = (1, 2, 3, 4, 5)
print("Sum of elements:", sum(my_tuple))
```

Question 14: Find the product of all elements in a tuple.

```
my_tuple = (1, 2, 3, 4)
product = 1
for x in my_tuple:
    product *= x
print("Product of elements:", product)
```

Question 15: Reverse a tuple.

```
my_tuple = (1, 2, 3, 4)
reversed_tuple = my_tuple[::-1]
print("Reversed tuple:", reversed_tuple)
```

Question 16: Count elements greater than a given number in a tuple.

```
my_tuple = (1, 5, 8, 2, 10)
count = sum(1 for x in my_tuple if x > 5)
print("Elements greater than 5:", count)
```

Question 17: Find all even numbers in a tuple.

```
my_tuple = (1, 2, 3, 4, 5, 6)
evens = tuple(x for x in my_tuple if x % 2 == 0)
print("Even numbers:", evens)
```

Question 18: Find all odd numbers in a tuple.

```
my_tuple = (1, 2, 3, 4, 5, 6)
odds = tuple(x for x in my_tuple if x % 2 != 0)
print("Odd numbers:", odds)
```

Question 19: Merge two tuples and sort the result.

```
tuple1 = (3, 1, 4)
tuple2 = (2, 5, 0)
merged_sorted = tuple(sorted(tuple1 + tuple2))
print("Merged and sorted tuple:", merged_sorted)
```

Question 20: Remove duplicates from a tuple.

```
my_tuple = (1, 2, 2, 3, 3, 4)
unique_tuple = tuple(set(my_tuple))
print("Tuple without duplicates:", unique_tuple)
```

Question 21: Find the second largest element in a tuple.

```
my_tuple = (4, 7, 1, 9, 2)
unique_tuple = tuple(set(my_tuple))
sorted_tuple = tuple(sorted(unique_tuple))
print("Second largest element:", sorted_tuple[-2])
```

Question 22: Check if a tuple is a palindrome.

```
my_tuple = (1, 2, 3, 2, 1)
if my_tuple == my_tuple[::-1]:
    print("The tuple is a palindrome")
else:
    print("The tuple is not a palindrome")
```

Question 23: Find all indices of an element in a tuple.

```
my_tuple = (1, 2, 3, 2, 4, 2)
element = 2
indices = tuple(i for i, x in enumerate(my_tuple) if x == element)
print("Indices of 2:", indices)
```

Question 24: Count elements greater than their previous element.

```
my_tuple = (1, 3, 2, 4, 5)
count = sum(1 for i in range(1, len(my_tuple)) if my_tuple[i] > my_tuple[i-1])
print("Count of elements greater than previous:", count)
```

Question 25: Find common elements between two tuples.

```
tuple1 = (1, 2, 3, 4)
tuple2 = (3, 4, 5, 6)
common = tuple(x for x in tuple1 if x in tuple2)
print("Common elements:", common)
```

Question 26: Find elements greater than a given number.

```
my_tuple = (1, 5, 8, 2, 10)
num = 5
greater = tuple(x for x in my_tuple if x > num)
print("Elements greater than 5:", greater)
```

Question 27: Find the sum of elements at even indices.

```
my_tuple = (1, 2, 3, 4, 5)
sum_even_index = sum(my_tuple[i] for i in range(0, len(my_tuple), 2))
print("Sum of elements at even indices:", sum_even_index)
```

Question 28: Find the product of elements at odd indices.

```
my_tuple = (1, 2, 3, 4, 5)
product_odd_index = 1
for i in range(1, len(my_tuple), 2):
    product_odd_index *= my_tuple[i]
print("Product of elements at odd indices:", product_odd_index)
```

Question 29: Check if all elements in a tuple are unique.

```
my_tuple = (1, 2, 3, 4)
if len(my_tuple) == len(set(my_tuple)):
    print("All elements are unique")
else:
    print("Duplicates exist in the tuple")
```

Question 30: Find the tuple with the maximum sum from a list of tuples.

```
tuple_list = [(1, 2), (3, 4), (5, 1)]
max_tuple = max(tuple_list, key=sum)
print("Tuple with maximum sum:", max_tuple)
```

Question 31: Find duplicates in a tuple.

```
my_tuple = (1, 2, 3, 2, 4, 3, 5)
duplicates = tuple(x for x in set(my_tuple) if my_tuple.count(x) > 1)
print("Duplicate elements:", duplicates)
```

Question 32: Count positive and negative numbers in a tuple.

```
my_tuple = (1, -2, 3, -4, 5)
positives = sum(1 for x in my_tuple if x > 0)
negatives = sum(1 for x in my_tuple if x < 0)
print("Positive numbers:", positives)
print("Negative numbers:", negatives)
```

Question 33: Separate even and odd numbers into two tuples.

```
my_tuple = (1, 2, 3, 4, 5, 6)
even = tuple(x for x in my_tuple if x % 2 == 0)
```

```
odd = tuple(x for x in my_tuple if x % 2 != 0)
print("Even numbers:", even)
print("Odd numbers:", odd)
```

Question 34: Merge two sorted tuples into a single sorted tuple.

```
tuple1 = (1, 3, 5)
tuple2 = (2, 4, 6)
merged_sorted = tuple(sorted(tuple1 + tuple2))
print("Merged and sorted tuple:", merged_sorted)
```

Question 35: Find the largest even number in a tuple.

```
my_tuple = (1, 4, 7, 8, 10)
even_numbers = tuple(x for x in my_tuple if x % 2 == 0)
if even_numbers:
    print("Largest even number:", max(even_numbers))
else:
    print("No even numbers found")
```

Question 36: Find the smallest odd number in a tuple.

```
my_tuple = (2, 4, 7, 9, 10)
odd_numbers = tuple(x for x in my_tuple if x % 2 != 0)
if odd_numbers:
    print("Smallest odd number:", min(odd_numbers))
else:
    print("No odd numbers found")
```

Question 37: Count elements greater than both neighbors in a tuple.

```
my_tuple = (1, 3, 2, 4, 1)
count = sum(1 for i in range(1, len(my_tuple)-1)
if my_tuple[i] > my_tuple[i-1] and my_tuple[i] > my_tuple[i+1])
print("Elements greater than neighbors:", count)
```

Question 38: Replace negative numbers with zero in a tuple.

```
my_tuple = (1, -2, 3, -4, 5)
new_tuple = tuple(x if x >= 0 else 0 for x in my_tuple)
print("Tuple after replacing negatives with 0:", new_tuple)
```

Question 39: Count frequency of all elements in a tuple.

```
my_tuple = (1, 2, 2, 3, 3, 3, 4)
frequency = {x: my_tuple.count(x) for x in set(my_tuple)}
print("Frequency of elements:", frequency)
```

Question 40: Find the second smallest element in a tuple.

```
my_tuple = (4, 7, 1, 9, 2)
unique_tuple = tuple(set(my_tuple))
sorted_tuple = tuple(sorted(unique_tuple))
print("Second smallest element:", sorted_tuple[1])
```

Question 41: Move all zeros to the end of a tuple.

```
my_tuple = (1, 0, 2, 0, 3, 0, 4)
non_zeros = tuple(x for x in my_tuple if x != 0)
zeros = tuple(x for x in my_tuple if x == 0)
result = non_zeros + zeros
print("Tuple after moving zeros to end:", result)
```

Question 42: Find missing numbers from 1 to n in a tuple.

```
my_tuple = (1, 2, 4, 6, 5)
n = 6
missing = tuple(x for x in range(1, n+1) if x not in my_tuple)
print("Missing numbers:", missing)
```

Question 43: Cumulative sum of a tuple.

```
my_tuple = (1, 2, 3, 4)
cumulative = []
total = 0
for x in my_tuple:
    total += x
    cumulative.append(total)
print("Cumulative sum tuple:", tuple(cumulative))
```

Question 44: Reverse a tuple without using slicing.

```
my_tuple = (1, 2, 3, 4)
reversed_tuple = tuple()
for i in range(len(my_tuple)-1, -1, -1):
    reversed_tuple += (my_tuple[i],)
print("Reversed tuple:", reversed_tuple)
```

Question 45: Rotate a tuple to the left by 3 positions.

```
my_tuple = (1, 2, 3, 4, 5, 6)
n = 3
rotated_tuple = my_tuple[n:] + my_tuple[:n]
print("Left rotated tuple:", rotated_tuple)
```

Question 46: Find the largest sum of consecutive elements of length 3 in a tuple.

```
my_tuple = (1, 2, 3, 4, 5)
```

```
k = 3
max_sum = max(sum(my_tuple[i:i+k]) for i in range(len(my_tuple)-k+1))
print("Largest sum of 3 consecutive elements:", max_sum)
```

Question 47: Count elements that are greater than both neighbors in a tuple.

```
my_tuple = (1, 3, 2, 4, 1)
count = sum(1 for i in range(1, len(my_tuple)-1) if my_tuple[i] > my_tuple[i-1] and my
my_tuple[i+1])
print("Elements greater than neighbors:", count)
```

Question 48: Find all pairs with a given sum in a tuple.

```
my_tuple = (1, 2, 3, 4, 5)
target = 5
pairs = tuple((my_tuple[i], my_tuple[j]) for i in range(len(my_tuple))
 for j in range(i+1, len(my_tuple)) if my_tuple[i]+my_tuple[j] == target)
print("Pairs with sum 5:", pairs)
```

Question 49: Flatten a nested tuple.

```
nested_tuple = (1, (2, (3, 4), 5), 6)
def flatten(t):
    flat = []
    for i in t:
        if isinstance(i, tuple):
            flat.extend(flatten(i))
        else:
            flat.append(i)
    return flat
print("Flattened tuple:", tuple(flatten(nested_tuple)))
```

Question 50: Find the missing number in a consecutive sequence in a tuple using formula.

```
my_tuple = (1, 2, 4, 5, 6)
n = len(my_tuple) + 1  # total numbers including missing
expected_sum = n * (n + 1) // 2
actual_sum = sum(my_tuple)
missing_number = expected_sum - actual_sum
print("Missing number:", missing_number)
```

TUPLES ADVANCED

Question 1: A company stores employee performance scores as tuples for different quarters:
`performance = ( (101, 85, 90), (102, 78, 88), (103, 95, 92) ).`
Find the employee with the **highest total score**.

```
performance = ((101, 85, 90), (102, 78, 88), (103, 95, 92))
highest = max(performance, key=lambda x: sum(x[1:]))
print("Employee with highest total score:", highest[0])
```

Question 2: Sales data for products across months is stored in a tuple of tuples:
`sales = ((1001, 1200, 1300), (1002, 1100, 1400), (1003, 1500, 1200)).`
Find the **product ID with maximum average monthly sales**.

```
sales = ((1001, 1200, 1300), (1002, 1100, 1400), (1003, 1500, 1200))
max_avg = max(sales, key=lambda x: sum(x[1:])/len(x[1:]))
print("Product with maximum average sales:", max_avg[0])
```

Question 3: Given tuples representing stock prices over days:
`stocks = (("AAPL", 150, 155, 160), ("GOOGL", 2700, 2720, 2710))`
Find the stock with **largest price increase between first and last day**.

```
stocks = (("AAPL", 150, 155, 160), ("GOOGL", 2700, 2720, 2710))
max_increase = max(stocks, key=lambda x: x[-1] - x[1])
print("Stock with largest increase:", max_increase[0])
```

Question 4: Employee attendance is tracked as tuples:
`attendance = ((101, 22), (102, 18), (103, 25))`
Print employees with **attendance below 20 days**.

```
attendance = ((101, 22), (102, 18), (103, 25))
low_attendance = tuple(emp[0] for emp in attendance if emp[1] < 20)
print("Employees with low attendance:", low_attendance)
```

Question 5: Server response times in milliseconds are stored as:
`servers = (("S1", 120, 150, 110), ("S2", 100, 130, 140))`
Find the server with the **minimum average response time**.

```
servers = (("S1", 120, 150, 110), ("S2", 100, 130, 140))
fastest = min(servers, key=lambda x: sum(x[1:])/len(x[1:]))
print("Fastest server:", fastest[0])
```

Question 6: Given tuples of project completion percentages:
`projects = ((1, 80, 90, 100), (2, 60, 70, 80), (3, 85, 95, 90))`
Find **projects with all months > 80%**.

```
projects = ((1, 80, 90, 100), (2, 60, 70, 80), (3, 85, 95, 90))
completed_projects = tuple(p[0] for p in projects if all(x > 80 for x in p[1:]))
print("Projects with all months >80%:", completed_projects)
```

Question 7: Transaction tuples:
`transactions = ((101, 5000), (102, 7000), (103, 6000), (104, 7000))`
Find **all transactions with the maximum amount**.

```
transactions = ((101, 5000), (102, 7000), (103, 6000), (104, 7000))
max_amount = max(transactions, key=lambda x: x[1])[1]
max_transactions = tuple(x for x in transactions if x[1] == max_amount)
print("Transactions with maximum amount:", max_transactions)
```

Question 8: System logs tuples contain `(timestamp, status_code)`:
`logs = (("10:00", 200), ("10:05", 500), ("10:10", 200))`
Count the number of **failed logs (status_code != 200)**.

```
logs = (("10:00", 200), ("10:05", 500), ("10:10", 200))
failed_count = sum(1 for log in logs if log[1] != 200)
print("Failed logs count:", failed_count)
```

Question 9: Orders tuple:
`orders = ((1, "Delivered"), (2, "Pending"), (3, "Delivered"))`
Extract all **order IDs which are pending**.

```
orders = ((1, "Delivered"), (2, "Pending"), (3, "Delivered"))
pending_orders = tuple(o[0] for o in orders if o[1] == "Pending")
print("Pending orders:", pending_orders)
```

Question 10: Tuple of employee salaries with potential bonuses:
`salaries = ((101, 50000, 5000), (102, 60000, 7000))`
Calculate **final salary (base + bonus) for each employee**.

```
salaries = ((101, 50000, 5000), (102, 60000, 7000))
final_salaries = tuple((e[0], e[1]+e[2]) for e in salaries)
print("Final salaries:", final_salaries)
```

Question 11:

A logistics company tracks the delivery times of packages for each driver every day of a week . The data is stored as a tuple of tuples:

```
delivery_times = (
    ("Driver1", 5, 7, 6, 8, 5, 6, 7),
    ("Driver2", 6, 6, 5, 7, 5, 6, 6),
    ("Driver3", 7, 8, 7, 9, 6, 8, 7)
)
```

Each inner tuple represents a driver followed by their delivery times (in hours) for 7 days.

Task: Find the driver who has the **least average delivery time** for the week.

```
delivery_times = (
    ("Driver1", 5, 7, 6, 8, 5, 6, 7),
    ("Driver2", 6, 6, 5, 7, 5, 6, 6),
    ("Driver3", 7, 8, 7, 9, 6, 8, 7)
)

fastest_driver = min(delivery_times, key=lambda x: sum(x[1:])/len(x[1:]))
print("Driver with least average delivery time:", fastest_driver[0])
```

Question 12:

A retail company stores weekly sales of multiple stores in a tuple of tuples.

Each store has its weekly sales for 4 weeks.

```
store_sales = (
    ("Store1", 12000, 13000, 12500, 14000),
    ("Store2", 15000, 14000, 13500, 14500),
    ("Store3", 11000, 12000, 11500, 12500)
)
```

Task: Identify **stores that consistently performed above 13000 in all weeks**.

```
store_sales = (
    ("Store1", 12000, 13000, 12500, 14000),
    ("Store2", 15000, 14000, 13500, 14500),
    ("Store3", 11000, 12000, 11500, 12500)
)

high_performance_stores = tuple(s[0] for s in store_sales
if all(x > 13000 for x in s[1:]))
print("Stores consistently above 13000 sales:", high_performance_stores)
```

Question 13:

A tech company tracks server uptime (in hours) for a month (30 days) for multiple servers in

a tuple of tuples.

```
servers = (
    ("Server1", *(24 for _ in range(30))),
    ("Server2", *(23 if i%5==0 else 24 for i in range(30))),
    ("Server3", *(22 + i%3 for i in range(30)))
)
```

Task: Determine which server had the **most number of full uptime days (24 hours)**.

```
servers = (
    ("Server1", *(24 for _ in range(30))),
    ("Server2", *(23 if i%5==0 else 24 for i in range(30))),
    ("Server3", *(22 + i%3 for i in range(30)))
)

full_uptime_server = max(servers, key=lambda x: sum(1 for h in x[1:] if h==24))
print("Server with most full uptime days:", full_uptime_server[0])
```

Question 14:

A company stores employee quarterly performance scores in tuples,

where each employee has multiple KPI scores.

```
employees = (
    (101, 80, 85, 90, 95),
    (102, 70, 75, 80, 85),
    (103, 90, 92, 88, 91),
    (104, 85, 87, 89, 90)
)
```

Task: Find **all employees whose average KPI score is above 90**.

```
employees = (
    (101, 80, 85, 90, 95),
    (102, 70, 75, 80, 85),
    (103, 90, 92, 88, 91),
    (104, 85, 87, 89, 90)
)

top_employees = tuple(e[0] for e in employees if sum(e[1:])/len(e[1:]) > 90)
print("Employees with average KPI > 90:", top_employees)
```

Question 15:

An online marketplace tracks product ratings from multiple customers.

Each tuple represents `(product_id, rating1, rating2, ..., ratingN)`.

```
products = (
    (201, 5, 4, 5, 3, 4),
    (202, 4, 4, 4, 5, 4),
    (203, 3, 2, 4, 3, 3)
)
```

Task: Identify the **product with the highest average rating**.

```
products = (
    (201, 5, 4, 5, 3, 4),
    (202, 4, 4, 4, 5, 4),
    (203, 3, 2, 4, 3, 3)
)

best_product = max(products, key=lambda x: sum(x[1:])/len(x[1:]))
print("Product with highest average rating:", best_product[0])
```

Question 16:

A logistics company stores tuples of shipments,

where each shipment contains `(shipment_id, weight, distance, delivery_time)`

```
shipments = (
    (301, 100, 500, 48),
    (302, 120, 400, 36),
    (303, 90, 450, 40),
    (304, 110, 550, 50)
)
```

Task: Identify the shipment with the **best efficiency**, defined as `(weight*distance)/delivery_time`.

```
shipments = (
    (301, 100, 500, 48),
    (302, 120, 400, 36),
    (303, 90, 450, 40),
    (304, 110, 550, 50)
)

best_efficiency = max(shipments, key=lambda x: (x[1]*x[2])/x[3])
print("Shipment with best efficiency:", best_efficiency[0])
```

Question 17:

A company tracks daily working hours of employees over a month.

The tuple stores `(employee_id, day1, day2, ..., day30).`

```
work_hours = (
    (401, *(8 for _ in range(30))),
    (402, *(7 + i%2 for i in range(30))),
    (403, *(6 + i%3 for i in range(30)))
)
```

Task: Find all employees who **never worked less than 7 hours on any day**.

```
work_hours = (
    (401, *(8 for _ in range(30))),
    (402, *(7 + i%2 for i in range(30))),
    (403, *(6 + i%3 for i in range(30)))
)

consistent_employees = tuple(e[0] for e in work_hours
if all(h >= 7 for h in e[1:]))
print("Employees never working less than 7 hours:", consistent_employees)
```

Question 18:

An e-commerce platform tracks tuple data of

`(customer_id, total_orders, total_amount_spent, loyalty_points)`

```
customers = (
    (501, 10, 5000, 100),
    (502, 15, 8000, 150),
    (503, 5, 3000, 50)
)
```

Task: Find **customers who spent above average amount AND have more than 100 loyalty points**.

```
customers = (
    (501, 10, 5000, 100),
    (502, 15, 8000, 150),
    (503, 5, 3000, 50)
)

avg_spent = sum(c[2] for c in customers)/len(customers)
premium_customers = tuple(c[0] for c in customers
if c[2] > avg_spent and c[3] > 100)
print("Premium customers:", premium_customers)
```

Question 19:

A hospital tracks patient vital signs over 3 days as tuples:

`(patient_id, day1_temp, day2_temp, day3_temp)`

```
patients = (
    (601, 98.6, 99.1, 98.9),
    (602, 100.5, 101.0, 100.0),
    (603, 97.5, 97.8, 98.0)
)
```

Task: Find **patients who had fever (>100°F) on any day**.

```
patients = (
    (601, 98.6, 99.1, 98.9),
    (602, 100.5, 101.0, 100.0),
    (603, 97.5, 97.8, 98.0)
)

fever_patients = tuple(p[0] for p in patients if any(t > 100 for t in p[1:]))
print("Patients with fever:", fever_patients)
```

Question 20:

A manufacturing company stores tuples of production units per shift

`(shift_id, machine1, machine2, machine3)`

```
shifts = (
    (701, 100, 120, 110),
    (702, 90, 130, 115),
    (703, 105, 125, 120)
)
```

Task: Identify the **shift with highest total production**.

```
shifts = (
    (701, 100, 120, 110),
    (702, 90, 130, 115),
    (703, 105, 125, 120)
)

best_shift = max(shifts, key=lambda x: sum(x[1:]))
print("Shift with highest total production:", best_shift[0])
```

Question 21:

A multinational company tracks quarterly revenue (in $ thousands) for

multiple branches across regions. Data is stored as:

```
branches = (
    ("Branch1", ("North", 120, 130, 125, 140)),
    ("Branch2", ("South", 150, 140, 135, 145)),
    ("Branch3", ("East", 110, 120, 115, 125)),
    ("Branch4", ("West", 130, 135, 140, 145))
)
```

Task: Identify the **region with the branch that had the highest total quarterly revenue**.

```
branches = (
    ("Branch1", ("North", 120, 130, 125, 140)),
    ("Branch2", ("South", 150, 140, 135, 145)),
    ("Branch3", ("East", 110, 120, 115, 125)),
    ("Branch4", ("West", 130, 135, 140, 145))
)

highest_branch = max(branches, key=lambda x: sum(x[1][1:]))
print("Region with highest revenue branch:", highest_branch[1][0])
```

Question 22:

An IT company tracks developer productivity over multiple sprints.

Each tuple is `(developer_id, ("Sprint1", tasks_completed),`

("Sprint2", tasks_completed), ...).

```
developers = (
    (101, ("Sprint1", 15), ("Sprint2", 18), ("Sprint3", 20)),
    (102, ("Sprint1", 12), ("Sprint2", 20), ("Sprint3", 19)),
    (103, ("Sprint1", 20), ("Sprint2", 22), ("Sprint3", 25))
)
```

Task: Find the **developer with the most consistent performance**

(lowest standard deviation across sprints).

```
import statistics

developers = (
    (101, ("Sprint1", 15), ("Sprint2", 18), ("Sprint3", 20)),
    (102, ("Sprint1", 12), ("Sprint2", 20), ("Sprint3", 19)),
    (103, ("Sprint1", 20), ("Sprint2", 22), ("Sprint3", 25))
)

consistency = min(developers, key=lambda x: statistics.stdev([s[1] for s in x[1:]]))
print("Developer with most consistent performance:", consistency[0])
```

Question 23:

A hospital tracks patients' daily readings for 7 days.

Each tuple is (patient_id, (day1_temp, day2_temp, ..., day7_temp)).

```
patients = (
    (201, (98.6, 99.1, 100.2, 98.9, 99.5, 100.5, 98.7)),
    (202, (97.8, 98.2, 97.5, 98.0, 98.5, 98.1, 97.9)),
    (203, (100.1, 101.0, 100.5, 101.2, 100.8, 101.5, 100.9))
)
```

Task: Identify **patients with fever (>100°F) on at least 3 days**.

```
patients = (
    (201, (98.6, 99.1, 100.2, 98.9, 99.5, 100.5, 98.7)),
    (202, (97.8, 98.2, 97.5, 98.0, 98.5, 98.1, 97.9)),
    (203, (100.1, 101.0, 100.5, 101.2, 100.8, 101.5, 100.9))
)

fever_patients = tuple(p[0] for p in patients if sum(1 for t in p[1] if t>100) >= 3)
print("Patients with fever on >=3 days:", fever_patients)
```

Question 24:

A manufacturing plant tracks production from multiple machines over a week. Data:

```
machines = (
    ("M1", (100, 120, 110, 115, 130, 125, 140)),
    ("M2", (90, 100, 95, 105, 110, 100, 115)),
    ("M3", (120, 130, 125, 135, 140, 150, 145))
)
```

Task: Identify **machines that exceeded 130 units production on any day**.

```
machines = (
    ("M1", (100, 120, 110, 115, 130, 125, 140)),
    ("M2", (90, 100, 95, 105, 110, 100, 115)),
    ("M3", (120, 130, 125, 135, 140, 150, 145))
)

high_output_machines = tuple(m[0] for m in machines if any(x>130 for x in m[1]))
print("Machines exceeding 130 units:", high_output_machines)
```

Question 25:

A university tracks student marks for 4 subjects. Data is stored as:

```
students = (
    (301, "Alice", (85, 90, 78, 92)),
    (302, "Bob", (88, 76, 85, 80)),
    (303, "Charlie", (95, 92, 90, 96))
)
```

Task: Find students who **scored above 90 in at least 2 subjects**.

```
students = (
    (301, "Alice", (85, 90, 78, 92)),
    (302, "Bob", (88, 76, 85, 80)),
    (303, "Charlie", (95, 92, 90, 96))
)

top_students = tuple(s[1] for s in students if sum(1 for mark in s[2] if mark>90) >=2
print("Students scoring >90 in at least 2 subjects:", top_students)
```

Question 26:

A delivery company tracks distance (km) covered by drivers per day for a week. Data:

```
drivers = (
    ("D1", (120, 130, 115, 140, 135, 150, 145)),
    ("D2", (100, 110, 105, 120, 115, 130, 125)),
    ("D3", (150, 160, 155, 165, 170, 160, 175))
)
```

Task: Find **driver with maximum total distance**.

```
drivers = (
    ("D1", (120, 130, 115, 140, 135, 150, 145)),
    ("D2", (100, 110, 105, 120, 115, 130, 125)),
    ("D3", (150, 160, 155, 165, 170, 160, 175))
)

max_distance_driver = max(drivers, key=lambda x: sum(x[1]))
print("Driver with maximum total distance:", max_distance_driver[0])
```

Question 27:

A software firm stores tuples of project durations in weeks `(project_id, (module1_weeks, module2_weeks, module3_weeks))`.

```
projects = (
    (401, (4, 5, 3)),
    (402, (6, 5, 4)),
    (403, (3, 2, 5))
)
```

Task: Identify **projects that took more than 12 weeks in total**.

```
projects = (
    (401, (4, 5, 3)),
    (402, (6, 5, 4)),
    (403, (3, 2, 5))
)

long_projects = tuple(p[0] for p in projects if sum(p[1]) > 12)
print("Projects taking >12 weeks:", long_projects)
```

Question 28:

A finance company tracks quarterly profits `(company_id, (Q1, Q2, Q3, Q4))`.

```
companies = (
    (501, (20000, 25000, 22000, 24000)),
    (502, (30000, 28000, 29000, 31000)),
    (503, (15000, 18000, 17000, 16000))
)
```

Task: Find the **company with maximum profit growth** (difference between Q4 and Q1).

```
companies = (
    (501, (20000, 25000, 22000, 24000)),
    (502, (30000, 28000, 29000, 31000)),
    (503, (15000, 18000, 17000, 16000))
)

max_growth_company = max(companies, key=lambda x: x[1][-1] - x[1][0])
print("Company with maximum
```

Question 29:

A logistics company tracks fuel consumption `(vehicle_id, (day1, day2, ..., day7))`.

```
vehicles = (
    ("V1", (15, 16, 15, 14, 17, 16, 15)),
    ("V2", (14, 15, 14, 15, 14, 15, 14)),
    ("V3", (16, 18, 17, 19, 18, 20, 17))
)
```

Task: Find **vehicles that consumed more than 18 liters on any day**.

```
vehicles = (
    ("V1", (15, 16, 15, 14, 17, 16, 15)),
    ("V2", (14, 15, 14, 15, 14, 15, 14)),
    ("V3", (16, 18, 17, 19, 18, 20, 17))
)

high_fuel_vehicles = tuple(v[0] for v in vehicles if any(f>18 for f in v[1]))
print("Vehicles consuming >18 liters:", high_fuel_vehicles)
```

Question 30:

A telecom company tracks customer call durations `(customer_id, (call1, call2, call3,...)).`

```
customers = (
    (601, (12, 15, 20, 10)),
    (602, (5, 6, 4, 5)),
    (603, (25, 30, 28, 32))
)
```

Task: Find **customers with average call duration above 20 minutes**.

```
customers = (
    (601, (12, 15, 20, 10)),
    (602, (5, 6, 4, 5)),
    (603, (25, 30, 28, 32))
)

long_call_customers = tuple(c[0] for c in customers if sum(c[1])/len(c[1]) > 20)
print("Customers with average call duration >20 mins:", long_call_customers)
```

UNIT-5 SETS

Program 1: Set Union, Intersection, and Difference

Aim:
To demonstrate set operations (union, intersection, difference).

Algorithm:

1. Start.
2. Define two sets.
3. Perform `union()`, `intersection()`, and `difference()`.
4. Print results.

Code:

```
A = {1, 2, 3, 4}
B = {3, 4, 5, 6}

print("Union:", A | B)
print("Intersection:", A & B)
print("Difference (A-B):", A - B)
```

Sample Output:

```
Union: {1, 2, 3, 4, 5, 6}
Intersection: {3, 4}
```

```
Difference (A-B): {1, 2}
```

Result:
Set operations executed successfully.

Program 2: Symmetric Difference of Sets

Aim:
To find elements present in either of two sets but not in both.

Algorithm:

1. Start.
2. Use `^` operator or `symmetric_difference()`.
3. Print result.

Code:

```
A = {10, 20, 30, 40}
B = {30, 40, 50, 60}

print("Symmetric Difference:", A ^ B)
```

Sample Output:

```
Symmetric Difference: {10, 20, 50, 60}
```

Result:
Symmetric difference found successfully.

Program 3: Cartesian Product of Two Sets

Aim:
To generate Cartesian product of two sets using list comprehension.

Algorithm:

1. Start.
2. Take two sets.
3. For each element in first set, pair with each element in second set.
4. Store as tuple list.

Code:

```
A = {1, 2}
B = {'a', 'b'}
cartesian = [(x, y) for x in A for y in B]
print("Cartesian Product:", cartesian)
```

Sample Output:

```
Cartesian Product: [(1, 'a'), (1, 'b'), (2, 'a'), (2, 'b')]
```

Result:
Cartesian product generated successfully.

Program 4: Frequency of Elements using Set

Aim:
To count frequency of elements in a list using set for uniqueness.

Algorithm:

1. Start.
2. Convert list into set for unique elements.
3. For each unique element, count occurrences in original list.
4. Print frequencies.

Code:

```
arr = [1, 2, 2, 3, 4, 3, 3, 5]
unique = set(arr)

for item in unique:
    print(f"{item} occurs {arr.count(item)} times")
```

Sample Output:

```
1 occurs 1 times
2 occurs 2 times
3 occurs 3 times
4 occurs 1 times
5 occurs 1 times
```

Result:
Frequencies counted successfully using set.

Program 5: Check Subset and Superset Relationship

Aim:
To check whether a set is subset or superset of another set.

Algorithm:

1. Start.
2. Use `issubset()` and `issuperset()`.
3. Print results.

Code:

```
A = {1, 2, 3}
B = {1, 2, 3, 4, 5}

print("A is subset of B:", A.issubset(B))
print("B is superset of A:", B.issuperset(A))
```

Sample Output:

```
A is subset of B: True
B is superset of A: True
```

Result:
Subset and superset relationships verified successfully.

Program 6: Frozen Set Demonstration

Aim:
To demonstrate usage of immutable sets (frozenset).

Algorithm:

1. Start.
2. Create `frozenset` from list.
3. Try performing operations like union and intersection.
4. Show immutability.

Code:

```
fs = frozenset([1, 2, 3, 4])
normal_set = {3, 4, 5, 6}

print("Frozen Set:", fs)
print("Union:", fs | normal_set)
print("Intersection:", fs & normal_set)
# fs.add(7)  # This will cause an error
```

Sample Output:

```
Frozen Set: frozenset({1, 2, 3, 4})
Union: {1, 2, 3, 4, 5, 6}
Intersection: {3, 4}
```

Result:
Frozen set demonstrated successfully.

Program 7: Remove Multiple Elements from a Set

Aim:
To remove multiple specific elements from a set.

Algorithm:

1. Start.
2. Use `discard()` or set difference.
3. Print final set.

Code:

```
A = {1, 2, 3, 4, 5, 6}
remove_items = {2, 4, 6}
A = A - remove_items
print("After Removal:", A)
```

Sample Output:

```
After Removal: {1, 3, 5}
```

Result:
Multiple elements removed successfully.

Program 8: Extract Unique Words from a String using Set

Aim:
To extract unique words from a given sentence.

Algorithm:

1. Start.
2. Split string into words.
3. Convert to set for uniqueness.
4. Print unique words.

Code:

```
text = "python is great and python is easy to learn"
words = text.split()
unique_words = set(words)
print("Unique Words:", unique_words)
```

Sample Output:

```
Unique Words: {'to', 'is', 'python', 'learn', 'great', 'easy', 'and'}
```

Result:
Unique words extracted successfully.

Problem 9:
You have two sets of employee IDs:

```
engineering = {101, 102, 103, 104, 105}
marketing = {104, 105, 106, 107}
```

Task: Find employees who are **only in one department** but not in both.

engineering = {101, 102, 103, 104, 105}

marketing = {104, 105, 106, 107}

# Employees only in one department (symmetric difference)

only_one_dept = engineering ^ marketing

print("Employees only in one department:", only_one_dept)

Problem 10:
You are tracking inventory in two warehouses:

```
warehouse_A = {"Laptop", "Mouse", "Keyboard", "Monitor"}
warehouse_B = {"Monitor", "Mouse", "Printer", "Scanner"}
```

Task: Find **items common in both warehouses**, **items unique to warehouse A**,

and **all unique items across both warehouses**.

warehouse_A = {"Laptop", "Mouse", "Keyboard", "Monitor"}

warehouse_B = {"Monitor", "Mouse", "Printer", "Scanner"}

common_items = warehouse_A & warehouse_B

unique_A = warehouse_A - warehouse_B

all_unique = warehouse_A | warehouse_B

print("Common items:", common_items)

print("Unique to A:", unique_A)

print("All unique items:", all_unique)

Problem 11: Average Marks of Students
Data:

```
students = [
    ("Alice", [85, 90, 78]),
    ("Bob", [75, 80, 85]),
    ("Charlie", [95, 92, 88])
]
```

Task: Calculate the average marks of each student.
Solution:

```
for name, marks in students:
    avg = sum(marks)/len(marks)
    print(f"{name}: Average = {avg}")
```

Problem 12: Total Sales of Each Store
Data:

```
stores = [
    ("Store1", [1200, 1300, 1250]),
    ("Store2", [1500, 1400, 1350]),
    ("Store3", [1100, 1200, 1150])
]
```

Task: Calculate total sales for each store.
Solution:

```
for store, sales in stores:
    total = sum(sales)
    print(f"{store}: Total Sales = {total}")
```

Problem 13: Filter Employees by Salary
Data:

```
employees = [
    ("E101", 50000),
    ("E102", 60000),
    ("E103", 45000),
    ("E104", 70000)
]
```

Task: List employees earning more than 55000.
Solution:

```
high_salary = [emp[0] for emp in employees if emp[1] > 55000]
print("High salary employees:", high_salary)
```

Problem 14: Flatten Nested List
Data:

```
nested_list = [[1, 2], [3, 4], [5, 6]]
```

Task: Convert it to a single flat list.
Solution:

```
flat = [item for sublist in nested_list for item in sublist]
print("Flattened list:", flat)
```

Problem 15: Highest Scoring Student
Data:

```
students = [
    ("Alice", (85, 90, 78)),
    ("Bob", (88, 76, 85)),
    ("Charlie", (95, 92, 90))
]
```

Task: Find the student with highest total marks.
Solution:

```
top_student = max(students, key=lambda s: sum(s[1]))
print("Top student:", top_student[0])
```

Problem 16: Count Products Above Threshold
Data:

```
products = [
    ("P1", [120, 130, 125]),
    ("P2", [150, 140, 135]),
    ("P3", [110, 120, 115])
]
```

Task: Count products with average sales above 125.
Solution:

```
count = sum(1 for p in products if sum(p[1])/len(p[1]) > 125)
print("Products above 125 avg:", count)
```

Problem 17: Max Temperature per City
Data:

```
cities = [
    ("CityA", [30, 32, 31, 29]),
    ("CityB", [28, 27, 26, 29]),
    ("CityC", [33, 34, 32, 35])
]
```

Task: Find the maximum temperature for each city.
Solution:

```
for city, temps in cities:
    print(f"{city}: Max Temp = {max(temps)}")
```

Problem 18: Employees with Consistent Attendance
Data:

```
attendance = [
    ("E101", [8, 8, 8, 8, 8]),
    ("E102", [8, 7, 8, 8, 8]),
    ("E103", [8, 8, 8, 8, 8])
]
```

Task: Identify employees with perfect attendance (all 8 hours).
Solution:

```
perfect = [emp[0] for emp in attendance if all(h==8 for h in emp[1])]
print("Perfect attendance:", perfect)
```

Problem 19: Students Passing All Subjects
Data:

```
students = [
    ("Alice", [85, 90, 78]),
    ("Bob", [55, 60, 65]),
    ("Charlie", [75, 80, 85])
]
```

Task: List students who scored >= 60 in all subjects.
Solution:

```
passed = [s[0] for s in students if all(m >= 60 for m in s[1])]
print("Students passing all subjects:", passed)
```

Problem 20: Average Quarterly Profit per Company
Data:

```
companies = [
    ("C1", [20000, 25000, 22000, 24000]),
    ("C2", [30000, 28000, 29000, 31000]),
    ("C3", [15000, 18000, 17000, 16000])
]
```

Task: Find the average quarterly profit for each company.
Solution:

```
for c, profits in companies:
    avg = sum(profits)/len(profits)
    print(f"{c}: Avg Profit = {avg}")
```

Problem 21: Branch Revenue Analysis
Data:

```
branches = (
    ("Branch1", ("North", 120, 130, 125, 140)),
    ("Branch2", ("South", 150, 140, 135, 145)),
    ("Branch3", ("East", 110, 120, 115, 125)),
    ("Branch4", ("West", 130, 135, 140, 145))
)
```

Task: Identify the **region with the branch that had the highest total quarterly revenue**.
Solution:

```
highest_branch = max(branches, key=lambda x: sum(x[1][1:]))
print("Region with highest revenue branch:", highest_branch[1][0])
```

Problem 22: Developer Productivity Consistency
Data:

```
import statistics

developers = (
    (101, ("Sprint1", 15), ("Sprint2", 18), ("Sprint3", 20)),
    (102, ("Sprint1", 12), ("Sprint2", 20), ("Sprint3", 19)),
    (103, ("Sprint1", 20), ("Sprint2", 22), ("Sprint3", 25))
)
```

Task: Find the **developer with the most consistent performance** (lowest standard

deviation across sprints).
Solution:

```
consistency = min(developers, key=lambda x: statistics.stdev([s[1] for s in x[1:]]))
print("Most consistent developer:", consistency[0])
```

Problem 23: Patient Fever Monitoring
Data:

```
patients = (
    (201, (98.6, 99.1, 100.2, 98.9, 99.5, 100.5, 98.7)),
    (202, (97.8, 98.2, 97.5, 98.0, 98.5, 98.1, 97.9)),
    (203, (100.1, 101.0, 100.5, 101.2, 100.8, 101.5, 100.9))
)
```

Task: Identify **patients with fever (>100°F) on at least 3 days**.
Solution:

```
fever_patients = tuple(p[0] for p in patients if sum(1 for t in p[1] if t>100) >= 3)
print("Patients with fever on >=3 days:", fever_patients)
```

Problem 24: Machine Production Monitoring
Data:

```
machines = (
    ("M1", (100, 120, 110, 115, 130, 125, 140)),
    ("M2", (90, 100, 95, 105, 110, 100, 115)),
    ("M3", (120, 130, 125, 135, 140, 150, 145))
)
```

Task: Identify **machines that exceeded 130 units production on any day**.
Solution:

```
high_output_machines = tuple(m[0] for m in machines if any(x>130 for x in m[1]))
print("Machines exceeding 130 units:", high_output_machines)
```

Problem 25: Student Top Performance
Data:

```
students = (
    (301, "Alice", (85, 90, 78, 92)),
    (302, "Bob", (88, 76, 85, 80)),
    (303, "Charlie", (95, 92, 90, 96))
)
```

Task: Find **students who scored above 90 in at least 2 subjects**.
Solution:

```
top_students = tuple(s[1] for s in students
if sum(1 for mark in s[2] if mark>90) >=2)
print("Students scoring >90 in at least 2 subjects:", top_students)
```

Problem 26: Driver Total Distance
Data:

```
drivers = (
    ("D1", (120, 130, 115, 140, 135, 150, 145)),
    ("D2", (100, 110, 105, 120, 115, 130, 125)),
    ("D3", (150, 160, 155, 165, 170, 160, 175))
)
```

Task: Find the **driver with maximum total distance**.
Solution:

```
max_distance_driver = max(drivers, key=lambda x: sum(x[1]))
print("Driver with maximum total distance:", max_distance_driver[0])
```

Problem 27: Project Duration Analysis
Data:

```
projects = (
    (401, (4, 5, 3)),
    (402, (6, 5, 4)),
    (403, (3, 2, 5))
)
```

Task: Identify **projects that took more than 12 weeks in total**.
Solution:

```
long_projects = tuple(p[0] for p in projects if sum(p[1]) > 12)
print("Projects taking >12 weeks:", long_projects)
```

Problem 28: Company Profit Growth
Data:

```
companies = (
    (501, (20000, 25000, 22000, 24000)),
    (502, (30000, 28000, 29000, 31000)),
    (503, (15000, 18000, 17000, 16000))
)
```

Task: Find the **company with maximum profit growth** (difference between Q4 and Q1).
Solution:

```
max_growth_company = max(companies, key=lambda x: x[1][-1] - x[1][0])
print("Company with maximum profit growth:", max_growth_company[0])
```

Problem 29: Vehicle Fuel Consumption
Data:

```
vehicles = (
    ("V1", (15, 16, 15, 14, 17, 16, 15)),
    ("V2", (14, 15, 14, 15, 14, 15, 14)),
    ("V3", (16, 18, 17, 19, 18, 20, 17))
)
```

Task: Find **vehicles that consumed more than 18 liters on any day**.
Solution:

```
high_fuel_vehicles = tuple(v[0] for v in vehicles if any(f>18 for f in v[1]))
print("Vehicles consuming >18 liters:", high_fuel_vehicles)
```

Problem 30: Customer Average Call Duration
Data:

```
customers = (
    (601, (12, 15, 20, 10)),
    (602, (5, 6, 4, 5)),
    (603, (25, 30, 28, 32))
)
```

Task: Find **customers with average call duration above 20 minutes**.
Solution:

```
long_call_customers = tuple(c[0] for c in customers if sum(c[1])/len(c[1]) > 20)
print("Customers with average call duration >20 mins:", long_call_customers)
```

Problem 31: Employee Quarterly Performance
Data:

```
employees = (
    (101, (80, 85, 90, 95)),
    (102, (70, 75, 80, 85)),
    (103, (90, 92, 88, 91)),
    (104, (85, 87, 89, 90))
)
```

Task: Find **employees whose average KPI score is above 90**.
Solution:

```
top_employees = tuple(e[0] for e in employees if sum(e[1])/len(e[1]) > 90)
print("Employees with avg KPI >90:", top_employees)
```

Problem 32: Shipment Efficiency
Data:

```
shipments = (
    (301, 100, 500, 48),
    (302, 120, 400, 36),
    (303, 90, 450, 40),
    (304, 110, 550, 50)
)
```

Task: Identify the **shipment with the best efficiency** `(weight*distance)/delivery_time`.
Solution:

```
best_shipment = max(shipments, key=lambda x: (x[1]*x[2])/x[3])
print("Best shipment ID:", best_shipment[0])
```

Problem 33: Student Attendance Monitoring
Data:

```
students = (
    (401, (1,1,1,1,1,1,1)),
    (402, (1,0,1,1,1,1,1)),
    (403, (1,1,1,1,1,1,1))
)
```

Task: Identify **students with perfect attendance** (all 1s).
Solution:

```
perfect_attendance = tuple(s[0] for s in students if all(d==1 for d in s[1]))
print("Students with perfect attendance:", perfect_attendance)
```

Problem 34: Delivery Time Improvement
Data:

```
drivers = (
    ("D1", (60, 58, 55, 53, 50)),
    ("D2", (50, 52, 51, 50, 49)),
    ("D3", (70, 68, 66, 64, 62))
)
```

Task: Identify **drivers who improved delivery time every consecutive day**.

Solution:

```
improving_drivers = tuple(d[0] for d in drivers if all(d[1][i] > d[1][i+1] for i in ra
print("Drivers improving daily:", improving_drivers)
```

Problem 35: Machine Error Monitoring
Data:

```
machines = (
    ("M1", (0,0,0,0,0,0,0)),
    ("M2", (0,1,0,0,1,0,0)),
    ("M3", (0,0,0,0,0,0,0))
)
```

Task: Find **machines with zero errors for the entire week**.
Solution:

```
zero_error_machines = tuple(m[0] for m in machines if all(e==0 for e in m[1]))
print("Machines with zero errors:", zero_error_machines)
```

Problem 36: Product Sales Growth
Data:

```
products = (
    ("P1", (100, 110, 120, 130)),
    ("P2", (90, 95, 100, 105)),
    ("P3", (150, 140, 145, 150))
)
```

Task: Identify **products with consistent growth every week**.
Solution:

```
growing_products = tuple(p[0] for p in products if all(p[1][i] < p[1][i+1] for i in ra
print("Products with consistent growth:", growing_products)
```

Problem 37: Flight Delay Monitoring
Data:

```
flights = (
    ("F1", (10, 20, 35, 40, 25, 30, 45)),
    ("F2", (5, 10, 15, 20, 25, 10, 15)),
    ("F3", (30, 40, 50, 35, 20, 45, 30))
)
```

Task: Find flights with **at least 3 days of delay >30 minutes**.
Solution:

```
delayed_flights = tuple(f[0] for f in flights if sum(1 for d in f[1] if d>30)>=3)
print("Flights with 3+ days of delay:", delayed_flights)
```

Problem 38: Patient Medication Doses

Data:

```
patients = (
    (601, (2,3,2,4,3,2,3)),
    (602, (1,2,1,2,1,2,1)),
    (603, (3,4,3,5,4,3,4))
)
```

Task: Identify patients who **received more than 3 doses on any day**.
Solution:

```
high_dose_patients = tuple(p[0] for p in patients if any(d>3 for d in p[1]))
print("Patients with >3 doses:", high_dose_patients)
```

Problem 39: Developer Bug Resolution
Data:

```
developers = (
    (701, (5,12,8)),
    (702, (15,9,11)),
    (703, (10,11,12))
)
```

Task: Identify developers who **resolved more than 10 bugs in at least 2 projects**.
Solution:

```
top_bug_fixers = tuple(d[0] for d in developers if sum(1 for x in d[1] if x>10)>=2)
print("Developers with >10 bugs in 2+ projects:", top_bug_fixers)
```

Problem 40: Telecom Data Usage Monitoring
Data:

```
users = (
    (801, (2,3,5,6,4,7,3)),
    (802, (1,2,3,2,1,2,3)),
    (803, (5,6,7,6,5,8,6))
)
```

Task: Identify **users who used more than 5 GB on more than 3 days**.
Solution:

```
high_data_users = tuple(u[0] for u in users if sum(1 for x in u[1] if x>5) > 3)
print("Users with >5GB on 3+ days:", high_data_users)
```

Problem 41: Multi-Branch Sales Analysis
Data:

```
branches = (
    ("North", [("Store1", [1200,1300,1250]), ("Store2", [1500,1400,1350])]),
    ("South", [("Store3", [1100,1200,1150]), ("Store4", [1600,1550,1500])]),
    ("East", [("Store5", [900,950,1000]), ("Store6", [1200,1250,1300])])
)
```

Task: Find **the store with the highest total sales across all branches**.

Solution:

```
all_stores = [(store, sum(sales)) for branch in branches for store,
 sales in branch[1]]
top_store = max(all_stores, key=lambda x: x[1])
print("Store with highest total sales:", top_store[0])
```

Problem 42: Employee Consistency Across Projects
Data:

```
employees = (
    (101, ("ProjA", 85), ("ProjB", 88), ("ProjC", 90)),
    (102, ("ProjA", 78), ("ProjB", 82), ("ProjC", 80)),
    (103, ("ProjA", 92), ("ProjB", 95), ("ProjC", 91))
)
```

Task: Find **employees with minimal variation across all project scores** (lowest standard deviation).
Solution:

```
import statistics
consistent_employee = min(employees, key=lambda e: statistics.stdev([p[1] for p in e[
print("Most consistent employee:", consistent_employee[0])
```

Problem 43: Hospital Patient Risk Monitoring
Data:

```
patients = (
    (201, (98, 102, 101, 99, 103)),
    (202, (97, 98, 96, 97, 99)),
    (203, (100, 101, 102, 100, 99))
)
```

Task: Identify **patients with fever >100°F on at least 3 consecutive days**.
Solution:

```
risky_patients = []
for p in patients:
    temps = p[1]
    for i in range(len(temps)-2):
        if all(t>100 for t in temps[i:i+3]):
            risky_patients.append(p[0])
            break
print("Patients with consecutive fever:", risky_patients)
```

Problem 44: Multi-Machine Production Analysis
Data:

```
machines = (
    ("M1", (100, 120, 130, 125, 140)),
    ("M2", (90, 110, 95, 105, 100)),
    ("M3", (130, 135, 140, 145, 150))
)
```

Task: Identify **machines with average production >130 units**.
Solution:

```
high_avg_machines = tuple(m[0] for m in machines if sum(m[1])/len(m[1]) > 130)
print("Machines with avg >130:", high_avg_machines)
```

Problem 45: Multi-Project Developer Performance
Data:

```
developers = (
    (301, ("Proj1", 10), ("Proj2", 12), ("Proj3", 8)),
    (302, ("Proj1", 15), ("Proj2", 16), ("Proj3", 18)),
    (303, ("Proj1", 8), ("Proj2", 7), ("Proj3", 9))
)
```

Task: Find **developers who completed >10 tasks in at least 2 projects**.
Solution:

```
top_developers = tuple(d[0] for d in developers
 if sum(1 for p in d[1:] if p[1]>10)>=2)
print("Top developers:", top_developers)
```

Problem 46: Telecom Data Usage Pattern
Data:

```
users = (
    (401, (2,3,5,6,4,7,3)),
    (402, (1,2,1,2,1,2,1)),
    (403, (5,6,7,6,5,8,6))
)
```

Task: Identify **users with >5GB usage on more than 3 days**.
Solution:

```
heavy_users = tuple(u[0] for u in users if sum(1 for x in u[1] if x>5) > 3)
print("Heavy data users:", heavy_users)
```

Problem 47: Multi-Branch Profit Growth
Data:

```
branches = (
    ("North", [20000, 22000, 24000, 26000]),
    ("South", [25000, 24000, 24500, 25500]),
    ("East", [15000, 16000, 17000, 18000])
)
```

Task: Find the **branch with maximum growth** (last quarter - first quarter).
Solution:

```
max_growth_branch = max(branches, key=lambda x: x[1][-1]-x[1][0])
print("Branch with max growth:", max_growth_branch[0])
```

Problem 48: Flight Delay Tracking
Data:

```
flights = (
    ("F1", (10, 20, 35, 40, 25, 30, 45)),
    ("F2", (5, 10, 15, 20, 25, 10, 15)),
    ("F3", (30, 40, 50, 35, 20, 45, 30))
)
```

Task: Identify **flights delayed >30 min on at least 3 days**.
Solution:

```
delayed_flights = tuple(f[0] for f in flights if sum(1 for d in f[1] if d>30)>=3)
print("Flights delayed >=3 days:", delayed_flights)
```

Problem 49: Hospital Medication Analysis
Data:

```
patients = (
    (501, (2,3,2,4,3,2,3)),
    (502, (1,2,1,2,1,2,1)),
    (503, (3,4,3,5,4,3,4))
)
```

Task: Identify **patients receiving >3 doses on any day**.
Solution:

```
high_dose_patients = tuple(p[0] for p in patients if any(d>3 for d in p[1]))
print("Patients with high dose:", high_dose_patients)
```

Problem 50: Multi-Day Developer Bug Resolution
Data:

```
developers = (
    (601, (5,12,8,14)),
    (602, (15,9,11,12)),
    (603, (10,11,12,13))
)
```

Task: Identify **developers resolving >10 bugs in at least 3 days**.
Solution:

```
top_bug_fixers = tuple(d[0] for d in developers if sum(1 for x in d[1] if x>10) >=3)
print("Developers resolving >10 bugs in 3+ days:", top_bug_fixers)
```

UNIT-6 PYTHON DICTIONARY

Q1. Create a dictionary with keys: `'name'`, `'age'`, `'city'` and assign any values.
Solution:

```
person = {'name': 'Alice', 'age': 25, 'city': 'Delhi'}
```

```
print(person)
```

Output:

```
{'name': 'Alice', 'age': 25, 'city': 'Delhi'}
```

Q2. Access the value of `'city'` in the above dictionary.

```
print(person['city'])
```

Output:

```
Delhi
```

Q3. Update `'age'` to 26.

```
person['age'] = 26
print(person)
```

Output:

```
{'name': 'Alice', 'age': 26, 'city': 'Delhi'}
```

Q4. Add a new key `'country'` with value `'India'`.

```
person['country'] = 'India'
print(person)
```

Output:

```
{'name': 'Alice', 'age': 26, 'city': 'Delhi', 'country': 'India'}
```

Q5. Delete the `'city'` key from the dictionary.

```
person.pop('city')
print(person)
```

Output:

```
{'name': 'Alice', 'age': 26, 'country': 'India'}
```

Q6. Check if `'age'` exists in the dictionary.

```
print('age' in person)
```

Output:

```
True
```

Q7. Get all keys from the dictionary.

```
print(person.keys())
```

Output:

```
dict_keys(['name', 'age', 'country'])
```

Q8. Get all values from the dictionary.

```
print(person.values())
```

Output:

```
dict_values(['Alice', 26, 'India'])
```

Q9. Iterate over dictionary and print key-value pairs.

```
for k, v in person.items():
    print(k, v)
```

Output:

```
name Alice
age 26
country India
```

Q10. Create a dictionary using `dict()` function.

```
student = dict(name='Bob', grade='A', marks=90)
print(student)
```

Output:

```
{'name': 'Bob', 'grade': 'A', 'marks': 90}
```

Q11. Merge two dictionaries `a = {'x':1, 'y':2}` and `b = {'y':3, 'z':4}` using Python 3.10+ syntax

```
a = {'x':1, 'y':2}
b = {'y':3, 'z':4}
merged = a | b
print(merged)
```

Output:

```
{'x': 1, 'y': 3, 'z': 4}
```

Q12. Create a dictionary of squares for numbers 1 to 5 using dictionary comprehension.

```
squares = {x: x**2 for x in range(1, 6)}
print(squares)
```

Output:

```
{1: 1, 2: 4, 3: 9, 4: 16, 5: 25}
```

Q13. Invert a dictionary `{'a':1, 'b':2, 'c':3}` (keys ↔ values).

```
d = {'a':1, 'b':2, 'c':3}
inverted = {v:k for k,v in d.items()}
print(inverted)
```

Output:

```
{1: 'a', 2: 'b', 3: 'c'}
```

Q14. Sort a dictionary by its values.

```
d = {'b': 3, 'a': 1, 'c': 2}
sorted_by_value = dict(sorted(d.items(), key=lambda x: x[1]))
print(sorted_by_value)
```

Output:

```
{'a': 1, 'c': 2, 'b': 3}
```

Q15. Count frequency of each character in a string `'hello'` using dictionary comprehension.

```
s = 'hello'
freq = {ch: s.count(ch) for ch in s}
print(freq)
```

Output:

```
{'h': 1, 'e': 1, 'l': 2, 'o': 1}
```

Q16. Create a nested dictionary for students with their marks.

```
students = {
    'S1': {'name':'Alice', 'marks':85},
    'S2': {'name':'Bob', 'marks':90}
}
print(students)
```

Output:

```
{'S1': {'name': 'Alice', 'marks': 85}, 'S2': {'name': 'Bob', 'marks': 90}}
```

Q17. Find all keys common to two dictionaries `a` and `b`.

```
a = {'x':1, 'y':2, 'z':3}
b = {'y':4, 'z':5, 'w':6}
common_keys = a.keys() & b.keys()
print(common_keys)
```

Output:

```
{'y', 'z'}
```

Q18. Remove duplicates from dictionary values if they are lists.

```
d = {'x':[1,2,2], 'y':[3,3,4]}
unique_vals = {k:list(set(v)) for k,v in d.items()}
print(unique_vals)
```

Output:

```
{'x': [1, 2], 'y': [3, 4]}
```

Q19. Rename a key `'name'` to `'fullname'` in a dictionary.

```
d = {'name':'John', 'age':25}
d['fullname'] = d.pop('name')
print(d)
```

Output:

```
{'age': 25, 'fullname': 'John'}
```

Q20. Combine two lists into a dictionary using `zip()`.

```
keys = ['a', 'b', 'c']
values = [1, 2, 3]
combined = dict(zip(keys, values))
print(combined)
```

Output:

```
{'a': 1, 'b': 2, 'c': 3}
```

Q21. Explain what happens when you assign one dictionary to another and modify it.

```
a = {'x': 1, 'y': 2}
b = a
b['x'] = 10
print(a)
print(b)
```

Output:

```
{'x': 10, 'y': 2}
{'x': 10, 'y': 2}
```

Explanation: Both `a` and `b` reference the same dictionary (shallow copy), so changes reflect in both.

Q22. Create a true copy of a dictionary to avoid reference issues.

```
import copy
a = {'x':1, 'y':[2,3]}
b = copy.deepcopy(a)
b['y'].append(4)
print(a)
print(b)
```

Output:

```
{'x': 1, 'y': [2, 3]}
{'x': 1, 'y': [2, 3, 4]}
```

Explanation: `deepcopy` creates an independent copy, so nested mutable objects don't affect the original.

Q23. Why can't lists be dictionary keys?

```
# a = {[1,2]: "value"}  # This will raise TypeError
```

Answer: Lists are mutable and unhashable, so they cannot be used as dictionary keys.

Only immutable objects (like int, string, tuple) can be keys.

Q24. Check if a value exists in a dictionary.

```
d = {'a': 1, 'b': 2}
print(2 in d.values())
```

Output:

```
True
```

Q25. Merge two dictionaries and sum values if keys are common.

```
a = {'x': 2, 'y': 3}
b = {'y': 4, 'z': 5}
merged = {k: a.get(k,0)+b.get(k,0) for k in set(a)|set(b)}
print(merged)
```

Output:

```
{'x': 2, 'y': 7, 'z': 5}
```

Q26. Demonstrate dictionary key uniqueness.

```
d = {'a':1, 'a':2}
print(d)
```

Output:

```
{'a': 2}
```

Explanation: Duplicate keys are not allowed; the last value overwrites previous ones.

Q27. Demonstrate that dictionary keys must be immutable.

```
# invalid: d = {[1,2]: "value"}
# valid: d = {(1,2): "value"}
d = {(1,2): "ok"}
print(d)
```

Output:

```
{(1, 2): 'ok'}
```

Q28. Using `setdefault()` to avoid KeyError.

```
d = {'a':1}
print(d.setdefault('b', 0))
print(d)
```

Output:

```
0
{'a': 1, 'b': 0}
```

Explanation: `setdefault` adds a key with default value if it doesn't exist.

Q29. Dictionary comprehension with conditional.

```
d = {x: x*x for x in range(6) if x%2==0}
print(d)
```

Output:

```
{0: 0, 2: 4, 4: 16}
```

Q30. Swap keys and values in a dictionary with duplicate values.

```
d = {'a':1, 'b':1, 'c':2}
inverted = {}
for k,v in d.items():
    inverted.setdefault(v, []).append(k)
print(inverted)
```

Output:

```
{1: ['a', 'b'], 2: ['c']}
```

Explanation: When values are not unique, we store keys in a list to prevent data loss.

Q31. Use `defaultdict` to count character frequency in a string.

```
from collections import defaultdict

s = "banana"
freq = defaultdict(int)
for ch in s:
    freq[ch] += 1
print(dict(freq))
```

Output:

```
{'b': 1, 'a': 3, 'n': 2}
```

Q32. Use `Counter` to find the most common element in a list.

```
from collections import Counter

lst = [1,2,2,3,3,3,4]
c = Counter(lst)
print(c.most_common(1))
```

Output:

```
[(3, 3)]
```

Q33. Merge a list of dictionaries into one.

```
list_dicts = [{'a':1}, {'b':2}, {'c':3}]
merged = {k:v for d in list_dicts for k,v in d.items()}
print(merged)
```

Output:

```
{'a': 1, 'b': 2, 'c': 3}
```

Q34. Convert JSON string to dictionary.

```
import json

json_str = '{"name": "Alice", "age": 25}'
data = json.loads(json_str)
print(data)
```

Output:

```
{'name': 'Alice', 'age': 25}
```

Q35. Flatten a nested dictionary.

```
nested = {'A': {'x':1, 'y':2}, 'B': {'x':3, 'y':4}}
flat = {f"{outer}_{inner}": val for outer, inner_dict in nested.items()
for inner, val in inner_dict.items()}
print(flat)
```

Output:

```
{'A_x': 1, 'A_y': 2, 'B_x': 3, 'B_y': 4}
```

Q36. Group values by key from a list of tuples.

```
data = [('a',1), ('b',2), ('a',3)]
grouped = {}
for k,v in data:
    grouped.setdefault(k, []).append(v)
print(grouped)
```

Output:

```
{'a': [1, 3], 'b': [2]}
```

Q37. Dictionary comprehension with multiple conditions.

```
d = {x: x**2 for x in range(10) if x%2==0 and x>4}
print(d)
```

Output:

```
{6: 36, 8: 64}
```

Q38. Reverse mapping a dictionary (values → keys) with unique values.

```
d = {'x':10, 'y':20, 'z':30}
rev = {v:k for k,v in d.items()}
print(rev)
```

Output:

```
{10: 'x', 20: 'y', 30: 'z'}
```

Q39. Sort a nested dictionary by the inner dictionary's value.

```
data = {'A': {'score': 90}, 'B': {'score': 75}, 'C': {'score':85}}
sorted_data = dict(sorted(data.items(), key=lambda x: x[1]['score']))
print(sorted_data)
```

Output:

```
{'B': {'score': 75}, 'C': {'score': 85}, 'A': {'score': 90}}
```

Q40. Merge two dictionaries and keep maximum value for common keys.

```
a = {'x':5, 'y':7}
b = {'y':10, 'z':3}
merged = {k: max(a.get(k,0), b.get(k,0)) for k in set(a)|set(b)}
print(merged)
```

Output:

```
{'x': 5, 'y': 10, 'z': 3}
```

Q41. Store student records with marks and calculate total marks.

```
students = {
    'Alice': {'Math':90, 'Science':85},
    'Bob': {'Math':75, 'Science':80}
}
totals = {name: sum(marks.values()) for name, marks in students.items()}
print(totals)
```

Output:

```
{'Alice': 175, 'Bob': 155}
```

Q42. Count frequency of words in a text using dictionary comprehension.

```
text = "apple banana apple orange banana apple"
words = text.split()
freq = {w: words.count(w) for w in set(words)}
print(freq)
```

Output:

```
{'orange': 1, 'banana': 2, 'apple': 3}
```

Q43. Track inventory updates in a store.

```
inventory = {'apple': 50, 'banana': 20}
sales = {'apple': 5, 'banana': 2, 'orange': 10}
for item, sold in sales.items():
    inventory[item] = inventory.get(item, 0) - sold
print(inventory)
```

Output:

```
{'apple': 45, 'banana': 18, 'orange': -10}
```

Q44. Create a dictionary to map employees to multiple departments.

```
data = [('Alice','HR'),('Bob','IT'),('Alice','Finance')]
emp_dept = {}
for emp, dept in data:
    emp_dept.setdefault(emp, []).append(dept)
print(emp_dept)
```

Output:

```
{'Alice': ['HR', 'Finance'], 'Bob': ['IT']}
```

Q45. Find students scoring above 80 in Math from a nested dictionary.

```
students = {'Alice':{'Math':90,'Science':70}, 'Bob':{'Math':75,'Science':85}}
high_math = {name: marks['Math'] for name, marks in students.items() if marks['Math']>
print(high_math)
```

Output:

```
{'Alice': 90}
```

Q46. Reverse lookup to find keys by value in a dictionary.

```
d = {'x':10, 'y':20, 'z':10}
rev = {}
for k,v in d.items():
    rev.setdefault(v, []).append(k)
print(rev)
```

Output:

```
{10: ['x', 'z'], 20: ['y']}
```

Q47. Merge multiple logs with timestamps, keeping the latest.

```
logs1 = {'10:00':'start','10:05':'load'}
logs2 = {'10:05':'update','10:10':'end'}
```

```
merged_logs = logs1 | logs2
print(merged_logs)
```

Output:

```
{'10:00': 'start', '10:05': 'update', '10:10': 'end'}
```

Q48. Build a dictionary of lists where keys are first letters of words.

```
words = ['apple','banana','apricot','berry']
d = {}
for w in words:
    d.setdefault(w[0], []).append(w)
print(d)
```

Output:

```
{'a': ['apple', 'apricot'], 'b': ['banana', 'berry']}
```

Q49. Sort students by total marks from nested dictionary.

```
students = {'Alice': {'Math':90,'Science':85}, 'Bob':{'Math':75,'Science':80}}
totals = {name: sum(marks.values()) for name, marks in students.items()}
sorted_totals = dict(sorted(totals.items(), key=lambda x: x[1], reverse=True))
print(sorted_totals)
```

Output:

```
{'Alice': 175, 'Bob': 155}
```

Q50. Find the most common value in a nested dictionary.

```
data = {'A': {'x':1,'y':2}, 'B':{'x':2,'y':2}, 'C':{'x':1,'y':3}}
flat_values = [v for inner in data.values() for v in inner.values()]
from collections import Counter
most_common = Counter(flat_values).most_common(1)
print(most_common)
```

Output:

```
[(2, 3)]
```

DICTIONARY ADVANCED

1. Student Performance Analyzer

Description: Store multiple students’ marks in nested dictionaries and calculate totals,

averages, and grades.

```
students = {
    'Alice': {'Math': 90, 'Physics': 85, 'Chemistry': 88},
    'Bob': {'Math': 75, 'Physics': 80, 'Chemistry': 70},
    'Charlie': {'Math': 95, 'Physics': 92, 'Chemistry': 90}
}

# Calculate total, average, and grade
for name, marks in students.items():
    total = sum(marks.values())
    avg = total / len(marks)
    grade = 'A' if avg>=90 else 'B' if avg>=75 else 'C'
    print(f"{name} -> Total: {total}, Avg: {avg:.2f}, Grade: {grade}")
```

Output Example:

```
Alice -> Total: 263, Avg: 87.67, Grade: B
Bob -> Total: 225, Avg: 75.00, Grade: B
Charlie -> Total: 277, Avg: 92.33, Grade: A
```

2. Inventory Management System

Description: Track stock for products using dictionaries, update stock with sales and purchases.

```
inventory = {'apple':50, 'banana':30, 'orange':20}
sales = {'apple':5, 'orange':2}
for item, sold in sales.items():
    inventory[item] = inventory.get(item, 0) - sold
print(inventory)
```

Output:

```
{'apple': 45, 'banana': 30, 'orange': 18}
```

3. Employee Department Tracker

Description: Map employees to multiple departments using dictionary of lists.

```
data = [('Alice','HR'),('Bob','IT'),('Alice','Finance'),('Charlie','IT')]
emp_dept = {}
for emp, dept in data:
    emp_dept.setdefault(emp, []).append(dept)
print(emp_dept)
```

Output:

```
{'Alice': ['HR', 'Finance'], 'Bob': ['IT'], 'Charlie': ['IT']}
```

4. Text Word Frequency Analyzer

Description: Count word frequency in a large text using dictionary and sort by frequency.

```
text = "apple banana apple orange banana apple"
words = text.split()
freq = {w: words.count(w) for w in set(words)}
sorted_freq = dict(sorted(freq.items(), key=lambda x: x[1], reverse=True))
print(sorted_freq)
```

Output:

```
{'apple': 3, 'banana': 2, 'orange': 1}
```

5. Library Book Management System

Description: Store book details with availability using nested dictionaries.

```
library = {
    'Book1': {'Author':'Author A','Available':True},
    'Book2': {'Author':'Author B','Available':False},
}
# Borrow Book1
library['Book1']['Available'] = False
print(library)
```

Output:

```
{'Book1': {'Author': 'Author A', 'Available': False}, 'Book2': {'Author': 'Author B',
```

6. Online Quiz Result Tracker

Description: Store quiz scores for multiple students and calculate top scorer.

```
scores = {'Alice':[90,85,88], 'Bob':[75,80,70], 'Charlie':[95,92,90]}
avg_scores = {name: sum(lst)/len(lst) for name,lst in scores.items()}
topper = max(avg_scores, key=avg_scores.get)
print("Top Scorer:", topper, "with avg score", avg_scores[topper])
```

Output:

```
Top Scorer: Charlie with avg score 92.33
```

7. E-Commerce Order Summary

Description: Track user orders (product, quantity) and calculate total items purchased.

```
orders = {'User1':[('Apple',2),('Banana',3)], 'User2':[('Apple',1),('Orange',5)]}
total_items = {user: sum(qty for _, qty in items) for user, items in orders.items()}
print(total_items)
```

Output:

```
{'User1': 5, 'User2': 6}
```

8. Movie Rating Database

Description: Store multiple users' movie ratings and find average ratings.

```
ratings = {'Alice':{'Titanic':5,'Avengers':4}, 'Bob':{'Titanic':4,'Avengers':5}}
avg_ratings = {}
for movie in set(k for v in ratings.values() for k in v):
    avg_ratings[movie] = sum(user.get(movie,0) for user in ratings.values()) / len(rat
print(avg_ratings)
```

Output:

```
{'Titanic': 4.5, 'Avengers': 4.5}
```

9. Sports Tournament Scoreboard

Description: Store teams and their match scores using nested dictionaries and calculate ranking.

```
scores = {'TeamA':{'Match1':2,'Match2':3}, 'TeamB':{'Match1':1,'Match2':4}}
total_scores = {team: sum(matches.values()) for team,matches in scores.items()}
ranking = sorted(total_scores.items(), key=lambda x:x[1], reverse=True)
print(ranking)
```

Output:

```
[('TeamB', 5), ('TeamA', 5)]
```

10. Nested Dictionary Data Analytics

Description: Analyze sales data stored in nested dictionary with multiple regions and products.

```
sales_data = {
    'Region1': {'ProductA': 100, 'ProductB': 150},
    'Region2': {'ProductA': 200, 'ProductB': 120}
}
# Total sales per product
total_per_product = {product: sum(region.get(product,0)
for region in sales_data.values())
                     for product in ['ProductA','ProductB']}
print(total_per_product)
```

Output:

```
{'ProductA': 300, 'ProductB': 270}
```

UNIT-7 PYTHON MODULE

1.**Write a Python program to generate a random color hex, a random alphabetical string, random value between two integers (inclusive) and a random multiple of 7 between 0 and 70.**

```
import random
import string
print("Generate a random color hex:")
print("#{:06x}".format(random.randint(0, 0xFFFFFF)))
print("\nGenerate a random alphabetical string:")
max_length = 255
s = ""
for i in range(random.randint(1, max_length)):
    s += random.choice(string.ascii_letters)
print(s)
print("Generate a random value between two integers, inclusive:")
print(random.randint(0, 10))
print(random.randint(-7, 7))
print(random.randint(1, 1))
print("Generate a random multiple of 7 between 0 and 70:")
print(random.randint(0, 10) * 7)
```

2.**Write a Python program to select a random element from a list, set, dictionary-value, and file from a directory.**

```
import random
import os
print("Select a random element from a list:")
```

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

print(random.choice(elements))

print(random.choice(elements))

print(random.choice(elements))

print("\nSelect a random element from a set:")

elements = set([1, 2, 3, 4, 5])

# convert to tuple because sets are invalid inputs

print(random.choice(tuple(elements)))

print(random.choice(tuple(elements)))

print(random.choice(tuple(elements)))

print("\nSelect a random value from a dictionary:")

d = {"a": 1, "b": 2, "c": 3, "d": 4, "e": 5}

key = random.choice(list(d))

print(d[key])

key = random.choice(list(d))

print(d[key])

key = random.choice(list(d))

print(d[key])

print("\nSelect a random file from a directory.:")

print(random.choice(os.listdir("/")))
```

3.Write a Python program that generates random alphabetical characters, alphabetical strings, and alphabetical strings of a fixed length.

```
import random

import string
```

```
length = 8

random_string = ''.join(random.choices(string.ascii_letters + string.digits, k=length))

print(random_string)
```

4.**Write a Python program to construct a seeded random number generator,**

also generate a float between 0 and 1, excluding 1

```
import random

print("Construct a seeded random number generator:")

print(random.Random().random())

print(random.Random(0).random())

print("\nGenerate a float between 0 and 1, excluding 1:")

print(random.random())
```

5. Write a Python program to generate a random integer between 0 and 6 - excluding 6, random integer between 5 and 10 - excluding 10, random integer between 0 and 10, with a step of 3 and random date between two dates.

```
import random

from datetime import datetime, timedelta

random_int_0_to_5 = random.randrange(0, 6)

print(f"Random integer between 0 and 6 (excluding 6): {random_int_0_to_5}"
```

6. Write a Python program to shuffle the elements of a given list

```
import random


a = [1, 2, 3, 4, 5]

random.shuffle(a)

print(a))
```

7. Write a Python program to generate a float between 0 and 1, inclusive and generate a random float within a specific range

```
import random
a = random.random()
print(a)
```

8.Write a Python program to set a random seed and get a random number between 0 and 1.

```
import random
for i in range(10):
    print(random.random())
```

9. Write a Python program to check if a function is a user-defined function or not.

Use types.FunctionType, types.LambdaType()

```
import types
def func():
   return 1


print(isinstance(func, types.FunctionType))
print(isinstance(func, types.LambdaType))
print(isinstance(lambda x: x, types.FunctionType))
print(isinstance(lambda x: x, types.LambdaType))
print(isinstance(max, types.FunctionType))
print(isinstance(max, types.LambdaType))
print(isinstance(abs, types.FunctionType))
print(isinstance(abs, types.LambdaType))
```

10.Write a Python program to check if a given value is a method of a user-defined class.

Use types.MethodType()

```
import types

class C:

   def x():

      return 1

   def y():

      return 1


def b():

   return 2


print(isinstance(C().x, types.MethodType))

print(isinstance(C().y, types.MethodType))

print(isinstance(b, types.MethodType))

print(isinstance(max, types.MethodType))

print(isinstance(abs, types.MethodType))
```

11. Write a Python program to check if a given function is a generator or not. Use types.GeneratorType()

```
import types

def a(x):

   yield x


def b(x):

   return x
```

```
def add(x, y):

    return x + y


print(isinstance(a(456), types.GeneratorType))

print(isinstance(b(823), types.GeneratorType))

print(isinstance(add(8,2), types.GeneratorType))
```

12.Write a Python program to check if a given value is compiled code or not.

Also check if a given value is a module or not. Use types.CodeType, types.ModuleType()

```
import types

print("Check if a given value is compiled code:")

code = compile("print('Hello')", "sample", "exec")

print(isinstance(code, types.CodeType))

print(isinstance("print(abs(-111))", types.CodeType))

print("\nCheck if a given value is a module:")

print(isinstance(types, types.ModuleType))
```

12.Write a Python program to construct a Decimal from a float and a Decimal from a string.

Also represent the decimal value as a tuple. Use decimal.Decimal

```
import decimal

print("Construct a Decimal from a float:")

pi_val = decimal.Decimal(3.14159)

print(pi_val)

print(pi_val.as_tuple())

print("\nConstruct a Decimal from a string:")
```

```
num_str = decimal.Decimal("123.25")
print(num_str)
print(num_str.as_tuple())
```

13.Write a Python program to configure rounding to round up and round down a given decimal value. Use decimal.Decimal

```
import decimal
print("Configure the rounding to round up:")
decimal.getcontext().prec = 1
decimal.getcontext().rounding = decimal.ROUND_UP
print(decimal.Decimal(30) / decimal.Decimal(4))
print("\nConfigure the rounding to round down:")
decimal.getcontext().prec = 3
decimal.getcontext().rounding = decimal.ROUND_DOWN
print(decimal.Decimal(30) / decimal.Decimal(4))
print("\nConfigure the rounding to round up:")
print(decimal.Decimal('8.325').quantize(decimal.Decimal('.01'), rounding=decimal.ROUND_UP))
print("\nConfigure the rounding to round down:")
print(decimal.Decimal('8.325').quantize(decimal.Decimal('.01'), rounding=decimal.ROUND_DOWN))
```

MODULE ADVANCED

1.Write a Python program to round a decimal value to the nearest multiple of 0.10, unless already an exact multiple of 0.05. Use decimal.Decimal

```
from decimal import Decimal

def round_to_10_cents(x):
    remainder = x.remainder_near(Decimal('0.10'))
    if abs(remainder) == Decimal('0.05'):
        return x
    else:
        return x - remainder


# Test code.
for x in range(80, 120):
    y = Decimal(x) / Decimal('1E2')
    print("{0} rounds to {1}".format(y, round_to_10_cents(y)))
```

2. Write a Python program to configure the rounding to round to the floor, ceiling. Use decimal.ROUND_FLOOR, decimal.ROUND_CEILING

```
import decimal
```

```
print("Configure the rounding to round to the floor:")

decimal.getcontext().prec = 4

decimal.getcontext().rounding = decimal.ROUND_FLOOR

print(decimal.Decimal(20) / decimal.Decimal(6))

print("\nConfigure the rounding to round to the ceiling:")

decimal.getcontext().prec = 4

decimal.getcontext().rounding = decimal.ROUND_CEILING

print(decimal.Decimal(20) / decimal.Decimal(6))
```

3. Write a Python program that can be configured to round to the nearest -

with ties going towards 0 and ties going away from 0. Use decimal.ROUND_

HALF_DOWN, decimal.ROUND_HALF_UP

```
import decimal

print("Configure the rounding to round to the nearest, with ties going towards 0:")

decimal.getcontext().prec = 1

decimal.getcontext().rounding = decimal.ROUND_HALF_DOWN

print(decimal.Decimal(10) / decimal.Decimal(4))

print("\nConfigure the rounding to round to the nearest, with ties going away from 0:")

decimal.getcontext().prec = 1

decimal.getcontext().rounding = decimal.ROUND_HALF_UP

print(decimal.Decimal(10) / decimal.Decimal(4))
```

4. **Write a Python program to configure rounding to round to the nearest integer, with ti even integer. Use decimal.ROUND_HALF_EVEN**

```
import decimal
```

```
print("Configure the rounding to round to the nearest, with ties going to the nearest even

decimal.getcontext().prec = 1

decimal.getcontext().rounding = decimal.ROUND_HALF_EVEN

print(decimal.Decimal(10) / decimal.Decimal(4))
```

.5 Write a Python program to create a shallow copy of a given list. Use copy.copy

```
import copy

nums_x = [1, [2, 3, 4]]

print("Original list: ", nums_x)

nums_y = copy.copy(nums_x)

print("\nCopy of the said list:")

print(nums_y)

print("\nChange the value of an element of the original list:")

nums_x[1][1] = 10

print(nums_x)

print("\nSecond list:")

print(nums_y)

nums =  [[1], [2]]

nums_copy = copy.copy(nums)

print("\nOriginal list:")

print(nums)

print("\nCopy of the said list:")

print(nums_copy)
print("\nChange the value of an element of the original list:")

nums[0][0] = 0

print("\nFirst list:")

print(nums)

print("\nSecond list:")

print(nums_copy)
```

6. Write a Python program to create a deep copy of a given list. Use copy.copy

```
import copy

nums_x = [1, [2, 3, 4]]

print("Original list: ", nums_x)

nums_y = copy.deepcopy(nums_x)

print("\nDeep copy of the said list:")

print(nums_y)

print("\nChange the value of an element of the original list:")

nums_x[1][1] = 10

print(nums_x)

print("\nCopy of the second list (Deep copy):")

print(nums_y)

nums = [[1, 2, 3], [4, 5, 6]]

deep_copy = copy.deepcopy(nums)

print("\nOriginal list:")

print(nums)
print("\nDeep copy of the said list:")

print(deep_copy)

print("\nChange the value of some elements of the original list:")

nums[0][2] = 55

nums[1][1] = 77

print("\nOriginal list:")

print(nums)

print("\nSecond list (Deep copy):")

print(deep_copy)
```

7. Write a Python program to create a shallow copy of a given dictionary. Use copy.copy

```
import copy

nums_x = {"a":1, "b":2, 'cc':{"c":3}}

print("Original dictionary: ", nums_x)

nums_y = copy.copy(nums_x)

print("\nCopy of the said list:")

print(nums_y)

print("\nChange the value of an element of the original dictionary:")

nums_x["cc"]["c"] = 10

print(nums_x)

print("\nSecond dictionary:")

print(nums_y)
```

```
nums = {"x":1, "y":2, 'zz':{"z":3}}

nums_copy = copy.copy(nums)

print("\nOriginal dictionary :")

print(nums)

print("\nCopy of the said list:")

print(nums_copy)

print("\nChange the value of an element of the original dictionary:")

nums["zz"]["z"] = 10

print("\nFirst dictionary:")

print(nums)

print("\nSecond dictionary (copy):")

print(nums_copy)
```

8. Write a Python program to create a deep copy of a given dictionary. Use copy.copy

```
import copy

nums_x = {"a":1, "b":2, 'cc':{"c":3}}

print("Original dictionary: ", nums_x)

nums_y = copy.deepcopy(nums_x)

print("\nDeep copy of the said list:")

print(nums_y)

print("\nChange the value of an element of the original dictionary:")

nums_x["cc"]["c"] = 10

print(nums_x)
print("\nSecond dictionary (Deep copy):")

print(nums_y)


nums = {"x":1, "y":2, 'zz':{"z":3}}

nums_copy = copy.deepcopy(nums)

print("\nOriginal dictionary :")

print(nums)

print("\nDeep copy of the said list:")

print(nums_copy)

print("\nChange the value of an element of the original dictionary:")

nums["zz"]["z"] = 10

print("\nFirst dictionary:")

print(nums)

print("\nSecond dictionary (Deep copy):")

print(nums_copy)
```

9. Write a Python program to read and display the content of a given CSV file.

Use csv.reader

```
import csv

reader = csv.reader(open("employees.csv"))

for row in reader:

    print(row)
```

10.Write a Python program to count the number of lines in a given CSV file. Use csv.read

```
import csv
reader = csv.reader(open("employees.csv"))
no_lines= len(list(reader))
print(no_lines)
```

11.Write a Python program to parse a given CSV string and get a list of lists of string values. Use csv.reader

```
import csv
csv_string = """1,2,3
4,5,6
7,8,9
"""
print("Original string:")
print(csv_string)
lines = csv_string.splitlines()
print("List of CSV formatted strings:")
print(lines)
reader = csv.reader(lines)
parsed_csv = list(reader)
print("\nList representation of the CSV file:")
print(parsed_csv)
```

12. Write a Python program to read the current line from a given CSV file. Use csv.reade

```
import csv
f = open("employees.csv", newline='')
```

```
csv_reader = csv.reader(f)

print(next(csv_reader))

print(next(csv_reader))

print(next(csv_reader))
```

13. Write a Python program to skip the headers of a given CSV file. Use csv.reader

```
import csv

f = open("employees.csv", "r")

reader = csv.reader(f)

next(reader)

for row in reader:

    print(row)
```

MODULE CASE BASED

1.Write a Python program to write (without writing separate lines between rows)

and read a CSV file with a specified delimiter. Use csv.reader

```
import csv

fw = open("test.csv", "w", newline='')

writer = csv.writer(fw, delimiter = ",")

writer.writerow(["a","b","c"])

writer.writerow(["d","e","f"])

writer.writerow(["g","h","i"])

fw.close()
```

```
fr = open("test.csv", "r")

csv = csv.reader(fr, delimiter = ",")

for row in csv:

  print(row)

fr.close()
```

2. Converting Python Objects to JSON Strings

```
import json


# Python object (a dictionary)

python_dict = {

    "name": "Alice",

    "age": 30,

    "is_student": False,

    "courses": ["Math", "Science"],

    "address": None

}


# Serialize the Python dictionary to a JSON string

json_string = json.dumps(python_dict)


print(type(json_string))

print(json_string)
```

```
# Output:
# <class 'str'>
# {"name": "Alice", "age": 30, "is_student": false, "courses":
["Math", "Science"], "address": null}
```

3. Converting JSON Strings to Python Objects

```
import json

# A JSON formatted string
json_data_string = '{"id": "09", "name": "Nitin", "department": "Finance"}'

# Deserialize the JSON string to a Python object (dictionary)
python_object = json.loads(json_data_string)

print(type(python_object))
print(python_object)
print("Name:", python_object['name'])

# Output:
# <class 'dict'>
# {'id': '09', 'name': 'Nitin', 'department': 'Finance'}
# Name: Nitin
```

4. Writing Python Objects to a File (Serialization to File)

```
import json


# Python object
data_to_write = {
    "book_id": 101,
    "title": "The Great Novel",
    "author": "J. Doe"
}


# Write the data to a file named 'data.json'
file_path = "data.json"
with open(file_path, 'w') as f:
    # Use indent for readability in the file
    json.dump(data_to_write, f, indent=4)


print(f"Data successfully written to {file_path}")


# The 'data.json' file content will be:
# {
#     "book_id": 101,
#     "title": "The Great Novel",
#     "author": "J. Doe"

# }
```

5. Convert a Python object containing all the legal data types

```
import json

print(json.dumps({"name": "John", "age": 30}))

print(json.dumps(["apple", "bananas"]))

print(json.dumps(("apple", "bananas")))

print(json.dumps("hello"))

print(json.dumps(42))

print(json.dumps(31.76))

print(json.dumps(True))

print(json.dumps(False))

print(json.dumps(None))
```

UNIT-8 PYTHON FUNCTIONS

1. **Write a Python function to find the maximum of three numbers**.

```
def max_of_two(x, y):
    # Check if x is greater than y
    if x > y:
        # If x is greater, return x
        return x
    # If y is greater or equal to x, return y
    return y

# Define a function that returns the maximum of three numbers
def max_of_three(x, y, z):
    # Call max_of_two function to find the maximum of y and z,
    # then compare it with x to find the overall maximum
```

```
return max_of_two(x, max_of_two(y, z))

# Print the result of calling max_of_three function with arguments 3, 6, and -5

print(max_of_three(3, 6, -5))
```

2. **Write a Python function to sum all the numbers in a list.**

```
def sum(numbers):
# Initialize a variable 'total' to store the sum of numbers, starting at 0
total = 0

# Iterate through each element 'x' in the 'numbers' list
for x in numbers:
# Add the current element 'x' to the 'total'
total += x

# Return the final sum stored in the 'total' variable
return total

# Print the result of calling the 'sum' function with a tuple of numbers (8, 2, 3, 0, 7)

print(sum((8, 2, 3, 0, 7)))
```

3. **Write a Python function to multiply all the numbers in a list.**

```
def multiply(numbers):
  # Initialize a variable 'total' to store the multiplication result, starting at 1
  total = 1

  # Iterate through each element 'x' in the 'numbers' list
  for x in numbers:
    # Multiply the current element 'x' with the 'total'
    total *= x

  # Return the final multiplication result stored in the 'total' variable
  return total

# Print the result of calling the 'multiply' function with a tuple of numbers (8, 2, 3, -1

print(multiply((8, 2, 3, -1, 7)))
```

4. **Write a Python program to reverse a string.**

```
def string_reverse(str1):
  # Initialize an empty string 'rstr1' to store the reversed string
  rstr1 = ''
```

```
    # Calculate the length of the input string 'str1'
    index = len(str1)

    # Execute a while loop until 'index' becomes 0
    while index > 0:
        # Concatenate the character at index - 1 of 'str1' to 'rstr1'
        rstr1 += str1[index - 1]

        # Decrement the 'index' by 1 for the next iteration
        index = index - 1

    # Return the reversed string stored in 'rstr1'
    return rstr1
# Print the result of calling the 'string_reverse' function with the input string '1234abcd'

print(string_reverse('1234abcd'))
```

5. **Write a Python function to calculate the factorial of a number (a non-negative inte**

```
def factorial(n):
    # Check if the number 'n' is 0
    if n == 0:
        # If 'n' is 0, return 1 (factorial of 0 is 1)
        return 1
    else:
        # If 'n' is not 0, recursively call the 'factorial' function with (n-1) and multiply it v
        return n * factorial(n - 1)

# Ask the user to input a number to compute its factorial and store
 it in variable 'n'
n = int(input("Input a number to compute the factorial: "))

# Print the factorial of the number entered by the user by calling
the 'factorial' function

print(factorial(n))
```

6. **Write a Python function to check whether a number falls within a given range**

```
def test_range(n):
    # Check if 'n' is within the range from 3 to 8 (inclusive) using
the 'in range()' statement
    if n in range(3, 9):
        # If 'n' is within the range, print that 'n' is within the given range
        print("%s is in the range" % str(n))
    else:
        # If 'n' is outside the range, print that the number is outside the given range
        print("The number is outside the given range.")

# Call the 'test_range' function with the argument 5

test_range(5)
```

7. **Write a Python function that accepts a string and counts the number of upper and lower case letters.**

```
def string_test(s):
    # Create a dictionary 'd' to store the count of upper and lower case characters
    d = {"UPPER_CASE": 0, "LOWER_CASE": 0}

    # Iterate through each character 'c' in the string 's'
    for c in s:
        # Check if the character 'c' is in upper case
        if c.isupper():
            # If 'c' is upper case, increment the count of upper case characters in the dicti
            d["UPPER_CASE"] += 1
        # Check if the character 'c' is in lower case
        elif c.islower():
            # If 'c' is lower case, increment the count of lower case characters in the dicti
            d["LOWER_CASE"] += 1
        else:
            # If 'c' is neither upper nor lower case (e.g., punctuation, spaces), do nothing
            pass

    # Print the original string 's'
    print("Original String: ", s)

    # Print the count of upper case characters
    print("No. of Upper case characters: ", d["UPPER_CASE"])

    # Print the count of lower case characters
    print("No. of Lower case Characters: ", d["LOWER_CASE"])

# Call the 'string_test' function with the input string 'The quick Brown Fox'

string_test('The quick Brown Fox')
```

8. **Write a Python function that takes a list and returns a new list with distinct**

elements from the first list.

```
def unique_list(l):
    # Create an empty list 'x' to store unique elements
    x = []

    # Iterate through each element 'a' in the input list 'l'
    for a in l:
        # Check if the element 'a' is not already present in the list 'x'
        if a not in x:
            # If 'a' is not in 'x', add it to the list 'x'
            x.append(a)

    # Return the list 'x' containing unique elements
    return x

# Print the result of calling the 'unique_list' function with a list containing duplicate

print(unique_list([1, 2, 3, 3, 3, 3, 4, 5]))
```

9. **Write a Python function that takes a number as a parameter and checks whether the number is prime or not**

```
def test_prime(n):
    # Check if 'n' is equal to 1
    if (n == 1):
        # If 'n' is 1, return False (1 is not a prime number)
        return False
    # Check if 'n' is equal to 2
    elif (n == 2):
        # If 'n' is 2, return True (2 is a prime number)
        return True
    else:
        # Iterate through numbers from 2 to (n-1) using 'x' as the iterator
        for x in range(2, n):
            # Check if 'n' is divisible by 'x' without any remainder
            if (n % x == 0):
                # If 'n' is divisible by 'x', return False (not a prime number)
                return False
        # If 'n' is not divisible by any number from 2 to (n-1), return True (prime number)
        return True

# Print the result of checking if 9 is a prime number by calling the 'test_prime' function
```

```
print(test_prime(9))
```

10. Write a Python program to print the even numbers from a given list.

```
def is_even_num(l):
    # Create an empty list 'enum' to store even numbers
    enum = []

    # Iterate through each number 'n' in the input list 'l'
    for n in l:
        # Check if the number 'n' is even (divisible by 2 without a remainder)
        if n % 2 == 0:
            # If 'n' is even, append it to the 'enum' list
            enum.append(n)

    # Return the list 'enum' containing even numbers
    return enum

# Print the result of calling the 'is_even_num' function with a list of numbers

print(is_even_num([1, 2, 3, 4, 5, 6, 7, 8, 9]))
```

11. **Write a Python function to check whether a number is "Perfect" or not.**

```
def perfect_number(n):
    # Initialize a variable 'sum' to store the sum of factors of 'n'
    sum = 0

    # Iterate through numbers from 1 to 'n-1' using 'x' as the iterator
    for x in range(1, n):
        # Check if 'x' is a factor of 'n' (divides 'n' without remainder)
        if n % x == 0:
            # If 'x' is a factor of 'n', add it to the 'sum'
            sum += x

    # Check if the 'sum' of factors is equal to the original number 'n'
    return sum == n

# Print the result of checking if 6 is a perfect number by calling the 'perfect_number' function

print(perfect_number(6))
```

12.**Write a Python function that checks whether a passed string is a palindrome or not.**

```
def isPalindrome(string):
    # Initialize left and right pointers to check characters from the start and end of
  the string
    left_pos = 0
    right_pos = len(string) - 1

    # Loop until the pointers meet or cross each other
    while right_pos >= left_pos:
        # Check if the characters at the left and right positions are not equal
        if not string[left_pos] == string[right_pos]:
            # If characters don't match, return False (not a palindrome)
            return False

        # Move the left pointer to the right and the right pointer to the left to
  continue checking
        left_pos += 1
        right_pos -= 1

    # If the loop finishes without returning False, the string is a palindrome,
 so return True
    return True

 # Print the result of checking if the string 'aza' is a palindrome by calling the
  'isPalindrome' function

print(isPalindrome('aza'))
```

13**. Write a Python function that prints out the first n rows of Pascal's triangle**

```
def pascal_triangle(n):
    # Initialize the first row of Pascal's Triangle with value 1 as a starting point
    trow = [1]

    # Create a list 'y' filled with zeros to be used for calculations
    y = [0]

    # Iterate through a range starting from 0 up to the maximum of 'n' or 0
(taking the maximum to handle negative 'n')
    for x in range(max(n, 0)):
        # Print the current row of Pascal's Triangle
        print(trow)

        # Update the current row based on the previous row by calculating the next row using list
comprehension
        # The formula for generating the next row in Pascal's Triangle is based on addition of
consecutive elements
        trow = [l + r for l, r in zip(trow + y, y + trow)]

    # Return True if 'n' is greater than or equal to 1, else return False
    return n >= 1
```

```
# Generate Pascal's Triangle up to row 6 by calling the 'pascal_triangle' function

pascal_triangle(6)
```

14. Write a Python function to check whether a string is a pangram or not.

```
import string
import sys

# Define a function named 'ispangram' that checks if a string is a pangram
def ispangram(str1, alphabet=string.ascii_lowercase):
    # Create a set 'alphaset' containing all lowercase letters from the provided alphabet
    alphaset = set(alphabet)

    # Convert the input string to lowercase and create a set from it
    str_set = set(str1.lower())

    # Check if the set of lowercase characters in the input string covers all characters in 'alphaset'
    return alphaset <= str_set

# Print the result of checking if the string is a pangram by calling the 'ispangram' function

print(ispangram('The quick brown fox jumps over the lazy dog'))
```

15.Write a Python program that accepts a hyphen-separated sequence of words as input and prints the words in a hyphen-separated sequence after sorting them alphabetically

```
items = [n for n in input().split('-')]

# Sort the elements in the 'items' list in lexicographical order (alphabetical and numerical sorting)
items.sort()

# Join the sorted elements in the 'items' list using the hyphen ("-") separator and print the resulting string

print('-'.join(items))
```

17. Write a Python function to create and print a list where the values are the squares of numbers between 1 and 30 (both included).

```
def printValues():
    # Create an empty list 'l'
    l = list()

    # Iterate through numbers from 1 to 20 (inclusive)
    for i in range(1, 21):
        # Calculate the square of 'i' and append it to the list 'l'
        l.append(i**2)

    # Print the list containing squares of numbers from 1 to 20
    print(l)

# Call the 'printValues' function to generate and print the list of squares

printValues()
```

PYTHON RECURSION

1. Factorial of a Number

```
def factorial(n):
    if n == 0:
        return 1
    return n * factorial(n-1)

print("Factorial:", factorial(5))
```

Output:

```
Factorial: 120
```

2. Print Numbers from 1 to N

```
def print_num(n):
    if n == 0:
        return
    print_num(n-1)
    print(n)

print_num(5)
```

Output:

```
1
2
3
4
5
```

3. Sum of First N Natural Numbers

```
def sum_n(n):
    if n == 0:
        return 0
    return n + sum_n(n-1)

print("Sum =", sum_n(10))
```

Output:

```
Sum = 55
```

4. Fibonacci Number

```
def fib(n):
    if n <= 1:
        return n
    return fib(n-1) + fib(n-2)

print("Fibonacci:", fib(6))
```

Output:

```
Fibonacci: 8
```

5. Power of Number (x^n)

```
def power(x, n):
    if n == 0:
        return 1
    return x * power(x, n-1)

print("Power:", power(2, 5))
```

Output:

```
Power: 32
```

6. Reverse a String

```
def rev_str(s):
    if len(s) == 0:
        return s
    return rev_str(s[1:]) + s[0]

print("Reversed:", rev_str("python"))
```

Output:

```
Reversed: nohtyp
```

7. Length of String

```
def str_len(s):
    if s == '':
        return 0
    return 1 + str_len(s[1:])

print("Length:", str_len("hello"))
```

Output:

```
Length: 5
```

8. Maximum in a List

```
def max_list(lst):
    if len(lst) == 1:
        return lst[0]
    sub_max = max_list(lst[1:])
    return lst[0] if lst[0] > sub_max else sub_max

print("Max:", max_list([3,7,2,9,5]))
```

Output:

```
Max: 9
```

9. Count Digits in a Number

```
def count_digits(n):
    if n == 0:
        return 0
    return 1 + count_digits(n//10)

print("Digits:", count_digits(12345))
```

Output:
```
Digits: 5
```

10. Sum of Digits

```python
def sum_digits(n):
    if n == 0:
        return 0
    return n % 10 + sum_digits(n // 10)

print("Sum of digits:", sum_digits(123))
```

Output:
```
Sum of digits: 6
```

11. Check if Number is Palindrome

```python
def reverse_num(n, temp=0):
    if n == 0:
        return temp
    return reverse_num(n // 10, temp * 10 + n % 10)

num = 121
if num == reverse_num(num):
    print("Palindrome")
else:
    print("Not Palindrome")
```

Output:
```
Palindrome
```

12. GCD of Two Numbers

```python
def gcd(a, b):
    if b == 0:
        return a
    return gcd(b, a % b)

print("GCD:", gcd(48, 18))
```

Output:
```
GCD: 6
```

13. Print List in Reverse

```python
def print_rev(lst):
    if not lst:
        return
    print_rev(lst[1:])
    print(lst[0])

print_rev([1, 2, 3, 4])
```

Output:

```
4
3
2
1
```

14. Check Palindrome String

```
def is_pal(s):
    if len(s) <= 1:
        return True
    return s[0] == s[-1] and is_pal(s[1:-1])

print(is_pal("madam"))
```

Output:

```
True
```

15. Decimal to Binary

```
def dec_bin(n):
    if n == 0:
        return ''
    return dec_bin(n // 2) + str(n % 2)

print("Binary:", dec_bin(13))
```

Output:

```
Binary: 1101
```

16. Count Occurrences in List

```
def count_occ(lst, x):
    if not lst:
        return 0
    return (1 if lst[0] == x else 0) + count_occ(lst[1:], x)

print("Occurrences:", count_occ([1,2,3,2,2], 2))
```

Output:

```
Occurrences: 3
```

17. Flatten a Nested List

```
def flatten(lst):
    res = []
    for i in lst:
        if isinstance(i, list):
            res += flatten(i)
        else:
            res.append(i)
    return res

print(flatten([1, [2, [3, 4]], 5]))
```

Output:

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

18. All Permutations of a String

```
def perm(s):
    if len(s) == 1:
        return [s]
    result = []
    for i in range(len(s)):
        for p in perm(s[:i] + s[i+1:]):
            result.append(s[i] + p)
    return result

print(perm("abc"))
```

Output:

```
['abc', 'acb', 'bac', 'bca', 'cab', 'cba']
```

19. Sum of Elements in Nested List

```
def sum_nested(lst):
    total = 0
    for i in lst:
        if isinstance(i, list):
            total += sum_nested(i)
        else:
            total += i
    return total

print("Sum:", sum_nested([1, [2, [3, 4]], 5]))
```

Output:

```
Sum: 15
```

20. Custom Recursive Sequence

```
f(n) = f(n-1) + 2*f(n-2)

def seq(n):
    if n == 1:
        return 1
    if n == 2:
        return 2
    return seq(n-1) + 2 * seq(n-2)

print("Term:", seq(5))
```

Output:

```
Term: 17
```

21. Binary Search

```
def bsearch(lst, low, high, x):
    if low > high:
        return -1
    mid = (low + high) // 2
    if lst[mid] == x:
        return mid
    elif lst[mid] > x:
        return bsearch(lst, low, mid-1, x)
    else:
        return bsearch(lst, mid+1, high, x)

print("Index:", bsearch([1,3,5,7,9], 0, 4, 7))
```

Output:

```
Index: 3
```

22. All Subsets of a List

```
def subsets(lst):
    if not lst:
        return [[]]
    sub = subsets(lst[1:])
    return sub + [[lst[0]] + s for s in sub]

print(subsets([1,2]))
```

Output:

```
[[], [2], [1], [1, 2]]
```

23. Sum of Series 1 + 1/2 + 1/3 + … + 1/n

```
def series_sum(n):
    if n == 1:
        return 1
    return 1/n + series_sum(n-1)

print("Sum:", round(series_sum(5), 2))
```

Output:

```
Sum: 2.28
```

24. All Combinations of List Elements

```
from itertools import combinations

def combine(lst):
    res = [[]]
    for i in range(1, len(lst)+1):
        res += list(combinations(lst, i))
    return res

print(combine([1, 2, 3]))
```

Output:

```
[[], (1,), (2,), (3,), (1, 2), (1, 3), (2, 3), (1, 2, 3)]
```

25. Count Vowels Using Recursion

```
def count_vowels(s):
    if not s:
        return 0
    return (1 if s[0].lower() in 'aeiou' else 0) + count_vowels(s[1:])

print("Vowels:", count_vowels("Recursion"))
```

Output:

```
Vowels: 4
```

Python Lambda Function

1. Check if a number is palindrome

```
is_palindrome = lambda n: str(n) == str(n)[::-1]
print(is_palindrome(121))
```

Output:

```
True
```

2. Extract all prime numbers from a list

```
nums = [2, 3, 4, 5, 6, 7, 11, 12]
is_prime = lambda x: all(x % i != 0 for i in range(2, int(x**0.5)+1)) and x > 1
primes = list(filter(is_prime, nums))
print(primes)
```

Output:

```
[2, 3, 5, 7, 11]
```

3. Maximum of three numbers

```
max3 = lambda a, b, c: a if a > b and a > c else (b if b > c else c)
print(max3(10, 25, 20))
```

Output:

```
25
```

4. Check if a string is a pangram

```
is_pangram = lambda s: len(set(s.lower())) >= 26
print(is_pangram("The quick brown fox jumps over the lazy dog"))
```

Output:

```
True
```

5. Convert Celsius → Fahrenheit

```
celsius = [0, 10, 20, 30, 40]
fahrenheit = list(map(lambda c: (9/5)*c + 32, celsius))
print(fahrenheit)
```

Output:

```
[32.0, 50.0, 68.0, 86.0, 104.0]
```

6. Sort tuples by second element

```
data = [(1, 3), (2, 1), (4, 5), (3, 2)]
sorted_data = sorted(data, key=lambda x: x[1])
print(sorted_data)
```

Output:
```
[(2, 1), (3, 2), (1, 3), (4, 5)]
```

7. Sort dictionary by values
```
d = {'a': 5, 'b': 2, 'c': 8}
sorted_dict = dict(sorted(d.items(), key=lambda x: x[1]))
print(sorted_dict)
```

Output:
```
{'b': 2, 'a': 5, 'c': 8}
```

8. Product of all numbers using reduce
```
from functools import reduce
nums = [2, 3, 4, 5]
prod = reduce(lambda x, y: x*y, nums)
print(prod)
```

Output:
```
120
```

9. Reverse words but keep order
```
sentence = "Python is powerful"
rev_words = ' '.join(map(lambda w: w[::-1], sentence.split()))
print(rev_words)
```

Output:
```
nohtyP si lufrewop
```

10. Count vowels in a string
```
s = "Recursion with Lambda"
vowels = list(filter(lambda ch: ch.lower() in 'aeiou', s))
print(len(vowels))
```

Output:
```
7
```

11. Capitalize alternate words
```
text = "python makes coding fun"
result = ' '.join(map(lambda w: w.upper() if text.split().index(w) % 2 == 0 else w, te
print(result)
```

Output:
```
PYTHON makes CODING fun
```

12. Even or Odd (conditional lambda)
```
even_odd = lambda x: "Even" if x % 2 == 0 else "Odd"
```

```
print(even_odd(13))
```

Output:
```
Odd
```

13. Factorial using reduce (lambda recursion-like)

```
from functools import reduce
fact = lambda n: reduce(lambda x, y: x*y, range(1, n+1))
print(fact(6))
```

Output:
```
720
```

14. Element-wise addition of lists

```
a, b = [1, 2, 3], [4, 5, 6]
result = list(map(lambda x, y: x + y, a, b))
print(result)
```

Output:
```
[5, 7, 9]
```

15. Strings that start & end with same letter

```
words = ["level", "data", "python", "madam"]
same_letter = list(filter(lambda w: w[0].lower() == w[-1].lower(), words))
print(same_letter)
```

Output:
```
['level', 'madam']
```

16. Validate Email (basic)

```
is_email = lambda e: "@" in e and "." in e.split('@')[-1]
print(is_email("user@gmail.com"))
print(is_email("wrongemail@com"))
```

Output:
```

True
False
```

17. Sort names by last character

```
names = ["Alex", "Charlie", "Bob", "David"]
sorted_names = sorted(names, key=lambda n: n[-1])
print(sorted_names)
```

Output:
```
['Charlie', 'David', 'Bob', 'Alex']
```

18. Sum of squares of even numbers

```
nums = [1, 2, 3, 4, 5, 6]
sum_sq_even = sum(map(lambda x: x**2, filter(lambda y: y % 2 == 0, nums)))
print(sum_sq_even)
```

Output:

```
56
```

19. Square of Sum using nested lambdas

```
square_of_sum = lambda a: (lambda b: (a + b)**2)
print(square_of_sum(3)(4))
```

Output:

```
49
```

20. Remove duplicate words from sentence

```
sentence = "python is great and python is fun"
unique = ' '.join(dict.fromkeys(sentence.split()))
print(unique)
```

Output:

```
python is great and fun
```

21. Prefix every word

```
words = ["AI", "ML", "Data"]
prefixed = list(map(lambda w: "Python_" + w, words))
print(prefixed)
```

Output:

```
['Python_AI', 'Python_ML', 'Python_Data']
```

22. Flatten a list of lists

```
from functools import reduce
lst = [[1,2],[3,4],[5,6]]
flat = reduce(lambda x, y: x + y, lst)
print(flat)
```

Output:

```
[1, 2, 3, 4, 5, 6]
```

23. Character frequency in string

```
s = "banana"
freq = {ch: s.count(ch) for ch in sorted(set(s))}
print(freq)
```

Output:

```
{'a': 3, 'b': 1, 'n': 2}
```

24. Fibonacci using lambda & recursion

```
fib = lambda n: n if n <= 1 else fib(n-1) + fib(n-2)
print(fib(7))
```

Output:

```
13
```

25. Square of odd numbers greater than 5

```
nums = [2, 5, 7, 8, 9, 11]
result = list(map(lambda x: x**2, filter(lambda y: y > 5 and y % 2 != 0, nums)))
print(result)
```

Output:

```
[49, 81, 121]
```

Advance python Function

1. Create a Chain of Function Decorators (Bold, Italic, Underline, etc.)

```
def make_bold(fn):
    def wrapped():
        return "<b>" + fn() + "</b>"
    return wrapped

# Define a decorator 'make_italic' that adds italic HTML tags to the wrapped function's return value
def make_italic(fn):
    def wrapped():
        return "<i>" + fn() + "</i>"
    return wrapped

# Define a decorator 'make_underline' that adds underline HTML tags to the wrapped function's return valu
def make_underline(fn):
    def wrapped():
        return "<u>" + fn() + "</u>"
    return wrapped

# Apply multiple decorators (@make_bold, @make_italic, @make_underline) to the 'hello' function
@make_bold
@make_italic
@make_underline
def hello():
    return "hello world"

# Print the result of the decorated 'hello' function, which adds HTML tags for bold, italic, and underline


print(hello()) ## returns "<b><i><u>hello world</u></i></b>"
```

2. **Access a Function Inside a Function**

```
def greet(name):
    return f"Hello, {name}!"

def welcome(name):
    message = greet(name)  # calling another function
    return f"{message} Welcome to Python."


print(welcome("Carlos"))
```

3. **Detect the Number of Local Variables Declared in a Function**

```
def abc():
    # Define and assign values to local variables 'x', 'y', and 'str1' inside the function 'abc'
    x = 1
    y = 2
    str1 = "w3resource"

    # Print the string "Python Exercises"
    print("Python Exercises")

# Access the number of local variables in the function 'abc' using the __code__.co_nlocals attri
print(abc.__code__.co_nlocals)
```

4. **Invoke a Function After a Specified Period of Time**

```
from time import sleep
import math

# Define a function named 'delay' that delays the execution of a function by the given milliseco
def delay(fn, ms, *args):
    # Sleep for the specified number of milliseconds
    sleep(ms / 1000)

    # Call the provided function 'fn' with the given arguments '*args' and return the result
    return fn(*args)

# Print a message indicating the operation that follows
print("Square root after specific milliseconds:")

# Call the 'delay' function with a lambda function to calculate square roots after specific delays
# Print the square root of 16 after a delay of 100 milliseconds
print(delay(lambda x: math.sqrt(x), 100, 16))

# Print the square root of 100 after a delay of 1000 milliseconds
print(delay(lambda x: math.sqrt(x), 1000, 100))

# Print the square root of 25100 after a delay of 2000 milliseconds

print(delay(lambda x: math.sqrt(x), 2000, 25100))
```

5. **Use a lambda with the filter() function to get all even numbers from a list**

```
nums = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]

# Display a message indicating that the following output will show the original list of integers
print("Original list of integers:")
print(nums)

# Display a message indicating that the following output will show even numbers from the list
print("\nEven numbers from the said list:")

# Use the 'filter()' function with a lambda function to filter even numbers from 'nums'
# Create a new list 'even_nums' containing only the even numbers from the original list
even_nums = list(filter(lambda x: x % 2 == 0, nums))
print(even_nums)

# Display a message indicating that the following output will show odd numbers from the list
print("\nOdd numbers from the said list:")

# Use the 'filter()' function with a lambda function to filter odd numbers from 'nums'
# Create a new list 'odd_nums' containing only the odd numbers from the original list
odd_nums = list(filter(lambda x: x % 2 != 0, nums))

print(odd_nums)
```

6. Use a lambda with the map() function to double each element in a list

```
nums1 = [1, 2, 3]
nums2 = [4, 5, 6]

# Display a message indicating that the following output will show the original lists
print("Original list:")
print(nums1)  # Print the contents of 'nums1'
print(nums2)  # Print the contents of 'nums2'

# Use the 'map()' function with a lambda function to add corresponding elements from 'nums1'
 and 'nums2'
# Apply the lambda function to pairs of elements from 'nums1' and 'nums2' and generate a
new 'result' iterable
result = map(lambda x, y: x + y, nums1, nums2)

# Display the result after adding elements from both lists together using 'map()' and a lambda function
print("\nResult: after adding two lists")

print(list(result))  # Print the result of adding corresponding elements from 'nums1' and 'nums2'.
```

7. Use a lambda with the sorted() function to sort a list of tuples based on the second element

```
subject_marks = [('English', 88), ('Science', 90), ('Maths', 97), ('Social sciences', 82)]

# Display the original list of tuples to the console
print("Original list of tuples:")
print(subject_marks)

# Sort the 'subject_marks' list of tuples based on the second element of each tuple (the marks),
# using a lambda function as the sorting key to extract the second element
subject_marks.sort(key=lambda x: x[1])
```

```
# Display the sorted list of tuples to the console
print("\nSorting the List of Tuples:")

print(subject_marks)
```

8. Create Higher-Order Function

```
def square(x):
    return x * x

numbers = [1, 2, 3, 4, 5]
squared_numbers = list(map(square, numbers))

print(f"Squared numbers: {squared_numbers}")
```

9. Python Function Practice Check Before You Append

```
a = [2, 5, 6, 7]
# Use append() to add the element 8 to the end of the list
a.append(8)

print(a)
```

10. Python Function Practice Removing Duplicates and Sorting

```
# initializing list
test_list = [5, 6, 2, 5, 3, 3, 6, 5, 5, 6, 5]

# printing original list
print("The original list : " + str(test_list))

# using sorted() + set() + count()
# sorting and removal of duplicates
res = sorted(set(test_list), key = lambda ele: test_list.count(ele))

# print result

print("The list after sorting and removal : " + str(res))
```

11. Python Function Practice Find the Second Occurrence

```
def find_string(txt, str1):
    # Use the find method to locate the second occurrence of str1 in txt.
    # The expression txt.find(str1) finds the first occurrence, and adding 1 finds the
starting position for searching the second occurrence.
    return txt.find(str1, txt.find(str1) + 1)

# Test the function with different strings and print the results.

# Test case 1
print(find_string("The quick brown fox jumps over the lazy dog", "the"))

# Test case 2

print(find_string("the quick brown fox jumps over the lazy dog", "the"))
```

12. Python Function Practice Sorting Non-Negative Number

```
def sort_non_negative(numbers):
    # Extract and sort the non-negative numbers
    non_negatives = sorted([num for num in numbers if num >= 0])

    result = []
    non_neg_index = 0

    for num in numbers:
        if num >= 0:
            # Replace with the next smallest non-negative
            result.append(non_negatives[non_neg_index])
            non_neg_index += 1
        else:
            # Keep negative numbers in original position
            result.append(num)

    return result
```

13. Python Function Practice Caesar Cipher

```
def caesar_cipher(text, shift):
    result = ""

    for char in text:
        if char.isalpha():
            # Determine if character is uppercase or lowercase
            offset = ord('A') if char.isupper() else ord('a')

            # Shift character and wrap around using modulo 26
            shifted = (ord(char) - offset + shift) % 26 + offset
            result += chr(shifted)
        else:
            # Leave non-alphabet characters unchanged
            result += char

    return result
```

UNIT-9 FILE HANDLING

Program 1 – Create a new text file and write a string into it

```
# Program 1: Create a text file and write content

file = open("sample.txt", "w")  # 'w' mode creates a new file
file.write("Hello! This is a sample file created using Python.\nWelcome to File Handli
file.close()

print("File 'sample.txt' created and data written successfully.")
```

Output:

```
File 'sample.txt' created and data written successfully.
```

Program 2 – Read and display file contents line by line

```
# Program 2: Read a file line by line

file = open("sample.txt", "r")
for line in file:
    print(line.strip())
file.close()
```

Output:

```
Hello! This is a sample file created using Python.
Welcome to File Handling.
```

Program 3 – Count lines, words, and characters

```
# Program 3: Count number of lines, words, and characters

file = open("sample.txt", "r")
lines = file.readlines()
file.close()

num_lines = len(lines)
num_words = sum(len(line.split()) for line in lines)
num_chars = sum(len(line) for line in lines)

print(f"Lines: {num_lines}, Words: {num_words}, Characters: {num_chars}")
```

Output Example:

```
Lines: 2, Words: 11, Characters: 78
```

Program 4 – Append user input to an existing text file

```
# Program 4: Append data to file

text = input("Enter text to append: ")
file = open("sample.txt", "a")
file.write("\n" + text)
file.close()

print("Data appended successfully!")
```

Output Example:

```
Enter text to append: This is new data added.
Data appended successfully!
```

Program 5 – Copy contents from one file to another

```python
# Program 5: Copy file content

src = open("sample.txt", "r")
dest = open("copy_sample.txt", "w")

for line in src:
    dest.write(line)

src.close()
dest.close()

print("File copied successfully.")
```

Output:

```
File copied successfully.
```

Program 6 – Read only the first n lines

```python
# Program 6: Read first n lines

n = int(input("Enter number of lines to read: "))
file = open("sample.txt", "r")

for i in range(n):
    print(file.readline().strip())

file.close()
```

Output Example:

```
Enter number of lines to read: 1
Hello! This is a sample file created using Python.
```

Program 7 – Read the last n lines of a file

```python
# Program 7: Read last n lines

n = int(input("Enter number of last lines to read: "))
with open("sample.txt", "r") as file:
    lines = file.readlines()
    for line in lines[-n:]:
        print(line.strip())
```

Output Example:

```
Enter number of last lines to read: 2
Welcome to File Handling.
This is new data added.
```

Program 8 – Check if a file exists before reading/writing

```python
# Program 8: Check if file exists

import os
```

```
filename = "sample.txt"
if os.path.exists(filename):
    print(f"File '{filename}' exists.")
else:
    print(f"File '{filename}' not found!")
```

Output Example:

```
File 'sample.txt' exists.
```

Program 9 – Rename and delete a file

```
# Program 9: Rename and delete file

import os

os.rename("copy_sample.txt", "renamed_sample.txt")
print("File renamed successfully!")

os.remove("renamed_sample.txt")
print("File deleted successfully!")
```

Output:

```
File renamed successfully!
File deleted successfully!
```

Program 10 – Find file size in bytes

```
# Program 10: Find file size

import os

filename = "sample.txt"
if os.path.exists(filename):
    size = os.path.getsize(filename)
    print(f"Size of '{filename}' = {size} bytes")
else:
    print("File not found!")
```

Output Example:

```
Size of 'sample.txt' = 88 bytes
```

Program 11 – Count frequency of each word in a text file

```
# Program 11: Count word frequency in a file

from collections import Counter

with open("sample.txt", "r") as f:
    words = f.read().lower().split()
    freq = Counter(words)

print("Word Frequency:\n", dict(freq))
```

Output Example

```
Word Frequency:
{'this': 2, 'is': 2, 'a': 1, 'sample': 1, 'file': 1, 'created': 1, 'using': 1, 'python
```

Program 12 – Find and replace a word in a file

```
# Program 12: Find and replace word in a file

old_word = input("Enter word to replace: ")
new_word = input("Enter new word: ")

with open("sample.txt", "r") as f:
    data = f.read()

data = data.replace(old_word, new_word)

with open("sample.txt", "w") as f:
    f.write(data)

print(f"'{old_word}' replaced with '{new_word}' successfully.")
```

Output Example

```
Enter word to replace: Python
Enter new word: PYTHON
'Python' replaced with 'PYTHON' successfully.
```

Program 13 – Find the longest word in a text file

```
# Program 13: Find longest word

with open("sample.txt", "r") as f:
    words = f.read().split()
    longest = max(words, key=len)

print("Longest word:", longest)
```

Output

```
Longest word: successfully.
```

Program 14 – Count how many times each vowel appears

```
# Program 14: Count vowels in a file

vowels = "aeiou"
count = {v: 0 for v in vowels}

with open("sample.txt", "r") as f:
    text = f.read().lower()
    for ch in text:
        if ch in count:
            count[ch] += 1

print("Vowel frequency:", count)
```

Output Example

```
Vowel frequency: {'a': 4, 'e': 6, 'i': 3, 'o': 2, 'u': 1}
```

Program 15 – Read numeric data and compute sum + average

```
# Program 15: Read numbers from file and find sum & average

with open("numbers.txt", "r") as f:
    nums = [float(x) for x in f.read().split()]

total = sum(nums)
avg = total / len(nums)

print("Sum =", total)
print("Average =", avg)
```

Output Example

```
Sum = 245.0
Average = 49.0
```

Program 16 – Remove all blank lines from a text file

```
# Program 16: Remove blank lines

with open("sample.txt", "r") as f:
    lines = f.readlines()

non_blank = [line for line in lines if line.strip()]

with open("sample.txt", "w") as f:
    f.writelines(non_blank)

print("Blank lines removed successfully.")
```

Output

```
Blank lines removed successfully.
```

Program 17 – Find all unique words and store in a new file

```
# Program 17: Store unique words in a new file

with open("sample.txt", "r") as f:
    words = set(f.read().split())

with open("unique_words.txt", "w") as out:
    out.write("\n".join(sorted(words)))

print("Unique words saved to 'unique_words.txt'")
```

Output

```
Unique words saved to 'unique_words.txt'
```

Program 18 – Compare two files and show differences

```
# Program 18: Compare files line by line

file1 = open("sample.txt", "r").readlines()
file2 = open("copy_sample.txt", "r").readlines()

for i, (l1, l2) in enumerate(zip(file1, file2), start=1):
    if l1 != l2:
        print(f"Line {i} differs:\nFile1: {l1.strip()}\nFile2: {l2.strip()}")
```

Output Example

```
Line 2 differs:
File1: Welcome to PYTHON.
File2: Welcome to Python.
```

Program 19 – Merge two text files into one

```
# Program 19: Merge two files

with open("sample.txt", "r") as f1, open("copy_sample.txt", "r") as f2, open("merged.t
    out.write(f1.read() + "\n" + f2.read())

print("Files merged successfully into 'merged.txt'")
```

Output

```
Files merged successfully into 'merged.txt'
```

Program 20 – Sort the lines of a file alphabetically

```
# Program 20: Sort file lines alphabetically

with open("sample.txt", "r") as f:
    lines = sorted(f.readlines())

with open("sorted_sample.txt", "w") as f:
    f.writelines(lines)

print("Lines sorted and saved to 'sorted_sample.txt'")
```

Output

```
Lines sorted and saved to 'sorted_sample.txt'
```

Program 21 – Read a CSV file and display its contents

```
# Program 21: Read a CSV file

import csv

with open("students.csv", "r") as f:
    reader = csv.reader(f)
    for row in reader:
        print(row)
```

Output Example

```
['Roll', 'Name', 'Marks']
['1', 'Ravi', '85']
['2', 'Anita', '90']
```

Program 22 – Write student records into a CSV file

```
# Program 22: Write student data to CSV

import csv

header = ['Roll', 'Name', 'Marks']
data = [
    [1, 'Ravi', 85],
    [2, 'Anita', 90],
    [3, 'Vikas', 78]
]
```

```
with open("students.csv", "w", newline="") as f:
    writer = csv.writer(f)
    writer.writerow(header)
    writer.writerows(data)

print("Student records saved to 'students.csv'")
```

Output

```
Student records saved to 'students.csv'
```

Program 23 – Find student with highest marks

```
# Program 23: Highest marks from CSV

import csv

with open("students.csv", "r") as f:
    reader = csv.DictReader(f)
    top_student = max(reader, key=lambda x: int(x['Marks']))

print("Topper:", top_student['Name'], "| Marks:", top_student['Marks'])
```

Output

```
Topper: Anita | Marks: 90
```

Program 24 – Count rows and columns in a CSV file

```
# Program 24: Count rows and columns

import csv

with open("students.csv", "r") as f:
    reader = list(csv.reader(f))
    rows = len(reader) - 1     # excluding header
    cols = len(reader[0])

print(f"Rows: {rows}, Columns: {cols}")
```

Output

```
Rows: 3, Columns: 3
```

Program 25 – Update a particular cell in CSV

```
# Program 25: Update a cell in CSV

import csv

with open("students.csv", "r") as f:
    data = list(csv.reader(f))

# update marks of roll 3
for row in data:
    if row[0] == '3':
        row[2] = '88'

with open("students.csv", "w", newline="") as f:
    csv.writer(f).writerows(data)
```

```
print("Marks updated for roll 3 successfully.")
```

Output

```
Marks updated for roll 3 successfully.
```

Program 26 – Convert CSV data to JSON

```
# Program 26: Convert CSV → JSON

import csv, json

with open("students.csv", "r") as f:
    reader = csv.DictReader(f)
    data = list(reader)

with open("students.json", "w") as jf:
    json.dump(data, jf, indent=4)

print("CSV converted to JSON and saved as 'students.json'")
```

Output

```
CSV converted to JSON and saved as 'students.json'
```

Program 27 – Read data from JSON file

```
# Program 27: Read JSON file

import json

with open("students.json", "r") as f:
    data = json.load(f)

for student in data:
    print(student)
```

Output

```
{'Roll': '1', 'Name': 'Ravi', 'Marks': '85'}
{'Roll': '2', 'Name': 'Anita', 'Marks': '90'}
```

Program 28 – Create JSON file with employee details

```
# Program 28: Create JSON file

import json

employees = [
    {"id": 101, "name": "Rahul", "dept": "IT", "salary": 75000},
    {"id": 102, "name": "Priya", "dept": "HR", "salary": 68000}
]

with open("employees.json", "w") as f:
    json.dump(employees, f, indent=4)

print("Employee details saved to 'employees.json'")
```

Output

```
Employee details saved to 'employees.json'
```

Program 29 – Update a key-value pair in a JSON file

```python
# Program 29: Update JSON data

import json

with open("employees.json", "r") as f:
    data = json.load(f)

# update salary for id 102
for emp in data:
    if emp["id"] == 102:
        emp["salary"] = 70000

with open("employees.json", "w") as f:
    json.dump(data, f, indent=4)

print("Employee data updated successfully.")
```

Output

```
Employee data updated successfully.
```

Program 30 – Merge two JSON files into one

```python
# Program 30: Merge JSON files

import json

with open("students.json", "r") as f1, open("employees.json", "r") as f2:
    data1 = json.load(f1)
    data2 = json.load(f2)

merged = {"students": data1, "employees": data2}

with open("merged_data.json", "w") as f:
    json.dump(merged, f, indent=4)

print("Two JSON files merged into 'merged_data.json'")
```

Output

```
Two JSON files merged into 'merged_data.json'
```

Program 31 – Encrypt and decrypt file content (Caesar cipher)

```python
# Program 31: Simple Caesar cipher encryption & decryption

def encrypt(text, shift):
    return "".join(chr((ord(ch) + shift) % 256) for ch in text)

def decrypt(text, shift):
    return "".join(chr((ord(ch) - shift) % 256) for ch in text)

with open("secret.txt", "w") as f:
    f.write("Python File Handling is fun!")
```

```
# Encrypt
with open("secret.txt", "r") as f:
    plain = f.read()
enc = encrypt(plain, 3)
open("secret.txt", "w").write(enc)

# Decrypt
with open("secret.txt", "r") as f:
    cipher = f.read()
dec = decrypt(cipher, 3)

print("Decrypted text:", dec)
```

Output

```
Decrypted text: Python File Handling is fun!
```

Program 32 – Remove duplicate lines from a file

```
# Program 32: Remove duplicate lines

with open("sample.txt", "r") as f:
    lines = f.readlines()

unique = list(dict.fromkeys(lines))  # preserve order

with open("sample.txt", "w") as f:
    f.writelines(unique)

print("Duplicate lines removed successfully.")
```

Program 33 – Split a large text file into smaller parts

```
# Program 33: Split file into chunks of n lines

n = 3
with open("sample.txt", "r") as f:
    lines = f.readlines()

for i in range(0, len(lines), n):
    part = lines[i:i+n]
    with open(f"part_{i//n + 1}.txt", "w") as out:
        out.writelines(part)

print("File split into parts successfully.")
```

Program 34 – Combine multiple files into one

```
# Program 34: Combine multiple files

files = ["part_1.txt", "part_2.txt", "part_3.txt"]

with open("combined.txt", "w") as out:
    for name in files:
        with open(name, "r") as f:
            out.write(f.read() + "\n")

print("Files combined successfully.")
```

Program 35 – Log all read/write actions to a separate file

```
# Program 35: Maintain an operation log
```

```
from datetime import datetime

def log_action(action):
    with open("actions.log", "a") as log:
        log.write(f"{datetime.now()} - {action}\n")

with open("logfile.txt", "w") as f:
    f.write("Initial content.")
log_action("Created logfile.txt")

with open("logfile.txt", "a") as f:
    f.write("\nAppended data.")
log_action("Appended data to logfile.txt")

print("Actions logged in 'actions.log'")
```

Program 36 – Search for a keyword in all files of a folder

```
# Program 36: Search keyword in all text files

import os

keyword = input("Enter keyword to search: ")
path = "."

for file in os.listdir(path):
    if file.endswith(".txt"):
        with open(file, "r") as f:
            if keyword in f.read():
                print(f"'{keyword}' found in {file}")
```

Output Example

```
Enter keyword to search: Python
'Python' found in sample.txt
'Python' found in combined.txt
```

Program 37 – Read a binary file and print hexadecimal representation

```
# Program 37: Binary → Hex dump

with open("image.png", "rb") as f:
    data = f.read(20)  # read first 20 bytes
print("Hex:", data.hex())
```

Output

```
Hex: 89504e470d0a1a0a0000000d494844520000...
```

Program 38 – Copy an image (binary file)

```
# Program 38: Copy binary file

with open("image.png", "rb") as src, open("copy_image.png", "wb") as dst:
    dst.write(src.read())

print("Image copied successfully.")
```

Program 39 – Track changes made to a file (version log)

```
# Program 39: Version control-like logging
```

```
import os, time

filename = "versioned.txt"
with open(filename, "w") as f:
    f.write("Initial data\n")

if not os.path.exists("version_log.txt"):
    open("version_log.txt", "w").close()

def log_version():
    timestamp = time.strftime("%Y-%m-%d_%H-%M-%S")
    with open(filename, "r") as src, open(f"backup_{timestamp}.txt", "w") as dst:
        dst.write(src.read())
    with open("version_log.txt", "a") as log:
        log.write(f"Backup created at {timestamp}\n")

log_version()
print("Version log updated.")
```

Program 40 – Compress and decompress a text file using gzip

```
# Program 40: Compress & decompress text

import gzip, shutil

# Compress
with open("sample.txt", "rb") as f_in, gzip.open("sample.txt.gz", "wb") as f_out:
    shutil.copyfileobj(f_in, f_out)
print("File compressed successfully.")

# Decompress
with gzip.open("sample.txt.gz", "rb") as f_in, open("uncompressed.txt", "wb")
as f_out:
    shutil.copyfileobj(f_in, f_out)
print("File decompressed successfully.")
```

Output

```
File compressed successfully.
File decompressed successfully.
```

Program 41 – Count total error lines in a log file

```
# Program 41: Count ERROR lines in server.log

count = 0
with open("server.log", "r") as f:
    for line in f:
        if "ERROR" in line:
            count += 1

print(f"Total ERROR lines: {count}")
```

Output Example

```
Total ERROR lines: 5
```

Program 42 – Calculate total debit and credit from transactions.txt

```
# Program 42: Sum debit and credit

debit_total = 0
credit_total = 0
```

```
with open("transactions.txt", "r") as f:
    for line in f:
        parts = line.strip().split(',')
        type_, amount = parts[0], float(parts[1])
        if type_.lower() == "debit":
            debit_total += amount
        elif type_.lower() == "credit":
            credit_total += amount

print(f"Total Debit: {debit_total}, Total Credit: {credit_total}")
```

Output Example

```
Total Debit: 1200.0, Total Credit: 3500.0
```

Program 43 – Department-wise average salary from CSV

```
# Program 43: Average salary per department

import csv
from collections import defaultdict

dept_salaries = defaultdict(list)

with open("employees.csv", "r") as f:
    reader = csv.DictReader(f)
    for row in reader:
        dept_salaries[row['dept']].append(float(row['salary']))

for dept, salaries in dept_salaries.items():
    avg = sum(salaries)/len(salaries)
    print(f"{dept}: Average Salary = {avg}")
```

Output Example

```
IT: Average Salary = 75000.0
HR: Average Salary = 70000.0
```

Program 44 – Generate a report file summarizing student grades

```
# Program 44: Student grade report

import csv

with open("students.csv", "r") as f:
    reader = csv.DictReader(f)
    report = []
    for row in reader:
        marks = int(row['Marks'])
        grade = 'A' if marks >= 90 else 'B' if marks >= 75 else 'C'
        report.append(f"{row['Name']} | Marks: {marks} | Grade: {grade}")

with open("student_report.txt", "w") as f:
    f.write("\n".join(report))

print("Student report generated: 'student_report.txt'")
```

Program 45 – Separate names by first letter into multiple files

```
# Program 45: Names separated by first letter
```

```
with open("names.txt", "r") as f:
    for name in f:
        name = name.strip()
        if name:
            with open(f"{name[0].upper()}.txt", "a") as out:
                out.write(name + "\n")

print("Names separated by first letter successfully.")
```

Program 46 – Count messages per user from a chat log

```
# Program 46: Count messages per user

from collections import Counter

user_count = Counter()

with open("chat.txt", "r") as f:
    for line in f:
        if ":" in line:
            user = line.split(":")[0].strip()
            user_count[user] += 1

print("Messages per user:", dict(user_count))
```

Program 47 – Parse configuration file into a dictionary

```
# Program 47: Parse config.txt

config = {}

with open("config.txt", "r") as f:
    for line in f:
        if "=" in line:
            key, value = line.strip().split("=", 1)
            config[key.strip()] = value.strip()

print("Configuration Loaded:", config)
```

Program 48 – Extract all email addresses from a text file

```
# Program 48: Extract emails

import re

emails = []
with open("data.txt", "r") as f:
    text = f.read()
    emails = re.findall(r"[a-zA-Z0-9._%+-]+@[a-zA-Z0-9.-]+\.[a-zA-Z]{2,}", text)

with open("emails.txt", "w") as out:
    out.write("\n".join(emails))

print(f"{len(emails)} email addresses saved to 'emails.txt'")
```

Program 49 – Mini text editor (open, edit, save)

```
# Program 49: Mini text editor

filename = input("Enter filename to edit: ")

print("Enter text (type 'SAVE' to finish):")
lines = []
```

```
while True:
    line = input()
    if line.upper() == "SAVE":
        break
    lines.append(line)

with open(filename, "w") as f:
    f.write("\n".join(lines))

print(f"File '{filename}' saved successfully.")
```

Program 50 – Backup system that copies file on modification

```
# Program 50: Automatic backup on modification

import shutil, os, time

filename = "important.txt"
backup_dir = "backup"

if not os.path.exists(backup_dir):
    os.makedirs(backup_dir)

last_modified = os.path.getmtime(filename)

while True:
    time.sleep(5)  # check every 5 seconds
    current_modified = os.path.getmtime(filename)
    if current_modified != last_modified:
        shutil.copy(filename, backup_dir)
        print(f"Backup created for '{filename}' at {time.ctime()}")
        last_modified = current_modified
```

Output Example

```
Backup created for 'important.txt' at Sun Oct 13 10:45:12 2025
```

UNIT-10 EXCEPTIONAL HANDLING

1. Handle division by zero

```
try:
    a = int(input("Enter numerator: "))
    b = int(input("Enter denominator: "))
    result = a / b
except ZeroDivisionError:
    print("Cannot divide by zero")
else:
    print("Result:", result)
```

2. Handle invalid integer input

```
try:
    num = int(input("Enter an integer: "))
except ValueError:
    print("Invalid input! Please enter a number.")
else:
    print("You entered:", num)
```

3. Handle IndexError in list

```
lst = [1, 2, 3]
try:
    print(lst[5])
except IndexError:
    print("Index out of range")
```

4. Handle KeyError in dictionary

```
d = {"a": 1, "b": 2}
try:
    print(d["c"])
except KeyError:
    print("Key not found in dictionary")
```

5. Multiple exceptions (ZeroDivisionError + ValueError)

```
try:
    x = int(input("Enter number: "))
    y = int(input("Enter divisor: "))
    print(x / y)
except ZeroDivisionError:
    print("Division by zero!")
except ValueError:
    print("Invalid number entered!")
```

6. Use `else` block

```
try:
    x = int(input("Enter number: "))
except ValueError:
    print("Invalid input!")
else:
    print("You entered:", x)
```

7. Use `finally` block with file

```
try:
    f = open("sample.txt", "r")
except FileNotFoundError:
    print("File not found")
else:
    print(f.read())
finally:
    print("Execution completed")
```

8. Handle TypeError (string + int)

```
try:
    result = "10" + 5
except TypeError:
    print("Cannot add string and integer")
```

9. Handle AttributeError

```
s = "hello"
try:
    s.append("world")
except AttributeError:
    print("Attribute does not exist for string")
```

10. Handle FileNotFoundError

```
try:
    with open("nofile.txt", "r") as f:
        print(f.read())
except FileNotFoundError:
    print("File does not exist")
```

11. Handle ImportError

```
try:
    import non_existent_module
except ImportError:
    print("Module not found")
```

12. Handle NameError

```
try:
    print(x)
except NameError:
    print("Variable is not defined")
```

13. Use `pass` in exception

```
try:
    print(10/0)
except ZeroDivisionError:
    pass  # Ignore division error
print("Program continues")
```

14. `try-except-else-finally` flow

```
try:
    x = int(input("Enter number: "))
except ValueError:
    print("Invalid input")
else:
    print("Number is:", x)
finally:
    print("Program ends")
```

15. Raise ValueError if negative

```
num = int(input("Enter number: "))
if num < 0:
    raise ValueError("Negative number not allowed")
else:
    print("Number is:", num)
```

16. Handle invalid string to int conversion

```
s = "abc"
try:
    num = int(s)
except ValueError:
    print("Cannot convert string to integer")
```

17. ZeroDivisionError inside loop

```
nums = [10, 0, 5]
for n in nums:
    try:
        print(10 / n)
    except ZeroDivisionError:
        print("Division by zero encountered")
```

18. IndexError in loop

```
lst = [1,2]
for i in range(5):
    try:
        print(lst[i])
    except IndexError:
        print(f"No element at index {i}")
```

19. Catch all exceptions with Exception class

```
try:
    x = int("abc") / 0
except Exception as e:
    print("Exception occurred:", e)
```

20. Validate user input until correct

```
while True:
    try:
        num = int(input("Enter integer: "))
        break
    except ValueError:
        print("Invalid input, try again")
print("Valid number entered:", num)
```

21. Handle KeyboardInterrupt gracefully

```
try:
    while True:
        pass
except KeyboardInterrupt:
    print("Program interrupted by user")
```

22. Handle OSError

```
try:
    f = open("/root/test.txt", "r")
except OSError:
    print("Cannot open file due to OS error")
```

23. Handle UnicodeEncodeError

```
s = "नमस्ते"
try:
    s.encode('ascii')
except UnicodeEncodeError:
    print("Cannot encode non-ASCII characters")
```

24. Handle ValueError and re-enter

```
while True:
    try:
        age = int(input("Enter your age: "))
        break
    except ValueError:
        print("Please enter a valid number")
print("Age:", age)
```

25. Nested try-except

```
try:
    x = int(input("Enter number: "))
    try:
        print(10 / x)
    except ZeroDivisionError:
        print("Inner: Division by zero")
except ValueError:
    print("Outer: Invalid input")
```

26. Handle multiple exceptions for file operations

```
try:
    f = open("nofile.txt", "r")
    num = int("abc")
except FileNotFoundError:
    print("File not found")
except ValueError:
    print("Invalid integer conversion")
```

27. Raise and catch IndexError manually

```
try:
    lst = [1, 2, 3]
    if len(lst) < 5:
        raise IndexError("List index out of range")
except IndexError as e:
    print("Caught exception:", e)
```

28. Handle KeyError in nested dictionaries

```
data = {"student": {"name": "Ravi", "age": 20}}
try:
    print(data["student"]["grade"])
except KeyError:
    print("Key 'grade' not found in nested dictionary")
```

29. Catch exception dividing number by user input

```
try:
    x = 10
    y = int(input("Enter divisor: "))
    print(x / y)
except ZeroDivisionError:
    print("Cannot divide by zero")
```

30. Handle exception on empty list pop

```
lst = []
try:
    lst.pop()
except IndexError:
    print("Cannot pop from empty list")
```

31. Handle exception opening multiple files

```
files = ["file1.txt", "file2.txt"]
for fname in files:
    try:
        f = open(fname, "r")
    except FileNotFoundError:
        print(f"{fname} not found")
```

32. Catch TypeError in function arguments

```
def add(a, b):
    try:
        return a + b
    except TypeError:
        print("Cannot add different data types")

add(10, "20")
```

33. Handle ZeroDivisionError in function and return None

```
def divide(a, b):
    try:
        return a / b
    except ZeroDivisionError:
        print("Division by zero")
        return None

print(divide(10, 0))
```

34. Handle exception in reading JSON file

```
import json
try:
    data = '{"name": "Ravi", "age": 20'  # Invalid JSON
    json.loads(data)
except json.JSONDecodeError:
    print("Invalid JSON format")
```

35. Raise TypeError if function argument is not string
```
def greet(name):
    if not isinstance(name, str):
        raise TypeError("Name must be string")
    print("Hello", name)

try:
    greet(123)
except TypeError as e:
    print("Caught exception:", e)
```

36. Handle AttributeError in object without method
```
class Person:
    def __init__(self, name):
        self.name = name

p = Person("Ravi")
try:
    p.say_hello()
except AttributeError:
    print("Method does not exist")
```

37. Catch ValueError parsing multiple inputs
```
s = "10,abc,30"
for val in s.split(","):
    try:
        num = int(val)
        print("Number:", num)
    except ValueError:
        print(f"Invalid integer: {val}")
```

38. Handle exception computing square roots
```
import math
nums = [16, -4, 25]
for n in nums:
    try:
        print(math.sqrt(n))
    except ValueError:
        print(f"Cannot compute square root of negative number: {n}")
```

39. File write with try-except-finally
```
try:
    f = open("test.txt", "w")
    f.write("Hello")
except IOError:
    print("Cannot write to file")
finally:
    f.close()
    print("File closed")
```

40. Handle IOError for reading non-existent file
```
try:
    with open("nofile.txt") as f:
        f.read()
except IOError:
    print("Error reading file")
```

41. Convert list of strings to integers

```
lst = ["10", "20", "abc"]
for val in lst:
    try:
        print(int(val))
    except ValueError:
        print(f"Cannot convert '{val}' to integer")
```

42. Raise ZeroDivisionError manually

```
try:
    raise ZeroDivisionError("Manual division by zero")
except ZeroDivisionError as e:
    print("Caught:", e)
```

43. Handle OverflowError in exponentiation

```
try:
    result = 2 ** 10000
except OverflowError:
    print("Number too large")
else:
    print("Exponentiation succeeded")
```

44. Handle FileNotFoundError and prompt

```
while True:
    try:
        fname = input("Enter filename: ")
        f = open(fname)
        break
    except FileNotFoundError:
        print("File not found. Try again")
```

45. Handle ImportError dynamically

```
module_name = "nonexistent"
try:
    __import__(module_name)
except ImportError:
    print(f"Module {module_name} not found")
```

46. Use finally to close file

```
try:
    f = open("sample.txt")
except FileNotFoundError:
    print("File not found")
finally:
    try:
        f.close()
    except NameError:
        pass
```

47. Catch StopIteration in iterator

```
lst = [1, 2]
it = iter(lst)
try:
    for _ in range(3):
        print(next(it))
except StopIteration:
    print("No more items in iterator")
```

48. Function returning division of two numbers

```
def safe_div(a, b):
    try:
        return a / b
    except ZeroDivisionError:
        print("Cannot divide by zero")
        return None

print(safe_div(10, 0))
```

49. KeyError in student database lookup

```
students = {"Ravi": 90, "Anita": 85}
try:
    print(students["Vikas"])
except KeyError:
    print("Student not found")
```

50. Access tuple element by index safely

```
t = (1, 2, 3)
try:
    print(t[5])
except IndexError:
    print("Tuple index out of range")
```

51. Banking system – withdraw exceeding balance

```
class Account:
    def __init__(self, name, balance):
        self.name = name
        self.balance = balance

    def withdraw(self, amount):
        try:
            if amount > self.balance:
                raise ValueError("Insufficient balance")
            self.balance -= amount
            print(f"{amount} withdrawn. Balance: {self.balance}")
        except ValueError as e:
            print("Error:", e)

acc = Account("Ravi", 5000)
acc.withdraw(6000)
```

52. ATM simulation – invalid PIN, insufficient balance

```
correct_pin = "1234"
balance = 3000

pin = input("Enter PIN: ")
try:
    if pin != correct_pin:
        raise ValueError("Invalid PIN")
    withdraw_amt = int(input("Enter amount to withdraw: "))
    if withdraw_amt > balance:
        raise ValueError("Insufficient balance")
    balance -= withdraw_amt
    print("Withdraw successful. Balance:", balance)
except ValueError as e:
    print("Error:", e)
```

53. Flight booking – unavailable seat

```
available_seats = ["A1", "A2"]
seat = input("Enter seat to book: ")
try:
    if seat not in available_seats:
        raise Exception("Seat not available")
    available_seats.remove(seat)
    print(f"Seat {seat} booked successfully")
except Exception as e:
    print("Error:", e)
```

54. Hotel room booking – no rooms left

```
rooms = [101, 102]
try:
    room_to_book = int(input("Enter room number: "))
    if room_to_book not in rooms:
        raise Exception("Room not available")
    rooms.remove(room_to_book)
    print(f"Room {room_to_book} booked")
except Exception as e:
    print("Error:", e)
```

55. Library management – book not available

```
books = {"Python": 2, "Java": 1}
book = input("Enter book to borrow: ")
try:
    if book not in books or books[book] == 0:
        raise Exception("Book not available")
    books[book] -= 1
    print(f"{book} borrowed successfully")
except Exception as e:
    print("Error:", e)
```

56. Online shopping cart – product out of stock

```
cart = {"Laptop": 1}
inventory = {"Laptop": 0, "Mouse": 10}

try:
    for item, qty in cart.items():
        if inventory.get(item, 0) < qty:
            raise Exception(f"{item} out of stock")
        inventory[item] -= qty
    print("Order placed successfully")
except Exception as e:
    print("Error:", e)
```

57. Payroll system – invalid employee data

```
employees = {"Ravi": 5000, "Anita": 6000}
emp = input("Enter employee name: ")
try:
    if emp not in employees:
        raise KeyError("Employee not found")
    print(f"Salary of {emp}: {employees[emp]}")
except KeyError as e:
    print("Error:", e)
```

58. Cinema ticket booking – seat already booked

```
booked_seats = ["A1", "A2"]
```

```
seat = input("Enter seat to book: ")
try:
    if seat in booked_seats:
        raise Exception("Seat already booked")
    booked_seats.append(seat)
    print(f"Seat {seat} booked successfully")
except Exception as e:
    print("Error:", e)
```

59. Quiz system – invalid answer input

```
valid_answers = ["a", "b", "c", "d"]
ans = input("Enter your answer (a/b/c/d): ").lower()
try:
    if ans not in valid_answers:
        raise ValueError("Invalid answer choice")
    print("Answer accepted")
except ValueError as e:
    print("Error:", e)
```

60. School grading system – marks out of range

```
marks = int(input("Enter marks: "))
try:
    if not (0 <= marks <= 100):
        raise ValueError("Marks must be between 0 and 100")
    print("Marks accepted")
except ValueError as e:
    print("Error:", e)
```

61. Gym membership – invalid member ID

```
members = ["M001", "M002"]
mid = input("Enter member ID: ")
try:
    if mid not in members:
        raise KeyError("Member ID not found")
    print("Member verified")
except KeyError as e:
    print("Error:", e)
```

62. Smart home devices – non-existent device

```
devices = ["TV", "AC", "Light"]
device = input("Enter device to switch ON: ")
try:
    if device not in devices:
        raise Exception("Device not found")
    print(f"{device} switched ON")
except Exception as e:
    print("Error:", e)
```

63. Digital library – borrowing already borrowed book

```
books = {"Python": "available", "Java": "borrowed"}
book = input("Enter book to borrow: ")
try:
    if books.get(book) != "available":
        raise Exception("Book already borrowed")
    books[book] = "borrowed"
    print(f"{book} borrowed successfully")
except Exception as e:
    print("Error:", e)
```

64. Multi-project task management – task already completed

```
tasks = {"Design": "Completed", "Implement": "Pending"}
task = input("Enter task to complete: ")
try:
    if tasks.get(task) == "Completed":
        raise Exception("Task already completed")
    tasks[task] = "Completed"
    print(f"{task} marked as completed")
except Exception as e:
    print("Error:", e)
```

65. Food ordering system – negative quantity

```
qty = int(input("Enter quantity: "))
try:
    if qty <= 0:
        raise ValueError("Quantity must be positive")
    print(f"Order placed for {qty} items")
except ValueError as e:
    print("Error:", e)
```

66. Flight fare calculation – invalid seat class

```
fares = {"Economy": 5000, "Business": 10000}
seat = input("Enter seat class: ")
try:
    if seat not in fares:
        raise Exception("Invalid seat class")
    print(f"Fare: {fares[seat]}")
except Exception as e:
    print("Error:", e)
```

67. Hotel multi-room booking – not enough rooms

```
available_rooms = [101, 102]
req = int(input("Enter number of rooms to book: "))
try:
    if req > len(available_rooms):
        raise Exception("Not enough rooms available")
    for _ in range(req):
        room = available_rooms.pop()
        print(f"Room {room} booked")
except Exception as e:
    print("Error:", e)
```

68. Vehicle parking – parking full

```
parking_slots = 2
vehicles = ["Car1", "Car2"]
vehicle = input("Enter vehicle name: ")
try:
    if len(vehicles) >= parking_slots:
        raise Exception("Parking full")
    vehicles.append(vehicle)
    print(f"{vehicle} parked")
except Exception as e:
    print("Error:", e)
```

69. Employee hierarchy – invalid employee type

```
emp_types = ["Manager", "Staff"]
etype = input("Enter employee type: ")
```

```
try:
    if etype not in emp_types:
        raise Exception("Invalid employee type")
    print(f"{etype} assigned")
except Exception as e:
    print("Error:", e)
```

70. Inventory management – negative stock

```
stock = {"Laptop": 5}
item = input("Enter item: ")
qty = int(input("Enter quantity to reduce: "))
try:
    if qty > stock.get(item, 0):
        raise Exception("Insufficient stock")
    stock[item] -= qty
    print(f"{qty} {item} sold")
except Exception as e:
    print("Error:", e)
```

71. E-commerce order – total exceeds limit

```
limit = 10000
total = int(input("Enter order total: "))
try:
    if total > limit:
        raise Exception("Order exceeds maximum limit")
    print("Order accepted")
except Exception as e:
    print("Error:", e)
```

72. Payroll with overtime – negative hours

```
def payroll(hours):
    try:
        if hours < 0:
            raise ValueError("Hours cannot be negative")
        return hours * 200
    except ValueError as e:
        print("Error:", e)

payroll(-5)
```

73. Smart home scenes – scene not defined

```
scenes = ["Morning", "Night"]
scene = input("Enter scene to activate: ")
try:
    if scene not in scenes:
        raise Exception("Scene not defined")
    print(f"{scene} scene activated")
except Exception as e:
    print("Error:", e)
```

74. Banking system with transaction history – invalid amount

```
transactions = []
amt = int(input("Enter amount to deposit: "))
try:
    if amt <= 0:
        raise ValueError("Amount must be positive")
    transactions.append(amt)
    print("Transaction successful")
```

```
except ValueError as e:
    print("Error:", e)
```

75. Quiz leaderboard – negative score / invalid participant

```
participants = {"Ravi": 5, "Anita": 8}
name = input("Enter participant name: ")
score = int(input("Enter score to update: "))
try:
    if name not in participants:
        raise Exception("Participant not found")
    if score < 0:
        raise ValueError("Score cannot be negative")
    participants[name] = score
    print(participants)
except (Exception, ValueError) as e:
    print("Error:", e)
```

76. Handle multiple exceptions for file parsing (JSON + CSV)

```
import json
import csv

files = ["data.json", "data.csv"]
for file in files:
    try:
        if file.endswith(".json"):
            with open(file) as f:
                data = json.load(f)
        elif file.endswith(".csv"):
            with open(file) as f:
                reader = csv.reader(f)
                data = list(reader)
    except FileNotFoundError:
        print(f"{file} not found")
    except (json.JSONDecodeError, csv.Error) as e:
        print(f"Error parsing {file}: {e}")
```

77. Handle exception reading corrupted Excel file

```
import pandas as pd
try:
    df = pd.read_excel("corrupt.xlsx")
except FileNotFoundError:
    print("File not found")
except Exception as e:
    print("Error reading Excel file:", e)
```

78. Multi-threaded application accessing shared data

```
import threading

counter = 0
lock = threading.Lock()

def increment():
    global counter
    try:
        lock.acquire()
        counter += 1
    finally:
        lock.release()

threads = [threading.Thread(target=increment) for _ in range(5)]
```

```
for t in threads:
    t.start()
for t in threads:
    t.join()
print("Counter:", counter)
```

79. Network socket – connection errors

```
import socket

try:
    s = socket.socket()
    s.connect(("localhost", 9999))
except ConnectionRefusedError:
    print("Connection refused")
except socket.error as e:
    print("Socket error:", e)
```

80. API response parsing – missing keys

```
response = {"name": "Ravi"}
try:
    age = response["age"]
except KeyError:
    print("Key 'age' missing in API response")
```

81. Multi-level inheritance – missing attribute

```
class A: pass
class B(A): pass
class C(B): pass

obj = C()
try:
    print(obj.value)
except AttributeError:
    print("Attribute not found in multi-level inheritance")
```

82. Database connection & query execution

```
import sqlite3

try:
    conn = sqlite3.connect("test.db")
    cursor = conn.cursor()
    cursor.execute("SELECT * FROM non_existent_table")
except sqlite3.OperationalError as e:
    print("Database error:", e)
finally:
    conn.close()
```

83. Web scraping – invalid URL / timeout

```
import requests

try:
    r = requests.get("https://invalid-url.com", timeout=2)
    r.raise_for_status()
except requests.exceptions.RequestException as e:
    print("Error fetching URL:", e)
```

84. Writing large binary files – disk full

```
try:
    with open("large.bin", "wb") as f:
        f.write(b"x" * 10**9)
except IOError as e:
    print("I/O error:", e)
```

85. Image processing – invalid format

```
from PIL import Image

try:
    img = Image.open("invalid_image.txt")
except IOError:
    print("Cannot open image, invalid format")
```

86. AI/ML dataset preprocessing – missing/NaN values

```
import pandas as pd
df = pd.DataFrame({"A": [1, None, 3]})
try:
    mean = df["A"].mean()
    if df["A"].isnull().any():
        raise ValueError("Missing values in dataset")
except ValueError as e:
    print("Error:", e)
```

87. Pandas DataFrame – missing column

```
import pandas as pd
df = pd.DataFrame({"A": [1, 2]})
try:
    print(df["B"])
except KeyError:
    print("Column 'B' does not exist")
```

88. Numpy array – shape mismatch

```
import numpy as np
a = np.array([1,2])
b = np.array([1,2,3])
try:
    c = a + b
except ValueError as e:
    print("Shape mismatch:", e)
```

89. Multi-file processing with logging

```
files = ["file1.txt", "file2.txt"]
for f in files:
    try:
        with open(f) as file:
            print(file.read())
    except FileNotFoundError:
        print(f"{f} not found, skipping")
```

90. JSON API with retry logic

```
import json
data = '{"name": "Ravi"}'
for i in range(3):
    try:
        d = json.loads(data)
        break
    except json.JSONDecodeError:
        print("JSON parse failed, retrying...")
```

91. Sending emails – SMTP errors

```
import smtplib
try:
    server = smtplib.SMTP("smtp.example.com")
    server.sendmail("from@example.com", "to@example.com", "Hello")
except smtplib.SMTPException as e:
    print("SMTP error:", e)
```

92. GUI (Tkinter) – missing widget

```
import tkinter as tk
root = tk.Tk()
try:
    btn = root.children["nonexistent"]
except KeyError:
    print("Widget not found")
```

93. Multiprocessing pool – worker failure

```
from multiprocessing import Pool

def f(x):
    if x == 2:
        raise ValueError("Invalid value")
    return x*x

with Pool(2) as p:
    results = []
    for i in range(3):
        try:
            results.append(p.apply(f, (i,)))
        except ValueError as e:
            print("Worker failed:", e)
```

94. Data serialization – pickle errors

```
import pickle
class A: pass

try:
    pickle.dumps(lambda x: x)  # Lambdas cannot be pickled
except pickle.PicklingError:
    print("Pickle error")
except Exception as e:
    print("Error:", e)
```

95. URL request – HTTPError / URLError

```
from urllib import request, error
try:
    response = request.urlopen("http://invalid-url")
except error.HTTPError as e:
    print("HTTP Error:", e)
except error.URLError as e:
    print("URL Error:", e)
```

96. Custom iterator – exception handling

```
class MyIter:
    def __init__(self, data):
        self.data = data
        self.index = 0
    def __iter__(self):
        return self
    def __next__(self):
        if self.index >= len(self.data):
            raise StopIteration
        value = self.data[self.index]
        self.index += 1
        return value

it = MyIter([1,2])
try:
    while True:
        print(next(it))
except StopIteration:
    print("Iteration completed")
```

97. Recursive function with base case validation

```
def factorial(n):
    if n < 0:
        raise ValueError("Negative numbers not allowed")
    return 1 if n == 0 else n * factorial(n-1)

try:
    print(factorial(-5))
except ValueError as e:
    print("Error:", e)
```

98. Threaded producer-consumer queue

```
import threading
import queue

q = queue.Queue()

def producer():
    for i in range(3):
        q.put(i)

def consumer():
    try:
        while True:
            item = q.get(timeout=1)
            print("Consumed:", item)
    except queue.Empty:
        print("Queue empty, consumer exits")

t1 = threading.Thread(target=producer)
t2 = threading.Thread(target=consumer)
t1.start()
t2.start()
t1.join()
t2.join()
```

99. Exception chaining (`raise ... from ...`)

```
try:
    try:
        x = int("abc")
    except ValueError as e:
        raise TypeError("Conversion failed") from e
except TypeError as e:
    print("Chained exception:", e)
```

100. Robust exception handling in mini banking project

```
class Bank:
    def __init__(self):
        self.accounts = {}

    def create_account(self, name, balance):
        if balance < 0:
            raise ValueError("Initial balance cannot be negative")
        self.accounts[name] = balance

    def deposit(self, name, amt):
        if amt <= 0:
            raise ValueError("Deposit must be positive")
        self.accounts[name] += amt

    def withdraw(self, name, amt):
        if amt <= 0:
            raise ValueError("Withdraw amount must be positive")
        if amt > self.accounts.get(name, 0):
            raise ValueError("Insufficient balance")
        self.accounts[name] -= amt

bank = Bank()
try:
    bank.create_account("Ravi", 5000)
    bank.deposit("Ravi", 2000)
    bank.withdraw("Ravi", 8000)
except ValueError as e:
    print("Error:", e)
finally:
    print("Account status:", bank.accounts)
```

UNIT-11 OOPS

Program 1 – Define a class `Car` and create an object

```
# Program 1: Define a class and create an object

class Car:
    pass

# Create an object
my_car = Car()
print("Car object created:", my_car)
```

Output

```
Car object created: <__main__.Car object at 0x...>
```

Program 2 – Add attributes and constructor

```
# Program 2: Class with attributes and constructor

class Car:
    def __init__(self, brand, model):
        self.brand = brand
        self.model = model

# Create object
my_car = Car("Toyota", "Corolla")
print(my_car.brand, my_car.model)
```

Output

```
Toyota Corolla
```

Program 3 – Method to display info

```
# Program 3: Add method display_info

class Car:
    def __init__(self, brand, model):
        self.brand = brand
        self.model = model

    def display_info(self):
        print(f"Car Brand: {self.brand}, Model: {self.model}")

my_car = Car("Honda", "Civic")
my_car.display_info()
```

Output

```
Car Brand: Honda, Model: Civic
```

Program 4 – Multiple objects

```
# Program 4: Multiple objects

car1 = Car("Ford", "Mustang")
car2 = Car("BMW", "X5")
```

```
car1.display_info()
car2.display_info()
```

Output

```
Car Brand: Ford, Model: Mustang
Car Brand: BMW, Model: X5
```

Program 5 – Class variable to track objects

```
# Program 5: Class variable

class Car:
    count = 0
    def __init__(self, brand, model):
        self.brand = brand
        self.model = model
        Car.count += 1

car1 = Car("Ford", "Figo")
car2 = Car("Tesla", "Model 3")
print("Total Cars created:", Car.count)
```

Output

```
Total Cars created: 2
```

Program 6 – Destructor method

```
# Program 6: Destructor

class Car:
    def __init__(self, brand):
        self.brand = brand
        print(f"{self.brand} created")

    def __del__(self):
        print(f"{self.brand} destroyed")

car = Car("Audi")
del car
```

Output

```
Audi created
Audi destroyed
```

Program 7 – Student class with marks and percentage

```
# Program 7: Student class with percentage

class Student:
    def __init__(self, name, marks):
        self.name = name
        self.marks = marks

    def percentage(self):
        total = sum(self.marks)
        percent = total / len(self.marks)
        return percent
```

```
s1 = Student("Ravi", [85, 90, 78])
print(f"{s1.name} Percentage: {s1.percentage():.2f}%")
```

Output

```
Ravi Percentage: 84.33%
```

Program 8 – Default attribute values

```
# Program 8: Default values

class Student:
    def __init__(self, name="Unknown", age=18):
        self.name = name
        self.age = age

s1 = Student()
s2 = Student("Anita", 20)
print(s1.name, s1.age)
print(s2.name, s2.age)
```

Output

```
Unknown 18
Anita 20
```

Program 9 – getattr and setattr

```
# Program 9: getattr and setattr

class Student:
    def __init__(self, name, age):
        self.name = name
        self.age = age

s = Student("Vikas", 21)
print(getattr(s, "name"))
setattr(s, "age", 22)
print(s.age)
```

Output

```
Vikas
22
```

Program 10 – Update marks method

```
# Program 10: Update marks

class Student:
    def __init__(self, name, marks):
        self.name = name
        self.marks = marks

    def update_marks(self, new_marks):
        self.marks = new_marks

s = Student("Priya", 80)
s.update_marks(90)
print(f"{s.name} marks: {s.marks}")
```

Output

```
Priya marks: 90
```

Program 11 – Circle class with area

```python
# Program 11: Circle area

import math

class Circle:
    def __init__(self, radius):
        self.radius = radius

    def area(self):
        return math.pi * self.radius ** 2

c = Circle(5)
print(f"Area of circle: {c.area():.2f}")
```

Output

```
Area of circle: 78.54
```

Program 12 – Circle class with circumference

```python
# Program 12: Circle circumference

class Circle:
    def __init__(self, radius):
        self.radius = radius

    def circumference(self):
        return 2 * 3.1416 * self.radius

c = Circle(7)
print(f"Circumference: {c.circumference():.2f}")
```

Output

```
Circumference: 43.98
```

Program 13 – Rectangle class with area

```python
# Program 13: Rectangle area

class Rectangle:
    def __init__(self, length, width):
        self.length = length
        self.width = width

    def area(self):
        return self.length * self.width

r = Rectangle(10, 5)
print(f"Rectangle Area: {r.area()}")
```

Output

```
Rectangle Area: 50
```

Program 14 – Check if rectangle is square

```
# Program 14: Rectangle check square

class Rectangle:
    def __init__(self, length, width):
        self.length = length
        self.width = width

    def is_square(self):
        return self.length == self.width

r1 = Rectangle(5, 5)
r2 = Rectangle(10, 5)
print("r1 is square:", r1.is_square())
print("r2 is square:", r2.is_square())
```

Output

```
r1 is square: True
r2 is square: False
```

Program 15 – BankAccount with deposit and withdraw

```
# Program 15: BankAccount

class BankAccount:
    def __init__(self, account_no, balance=0):
        self.account_no = account_no
        self.balance = balance

    def deposit(self, amount):
        self.balance += amount
        print(f"Deposited {amount}. New balance: {self.balance}")

    def withdraw(self, amount):
        if amount > self.balance:
            print("Insufficient balance!")
        else:
            self.balance -= amount
            print(f"Withdrawn {amount}. New balance: {self.balance}")

acc = BankAccount("12345", 1000)
acc.deposit(500)
acc.withdraw(2000)
acc.withdraw(800)
```

Output

```
Deposited 500. New balance: 1500
Insufficient balance!
Withdrawn 800. New balance: 700
```

Program 16 – Person and Employee inheritance

```
# Program 16: Inheritance

class Person:
    def __init__(self, name, age):
        self.name = name
        self.age = age
```

```
class Employee(Person):
    def __init__(self, name, age, salary):
        super().__init__(name, age)
        self.salary = salary

e = Employee("Rohit", 28, 50000)
print(e.name, e.age, e.salary)
```

Output

```
Rohit 28 50000
```

Program 17 – Library class with add/show books

```
# Program 17: Library management

class Library:
    def __init__(self):
        self.books = []

    def add_book(self, book):
        self.books.append(book)
        print(f"Added {book}")

    def show_books(self):
        print("Books in Library:", self.books)

lib = Library()
lib.add_book("Python 101")
lib.add_book("Data Science")
lib.show_books()
```

Output

```
Added Python 101
Added Data Science
Books in Library: ['Python 101', 'Data Science']
```

Program 18 – Temperature conversion

```
# Program 18: Temperature conversion

class Temperature:
    def __init__(self, celsius):
        self.celsius = celsius

    def to_fahrenheit(self):
        return (self.celsius * 9/5) + 32

t = Temperature(37)
print(f"{t.celsius}°C = {t.to_fahrenheit():.2f}°F")
```

Output

```
37°C = 98.60°F
```

Program 19 – Time class with 12-hour display

```
# Program 19: Time 12-hour format
```

```
class Time:
    def __init__(self, hours, minutes):
        self.hours = hours
        self.minutes = minutes

    def display_12hr(self):
        period = "AM"
        h = self.hours
        if h >= 12:
            period = "PM"
            if h > 12:
                h -= 12
        elif h == 0:
            h = 12
        print(f"{h:02d}:{self.minutes:02d} {period}")

t = Time(14, 30)
t.display_12hr()
```

Output

```
02:30 PM
```

Program 20 – Fraction class addition/subtraction

```
# Program 20: Fraction operations

from math import gcd

class Fraction:
    def __init__(self, num, den):
        self.num = num
        self.den = den

    def add(self, other):
        new_num = self.num*other.den + other.num*self.den
        new_den = self.den * other.den
        g = gcd(new_num, new_den)
        return Fraction(new_num//g, new_den//g)

    def subtract(self, other):
        new_num = self.num*other.den - other.num*self.den
        new_den = self.den * other.den
        g = gcd(new_num, new_den)
        return Fraction(new_num//g, new_den//g)

    def display(self):
        print(f"{self.num}/{self.den}")

f1 = Fraction(1,2)
f2 = Fraction(1,3)
f3 = f1.add(f2)
f4 = f1.subtract(f2)
f3.display()
f4.display()
```

Output

```
5/6
1/6
```

Program 21 – Point class distance from origin

```
# Program 21: Point distance

import math

class Point:
    def __init__(self, x, y):
        self.x = x
        self.y = y

    def distance_from_origin(self):
        return math.sqrt(self.x**2 + self.y**2)

p = Point(3, 4)
print(f"Distance from origin: {p.distance_from_origin()}")
```

Output

```
Distance from origin: 5.0
```

Program 22 – Person class with is_adult method

```
# Program 22: Check adult

class Person:
    def __init__(self, name, age):
        self.name = name
        self.age = age

    def is_adult(self):
        return self.age >= 18

p = Person("Anita", 17)
print(f"{p.name} is adult? {p.is_adult()}")
```

Output

```
Anita is adult? False
```

Program 23 – BankAccount with deposit/withdraw edge cases

```
# Program 23: BankAccount edge cases

acc = BankAccount("9999", 1000)
acc.withdraw(1200)  # insufficient
acc.deposit(500)
acc.withdraw(1200)
```

Output

```
Insufficient balance!
Deposited 500. New balance: 1500
Withdrawn 1200. New balance: 300
```

Program 24 – Rectangle perimeter method

```
# Program 24: Rectangle perimeter

class Rectangle:
    def __init__(self, l, w):
        self.length = l
        self.width = w

    def perimeter(self):
        return 2*(self.length + self.width)

r = Rectangle(5, 10)
print("Perimeter:", r.perimeter())
```

Output

```
Perimeter: 30
```

Program 25 – Circle class with both area and circumference

```
# Program 25: Circle area & circumference

class Circle:
    def __init__(self, radius):
        self.radius = radius

    def area(self):
        return 3.1416 * self.radius ** 2

    def circumference(self):
        return 2 * 3.1416 * self.radius

c = Circle(3)
print("Area:", c.area())
print("Circumference:", c.circumference())
```

Output

```
Area: 28.2744
Circumference: 18.8496
```

Program 26 – Shape class with Square and Circle subclasses

```
# Program 26: Shape inheritance

import math

class Shape:
    def area(self):
        pass

class Square(Shape):
    def __init__(self, side):
        self.side = side

    def area(self):
        return self.side ** 2

class Circle(Shape):
    def __init__(self, radius):
        self.radius = radius

    def area(self):
        return math.pi * self.radius ** 2

s = Square(5)
c = Circle(3)
print("Square area:", s.area())
print("Circle area:", round(c.area(), 2))
```

Output

```
Square area: 25
Circle area: 28.27
```

Program 27 – Override area method in subclasses

(Already done in Program 26 using `area()` override)

Program 28 – Single inheritance

```
# Program 28: Single inheritance

class Person:
    def __init__(self, name):
        self.name = name

    def show_name(self):
        print("Name:", self.name)

class Employee(Person):
    def __init__(self, name, salary):
        super().__init__(name)
        self.salary = salary

    def show_salary(self):
        print("Salary:", self.salary)

e = Employee("Ravi", 50000)
e.show_name()
e.show_salary()
```

Output

```
Name: Ravi
Salary: 50000
```

Program 29 – Multiple inheritance

```
# Program 29: Multiple inheritance

class Person:
    def __init__(self, name):
        self.name = name

class Employee:
    def __init__(self, salary):
        self.salary = salary

class Manager(Person, Employee):
    def __init__(self, name, salary, department):
        Person.__init__(self, name)
        Employee.__init__(self, salary)
        self.department = department

m = Manager("Anita", 80000, "IT")
print(m.name, m.salary, m.department)
```

Output

```
Anita 80000 IT
```

Program 30 – Multilevel inheritance

```
# Program 30: Multilevel inheritance

class Grandparent:
    def __init__(self, gp_name):
        self.gp_name = gp_name

class Parent(Grandparent):
    def __init__(self, gp_name, parent_name):
        super().__init__(gp_name)
        self.parent_name = parent_name

class Child(Parent):
    def __init__(self, gp_name, parent_name, child_name):
        super().__init__(gp_name, parent_name)
        self.child_name = child_name

c = Child("John Sr.", "John Jr.", "Johnny")
print(c.gp_name, c.parent_name, c.child_name)
```

Output

```
John Sr. John Jr. Johnny
```

Program 31 – Hierarchical inheritance

```
# Program 31: Hierarchical inheritance

class Parent:
    def greet(self):
        print("Hello from Parent")

class Child1(Parent):
    def greet1(self):
        print("Hello from Child1")

class Child2(Parent):
    def greet2(self):
        print("Hello from Child2")

c1 = Child1()
c2 = Child2()
c1.greet()
c2.greet()
```

Output

```
Hello from Parent
Hello from Parent
```

Program 32 – Method overriding

```
# Program 32: Method overriding

class Animal:
    def sound(self):
        print("Some sound")

class Dog(Animal):
    def sound(self):
        print("Bark Bark")

d = Dog()
d.sound()
```

Output

```
Bark Bark
```

Program 33 – Method overloading (via default args)

```
# Program 33: Method overloading simulation

class Calculator:
    def add(self, a, b=0, c=0):
        return a + b + c

calc = Calculator()
print(calc.add(5))
print(calc.add(5, 10))
print(calc.add(5, 10, 15))
```

Output

```
5
15
30
```

Program 34 – Private attribute with getter and setter

```
# Program 34: Encapsulation

class Person:
    def __init__(self, name):
        self.__name = name

    def get_name(self):
        return self.__name

    def set_name(self, name):
        self.__name = name

p = Person("Rohit")
print(p.get_name())
p.set_name("Anita")
print(p.get_name())
```

Output

```
Rohit
Anita
```

Program 35 – Property decorator

```
# Program 35: @property
```

```
class Person:
    def __init__(self, age):
        self._age = age

    @property
    def age(self):
        return self._age

    @age.setter
    def age(self, value):
        if value < 0:
            print("Invalid age")
        else:
            self._age = value

p = Person(25)
print(p.age)
p.age = -5  # Invalid
p.age = 30
print(p.age)
```

Output

```
25
Invalid age
30
```

Program 36 – Static method

```
# Program 36: @staticmethod

class MathUtils:
    @staticmethod
    def add(a, b):
        return a + b

print(MathUtils.add(5, 10))
```

Output

```
15
```

Program 37 – Class method

```
# Program 37: @classmethod

class Employee:
    company = "ABC Corp"

    @classmethod
    def show_company(cls):
        print("Company:", cls.company)

Employee.show_company()
```

Output

```
Company: ABC Corp
```

Program 38 – Counter class tracking objects

```
# Program 38: Counter class

class Counter:
    count = 0
    def __init__(self):
        Counter.count += 1

a = Counter()
b = Counter()
c = Counter()
print("Objects created:", Counter.count)
```

Output

```
Objects created: 3
```

Program 39 – str method

```
# Program 39: __str__

class Person:
    def __init__(self, name, age):
        self.name = name
        self.age = age

    def __str__(self):
        return f"Person(Name: {self.name}, Age: {self.age})"

p = Person("Ravi", 28)
print(p)
```

Output

```
Person(Name: Ravi, Age: 28)
```

Program 40 – repr method

```
# Program 40: __repr__

class Person:
    def __init__(self, name, age):
        self.name = name
        self.age = age

    def __repr__(self):
        return f"Person('{self.name}', {self.age})"

p = Person("Anita", 25)
print(repr(p))
```

Output

```
Person('Anita', 25)
```

Program 41 – eq method to compare employee IDs

```
# Program 41: __eq__ for Employee

class Employee:
    def __init__(self, emp_id, name):
        self.emp_id = emp_id
        self.name = name

    def __eq__(self, other):
        return self.emp_id == other.emp_id

e1 = Employee(101, "Ravi")
e2 = Employee(102, "Anita")
e3 = Employee(101, "Vikas")
print(e1 == e2)
print(e1 == e3)
```

Output

```
False
True
```

Program 42 – lt and gt for salary comparison

```
# Program 42: Compare salaries

class Employee:
    def __init__(self, name, salary):
        self.name = name
        self.salary = salary

    def __lt__(self, other):
        return self.salary < other.salary

    def __gt__(self, other):
        return self.salary > other.salary

e1 = Employee("Ravi", 50000)
e2 = Employee("Anita", 60000)
print(e1 < e2)
print(e1 > e2)
```

Output

```
True
False
```

Program 43 – Encapsulation salary validation

```
# Program 43: Private salary with validation

class Employee:
    def __init__(self, salary):
        self.__salary = salary

    def get_salary(self):
        return self.__salary

    def set_salary(self, value):
        if value < 0:
            print("Invalid salary")
        else:
            self.__salary = value

e = Employee(50000)
```

```
print(e.get_salary())
e.set_salary(-1000)
e.set_salary(60000)
print(e.get_salary())
```

Output

```
50000
Invalid salary
60000
```

Program 44 – Book class with del

```
# Program 44: __del__

class Book:
    def __init__(self, title):
        self.title = title
        print(f"Book {self.title} created")

    def __del__(self):
        print(f"Book {self.title} deleted")

b = Book("Python 101")
del b
```

Output

```
Book Python 101 created
Book Python 101 deleted
```

Program 45 – Student class with class variable and class method

```
# Program 45: Class variable & class method

class Student:
    college = "XYZ College"

    @classmethod
    def set_college(cls, name):
        cls.college = name

    @classmethod
    def get_college(cls):
        return cls.college

print(Student.get_college())
Student.set_college("ABC University")
print(Student.get_college())
```

Output

```
XYZ College
ABC University
```

Program 46 – Abstract base class Vehicle

```
# Program 46: ABC

from abc import ABC, abstractmethod
```

```
class Vehicle(ABC):
    @abstractmethod
    def move(self):
        pass

class Car(Vehicle):
    def move(self):
        print("Car moves on road")

class Bike(Vehicle):
    def move(self):
        print("Bike moves on road")

c = Car()
b = Bike()
c.move()
b.move()
```

Output

```
Car moves on road
Bike moves on road
```

Program 47 – Polymorphism function with multiple objects

```
# Program 47: Polymorphism

class Cat:
    def speak(self):
        print("Meow")

class Dog:
    def speak(self):
        print("Bark")

def make_sound(animal):
    animal.speak()

make_sound(Cat())
make_sound(Dog())
```

Output

```
Meow
Bark
```

Program 48 – Operator overloading + for Vector

```
# Program 48: Vector addition

class Vector:
    def __init__(self, x, y):
        self.x = x
        self.y = y

    def __add__(self, other):
        return Vector(self.x + other.x, self.y + other.y)

    def show(self):
        print(f"({self.x}, {self.y})")
```

```
v1 = Vector(2, 3)
v2 = Vector(4, 5)
v3 = v1 + v2
v3.show()
```

Output

```
(6, 8)
```

Program 49 – Operator overloading * for Point

```
# Program 49: Point multiplication

class Point:
    def __init__(self, x, y):
        self.x = x
        self.y = y

    def __mul__(self, scalar):
        return Point(self.x * scalar, self.y * scalar)

    def show(self):
        print(f"({self.x}, {self.y})")

p = Point(3, 4)
p2 = p * 3
p2.show()
```

Output

```
(9, 12)
```

Program 50 – len method for Book

```
# Program 50: __len__ for Book

class Book:
    def __init__(self, title):
        self.title = title

    def __len__(self):
        return len(self.title)

b = Book("Python Programming")
print(len(b))
```

Output

```
18
```

Program 51 – Bank class handling multiple accounts

```
# Program 51: Bank with multiple accounts

class Account:
    def __init__(self, acc_no, name, balance=0):
        self.acc_no = acc_no
        self.name = name
        self.balance = balance

    def deposit(self, amount):
        self.balance += amount
        print(f"{self.name} deposited {amount}. Balance: {self.balance}")

    def withdraw(self, amount):
        if amount > self.balance:
            print(f"{self.name} insufficient balance!")
        else:
            self.balance -= amount
            print(f"{self.name} withdrew {amount}. Balance: {self.balance}")

class Bank:
    def __init__(self):
        self.accounts = []

    def add_account(self, account):
        self.accounts.append(account)

    def show_accounts(self):
        for acc in self.accounts:
            print(f"{acc.acc_no} - {acc.name} - {acc.balance}")

# Usage
b = Bank()
a1 = Account(101, "Ravi", 1000)
a2 = Account(102, "Anita", 2000)
b.add_account(a1)
b.add_account(a2)
b.show_accounts()
a1.deposit(500)
a2.withdraw(2500)
```

Output

```
101 - Ravi - 1000
102 - Anita - 2000
Ravi deposited 500. Balance: 1500
Anita insufficient balance!
```

Program 52 – Library Management System

```
# Program 52: Library Management

class Book:
    def __init__(self, title):
        self.title = title
        self.available = True

class Member:
    def __init__(self, name):
        self.name = name
        self.books_borrowed = []

class Library:
    def __init__(self):
        self.books = []

    def add_book(self, book):
        self.books.append(book)

    def borrow_book(self, member, title):
        for book in self.books:
            if book.title == title and book.available:
                book.available = False
                member.books_borrowed.append(book)
                print(f"{member.name} borrowed {title}")
                return
        print(f"{title} not available")

    def show_books(self):
        for book in self.books:
            status = "Available" if book.available else "Borrowed"
            print(f"{book.title}: {status}")

lib = Library()
b1 = Book("Python 101")
b2 = Book("Data Science")
lib.add_book(b1)
lib.add_book(b2)
m = Member("Ravi")
lib.borrow_book(m, "Python 101")
lib.show_books()
```

Output

```
Ravi borrowed Python 101
Python 101: Borrowed
Data Science: Available
```

Program 53 – Student Management System

```
# Program 53: Student Management

class Student:
    def __init__(self, name, roll_no, marks):
        self.name = name
        self.roll_no = roll_no
        self.marks = marks

    def percentage(self):
        return sum(self.marks)/len(self.marks)

class Course:
    def __init__(self, course_name):
        self.course_name = course_name
        self.students = []

    def add_student(self, student):
        self.students.append(student)

    def show_results(self):
        for s in self.students:
            print(f"{s.name} ({s.roll_no}) - {s.percentage():.2f}%")

c = Course("Python")
s1 = Student("Ravi", 101, [80, 90, 70])
s2 = Student("Anita", 102, [85, 75, 95])
c.add_student(s1)
c.add_student(s2)
c.show_results()
```

Output

```
Ravi (101) - 80.00%
Anita (102) - 85.00%
```

Program 54 – Car Rental System

```
# Program 54: Car Rental

class Car:
    def __init__(self, model):
        self.model = model
        self.available = True

class Customer:
    def __init__(self, name):
        self.name = name

class Rental:
    def __init__(self):
        self.cars = []

    def add_car(self, car):
        self.cars.append(car)

    def rent_car(self, customer, model):
        for car in self.cars:
            if car.model == model and car.available:
                car.available = False
                print(f"{customer.name} rented {model}")
                return
        print(f"{model} not available")

rental = Rental()
c1 = Car("Honda Civic")
c2 = Car("Toyota Corolla")
rental.add_car(c1)
rental.add_car(c2)
cust = Customer("Ravi")
rental.rent_car(cust, "Honda Civic")
rental.rent_car(cust, "Honda Civic")
```

Output

```
Ravi rented Honda Civic
Honda Civic not available
```

Program 55 – Shopping Cart

```
# Program 55: Shopping Cart

class Product:
    def __init__(self, name, price):
        self.name = name
        self.price = price

class Cart:
    def __init__(self):
        self.items = []

    def add_product(self, product, qty=1):
        self.items.append((product, qty))

    def total(self):
        return sum(p.price * q for p, q in self.items)
```

```
p1 = Product("Laptop", 50000)
p2 = Product("Mouse", 500)
cart = Cart()
cart.add_product(p1)
cart.add_product(p2, 2)
print("Total:", cart.total())
```

Output

```
Total: 51000
```

Program 56 – Zoo Management

```
# Program 56: Zoo Management

class Animal:
    def __init__(self, name, species):
        self.name = name
        self.species = species

class Cage:
    def __init__(self):
        self.animals = []

    def add_animal(self, animal):
        self.animals.append(animal)

cage = Cage()
a1 = Animal("Leo", "Lion")
a2 = Animal("Ella", "Elephant")
cage.add_animal(a1)
cage.add_animal(a2)
for a in cage.animals:
    print(f"{a.name} - {a.species}")
```

Output

```
Leo - Lion
Ella - Elephant
```

Program 57 – Hospital Management System

```
# Program 57: Hospital Management

class Patient:
    def __init__(self, name):
        self.name = name

class Doctor:
    def __init__(self, name, specialty):
        self.name = name
        self.specialty = specialty

class Appointment:
    def __init__(self, patient, doctor):
        self.patient = patient
        self.doctor = doctor

p = Patient("Ravi")
d = Doctor("Dr. Anita", "Cardiology")
app = Appointment(p, d)
```

```
print(f"Appointment: {app.patient.name} with {app.doctor.name} ({app.doctor.specialty
```

Output

```
Appointment: Ravi with Dr. Anita (Cardiology)
```

Program 58 – Flight Reservation System

```
# Program 58: Flight Reservation

class Flight:
    def __init__(self, flight_no):
        self.flight_no = flight_no
        self.seats = 10

class Passenger:
    def __init__(self, name):
        self.name = name

class Reservation:
    def book_seat(self, flight, passenger):
        if flight.seats > 0:
            flight.seats -= 1
            print(f"{passenger.name} booked on flight {flight.flight_no}")
        else:
            print(f"No seats available on flight {flight.flight_no}")

f = Flight("AI101")
p = Passenger("Ravi")
r = Reservation()
r.book_seat(f, p)
```

Output

```
Ravi booked on flight AI101
```

Program 59 – Cinema Ticket Booking

```
# Program 59: Cinema Ticket Booking

class Movie:
    def __init__(self, title):
        self.title = title

class Theater:
    def __init__(self, name, seats=100):
        self.name = name
        self.seats = seats

    def book_ticket(self, qty):
        if qty <= self.seats:
            self.seats -= qty
            print(f"{qty} tickets booked. Remaining: {self.seats}")
        else:
            print("Not enough seats available")

theater = Theater("PVR", 50)
theater.book_ticket(20)
theater.book_ticket(40)
```

Output

```
20 tickets booked. Remaining: 30
Not enough seats available
```

Program 60 – Payroll System

```
# Program 60: Payroll System

class Employee:
    def __init__(self, name, salary):
        self.name = name
        self.salary = salary

    def annual_salary(self):
        return self.salary * 12

e = Employee("Anita", 5000)
print(f"{e.name} annual salary: {e.annual_salary()}")
```

Output

```
Anita annual salary: 60000
```

Program 61 – School Report Card

```
# Program 61: School Report Card

class Student:
    def __init__(self, name, marks):
        self.name = name
        self.marks = marks

    def grade(self):
        avg = sum(self.marks)/len(self.marks)
        if avg >= 90:
            return "A"
        elif avg >= 75:
            return "B"
        elif avg >= 50:
            return "C"
        else:
            return "D"

s = Student("Ravi", [80, 90, 85])
print(f"{s.name} grade: {s.grade()}")
```

Output

```
Ravi grade: B
```

Program 62 – Quiz System

```
# Program 62: Quiz System

class Question:
    def __init__(self, q, ans):
        self.q = q
        self.ans = ans

    def check(self, answer):
        return self.ans.lower() == answer.lower()

q1 = Question("Capital of India?", "New Delhi")
```

```
print(q1.check("new delhi"))
print(q1.check("Mumbai"))
```

Output

```
True
False
```

Program 63 – E-commerce Product Catalog

```
# Program 63: E-commerce Catalog

class Product:
    def __init__(self, name, price):
        self.name = name
        self.price = price

class Catalog:
    def __init__(self):
        self.products = []

    def add_product(self, product):
        self.products.append(product)

    def show_products(self):
        for p in self.products:
            print(f"{p.name}: {p.price}")

c = Catalog()
c.add_product(Product("Laptop", 50000))
c.add_product(Product("Mouse", 500))
c.show_products()
```

Output

```
Laptop: 50000
Mouse: 500
```

Program 64 – ATM Simulation

```
# Program 64: ATM Simulation

class ATM:
    def __init__(self, balance):
        self.balance = balance

    def withdraw(self, amount):
        if amount > self.balance:
            print("Insufficient balance")
        else:
            self.balance -= amount
            print(f"Withdrawn: {amount}. Balance: {self.balance}")

atm = ATM(5000)
atm.withdraw(2000)
atm.withdraw(4000)
```

Output

```
Withdrawn: 2000. Balance: 3000
Insufficient balance
```

Program 65 – Smart Home Devices

```
# Program 65: Smart Home

class Device:
    def __init__(self, name):
        self.name = name
        self.status = "OFF"

    def switch_on(self):
        self.status = "ON"

    def switch_off(self):
        self.status = "OFF"

    def show_status(self):
        print(f"{self.name} is {self.status}")

light = Device("Living Room Light")
fan = Device("Ceiling Fan")
light.switch_on()
fan.switch_off()
light.show_status()
fan.show_status()
```

Output

```
Living Room Light is ON
Ceiling Fan is OFF
```

Program 66 – Hotel Room Booking

```
# Program 66: Hotel Booking

class Room:
    def __init__(self, number):
        self.number = number
        self.occupied = False

class Hotel:
    def __init__(self, name):
        self.name = name
        self.rooms = []

    def add_room(self, room):
        self.rooms.append(room)

    def book_room(self):
        for room in self.rooms:
            if not room.occupied:
                room.occupied = True
                print(f"Room {room.number} booked")
                return
        print("No rooms available")

hotel = Hotel("Sunrise Hotel")
hotel.add_room(Room(101))
hotel.add_room(Room(102))
hotel.book_room()
hotel.book_room()
hotel.book_room()
```

Output

```
Room 101 booked
Room 102 booked
No rooms available
```

Program 67 – Online Course Enrollment

```python
# Program 67: Course Enrollment

class Course:
    def __init__(self, name):
        self.name = name
        self.students = []

    def enroll(self, student):
        self.students.append(student)
        print(f"{student} enrolled in {self.name}")

python_course = Course("Python")
python_course.enroll("Ravi")
python_course.enroll("Anita")
```

Output

```
Ravi enrolled in Python
Anita enrolled in Python
```

Program 68 – Vehicle Parking System

```python
# Program 68: Parking System

class Vehicle:
    def __init__(self, number):
        self.number = number

class ParkingLot:
    def __init__(self, capacity):
        self.capacity = capacity
        self.vehicles = []

    def park(self, vehicle):
        if len(self.vehicles) < self.capacity:
            self.vehicles.append(vehicle)
            print(f"Vehicle {vehicle.number} parked")
        else:
            print("Parking Full")

p = ParkingLot(2)
v1 = Vehicle("KA01AB1234")
v2 = Vehicle("KA01AB5678")
v3 = Vehicle("KA01AB9999")
p.park(v1)
p.park(v2)
p.park(v3)
```

Output

```
Vehicle KA01AB1234 parked
Vehicle KA01AB5678 parked
```

```
Parking Full
```

Program 69 – Gym Membership Management

```python
# Program 69: Gym Membership

class Member:
    def __init__(self, name):
        self.name = name
        self.active = True

    def cancel_membership(self):
        self.active = False
        print(f"{self.name} membership canceled")

m = Member("Ravi")
m.cancel_membership()
```

Output

```
Ravi membership canceled
```

Program 70 – Ticketing System

```python
# Program 70: Ticketing System

class Ticket:
    def __init__(self, ticket_id, status="Open"):
        self.ticket_id = ticket_id
        self.status = status

    def close_ticket(self):
        self.status = "Closed"

t1 = Ticket(101)
print(t1.ticket_id, t1.status)
t1.close_ticket()
print(t1.ticket_id, t1.status)
```

Output

```
101 Open
101 Closed
```

Program 71 – Car Service Management

```python
# Program 71: Car Service

class Car:
    def __init__(self, model):
        self.model = model

class ServiceCenter:
    def __init__(self):
        self.cars_serviced = []

    def service_car(self, car):
        self.cars_serviced.append(car)
        print(f"{car.model} serviced")

c1 = Car("Honda")
c2 = Car("Toyota")
```

```
sc = ServiceCenter()
sc.service_car(c1)
sc.service_car(c2)
```

Output

```
Honda serviced
Toyota serviced
```

Program 72 – Restaurant Table Booking

```
# Program 72: Restaurant Booking

class Table:
    def __init__(self, number):
        self.number = number
        self.booked = False

class Restaurant:
    def __init__(self, name):
        self.name = name
        self.tables = []

    def add_table(self, table):
        self.tables.append(table)

    def book_table(self):
        for table in self.tables:
            if not table.booked:
                table.booked = True
                print(f"Table {table.number} booked")
                return
        print("No tables available")

rest = Restaurant("Spice Hut")
rest.add_table(Table(1))
rest.add_table(Table(2))
rest.book_table()
rest.book_table()
rest.book_table()
```

Output

```
Table 1 booked
Table 2 booked
No tables available
```

Program 73 – Online Quiz System with score

```
# Program 73: Quiz System with Score

class Question:
    def __init__(self, q, ans):
        self.q = q
        self.ans = ans

class Quiz:
    def __init__(self):
        self.questions = []

    def add_question(self, question):
        self.questions.append(question)

    def take_quiz(self, answers):
        score = 0
        for q, a in zip(self.questions, answers):
            if q.ans.lower() == a.lower():
                score += 1
        print(f"Score: {score}/{len(self.questions)}")

q1 = Question("2+2?", "4")
q2 = Question("Capital of India?", "New Delhi")
quiz = Quiz()
quiz.add_question(q1)
quiz.add_question(q2)
quiz.take_quiz(["4", "Mumbai"])
```

Output

```
Score: 1/2
```

Program 74 – E-Library with digital borrowing

```
# Program 74: E-Library Digital Borrowing

class DigitalBook:
    def __init__(self, title):
        self.title = title
        self.borrowed = False

class ELibrary:
    def __init__(self):
        self.books = []

    def add_book(self, book):
        self.books.append(book)

    def borrow_book(self, title):
        for book in self.books:
            if book.title == title and not book.borrowed:
                book.borrowed = True
                print(f"{title} borrowed")
                return
        print(f"{title} not available")

lib = ELibrary()
lib.add_book(DigitalBook("Python 101"))
lib.add_book(DigitalBook("AI Basics"))
lib.borrow_book("Python 101")
lib.borrow_book("Python 101")
```

Output

```
Python 101 borrowed
Python 101 not available
```

Program 75 – Online Food Ordering System

```
# Program 75: Food Ordering System

class Item:
    def __init__(self, name, price):
        self.name = name
        self.price = price

class Order:
    def __init__(self):
        self.items = []

    def add_item(self, item, qty=1):
        self.items.append((item, qty))

    def total(self):
        return sum(item.price * qty for item, qty in self.items)

pizza = Item("Pizza", 250)
burger = Item("Burger", 100)
order = Order()
order.add_item(pizza, 2)
order.add_item(burger)
print("Total Bill:", order.total())
```

Output

```
Total Bill: 600
```

Program 76 – Employee Hierarchy with Bonus Calculation

```
# Program 76: Employee Hierarchy

class Employee:
    def __init__(self, name, salary):
        self.name = name
        self.salary = salary

    def bonus(self):
        return self.salary * 0.1

class Manager(Employee):
    def bonus(self):
        return self.salary * 0.2  # Managers get 20% bonus

class Developer(Employee):
    def bonus(self):
        return self.salary * 0.15

employees = [Manager("Ravi", 50000), Developer("Anita", 40000), Employee("Vikas", 300
for emp in employees:
    print(f"{emp.name} bonus: {emp.bonus()}")
```

Output

```
Ravi bonus: 10000.0
Anita bonus: 6000.0
Vikas bonus: 3000.0
```

Program 77 – Inventory Management System

```
# Program 77: Inventory Management

class Product:
    def __init__(self, name, qty):
        self.name = name
        self.qty = qty
```

```
class Inventory:
    def __init__(self):
        self.products = []

    def add_product(self, product):
        self.products.append(product)

    def sell_product(self, name, qty):
        for p in self.products:
            if p.name == name:
                if p.qty >= qty:
                    p.qty -= qty
                    print(f"{qty} {name} sold")
                else:
                    print(f"Not enough {name} in stock")
                return
        print(f"{name} not found")

inv = Inventory()
inv.add_product(Product("Laptop", 10))
inv.add_product(Product("Mouse", 50))
inv.sell_product("Laptop", 5)
inv.sell_product("Mouse", 60)
```

Output

```
5 Laptop sold
Not enough Mouse in stock
```

Program 78 – Hotel Room Booking with Multiple Customers

```
# Program 78: Hotel Advanced Booking

class Room:
    def __init__(self, number):
        self.number = number
        self.occupied = False

class Customer:
    def __init__(self, name):
        self.name = name

class Hotel:
    def __init__(self, name):
        self.name = name
        self.rooms = []

    def add_room(self, room):
        self.rooms.append(room)

    def book_room(self, customer):
        for room in self.rooms:
            if not room.occupied:
                room.occupied = True
                print(f"{customer.name} booked room {room.number}")
                return
        print(f"No rooms available for {customer.name}")

hotel = Hotel("Sunrise")
hotel.add_room(Room(101))
hotel.add_room(Room(102))
hotel.book_room(Customer("Ravi"))
hotel.book_room(Customer("Anita"))
hotel.book_room(Customer("Vikas"))
```

Output

```
Ravi booked room 101
Anita booked room 102
No rooms available for Vikas
```

Program 79 – Library with Return & Fine Calculation

```
# Program 79: Library with fines

class Book:
    def __init__(self, title):
        self.title = title
        self.borrowed = False

class Member:
    def __init__(self, name):
        self.name = name
        self.borrowed_books = []

class Library:
    def __init__(self):
        self.books = []

    def add_book(self, book):
        self.books.append(book)

    def borrow(self, member, title):
        for book in self.books:
            if book.title == title and not book.borrowed:
                book.borrowed = True
                member.borrowed_books.append(book)
                print(f"{member.name} borrowed {title}")
                return
        print(f"{title} not available")

    def return_book(self, member, title, days):
        for book in member.borrowed_books:
            if book.title == title:
                book.borrowed = False
                member.borrowed_books.remove(book)
                fine = 0
                if days > 7:  # 7 days free
                    fine = (days-7)*10
                print(f"{member.name} returned {title}. Fine: {fine}")
                return
        print(f"{member.name} did not borrow {title}")

lib = Library()
b1 = Book("Python")
b2 = Book("AI")
lib.add_book(b1)
lib.add_book(b2)
m = Member("Ravi")
lib.borrow(m, "Python")
lib.return_book(m, "Python", 10)
```

Output

```
Ravi borrowed Python
Ravi returned Python. Fine: 30
```

Program 80 – Online Shopping Cart with Discount

```
# Program 80: Shopping Cart with Discount

class Product:
    def __init__(self, name, price):
        self.name = name
        self.price = price

class Cart:
    def __init__(self):
        self.items = []

    def add_item(self, product, qty=1):
        self.items.append((product, qty))

    def total(self):
        total = sum(p.price*q for p, q in self.items)
        if total > 5000:  # 10% discount
            total *= 0.9
        return total

cart = Cart()
cart.add_item(Product("Laptop", 4000))
cart.add_item(Product("Mouse", 1000))
print("Total after discount:", cart.total())
```

Output

```
Total after discount: 4500.0
```

Program 81 – ATM with PIN Validation

```
# Program 81: ATM with PIN

class ATM:
    def __init__(self, balance, pin):
        self.balance = balance
        self.pin = pin

    def validate_pin(self, input_pin):
        return self.pin == input_pin

    def withdraw(self, amount, input_pin):
        if not self.validate_pin(input_pin):
            print("Invalid PIN")
            return
        if amount > self.balance:
            print("Insufficient Balance")
        else:
            self.balance -= amount
            print(f"Withdrawn: {amount}. Balance: {self.balance}")

atm = ATM(5000, 1234)
atm.withdraw(1000, 1111)
atm.withdraw(2000, 1234)
```

Output

```
Invalid PIN
Withdrawn: 2000. Balance: 3000
```

Program 82 – Flight Reservation with Multiple Seats

```
# Program 82: Flight Reservation Advanced

class Flight:
    def __init__(self, flight_no, capacity):
        self.flight_no = flight_no
        self.capacity = capacity
        self.passengers = []

    def book_seat(self, passenger):
        if len(self.passengers) < self.capacity:
            self.passengers.append(passenger)
            print(f"{passenger} booked on {self.flight_no}")
        else:
            print("No seats available")

f = Flight("AI101", 2)
f.book_seat("Ravi")
f.book_seat("Anita")
f.book_seat("Vikas")
```

Output

```
Ravi booked on AI101
Anita booked on AI101
No seats available
```

Program 83 – Cinema Ticket Booking with Seat Selection

```
# Program 83: Cinema Ticket with seats

class Cinema:
    def __init__(self, rows, cols):
        self.rows = rows
        self.cols = cols
        self.seats = [[False]*cols for _ in range(rows)]

    def book_seat(self, row, col):
        if self.seats[row][col]:
            print("Seat already booked")
        else:
            self.seats[row][col] = True
            print(f"Seat ({row},{col}) booked")

cinema = Cinema(5,5)
cinema.book_seat(2,3)
cinema.book_seat(2,3)
```

Output

```
Seat (2,3) booked
Seat already booked
```

Program 84 – Gym Membership with Subscription & Classes

```
# Program 84: Gym Membership Advanced

class Member:
    def __init__(self, name):
        self.name = name
        self.subscribed = False
        self.classes = []

    def subscribe(self):
        self.subscribed = True
        print(f"{self.name} subscribed to gym")

    def enroll_class(self, class_name):
        if self.subscribed:
            self.classes.append(class_name)
            print(f"{self.name} enrolled in {class_name}")
        else:
            print("Subscription required")

m = Member("Ravi")
m.enroll_class("Yoga")
m.subscribe()
m.enroll_class("Yoga")
```

Output

```
Subscription required
Ravi subscribed to gym
Ravi enrolled in Yoga
```

Program 85 – Payroll with Overtime & Bonus

```
# Program 85: Payroll Advanced

class Employee:
    def __init__(self, name, salary):
        self.name = name
        self.salary = salary

    def monthly_pay(self, overtime_hours=0):
        overtime_pay = overtime_hours * 200
        return self.salary + overtime_pay + self.bonus()

    def bonus(self):
        return self.salary * 0.1

e = Employee("Anita", 5000)
print(f"{e.name} monthly pay with 10 hrs overtime: {e.monthly_pay(10)}")
```

Output

```
Anita monthly pay with 10 hrs overtime: 7200.0
```

Program 86 – Library with Reservation & Waiting List

```
# Program 86: Library Waiting List

class Book:
    def __init__(self, title):
        self.title = title
        self.available = True
        self.waiting_list = []

class Member:
    def __init__(self, name):
        self.name = name

class Library:
    def borrow(self, book, member):
        if book.available:
            book.available = False
            print(f"{member.name} borrowed {book.title}")
        else:
            book.waiting_list.append(member.name)
            print(f"{member.name} added to waiting list for {book.title}")

b = Book("Python 101")
m1 = Member("Ravi")
m2 = Member("Anita")
lib = Library()
lib.borrow(b, m1)
lib.borrow(b, m2)
print("Waiting list:", b.waiting_list)
```

Output

```
Ravi borrowed Python 101
Anita added to waiting list for Python 101
Waiting list: ['Anita']
```

Program 87 – Multi-level Flight Booking with Seat Class

```
# Program 87: Flight Booking with classes

class Flight:
    def __init__(self, flight_no):
        self.flight_no = flight_no
        self.seats = {"Economy": 2, "Business": 1}

    def book_seat(self, passenger, seat_class):
        if self.seats.get(seat_class,0) > 0:
            self.seats[seat_class] -= 1
            print(f"{passenger} booked {seat_class} on {self.flight_no}")
        else:
            print(f"No {seat_class} seats available")

f = Flight("AI101")
f.book_seat("Ravi", "Economy")
f.book_seat("Anita", "Business")
f.book_seat("Vikas", "Business")
```

Output

```
Ravi booked Economy on AI101
Anita booked Business on AI101
No Business seats available
```

Program 88 – E-commerce Order with Multiple Items

```
# Program 88: E-commerce Orders
```

```
class Product:
    def __init__(self, name, price):
        self.name = name
        self.price = price

class Order:
    def __init__(self):
        self.items = []

    def add_item(self, product, qty):
        self.items.append((product, qty))

    def total_bill(self):
        return sum(p.price*q for p,q in self.items)

o = Order()
o.add_item(Product("Laptop", 40000), 1)
o.add_item(Product("Mouse", 500), 2)
print("Total Bill:", o.total_bill())
```

Output

```
Total Bill: 41000
```

Program 89 – Hotel with Multi-Room Booking

```
# Program 89: Hotel Multi-Room

class Room:
    def __init__(self, number):
        self.number = number
        self.occupied = False

class Hotel:
    def __init__(self):
        self.rooms = []

    def add_room(self, room):
        self.rooms.append(room)

    def book_rooms(self, count):
        booked = 0
        for room in self.rooms:
            if not room.occupied and booked < count:
                room.occupied = True
                booked += 1
                print(f"Room {room.number} booked")
        if booked < count:
            print(f"{count - booked} rooms not available")

hotel = Hotel()
for i in range(101,106):
    hotel.add_room(Room(i))
hotel.book_rooms(3)
hotel.book_rooms(3)
```

Output

```
Room 101 booked
Room 102 booked
Room 103 booked
```

```
3 rooms not available
```

Program 90 – Smart Home with Multiple Devices & Scenes

```
# Program 90: Smart Home Automation

class Device:
    def __init__(self, name):
        self.name = name
        self.status = "OFF"

    def switch_on(self):
        self.status = "ON"

    def switch_off(self):
        self.status = "OFF"

class SmartHome:
    def __init__(self):
        self.devices = []

    def add_device(self, device):
        self.devices.append(device)

    def activate_scene(self, scene):
        if scene == "Movie":
            for d in self.devices:
                if d.name in ["TV","Lights"]:
                    d.switch_on()
        elif scene == "Sleep":
            for d in self.devices:
                d.switch_off()
        for d in self.devices:
            print(f"{d.name}: {d.status}")

home = SmartHome()
home.add_device(Device("TV"))
home.add_device(Device("Lights"))
home.add_device(Device("AC"))
home.activate_scene("Movie")
```

Output

```
TV: ON
Lights: ON
AC: OFF
```

Program 91 – Quiz System with Multiple Rounds

```
# Program 91: Quiz Multi-Round

class Question:
    def __init__(self, q, ans):
        self.q = q
        self.ans = ans

class Quiz:
    def __init__(self):
        self.rounds = {}

    def add_question(self, round_no, question):
        self.rounds.setdefault(round_no, []).append(question)
    def take_quiz(self, answers):
        score = 0
        for round_no, qs in self.rounds.items():
            for q,a in zip(qs, answers.get(round_no,[])):
                if q.ans.lower() == a.lower():
                    score += 1
        print(f"Total Score: {score}")

q1 = Question("2+2?", "4")
q2 = Question("Capital?", "New Delhi")
quiz = Quiz()
quiz.add_question(1, q1)
quiz.add_question(2, q2)
quiz.take_quiz({1:["4"], 2:["Mumbai"]})
```

Output

```
Total Score: 1
```

Program 92 – School with Multiple Classes and Students

```
# Program 92: School Management

class Student:
    def __init__(self, name):
        self.name = name

class SchoolClass:
    def __init__(self, name):
        self.name = name
        self.students = []

    def add_student(self, student):
        self.students.append(student)

class School:
    def __init__(self):
        self.classes = []

    def add_class(self, school_class):
        self.classes.append(school_class)

school = School()
class1 = SchoolClass("10A")
class1.add_student(Student("Ravi"))
class1.add_student(Student("Anita"))
school.add_class(class1)
for c in school.classes:
    for s in c.students:
        print(f"{s.name} in {c.name}")
```

Output

```
Ravi in 10A
Anita in 10A
```

Program 93 – Banking System with Multiple Accounts & Transfers

```
# Program 93: Bank Transfer
```

```
class Account:
    def __init__(self, name, balance):
        self.name = name
        self.balance = balance

    def deposit(self, amt):
        self.balance += amt

    def withdraw(self, amt):
        if amt > self.balance:
            print("Insufficient Balance")
        else:
            self.balance -= amt

    def transfer(self, other, amt):
        if amt > self.balance:
            print("Cannot transfer, insufficient balance")
        else:
            self.withdraw(amt)
            other.deposit(amt)
            print(f"{amt} transferred from {self.name} to {other.name}")

a1 = Account("Ravi", 5000)
a2 = Account("Anita", 3000)
a1.transfer(a2, 2000)
a1.transfer(a2, 4000)
```

Output

```
2000 transferred from Ravi to Anita
Cannot transfer, insufficient balance
```

Program 94 – Library with Multiple Copies

```
# Program 94: Library Multiple Copies

class Book:
    def __init__(self, title, copies):
        self.title = title
        self.copies = copies

class Library:
    def __init__(self):
        self.books = []

    def add_book(self, book):
        self.books.append(book)

    def borrow(self, title):
        for b in self.books:
            if b.title == title and b.copies > 0:
                b.copies -= 1
                print(f"{title} borrowed. Remaining: {b.copies}")
                return
        print(f"{title} not available")

lib = Library()
lib.add_book(Book("Python", 2))
lib.borrow("Python")
lib.borrow("Python")
lib.borrow("Python")
```

Output

```
Python borrowed. Remaining: 1
Python borrowed. Remaining: 0
Python not available
```

Program 95 – Flight with Passenger Classes & Fare Calculation

```
# Program 95: Flight Fare

class Passenger:
    def __init__(self, name, seat_class):
        self.name = name
        self.seat_class = seat_class

class Flight:
    fares = {"Economy":5000, "Business":10000}

    def __init__(self, flight_no):
        self.flight_no = flight_no
        self.passengers = []

    def book(self, passenger):
        self.passengers.append(passenger)
        print(f"{passenger.name} booked {passenger.seat_class} seat. Fare:
{Flight.fares[passenger.seat_class]}")

f = Flight("AI101")
f.book(Passenger("Ravi","Economy"))
f.book(Passenger("Anita","Business"))
```

Output

```
Ravi booked Economy seat. Fare: 5000
Anita booked Business seat. Fare: 10000
```

Program 96 – Smart Home with Scenes & Scheduling

```
# Program 96: Smart Home with scheduling

class Device:
    def __init__(self, name):
        self.name = name
        self.status = "OFF"

class SmartHome:
    def __init__(self):
        self.devices = []

    def add_device(self, device):
        self.devices.append(device)

    def schedule_scene(self, scene):
        if scene == "Morning":
            for d in self.devices:
                if d.name in ["Coffee Maker","Lights"]:
                    d.status="ON"
        for d in self.devices:
            print(f"{d.name}: {d.status}")
```

```
home = SmartHome()
home.add_device(Device("Coffee Maker"))
home.add_device(Device("Lights"))
home.add_device(Device("Fan"))
home.schedule_scene("Morning")
```

Output

```
Coffee Maker: ON
Lights: ON
Fan: OFF
```

Program 97 – Quiz with Leaderboard

```
# Program 97: Quiz Leaderboard

class Participant:
    def __init__(self, name):
        self.name = name
        self.score = 0

    def update_score(self, points):
        self.score += points

participants = [Participant("Ravi"), Participant("Anita")]
participants[0].update_score(5)
participants[1].update_score(8)
participants.sort(key=lambda x: x.score, reverse=True)
for p in participants:
    print(f"{p.name}: {p.score}")
```

Output

```
Anita: 8
Ravi: 5
```

Program 98 – School with Subjects & Student Marks

```
# Program 98: School with Subjects & Marks

class Student:
    def __init__(self, name):
        self.name = name
        self.marks = {}

    def add_marks(self, subject, mark):
        self.marks[subject] = mark

    def percentage(self):
        return sum(self.marks.values())/len(self.marks) if self.marks else 0

class School:
    def __init__(self):
        self.students = []

    def add_student(self, student):
        self.students.append(student)

    def show_results(self):
        for s in self.students:
            print(f"{s.name} - Marks: {s.marks} - Percentage: {s.percentage():.2f}%")
```

```
s1 = Student("Ravi")
s1.add_marks("Math", 90)
s1.add_marks("Physics", 80)

s2 = Student("Anita")
s2.add_marks("Math", 85)
s2.add_marks("Physics", 95)

school = School()
school.add_student(s1)
school.add_student(s2)
school.show_results()
```

Output

```
Ravi - Marks: {'Math': 90, 'Physics': 80} - Percentage: 85.00%
Anita - Marks: {'Math': 85, 'Physics': 95} - Percentage: 90.00%
```

Program 99 – Advanced Banking System with Transactions History

```
# Program 99: Banking System with History

class Account:
    def __init__(self, name, balance=0):
        self.name = name
        self.balance = balance
        self.transactions = []

    def deposit(self, amount):
        self.balance += amount
        self.transactions.append(f"Deposited {amount}")

    def withdraw(self, amount):
        if amount > self.balance:
            print("Insufficient balance")
        else:
            self.balance -= amount
            self.transactions.append(f"Withdrawn {amount}")

    def show_transactions(self):
        print(f"Transactions of {self.name}:")
        for t in self.transactions:
            print(t)
        print(f"Current Balance: {self.balance}")

a = Account("Ravi", 5000)
a.deposit(2000)
a.withdraw(3000)
a.show_transactions()
```

Output

```
Transactions of Ravi:
Deposited 2000
Withdrawn 3000
Current Balance: 4000
```

Program 100 – Multi-Project Task Management System

```
# Program 100: Task Management System
```

```
class Task:
    def __init__(self, title, status="Pending"):
        self.title = title
        self.status = status

    def complete(self):
        self.status = "Completed"

class Project:
    def __init__(self, name):
        self.name = name
        self.tasks = []

    def add_task(self, task):
        self.tasks.append(task)

    def show_tasks(self):
        print(f"Project: {self.name}")
        for t in self.tasks:
            print(f"{t.title} - {t.status}")

class Company:
    def __init__(self):
        self.projects = []

    def add_project(self, project):
        self.projects.append(project)

    def show_all_projects(self):
        for p in self.projects:
            p.show_tasks()

# Usage
task1 = Task("Design Module")
task2 = Task("Implement Module")
task2.complete()

proj = Project("CRM System")
proj.add_task(task1)
proj.add_task(task2)

company = Company()
company.add_project(proj)
company.show_all_projects()
```

Output

```
Project: CRM System
Design Module - Pending
Implement Module - Completed
```

UNIT-12(DECORATOR, GENERATOR, CONSTRUCTOR)

Decorators (1–10)(Questions)

1. Simple decorator that prints "Before" and "After"

```
def decorator(func):
    def wrapper():
        print("Before function call")
        func()
        print("After function call")
    return wrapper

@decorator
def greet():
    print("Hello, World!")

greet()
```

2. Decorator to measure execution time

```
import time

def timer(func):
    def wrapper():
        start = time.time()
        func()
        end = time.time()
        print(f"Execution time: {end - start:.4f} seconds")
    return wrapper

@timer
def compute():
    sum(range(1000000))

compute()
```

3. Decorator that doubles return value

```
def double_return(func):
    def wrapper():
        return 2 * func()
    return wrapper

@double_return
def number():
    return 5

print(number())  # Output: 10
```

4. Decorator that prints function name

```
def print_name(func):
    def wrapper():
        print(f"Function name: {func.__name__}")
        func()
    return wrapper

@print_name
def say_hello():
    print("Hello")

say_hello()
```

5. Decorator that repeats function 3 times

```
def repeat_three(func):
    def wrapper():
        for _ in range(3):
            func()
    return wrapper

@repeat_three
def hello():
    print("Hello!")

hello()
```

6. Decorator that logs arguments

```
def log_args(func):
    def wrapper(*args, **kwargs):
        print(f"Arguments: {args}, {kwargs}")
        return func(*args, **kwargs)
    return wrapper

@log_args
def add(a, b):
    return a + b

print(add(5, 10))
```

7. Decorator converting return value to uppercase

```
def uppercase(func):
    def wrapper():
        return func().upper()
    return wrapper

@uppercase
def greet():
    return "hello world"

print(greet())
```

8. Decorator counting function calls

```
def counter(func):
    count = 0
    def wrapper():
        nonlocal count
        count += 1
        print(f"Function called {count} times")
        func()
    return wrapper

@counter
def hello():
    print("Hi!")

hello()
hello()
```

9. Decorator printing current datetime

```
from datetime import datetime

def log_time(func):
    def wrapper():
        print(f"Current time: {datetime.now()}")
        func()
    return wrapper

@log_time
def greet():
    print("Hello!")

greet()
```

10. Decorator validating positive integer input

```
def positive_input(func):
    def wrapper(n):
        if n <= 0:
            print("Input must be positive!")
        else:
            func(n)
    return wrapper

@positive_input
def square(n):
    print(n**2)

square(5)
square(-3)
```

Generators (11–18)(Questions)

11. Generator for first 10 natural numbers

```
def natural_numbers():
    for i in range(1, 11):
        yield i

for n in natural_numbers():
    print(n)
```

12. Generator for squares 1–10

```
def squares():
    for i in range(1, 11):
        yield i**2

for s in squares():
    print(s)
```

13. Fibonacci generator up to n

```
def fibonacci(n):
    a, b = 0, 1
    while a <= n:
        yield a
        a, b = b, a + b

for f in fibonacci(20):
    print(f)
```

14. Generator for even numbers 1–20

```
def even_numbers():
    for i in range(2, 21, 2):
        yield i

for e in even_numbers():
    print(e)
```

15. Generator yielding characters from string

```
def char_gen(s):
    for c in s:
        yield c

for ch in char_gen("hello"):
    print(ch)
```

16. Generator for positive numbers in list

```
def positive_gen(lst):
    for n in lst:
        if n > 0:
            yield n

lst = [-5, 3, 0, 7]
for n in positive_gen(lst):
    print(n)
```

17. Generator for primes up to n

```
def is_prime(n):
    if n < 2:
        return False
    for i in range(2, int(n**0.5)+1):
        if n % i == 0:
            return False
    return True

def prime_gen(limit):
    for i in range(2, limit+1):
        if is_prime(i):
            yield i

for p in prime_gen(20):
    print(p)
```

18. Generator for factorials 1–n

```
def factorial_gen(n):
    fact = 1
    for i in range(1, n+1):
        fact *= i
        yield fact

for f in factorial_gen(5):
    print(f)
```

Constructors (19–25)(Questions)

19. Class constructor initializing name and age

```
class Person:
    def __init__(self, name, age):
        self.name = name
        self.age = age

p = Person("Ravi", 25)
print(p.name, p.age)
```

20. Constructor printing welcome message

```
class Welcome:
    def __init__(self):
        print("Welcome to the program!")

w = Welcome()
```

21. Constructor initializing empty list

```
class MyList:
    def __init__(self):
        self.data = []

obj = MyList()
print(obj.data)
```

22. Constructor with default values

```
class Student:
    def __init__(self, name="Unknown", age=18):
        self.name = name
        self.age = age

s1 = Student()
s2 = Student("Anita", 20)
print(s1.name, s1.age)
print(s2.name, s2.age)
```

23. Constructor validating age

```
class Person:
    def __init__(self, name, age):
        if age < 0:
            raise ValueError("Age must be positive")
        self.name = name
        self.age = age

p = Person("Ravi", 25)
# p2 = Person("Anita", -5)  # Raises exception
```

24. Constructor chaining using `__init__`

```
class Base:
    def __init__(self):
        print("Base constructor")

class Child(Base):
    def __init__(self):
        super().__init__()
        print("Child constructor")

c = Child()
```

25. Constructor incrementing class-level counter

```
class Person:
    count = 0
    def __init__(self, name):
        self.name = name
        Person.count += 1

p1 = Person("Ravi")
p2 = Person("Anita")
print(Person.count)
```

Decorators (26–35)(Questions)

26. Decorator for caching function results (memoization)

```
def memoize(func):
    cache = {}
    def wrapper(n):
        if n not in cache:
            cache[n] = func(n)
        return cache[n]
    return wrapper

@memoize
def fib(n):
    if n <= 1:
        return n
    return fib(n-1) + fib(n-2)

print(fib(10))  # Efficient computation with caching
```

27. Decorator to log execution time in milliseconds

```
import time

def time_ms(func):
    def wrapper(*args, **kwargs):
        start = time.time()
        result = func(*args, **kwargs)
        end = time.time()
        print(f"Execution time: {(end-start)*1000:.2f} ms")
        return result
    return wrapper

@time_ms
def compute():
    sum(range(100000))

compute()
```

28. Decorator to ensure only integers as arguments

```
def int_only(func):
    def wrapper(*args):
        if not all(isinstance(x, int) for x in args):
            print("All arguments must be integers")
            return
        return func(*args)
    return wrapper

@int_only
def add(a, b):
    return a + b

print(add(5, 10))
print(add(5, "10"))  # Invalid
```

29. Decorator restricting access based on role

```
def role_check(required_role):
    def decorator(func):
        def wrapper(role):
            if role != required_role:
                print("Access denied")
            else:
                func(role)
        return wrapper
    return decorator

@role_check("admin")
def access_system(role):
    print("Access granted")

access_system("user")
access_system("admin")
```

30. Decorator that retries function up to 3 times

```
def retry(func):
    def wrapper():
        for i in range(3):
            try:
                func()
                break
            except Exception as e:
                print(f"Attempt {i+1} failed: {e}")
    return wrapper

@retry
def risky():
    import random
    if random.choice([True, False]):
        raise ValueError("Random failure")
    print("Success")

risky()
```

31. Decorator printing function docstring

```
def print_doc(func):
    def wrapper():
        print(func.__doc__)
        return func()
    return wrapper

@print_doc
def greet():
    """This function prints a greeting"""
    print("Hello!")

greet()
```

32. Decorator applying another decorator to all class methods

```
def log_method(func):
    def wrapper(*args, **kwargs):
        print(f"Calling {func.__name__}")
        return func(*args, **kwargs)
    return wrapper

def decorate_all_methods(cls):
    for attr in cls.__dict__:
        if callable(getattr(cls, attr)):
            setattr(cls, attr, log_method(getattr(cls, attr)))
    return cls

@decorate_all_methods
class MyClass:
    def method1(self):
        print("Method1 executed")
    def method2(self):
        print("Method2 executed")

obj = MyClass()
obj.method1()
obj.method2()
```

33. Decorator multiplying numeric outputs by 5

```
def multiply_by_5(func):
    def wrapper(*args, **kwargs):
        return func(*args, **kwargs) * 5
    return wrapper

@multiply_by_5
def get_number():
    return 4

print(get_number())
```

34. Decorator converting all string arguments to uppercase

```
def uppercase_args(func):
    def wrapper(*args):
        args = [a.upper() if isinstance(a, str) else a for a in args]
        return func(*args)
    return wrapper

@uppercase_args
def print_name(name):
    print(name)

print_name("Ravi")
```

35. Decorator printing message only on successful execution

```
def success_message(func):
    def wrapper():
        try:
            func()
            print("Function executed successfully!")
        except:
            print("Function failed")
    return wrapper

@success_message
def func1():
    print("Running func1")
    # raise ValueError("Error")  # Uncomment to test failure

func1()
```

Generators (36–45)(Questions)

36. Generator reading large file line by line

```
def read_file(file):
    with open(file) as f:
        for line in f:
            yield line.strip()

# for line in read_file("large.txt"):
#     print(line)
```

37. Generator for numbers divisible by 3 (1–100)

```
def divisible_by_3():
    for i in range(1, 101):
        if i % 3 == 0:
            yield i

for n in divisible_by_3():
    print(n)
```

38. Infinite sequence of natural numbers

```
def infinite_natural():
    n = 1
    while True:
        yield n
        n += 1

gen = infinite_natural()
for _ in range(10):
    print(next(gen))
```

39. Generator to iterate over nested lists

```
def flatten(nested_list):
    for sublist in nested_list:
        for item in sublist:
            yield item

nested = [[1,2],[3,4],[5]]
for x in flatten(nested):
    print(x)
```

40. Generator merging two sorted lists

```
def merge_sorted(a, b):
    i = j = 0
    while i < len(a) and j < len(b):
        if a[i] < b[j]:
            yield a[i]
            i += 1
        else:
            yield b[j]
            j += 1
    while i < len(a):
        yield a[i]
        i += 1
    while j < len(b):
        yield b[j]
        j += 1

for x in merge_sorted([1,3,5],[2,4,6]):
    print(x)
```

41. Generator for squares of odd numbers

```
def odd_squares(n):
    for i in range(1, n+1):
        if i % 2 != 0:
            yield i**2

for s in odd_squares(10):
    print(s)
```

42. Generator for cumulative sum

```
def cumulative_sum(lst):
    total = 0
    for n in lst:
        total += n
        yield total

for s in cumulative_sum([1,2,3,4]):
    print(s)
```

43. Generator yielding numbers in reverse

```
def reverse_gen(n):
    for i in range(n, 0, -1):
        yield i

for x in reverse_gen(5):
    print(x)
```

44. Generator for powers of 2

```
def powers_of_2(n):
    for i in range(n+1):
        yield 2**i

for x in powers_of_2(5):
    print(x)
```

45. Generator yielding random numbers

```
import random
def random_gen(n):
    for _ in range(n):
        yield random.randint(1,100)

for r in random_gen(5):
    print(r)
```

Constructors (46–50)(Questions)

46. Constructor with variable number of arguments

```
class VarArgs:
    def __init__(self, *args):
        self.args = args

obj = VarArgs(1,2,3,4)
print(obj.args)
```

47. Constructor reading data from a file

```
class FileData:
    def __init__(self, filename):
        with open(filename) as f:
            self.data = f.read().splitlines()

# obj = FileData("data.txt")
# print(obj.data)
```

48. Constructor copying attributes from another object

```
class CopyObj:
    def __init__(self, other):
        self.name = other.name
        self.age = other.age

class Person:
    def __init__(self, name, age):
        self.name = name
        self.age = age
```

```
p1 = Person("Ravi", 25)
p2 = CopyObj(p1)
print(p2.name, p2.age)
```

49. Constructor raising exception if required field missing

```
class Person:
    def __init__(self, name=None):
        if not name:
            raise ValueError("Name is required")
        self.name = name

# p = Person()  # Raises exception
p = Person("Ravi")
print(p.name)
```

50. Constructor initializing attributes from dictionary

```
class Person:
    def __init__(self, data):
        self.name = data.get("name")
        self.age = data.get("age")

p = Person({"name":"Ravi","age":25})
print(p.name, p.age)
```

Decorators (51–60)(questions)

51. Decorator that logs exceptions raised by a function

```
def log_exceptions(func):
    def wrapper(*args, **kwargs):
        try:
            return func(*args, **kwargs)
        except Exception as e:
            print(f"Exception in {func.__name__}: {e}")
    return wrapper

@log_exceptions
def divide(a, b):
    return a / b

print(divide(10, 2))
print(divide(5, 0))
```

52. Decorator to enforce type hints at runtime

```
def enforce_types(func):
    def wrapper(*args, **kwargs):
        hints = func.__annotations__
        for i, arg in enumerate(args):
            arg_name = list(hints.keys())[i]
            if not isinstance(arg, hints[arg_name]):
                raise TypeError(f"{arg_name} must be {hints[arg_name]}")
        return func(*args, **kwargs)
    return wrapper

@enforce_types
def add(a: int, b: int):
    return a + b

print(add(5, 3))
# add(5, '3')  # Raises TypeError
```

53. Decorator measuring memory usage

```
import tracemalloc

def memory_usage(func):
    def wrapper(*args, **kwargs):
        tracemalloc.start()
        result = func(*args, **kwargs)
        current, peak = tracemalloc.get_traced_memory()
        print(f"Current memory: {current/1024:.2f} KB; Peak: {peak/1024:.2f} KB")
        tracemalloc.stop()
        return result
    return wrapper

@memory_usage
def create_list(n):
    return [i for i in range(n)]

create_list(10000)
```

54. Decorator that limits number of calls

```
def limit_calls(max_calls):
    def decorator(func):
        count = 0
        def wrapper(*args, **kwargs):
            nonlocal count
            if count >= max_calls:
                print("Function call limit reached")
                return
            count += 1
            return func(*args, **kwargs)
        return wrapper
    return decorator

@limit_calls(3)
def greet():
    print("Hello!")

greet()
greet()
greet()
greet()  # Limit reached
```

55. Decorator ensuring execution order in pipeline

```
def pipeline(func):
    def wrapper(data):
        print("Preprocessing")
        data = data.strip().lower()
        result = func(data)
        print("Postprocessing")
        return result
    return wrapper

@pipeline
def process(data):
    return data[::-1]

print(process(" Hello World "))
```

56. Decorator adding retry with exponential backoff

```
import time
def retry_backoff(func):
    def wrapper(*args, **kwargs):
        delay = 1
        for i in range(3):
            try:
                return func(*args, **kwargs)
            except Exception as e:
                print(f"Attempt {i+1} failed: {e}")
                time.sleep(delay)
                delay *= 2
    return wrapper

@retry_backoff
def risky():
    import random
    if random.choice([True, False]):
        raise ValueError("Random failure")
    print("Success")

risky()
```

57. Decorator validating email input

```
import re
def validate_email(func):
    def wrapper(email):
        if not re.match(r"[^@]+@[^@]+\.[^@]+", email):
            print("Invalid email")
        else:
            func(email)
    return wrapper

@validate_email
def send_email(email):
    print(f"Email sent to {email}")

send_email("test@example.com")
send_email("invalid-email")
```

58. Decorator formatting function output (currency)

```
def format_currency(func):
    def wrapper(*args, **kwargs):
        return f"${func(*args, **kwargs):.2f}"
    return wrapper

@format_currency
def get_price():
    return 123.456

print(get_price())
```

59. Decorator caching results with expiration

```
import time
def cache_with_expiry(expiry):
    cache = {}
    def decorator(func):
        def wrapper(n):
            now = time.time()
            if n in cache and now - cache[n][1] < expiry:
                return cache[n][0]
            result = func(n)
            cache[n] = (result, now)
            return result
        return wrapper
    return decorator

@cache_with_expiry(5)
def square(n):
    print("Computing...")
    return n*n

print(square(4))
print(square(4))  # Cached
```

60. Decorator logging args and return value

```
def log_args_return(func):
    def wrapper(*args, **kwargs):
        print(f"Arguments: {args}, {kwargs}")
        result = func(*args, **kwargs)
        print(f"Return value: {result}")
        return result
    return wrapper

@log_args_return
def multiply(a, b):
    return a*b

multiply(5, 6)
```

Generators (61–70)

61. Generator streaming large CSV as dictionaries

```
import csv
def read_csv(file):
    with open(file) as f:
        reader = csv.DictReader(f)
        for row in reader:
            yield row

# for row in read_csv("data.csv"):
#     print(row)
```

62. Generator yielding sentences from a text file

```
def sentences(file):
    with open(file) as f:
        for line in f:
            for sentence in line.strip().split('.'):
                if sentence:
                    yield sentence.strip()

# for s in sentences("text.txt"):
#     print(s)
```

63. Generator simulating live stock price feed

```
import random
def stock_feed():
    price = 100
    while True:
        price += random.uniform(-1, 1)
        yield round(price, 2)

feed = stock_feed()
for _ in range(5):
    print(next(feed))
```

64. Generator for primes lazily for large n

```
def prime_gen(n):
    def is_prime(num):
        if num < 2:
            return False
        for i in range(2, int(num**0.5)+1):
            if num % i == 0:
                return False
        return True
    for i in range(2, n+1):
        if is_prime(i):
            yield i

for p in prime_gen(50):
    print(p)
```

65. Generator flattening nested dictionaries

```
def flatten_dict(d, parent_key=''):
    for k, v in d.items():
        new_key = f"{parent_key}.{k}" if parent_key else k
        if isinstance(v, dict):
            yield from flatten_dict(v, new_key)
        else:
            yield (new_key, v)

nested = {"a":1, "b":{"c":2,"d":3}}
for k,v in flatten_dict(nested):
    print(k,v)
```

66. Generator reading multiple files alternately

```
def alternate_files(files):
    files_objs = [open(f) for f in files]
    while True:
        done = True
        for f in files_objs:
            line = f.readline()
            if line:
                done = False
                yield line.strip()
        if done:
            break
    for f in files_objs:
        f.close()

# for line in alternate_files(["file1.txt","file2.txt"]):
#     print(line)
```

67. Generator producing unique combinations of a list

```
import itertools
def combinations(lst, r):
    for combo in itertools.combinations(lst, r):
        yield combo

for c in combinations([1,2,3], 2):
    print(c)
```

68. Generator for Fibonacci numbers indefinitely until limit

```
def fib_limit(limit):
    a,b=0,1
    while a <= limit:
        yield a
        a,b = b, a+b

for f in fib_limit(20):
    print(f)
```

69. Generator merging multiple sorted files

```
import heapq
def merge_files(*files):
    file_iters = [open(f) for f in files]
    merged = heapq.merge(*file_iters)
    for line in merged:
        yield line.strip()
    for f in file_iters:
        f.close()

# for l in merge_files("f1.txt","f2.txt"):
#     print(l)
```

70. Generator yielding moving averages

```
def moving_average(lst, k):
    for i in range(len(lst)-k+1):
        yield sum(lst[i:i+k])/k

for ma in moving_average([1,2,3,4,5],3):
    print(ma)
```

Constructors (71–75)

71. Constructor initializing from JSON

```
import json
class Person:
    def __init__(self, json_data):
        data = json.loads(json_data)
        self.name = data["name"]
        self.age = data["age"]

p = Person('{"name":"Ravi","age":25}')
print(p.name,p.age)
```

72. Constructor connecting to database

```
import sqlite3
class DB:
    def __init__(self, dbname):
        self.conn = sqlite3.connect(dbname)
        self.cursor = self.conn.cursor()

db = DB(":memory:")
print(db.conn, db.cursor)
```

73. Constructor raising exception for duplicates

```
class Registry:
    existing = set()
    def __init__(self, name):
        if name in Registry.existing:
            raise ValueError("Duplicate entry")
        self.name = name
        Registry.existing.add(name)

r1 = Registry("Ravi")
# r2 = Registry("Ravi")  # Raises ValueError
```

74. Constructor initializing nested objects

```
class Address:
    def __init__(self, city, country):
        self.city = city
        self.country = country

class Person:
    def __init__(self, name, addr):
        self.name = name
        self.address = addr

a = Address("Delhi","India")
p = Person("Ravi", a)
print(p.name, p.address.city, p.address.country)
```

75. Constructor validating multiple attributes

```
class Employee:
    def __init__(self, name, age, salary):
        if not name or age <= 0 or salary < 0:
            raise ValueError("Invalid input")
        self.name = name
        self.age = age
        self.salary = salary

e = Employee("Ravi",30,50000)
print(e.name,e.age,e.salary)
```

Decorators (76–85)

76. Decorator logging function execution asynchronously

```
import asyncio

def async_log(func):
    async def wrapper(*args, **kwargs):
        print(f"Starting {func.__name__}")
        result = await func(*args, **kwargs)
        print(f"Finished {func.__name__}")
        return result
    return wrapper

@async_log
async def say_hello():
    await asyncio.sleep(1)
    print("Hello World!")

asyncio.run(say_hello())
```

77. Decorator converting blocking function to coroutine

```
import asyncio
def to_coroutine(func):
    async def wrapper(*args, **kwargs):
        loop = asyncio.get_event_loop()
        return await loop.run_in_executor(None, func, *args)
    return wrapper

@to_coroutine
def compute(x, y):
    import time
    time.sleep(1)
    return x + y

result = asyncio.run(compute(5,10))
print(result)
```

78. Decorator throttling function calls (1 call/sec)

```
import time
def throttle(func):
    last_called = 0
    def wrapper(*args, **kwargs):
        nonlocal last_called
        now = time.time()
        if now - last_called >= 1:
            last_called = now
            return func(*args, **kwargs)
        else:
            print("Throttled")
    return wrapper

@throttle
def hello():
    print("Hello")

hello()
time.sleep(1.1)
hello()
```

79. Decorator validating JSON payload

```
import json
def validate_json(func):
    def wrapper(data):
        try:
            obj = json.loads(data)
        except json.JSONDecodeError:
            print("Invalid JSON")
            return
        return func(obj)
    return wrapper

@validate_json
def process_data(obj):
    print(obj)

process_data('{"name":"Ravi"}')
process_data('invalid json')
```

80. Decorator retrying API calls with exponential backoff

```
import time, random
def retry_api(func):
    def wrapper(*args, **kwargs):
        delay = 1
        for i in range(3):
            try:
                return func(*args, **kwargs)
            except Exception as e:
                print(f"Attempt {i+1} failed: {e}")
                time.sleep(delay)
                delay *= 2
    return wrapper

@retry_api
def fetch_data():
    if random.choice([True, False]):
        raise ValueError("API error")
    print("Data fetched")

fetch_data()
```

81. Decorator monitoring performance metrics

```
import time
def monitor(func):
    def wrapper(*args, **kwargs):
        start = time.time()
        result = func(*args, **kwargs)
        end = time.time()
        print(f"{func.__name__} took {end-start:.4f} sec")
        return result
    return wrapper

@monitor
def heavy_computation():
    sum(range(1000000))

heavy_computation()
```

82. Decorator dynamically adding authentication check

```
def auth_required(func):
    def wrapper(user_role):
        if user_role != "admin":
            print("Authentication failed")
            return
        return func()
    return wrapper

@auth_required
```

```
def sensitive_task():
    print("Sensitive data accessed")

sensitive_task("user")
sensitive_task("admin")
```

83. Decorator handling multiple exception types

```
def handle_exceptions(func):
    def wrapper(*args, **kwargs):
        try:
            return func(*args, **kwargs)
        except (ValueError, ZeroDivisionError) as e:
            print(f"Caught exception: {e}")
    return wrapper

@handle_exceptions
def risky(a, b):
    if a < 0:
        raise ValueError("Negative")
    return a / b

risky(-5,2)
risky(5,0)
```

84. Decorator converting outputs to XML

```
def to_xml(func):
    def wrapper(*args, **kwargs):
        data = func(*args, **kwargs)
        xml = "<data>"
        for k,v in data.items():
            xml += f"<{k}>{v}</{k}>"
        xml += "</data>"
        return xml
    return wrapper

@to_xml
def get_data():
    return {"name":"Ravi","age":25}

print(get_data())
```

85. Decorator enabling multi-threaded execution

```
from threading import Thread

def run_in_thread(func):
    def wrapper(*args, **kwargs):
        t = Thread(target=func, args=args, kwargs=kwargs)
        t.start()
        return t
    return wrapper

@run_in_thread
def print_numbers():
    for i in range(5):
        print(i)

t = print_numbers()
t.join()
```

Generators (86–95)

86. Generator streaming live tweets (simulation)

```
import random
def twitter_stream():
    tweets = ["Hello","Python","AI","OpenAI"]
    while True:
        yield random.choice(tweets)

gen = twitter_stream()
for _ in range(5):
    print(next(gen))
```

87. Generator yielding streaming sensor data

```
import random
def sensor_data():
    while True:
        yield {"temperature":random.randint(20,30),"humidity":random.randint(40,60)}

gen = sensor_data()
for _ in range(3):
    print(next(gen))
```

88. Generator lazily evaluating large math series

```
def harmonic_series(n):
    for i in range(1,n+1):
        yield 1/i

for h in harmonic_series(5):
    print(h)
```

89. Generator producing batch of data for ML training

```
def batch_generator(data, batch_size):
    for i in range(0, len(data), batch_size):
        yield data[i:i+batch_size]

for batch in batch_generator(list(range(10)),3):
    print(batch)
```

90. Generator streaming video frames (simulation)

```
def video_frames():
    frame = 0
    while frame < 5:
        yield f"Frame {frame}"
        frame += 1

for f in video_frames():
    print(f)
```

91. Generator producing random UUIDs indefinitely

```
import uuid
def uuid_gen():
    while True:
        yield uuid.uuid4()

gen = uuid_gen()
for _ in range(3):
    print(next(gen))
```

92. Generator monitoring log files

```
import time
def log_monitor(file):
    with open(file) as f:
        f.seek(0,2)  # Move to end
        while True:
            line = f.readline()
            if line:
                yield line.strip()
            else:
                time.sleep(0.1)
```

93. Generator yielding sliding windows of size k

```
def sliding_window(lst, k):
    for i in range(len(lst)-k+1):
        yield lst[i:i+k]

for w in sliding_window([1,2,3,4,5],3):
    print(w)
```

94. Generator producing unique pairs of users

```
import itertools
def user_pairs(users):
    for pair in itertools.combinations(users,2):
        yield pair

for p in user_pairs(["Alice","Bob","Charlie"]):
    print(p)
```

95. Generator yielding real-time stock prices (simulation)

```
import random
def stock_price():
    price = 100
    while True:
        price += random.uniform(-1,1)
        yield round(price,2)

gen = stock_price()
for _ in range(5):
    print(next(gen))
```

Constructors (96–100)

96. Constructor initializing multi-threaded logging system

```
import logging, threading
class Logger:
    def __init__(self, name):
        self.logger = logging.getLogger(name)
        handler = logging.StreamHandler()
        self.logger.addHandler(handler)
        self.logger.setLevel(logging.INFO)
        self.lock = threading.Lock()

    def log(self, msg):
        with self.lock:
            self.logger.info(msg)

log = Logger("App")
log.log("System started")
```

97. Constructor initializing configuration manager

```
class Config:
    def __init__(self, configs):
        self.config = {}
        for cfg in configs:
            self.config.update(cfg)

cfg = Config([{"db":"sqlite"},{"host":"localhost"}])
print(cfg.config)
```

98. Constructor initializing ML model pipeline

```
class MLModel:
    def __init__(self, preprocess=True):
        self.pipeline = []
        if preprocess:
            self.pipeline.append("Scaling")
            self.pipeline.append("Normalization")
        self.pipeline.append("Model")

model = MLModel()
print(model.pipeline)
```

99. Constructor validating complex user input

```
class User:
    def __init__(self, username, age, email):
        if not username or age<0 or "@" not in email:
            raise ValueError("Invalid input")
        self.username = username
        self.age = age
        self.email = email

u = User("Ravi",25,"ravi@example.com")
print(u.username,u.age,u.email)
```

100. Constructor initializing mini banking system

```
class BankAccount:
    accounts = {}
    def __init__(self, name, balance=0):
        if name in BankAccount.accounts:
            raise ValueError("Account exists")
        self.name = name
        self.balance = balance
        BankAccount.accounts[name] = self
    def deposit(self, amount):
        self.balance += amount
    def withdraw(self, amount):
        if amount > self.balance:
            print("Insufficient funds")
        else:
            self.balance -= amount

acc = BankAccount("Ravi",1000)
acc.deposit(500)
acc.withdraw(200)
print(acc.balance)
```

UNIT13-NUMPY

Beginner to Intermediate

1. Create a 1D NumPy array of integers from 1 to 50

```
import numpy as np
arr = np.arange(1, 51)
print(arr)
```

2. Create a 2D NumPy array of shape (5,5) filled with zeros

```
arr = np.zeros((5,5))
print(arr)
```

3. Create a 3D NumPy array of ones with shape (3,3,3)

```
arr = np.ones((3,3,3))
print(arr)
```

4. Create an identity matrix of size 6×6

```
arr = np.eye(6)
print(arr)
```

5. Create a NumPy array of numbers from 10 to 100 with step 10

```
arr = np.arange(10, 101, 10)
print(arr)
```

6. Generate 10 random integers between 0 and 50

```
arr = np.random.randint(0, 51, size=10)
print(arr)
```

7. Generate a 3×3 matrix of random floats between 0 and 1

```
arr = np.random.rand(3,3)
print(arr)
```

8. Find the shape, dimension, and size of a given array

```
arr = np.random.rand(4,5)
print("Shape:", arr.shape)
print("Dimensions:", arr.ndim)
print("Size:", arr.size)
```

9. Reshape a 1D array of size 12 to a 3×4 2D array

```
arr = np.arange(12)
arr2d = arr.reshape(3,4)
print(arr2d)
```

10. Flatten a 2D array into a 1D array

```
arr_flat = arr2d.flatten()
print(arr_flat)
```

11. Concatenate two 2D arrays vertically and horizontally

```
a = np.ones((2,2))
b = np.zeros((2,2))
print("Vertical:\n", np.vstack((a,b)))
print("Horizontal:\n", np.hstack((a,b)))
```

12. Split a 1D array into 3 equal parts

```
arr = np.arange(9)
parts = np.array_split(arr, 3)
for p in parts:
    print(p)
```

13. Find the maximum, minimum, and their indices in an array

```
arr = np.random.randint(0,100,10)
```

```
print("Max:", arr.max(), "at index", arr.argmax())
print("Min:", arr.min(), "at index", arr.argmin())
```

14. Compute the mean, median, and standard deviation of an array

```
arr = np.random.randint(0,50,10)
print("Mean:", arr.mean())
print("Median:", np.median(arr))
print("Std Dev:", arr.std())
```

15. Compute the sum along rows and columns of a 2D array

```
arr = np.random.randint(1,10,(3,4))
print("Sum along axis 0 (columns):", arr.sum(axis=0))
print("Sum along axis 1 (rows):", arr.sum(axis=1))
```

16. Multiply two matrices using `np.dot`

```
a = np.array([[1,2],[3,4]])
b = np.array([[5,6],[7,8]])
result = np.dot(a,b)
print(result)
```

17. Find the element-wise square and square root of an array

```
arr = np.array([1,4,9,16])
print("Square:", np.square(arr))
print("Square root:", np.sqrt(arr))
```

18. Replace all negative numbers in an array with 0

```
arr = np.array([1,-2,3,-4])
arr[arr<0] = 0
print(arr)
```

19. Select all even numbers from an array

```
arr = np.arange(10)
even = arr[arr%2==0]
print(even)
```

20. Create a boolean mask to select elements greater than a threshold

```
arr = np.random.randint(0,20,10)
mask = arr>10
print(arr[mask])
```

21. Sort a 1D array in ascending and descending order

```
arr = np.random.randint(0,50,10)
print("Ascending:", np.sort(arr))
print("Descending:", np.sort(arr)[::-1])
```

22. Reverse a 1D array

```
arr = np.arange(5)
print(arr[::-1])
```

23. Round an array of floats to 2 decimal places

```
arr = np.random.rand(5)*10
print(np.round(arr,2))
```

24. Generate 20 equally spaced numbers between 0 and 5 using `linspace`

```
arr = np.linspace(0,5,20)
print(arr)
```

25. Create a 2D array and swap its rows and columns

```
arr = np.array([[1,2,3],[4,5,6]])
arr_T = arr.T
print(arr_T)
```

26. Find the unique elements and their counts in a NumPy array

```
arr = np.array([1,2,2,3,3,3,4])
unique, counts = np.unique(arr, return_counts=True)
print("Unique elements:", unique)
print("Counts:", counts)
```

27. Compute cumulative sum and cumulative product of an array

```
arr = np.array([1,2,3,4])
print("Cumulative sum:", np.cumsum(arr))
print("Cumulative product:", np.cumprod(arr))
```

28. Compute the dot product of two vectors

```
a = np.array([1,2,3])
b = np.array([4,5,6])
print(np.dot(a,b))
```

29. Compute the cross product of two 3D vectors

```
a = np.array([1,0,0])
b = np.array([0,1,0])
print(np.cross(a,b))
```

30. Compute the determinant of a 3×3 matrix

```
a = np.array([[1,2,3],[0,1,4],[5,6,0]])
print(np.linalg.det(a))
```

31. Compute the eigenvalues and eigenvectors of a square matrix

```
a = np.array([[2,0],[0,3]])
eigvals, eigvecs = np.linalg.eig(a)
print("Eigenvalues:", eigvals)
print("Eigenvectors:\n", eigvecs)
```

32. Solve a system of linear equations using NumPy

```
# 2x + 3y = 8, 3x + y = 5
a = np.array([[2,3],[3,1]])
b = np.array([8,5])
x = np.linalg.solve(a,b)
print("Solution:", x)
```

33. Compute the inverse of a matrix

```
a = np.array([[1,2],[3,4]])
inv_a = np.linalg.inv(a)
print(inv_a)
```

34. Create a diagonal matrix from a given 1D array

```
arr = np.array([1,2,3,4])
diag = np.diag(arr)
print(diag)
```

35. Extract the diagonal of a square matrix

```
a = np.array([[1,2,3],[4,5,6],[7,8,9]])
print(np.diag(a))
```

36. Repeat elements of an array multiple times

```
arr = np.array([1,2,3])
print(np.repeat(arr,3))
```

37. Tile an array to form a larger array

```
arr = np.array([1,2])
print(np.tile(arr, 3))
```

38. Compute element-wise logarithm and exponential of an array

```
arr = np.array([1,2,3])
print("Log:", np.log(arr))
print("Exp:", np.exp(arr))
```

39. Compute element-wise sine, cosine, and tangent of an array

```
arr = np.array([0, np.pi/2, np.pi])
print("Sin:", np.sin(arr))
print("Cos:", np.cos(arr))
print("Tan:", np.tan(arr))
```

40. Create a structured array with fields `name`, `age`, `salary`

```
data = np.array([('Alice',25,50000),('Bob',30,60000)],
                dtype=[('name','U10'),('age','i4'),('salary','f4')])
print(data)
```

41. Extract all rows where age > 30 in a structured array

```
print(data[data['age']>30])
```

42. Use `np.where` to replace all negative numbers with mean of positive numbers

```
arr = np.array([-1,2,-3,4])
mean_pos = arr[arr>0].mean()
arr_new = np.where(arr<0, mean_pos, arr)
print(arr_new)
```

43. Generate a random 5×5 matrix and normalize it to range [0,1]

```
arr = np.random.randint(0,100,(5,5))
arr_norm = (arr - arr.min()) / (arr.max() - arr.min())
print(arr_norm)
```

44. Create a checkerboard matrix of size 8×8

```
checker = np.zeros((8,8),dtype=int)
checker[1::2, ::2] = 1
checker[::2,1::2] = 1
```

```
print(checker)
```

45. Generate a random permutation of numbers from 0 to 19

```
perm = np.random.permutation(20)
print(perm)
```

46. Find all prime numbers in a 1D NumPy array

```
arr = np.arange(2,21)
def is_prime(n):
    return np.all(n % np.arange(2,n) != 0)
primes = np.array([x for x in arr if is_prime(x)])
print(primes)
```

47. Compute the rank of a matrix

```
a = np.array([[1,2,3],[4,5,6],[7,8,9]])
print(np.linalg.matrix_rank(a))
```

48. Compute the trace of a square matrix

```
a = np.array([[1,2],[3,4]])
print(np.trace(a))
```

49. Broadcast a 1D array to add to each row of a 2D matrix

```
a = np.ones((3,3))
b = np.array([1,2,3])
print(a + b)  # Broadcasting
```

50. Compute pairwise distances between rows of a 2D array

```
from scipy.spatial.distance import cdist
arr = np.array([[1,2],[3,4],[5,6]])
dist = cdist(arr, arr, metric='euclidean')
print(dist)
```

Critical / Industry-Level

51. Implement element-wise conditional operations using `np.where` (apply tax if salary > 50000)

```
salary = np.array([40000, 60000, 55000, 30000])
taxed_salary = np.where(salary > 50000, salary*0.9, salary)  # 10% tax if >50k
print(taxed_salary)
```

52. Vectorize a loop that computes y = x^2 + 3x + 2 for a large array

```
x = np.arange(1,1000001)
y = x**2 + 3*x + 2  # vectorized
print(y[:5])
```

53. Create a large 2D random array (10^6 × 10) and compute column-wise mean efficiently

```
arr = np.random.rand(10**6,10)
col_mean = arr.mean(axis=0)
print(col_mean)
```

54. Compute covariance matrix of a dataset

```
arr = np.random.rand(5,3)  # 5 samples, 3 features
cov_matrix = np.cov(arr, rowvar=False)
print(cov_matrix)
```

55. Perform PCA on a dataset using NumPy linear algebra

```
X = np.random.rand(100,3)
X_mean = X - X.mean(axis=0)
cov = np.cov(X_mean, rowvar=False)
eigvals, eigvecs = np.linalg.eigh(cov)
# Project data onto principal components
X_pca = X_mean @ eigvecs[:,::-1]
print(X_pca[:5])
```

56. Implement Min-Max scaling manually using NumPy

```
arr = np.random.randint(0,100,(5,5))
arr_scaled = (arr - arr.min()) / (arr.max() - arr.min())
print(arr_scaled)
```

57. Implement Z-score normalization manually using NumPy

```
arr = np.random.rand(5,5)
arr_norm = (arr - arr.mean(axis=0)) / arr.std(axis=0)
print(arr_norm)
```

58. Find the top 5 largest elements in a large array without sorting entire array

```
arr = np.random.randint(0,1000,1000)
top5 = arr[np.argpartition(-arr, 5)[:5]]
print(top5)
```

59. Compute matrix exponentiation (A^n) for a square matrix

```
A = np.array([[1,2],[3,4]])
n = 3
A_pow = np.linalg.matrix_power(A,n)
print(A_pow)
```

60. Simulate 1000 random walks using NumPy arrays

```
steps = np.random.choice([-1,1], size=(1000,100))
walks = np.cumsum(steps, axis=1)
print(walks[:5])
```

61. Generate a random symmetric matrix

```
A = np.random.rand(4,4)
symm = (A + A.T)/2
print(symm)
```

62. Find the indices of the top 3 maximum values in a 2D array

```
arr = np.random.randint(0,100,(3,3))
flat_indices = np.argpartition(-arr.flatten(), 3)[:3]
indices = np.array(np.unravel_index(flat_indices, arr.shape)).T
print(indices)
```

63. Compute moving average of a 1D time-series array using convolution

```
data = np.random.rand(10)
window_size = 3
mov_avg = np.convolve(data, np.ones(window_size)/window_size, mode='valid')
print(mov_avg)
```

64. Generate a 2D Gaussian kernel matrix

```
def gaussian_kernel(size, sigma=1):
    ax = np.linspace(-(size-1)/2,(size-1)/2,size)
    xx, yy = np.meshgrid(ax,ax)
    kernel = np.exp(-(xx**2 + yy**2)/(2*sigma**2))
    return kernel/kernel.sum()

kernel = gaussian_kernel(5)
print(kernel)
```

65. Implement batch matrix multiplication efficiently using broadcasting

```
A = np.random.rand(10,3,4)
B = np.random.rand(10,4,2)
C = np.matmul(A,B)
print(C.shape)
```

66. Compute the Moore-Penrose pseudo-inverse of a non-square matrix

```
A = np.random.rand(3,5)
A_pinv = np.linalg.pinv(A)
print(A_pinv)
```

67. Implement one-hot encoding for a 1D array of categorical labels

```
labels = np.array([0,1,2,1,0])
one_hot = np.eye(labels.max()+1)[labels]
print(one_hot)
```

68. Compute correlation coefficients between all pairs of columns in a 2D dataset

```
X = np.random.rand(5,3)
corr = np.corrcoef(X, rowvar=False)
print(corr)
```

69. Identify outliers in a dataset using the 1.5*IQR rule

```
arr = np.random.rand(10)*10
Q1 = np.percentile(arr,25)
Q3 = np.percentile(arr,75)
IQR = Q3-Q1
outliers = arr[(arr < Q1 - 1.5*IQR) | (arr > Q3 + 1.5*IQR)]
print(outliers)
```

70. Compute the cumulative distribution function (CDF) of a 1D array

```
arr = np.random.rand(10)
sorted_arr = np.sort(arr)
cdf = np.arange(1, len(arr)+1)/len(arr)
print("CDF:", list(zip(sorted_arr, cdf)))
```

71. Rotate a 2D matrix by 90, 180, 270 degrees without using loops

```
arr = np.array([[1,2,3],[4,5,6],[7,8,9]])
print("90°:\n", np.rot90(arr, -1))
print("180°:\n", np.rot90(arr, 2))
print("270°:\n", np.rot90(arr, 1))
```

72. Flip a 2D array along vertical and horizontal axes

```
arr = np.array([[1,2],[3,4]])
print("Vertical flip:\n", np.flipud(arr))
print("Horizontal flip:\n", np.fliplr(arr))
```

73. Implement a 2D convolution operation manually using NumPy arrays

```
def conv2d(mat, kernel):
    m,n = mat.shape
    kx,ky = kernel.shape
    out = np.zeros((m-kx+1, n-ky+1))
    for i in range(m-kx+1):
        for j in range(n-ky+1):
            out[i,j] = np.sum(mat[i:i+kx,j:j+ky]*kernel)
    return out

mat = np.array([[1,2,3],[4,5,6],[7,8,9]])
kernel = np.array([[1,0],[0,-1]])
print(conv2d(mat,kernel))
```

74. Perform eigen decomposition and reconstruct the original matrix

```
A = np.array([[2,0],[0,3]])
eigvals, eigvecs = np.linalg.eig(A)
A_reconstructed = eigvecs @ np.diag(eigvals) @ np.linalg.inv(eigvecs)
print(A_reconstructed)
```

75. Simulate and vectorize Monte Carlo estimation of Pi using NumPy

```
N = 1000000
x = np.random.rand(N)
y = np.random.rand(N)
inside = np.sum(x**2 + y**2 <= 1)
pi_estimate = (inside/N)*4
print("Estimated Pi:", pi_estimate)
```

UNIT-14 Pandas Solutions

1. Create a Pandas Series from a Python list of numbers

```
import pandas as pd
lst = [10, 20, 30, 40]
s = pd.Series(lst)
print(s)
```

2. Create a Pandas DataFrame from a Python dictionary

```
data = {'Name':['Alice','Bob'],'Age':[25,30]}
df = pd.DataFrame(data)
print(df)
```

3. Load a CSV file into a Pandas DataFrame

```
df = pd.read_csv('https://raw.githubusercontent.com/
mwaskom/seaborn-data/master/tips.csv')
print(df.head())
```

4. Display the first 5 and last 5 rows of a DataFrame

```
print(df.head())
print(df.tail())
```

5. Get column names, index, and basic info of a DataFrame

```
print(df.columns)
print(df.index)
print(df.info())
```

6. Select a single column as Series and as DataFrame

```
print(df['total_bill'])        # Series
print(df[['total_bill']])      # DataFrame
```

7. Select multiple columns from a DataFrame

```
print(df[['total_bill','tip']])
```

8. Filter rows based on a condition (e.g., tip > 5)

```
print(df[df['tip'] > 5])
```

9. Filter rows based on multiple conditions

```
print(df[(df['tip']>5) & (df['total_bill']>20)])
```

10. Add a new column to a DataFrame (tax = 10% of total_bill)

```
df['tax'] = df['total_bill'] * 0.1
print(df.head())
```

11. Delete a column and a row from a DataFrame

```
df_drop = df.drop('tax', axis=1)
df_drop_row = df_drop.drop(0, axis=0)
print(df_drop_row.head())
```

12. Rename columns of a DataFrame`df.rename(columns={'total_bill':'TotalBill','tip':'Tip'},`

```
 inplace=True)

print(df.head())
```

13. Sort a DataFrame by one column and multiple columns

```
print(df.sort_values('Tip'))
print(df.sort_values(['Tip','TotalBill'], ascending=[False, True]))
```

14. Get basic statistics of numeric columns

```
print(df.describe())
```

15. Count unique values and value counts of a column

```
print(df['day'].unique())
print(df['day'].value_counts())
```

16. Check for missing values and count them

```
print(df.isnull().sum())
```

17. Fill missing values with mean, median, or mode

```
df['tip'].fillna(df['tip'].mean(), inplace=True)
```

18. Drop rows or columns with missing values

```
df.dropna(axis=0, inplace=True)  # drop rows
df.dropna(axis=1, inplace=True)  # drop columns
```

19. Replace values in a column

```
df['sex'].replace({'Male':'M','Female':'F'}, inplace=True)
```

20. Select rows by position using `iloc`

```
print(df.iloc[0:5,0:2])
```

21. Select rows by label using `loc`

```
print(df.loc[0:5,['TotalBill','Tip']])
```

22. Reset index and set a new index

```
df_reset = df.reset_index(drop=True)
df.set_index('day', inplace=True)
```

23. Apply a function to a column using `apply()`

```
df['TotalBillSquared'] = df['TotalBill'].apply(lambda x: x**2)
```

24. Map a function or dictionary to a column using `map()`

```
df['SexMapped'] = df['sex'].map({'M':1,'F':0})
```

25. Filter rows using `isin()`

```
print(df[df['day'].isin(['Sun','Sat'])])
```

26. Group by a column and compute aggregate statistics

```
grouped = df.groupby('day')['TotalBill'].mean()
print(grouped)
```

27. Group by multiple columns and aggregate using sum, mean, count

```
agg = df.groupby(['day','sex']).agg({'TotalBill':['sum','mean'],'Tip':'count'})
print(agg)
```

28. Pivot a DataFrame using `pivot_table()`

```
pivot = df.pivot_table(values='TotalBill', index='day', columns='sex', aggfunc='mean')
print(pivot)
```

29. Melt a DataFrame from wide to long format

```
melted = pd.melt(df.reset_index(), id_vars=['day'], value_vars=['TotalBill','Tip'])
print(melted.head())
```

30. Merge two DataFrames on a common column (inner join)

```
df1 = pd.DataFrame({'ID':[1,2,3],'Name':['A','B','C']})
df2 = pd.DataFrame({'ID':[2,3,4],'Score':[90,80,70]})
merged = pd.merge(df1, df2, on='ID', how='inner')
print(merged)
```

31. Merge two DataFrames with outer, left, and right joins

```
outer = pd.merge(df1, df2, on='ID', how='outer')
left = pd.merge(df1, df2, on='ID', how='left')
right = pd.merge(df1, df2, on='ID', how='right')
```

32. Concatenate two DataFrames vertically and horizontally

```
df_v = pd.concat([df1, df2], axis=0, ignore_index=True)
df_h = pd.concat([df1, df2], axis=1)
```

33. Perform an outer join and handle missing values

```
merged = pd.merge(df1, df2, on='ID', how='outer').fillna(0)
print(merged)
```

34. Create a categorical column from a numeric column using `cut()`

```
df['TotalBillCategory'] = pd.cut(df['TotalBill'], bins=[0,10,20,50], labels=['Low','Me
```

35. Create a rank column based on another column

```
df['Rank'] = df['Tip'].rank(ascending=False)
```

36. Find correlation between numeric columns

```
print(df[['TotalBill','Tip']].corr())
```

37. Compute rolling mean and rolling sum for time-series data

```
df['RollingMean'] = df['TotalBill'].rolling(window=3).mean()
df['RollingSum'] = df['TotalBill'].rolling(window=3).sum()
```

38. Shift and lag columns in a DataFrame

```
df['PrevTip'] = df['Tip'].shift(1)
```

39. Compute cumulative sum and cumulative product of a column

```
df['Cumsum'] = df['Tip'].cumsum()
df['Cumprod'] = df['Tip'].cumprod()
```

40. Drop duplicate rows based on a subset of columns

```
df.drop_duplicates(subset=['day','sex'], inplace=True)
```

41. Extract year, month, day, weekday from a datetime column

```
df['Date'] = pd.to_datetime('2025-10-01') + pd.to_timedelta(df.index, unit='D')
df['Year'] = df['Date'].dt.year
df['Month'] = df['Date'].dt.month
df['Day'] = df['Date'].dt.day
df['Weekday'] = df['Date'].dt.day_name()
```

42. Filter rows by date range

```
mask = (df['Date'] > '2025-10-03') & (df['Date'] < '2025-10-05')
print(df.loc[mask])
```

43. Resample time-series data by month or week

```
ts = df.set_index('Date')['TotalBill']
monthly = ts.resample('M').sum()
weekly = ts.resample('W').mean()
```

44. Compute percentage change in a numeric column

```
df['PctChange'] = df['TotalBill'].pct_change() * 100
```

45. Create a flag column based on conditions

```
df['BonusFlag'] = df['Tip'].apply(lambda x: 1 if x > 5 else 0)
```

46. Apply multiple functions to a column using `agg()`

```
df['Tip'].agg(['sum','mean','max'])
```

47. Sort by index and column values together

```
df.sort_index(inplace=True)
df.sort_values(['TotalBill','Tip'], ascending=[True,False], inplace=True)
```

48. Use `query()` to filter rows with a string expression

```
print(df.query('Tip > 5 & TotalBill < 20'))
```

49. Sample random rows from a DataFrame

```
sampled = df.sample(n=5, random_state=1)
```

50. Convert a column to categorical and perform operations

```
df['TotalBillCategory'] = df['TotalBillCategory'].astype('category')
print(df['TotalBillCategory'].value_counts())
```

51. Handle large CSV files efficiently using chunksize

```
# Reading a large CSV file in chunks to avoid memory issues
import pandas as pd

chunksize = 10000
chunk_list = []

for chunk in pd.read_csv('large_file.csv', chunksize=chunksize):
    # Example: filter rows where column 'value' > 100
    filtered_chunk = chunk[chunk['value'] > 100]
    chunk_list.append(filtered_chunk)

# Concatenate all filtered chunks
df_filtered = pd.concat(chunk_list)
print(df_filtered.head())
```

Explanation: Processes large CSVs in manageable chunks and concatenates filtered results.

52. Read multiple CSV files and concatenate into a single DataFrame

```
import glob

all_files = glob.glob("data_folder/*.csv")
df_list = [pd.read_csv(f) for f in all_files]
```

```
df_all = pd.concat(df_list, ignore_index=True)
print(df_all.shape)
```

Explanation: Reads all CSVs in a folder and merges them into one DataFrame.

53. Pivot multi-index DataFrame and perform aggregation

```
data = {
    'Region': ['North','North','South','South'],
    'Product': ['A','B','A','B'],
    'Sales': [100,150,200,250]
}
df = pd.DataFrame(data)
pivot = df.pivot_table(values='Sales', index='Region', columns='Product',
 aggfunc='sum')
print(pivot)
```

Explanation: Creates a table of aggregated sales per region and product.

54. Perform cross-tabulation of two categorical columns

```
cross_tab = pd.crosstab(df['Region'], df['Product'])
print(cross_tab)
```

Explanation: Shows counts of occurrences for combinations of two categorical variables.

55. Handle missing data using interpolation

```
df = pd.DataFrame({'Value':[10,None,30,None,50]})
df['Interpolated'] = df['Value'].interpolate(method='linear')
print(df)
```

Explanation: Fills missing values by linear interpolation.

56. Forward-fill and backward-fill missing values

```
df['FFill'] = df['Value'].fillna(method='ffill')
df['BFill'] = df['Value'].fillna(method='bfill')
print(df)
```

Explanation: Forward-fill fills with previous value, backward-fill fills with next value.

57. Detect and remove outliers using IQR method

```
df = pd.DataFrame({'Score':[10,12,14,100,15,13,11]})
```

```
Q1 = df['Score'].quantile(0.25)
Q3 = df['Score'].quantile(0.75)
IQR = Q3 - Q1
df_no_outliers = df[~((df['Score'] < Q1 - 1.5*IQR) | (df['Score'] > Q3 + 1.5*IQR))]
print(df_no_outliers)
```

Explanation: Outliers beyond 1.5*IQR are removed.

58. Apply a custom function to multiple columns simultaneously

```
df = pd.DataFrame({'A':[1,2,3],'B':[4,5,6]})
def custom_func(x):
    return x['A'] + 2*x['B']

df['Result'] = df.apply(custom_func, axis=1)
print(df)
```

Explanation: Combines multiple columns to calculate a new column.

59. Vectorize operations on multiple columns for performance

```
df['ResultVec'] = df['A'] + 2*df['B']  # vectorized version
```

Explanation: Faster than using `apply()` for large DataFrames.

60. Merge multiple DataFrames iteratively using a loop

```
df1 = pd.DataFrame({'ID':[1,2],'Value1':[10,20]})
df2 = pd.DataFrame({'ID':[1,2],'Value2':[30,40]})
df3 = pd.DataFrame({'ID':[1,2],'Value3':[50,60]})

dfs = [df1, df2, df3]
from functools import reduce
df_merged = reduce(lambda left,right: pd.merge(left,right,on='ID'), dfs)
print(df_merged)
```

Explanation: Merges multiple DataFrames on a common key iteratively.

61. Perform groupby-apply operations with a custom function

```
df = pd.DataFrame({'Group':['A','A','B','B'],'Value':[10,20,30,40]})
def top_value(x):
    return x.max()

result = df.groupby('Group')['Value'].apply(top_value)
print(result)
```

Explanation: Computes max value per group.

62. Detect duplicate rows and retain first/last occurrences

```
df = pd.DataFrame({'A':[1,2,2,3],'B':[5,6,6,7]})
df_first = df.drop_duplicates()
df_last = df.drop_duplicates(keep='last')
print(df_first)
print(df_last)
```

Explanation: Removes duplicate rows while retaining first or last occurrence.

63. Perform one-hot encoding for categorical columns

```
df = pd.DataFrame({'Color':['Red','Blue','Green']})
df_encoded = pd.get_dummies(df, columns=['Color'])
print(df_encoded)
```

Explanation: Converts categorical column to binary columns.

64. Compute weighted average for grouped data

```
df = pd.DataFrame({'Group':['A','A','B','B'],'Value':[10,20,30,40],'Weight':[1,2,1,3]
df['Weighted'] = df['Value']*df['Weight']
result = df.groupby('Group').apply(lambda x: x['Weighted'].sum()/x['Weight'].sum())
print(result)
```

Explanation: Computes weighted mean per group.

65. Join DataFrames on multiple keys

```
df1 = pd.DataFrame({'ID':[1,2],'Dept':['HR','IT'],'Score':[90,80]})
df2 = pd.DataFrame({'ID':[1,2],'Dept':['HR','IT'],'Bonus':[1000,2000]})
df_joined = pd.merge(df1, df2, on=['ID','Dept'])
print(df_joined)
```

Explanation: Joins DataFrames using composite keys.

66. Perform time-series analysis using rolling window and expanding window

```
df = pd.DataFrame({'Value':[10,20,30,40,50]})
df['RollingMean'] = df['Value'].rolling(window=3).mean()
df['ExpandingSum'] = df['Value'].expanding().sum()
```

```
print(df)
```

Explanation: Rolling computes moving statistics; expanding computes cumulative statistics.

67. Reshape a dataset from long to wide using pivot

```
df = pd.DataFrame({'Date':['2025-10-01','2025-10-01','2025-10-02'],
'Metric':['A','B','A'],'Value':[10,20,30]})
df_wide = df.pivot(index='Date', columns='Metric', values='Value')
print(df_wide)
```

Explanation: Converts long-format data to wide format.

68. Create hierarchical indexing (multi-index) and select subsets

```
df = pd.DataFrame({'Region':['North','North','South','South'],
'Product':['A','B','A','B'],
'Sales':[100,150,200,250]})
df_multi = df.set_index(['Region','Product'])
print(df_multi.loc['North'])
```

Explanation: Multi-index allows hierarchical selection.

69. Stack and unstack a multi-index DataFrame

```
unstacked = df_multi.unstack()
stacked = unstacked.stack()
print(unstacked)
print(stacked)
```

Explanation: Converts between long and wide hierarchical formats.

70. Apply lambda functions with multiple arguments across columns

```
df = pd.DataFrame({'A':[1,2],'B':[3,4]})
df['C'] = df.apply(lambda x: x['A'] + 2*x['B'], axis=1)
```

71. Perform conditional updates for multiple columns using `np.where`

```
import numpy as np
df['New'] = np.where(df['A']>1, df['B']*2, df['B']+2)
```

72. Merge datasets with fuzzy matching

```
# Example using 'fuzzywuzzy' library
```

```
from fuzzywuzzy import process
choices = ['apple','banana','grape']
query = 'aple'
best_match = process.extractOne(query, choices)
print(best_match)
```

Explanation: Approximate string matching for merging datasets.

73. Optimize memory usage by converting dtypes

```
df = pd.DataFrame({'A':[1,2,3],'B':[1000,2000,3000]})
df['A'] = df['A'].astype('int8')
df['B'] = df['B'].astype('int16')
print(df.dtypes)
```

74. Simulate a small ETL pipeline: load, clean, transform, aggregate

```
# Load
df = pd.read_csv('sales.csv')
# Clean
df.dropna(subset=['Sales'], inplace=True)
# Transform
df['SalesTax'] = df['Sales']*0.1
# Aggregate
summary = df.groupby('Region')['Sales'].sum()
print(summary)
```

75. Perform exploratory data analysis (EDA)

```
import seaborn as sns
import matplotlib.pyplot as plt

df = sns.load_dataset('tips')
print(df.describe())
print(df.info())
sns.histplot(df['total_bill'], bins=20)
plt.show()
sns.boxplot(x='day', y='total_bill', data=df)
plt.show()
sns.scatterplot(x='total_bill', y='tip', hue='sex', data=df)
plt.show()
```

Explanation: Covers statistical summary, info, and visual exploration using Pandas + Seaborn.

UNIT-15
Scikit-learn Solutions – Beginner (1–25)

1. Import `scikit-learn` and check its version

```
import sklearn
print("Scikit-learn version:", sklearn.__version__)
```

Explanation: Verifies the installed version of Scikit-learn.

2. Load the `iris` dataset and display its features

```
from sklearn.datasets import load_iris
iris = load_iris()
X = iris.data      # features
y = iris.target    # target labels

print("Feature names:", iris.feature_names)
print("First 5 rows of features:\n", X[:5])
print("Target labels:\n", y[:5])
```

Explanation: Loads built-in Iris dataset and displays first few entries.

3. Split a dataset into train and test sets

```
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3,
random_state=42)
print("Train shape:", X_train.shape, "Test shape:", X_test.shape)
```

Explanation: Splits dataset into 70% train, 30% test.

4. Standardize features using `StandardScaler`

```
from sklearn.preprocessing import StandardScaler
scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)
print("Mean after scaling:", X_train_scaled.mean(axis=0))
print("Std after scaling:", X_train_scaled.std(axis=0))
```

Explanation: StandardScaler standardizes features to zero mean and unit variance.

5. Apply `MinMaxScaler` to scale features between 0–1

```
from sklearn.preprocessing import MinMaxScaler
scaler = MinMaxScaler()
X_scaled = scaler.fit_transform(X)
print("Min of scaled features:", X_scaled.min(axis=0))
print("Max of scaled features:", X_scaled.max(axis=0))
```

Explanation: Scales data to a fixed range [0,1].

6. Encode categorical variables using `LabelEncoder`

```
from sklearn.preprocessing import LabelEncoder
le = LabelEncoder()
y_encoded = le.fit_transform(y)
print("Encoded labels:", y_encoded[:10])
```

Explanation: Converts categorical labels to integers.

7. Apply `OneHotEncoder` to a categorical column

```
from sklearn.preprocessing import OneHotEncoder
import numpy as np

# Example categorical feature
colors = np.array(['Red','Blue','Green','Red','Blue']).reshape(-1,1)
ohe = OneHotEncoder(sparse=False)
colors_encoded = ohe.fit_transform(colors)
print(colors_encoded)
```

Explanation: Converts categorical feature into binary (dummy) variables.

8. Fit a `LinearRegression` model on a dataset

```
from sklearn.linear_model import LinearRegression
# Using iris petal length to predict petal width (example)
X_lr = X[:,2].reshape(-1,1)
y_lr = X[:,3]
lr = LinearRegression()
lr.fit(X_lr, y_lr)
print("Coefficients:", lr.coef_)
print("Intercept:", lr.intercept_)
```

Explanation: Fits a simple linear regression model.

9. Predict outputs using a trained `LinearRegression` model

```
y_pred = lr.predict(X_lr[:5])
print("Predictions for first 5 samples:", y_pred)
```

10. Calculate mean squared error (MSE) of predictions

```
from sklearn.metrics import mean_squared_error
```

```
mse = mean_squared_error(y_lr, lr.predict(X_lr))
print("Mean Squared Error:", mse)
```

11. Fit a `LogisticRegression` model for classification

```
from sklearn.linear_model import LogisticRegression
clf = LogisticRegression(max_iter=200)
clf.fit(X_train_scaled, y_train)
print("Training accuracy:", clf.score(X_train_scaled, y_train))
```

12. Compute accuracy score for classification predictions

```
from sklearn.metrics import accuracy_score
y_pred = clf.predict(X_test_scaled)
print("Test accuracy:", accuracy_score(y_test, y_pred))
```

13. Split data with stratified sampling for classification

```
X_train_strat, X_test_strat, y_train_strat, y_test_strat = train_test_split(
    X, y, test_size=0.3, random_state=42, stratify=y)
print("Class distribution in test set:", np.bincount(y_test_strat))
```

Explanation: Ensures class proportions are preserved in train/test sets.

14. Use `KNeighborsClassifier` for classification

```
from sklearn.neighbors import KNeighborsClassifier
knn = KNeighborsClassifier(n_neighbors=3)
knn.fit(X_train_scaled, y_train)
print("KNN test accuracy:", knn.score(X_test_scaled, y_test))
```

15. Apply `KMeans` clustering on a dataset

```
from sklearn.cluster import KMeans
kmeans = KMeans(n_clusters=3, random_state=42)
kmeans.fit(X)
print("Cluster centers:\n", kmeans.cluster_centers_)
print("Labels:", kmeans.labels_[:10])
```

16. Visualize clusters using PCA for dimensionality reduction

```
from sklearn.decomposition import PCA
import matplotlib.pyplot as plt

pca = PCA(n_components=2)
X_pca = pca.fit_transform(X)
plt.scatter(X_pca[:,0], X_pca[:,1], c=kmeans.labels_)
plt.xlabel('PC1')
plt.ylabel('PC2')
plt.title('KMeans Clusters (PCA Reduced)')
plt.show()
```

17. Fit a `DecisionTreeClassifier` and plot its tree

```
from sklearn.tree import DecisionTreeClassifier, plot_tree
dt = DecisionTreeClassifier(random_state=42)
dt.fit(X_train_scaled, y_train)
plt.figure(figsize=(12,8))
plot_tree(dt, feature_names=iris.feature_names, class_names=iris.target_names,
 filled=True)
plt.show()
```

18. Apply `RandomForestClassifier` on a dataset

```
from sklearn.ensemble import RandomForestClassifier
rf = RandomForestClassifier(n_estimators=100, random_state=42)
rf.fit(X_train_scaled, y_train)
print("Random Forest test accuracy:", rf.score(X_test_scaled, y_test))
```

19. Check feature importances from a Random Forest model

```
import pandas as pd
feat_importances = pd.Series(rf.feature_importances_, index=iris.feature_names)
print(feat_importances.sort_values(ascending=False))
```

20. Fit a `SupportVectorMachine` classifier with linear kernel

```
from sklearn.svm import SVC
svc = SVC(kernel='linear')
svc.fit(X_train_scaled, y_train)
print("SVM test accuracy:", svc.score(X_test_scaled, y_test))
```

21. Apply `PolynomialFeatures` to transform input features

```
from sklearn.preprocessing import PolynomialFeatures
poly = PolynomialFeatures(degree=2)
X_poly = poly.fit_transform(X_lr)
print("Original shape:", X_lr.shape, "Transformed shape:", X_poly.shape)
```

22. Fit `Ridge` regression and check coefficients

```
from sklearn.linear_model import Ridge
ridge = Ridge(alpha=1.0)
ridge.fit(X_lr, y_lr)
print("Ridge coefficients:", ridge.coef_)
```

23. Fit `Lasso` regression and check coefficients

```
from sklearn.linear_model import Lasso
lasso = Lasso(alpha=0.1)
```

```
lasso.fit(X_lr, y_lr)
print("Lasso coefficients:", lasso.coef_)
```

24. Use `train_test_split` with different `random_state` values

```
X_train1, X_test1, y_train1, y_test1 = train_test_split(X, y, random_state=0)
X_train2, X_test2, y_train2, y_test2 = train_test_split(X, y, random_state=42)
print("Train shapes with different random states:", X_train1.shape, X_train2.shape)
```

25. Handle missing data using `SimpleImputer` and fit a model

```
from sklearn.impute import SimpleImputer
import numpy as np

# Introduce missing values
X_missing = X.copy()
X_missing[0,0] = np.nan
imputer = SimpleImputer(strategy='mean')
X_imputed = imputer.fit_transform(X_missing)

lr.fit(X_imputed, y)
print("Model fitted on imputed data, first 5 predictions:", lr.predict(X_imputed[:5]))
```

26. Apply cross-validation using `cross_val_score`

```
from sklearn.model_selection import cross_val_score
scores = cross_val_score(clf, X_train_scaled, y_train, cv=5)  # 5-fold CV
print("Cross-validation scores:", scores)
print("Mean CV accuracy:", scores.mean())
```

Explanation: Evaluates model performance across 5 folds for reliability.

27. Tune hyperparameters using `GridSearchCV`

```
from sklearn.model_selection import GridSearchCV
param_grid = {'C':[0.1,1,10], 'kernel':['linear','rbf']}
grid = GridSearchCV(SVC(), param_grid, cv=5)
grid.fit(X_train_scaled, y_train)
print("Best params:", grid.best_params_)
print("Best CV score:", grid.best_score_)
```

Explanation: Finds optimal hyperparameters for SVC.

28. Use `RandomizedSearchCV` for hyperparameter tuning

```
from sklearn.model_selection import RandomizedSearchCV
import scipy.stats as stats

param_dist = {'C': stats.uniform(0.1,10), 'kernel':['linear','rbf']}
rand_search = RandomizedSearchCV(SVC(), param_dist, n_iter=10, cv=5, random_state=42)
rand_search.fit(X_train_scaled, y_train)
print("Best params (random search):", rand_search.best_params_)
```

Explanation: Efficient search using random combinations.

29. Fit a `GradientBoostingClassifier` and evaluate accuracy

```
from sklearn.ensemble import GradientBoostingClassifier
gb = GradientBoostingClassifier(n_estimators=100, learning_rate=0.1,
random_state=42)
gb.fit(X_train_scaled, y_train)
print("Gradient Boosting test accuracy:", gb.score(X_test_scaled, y_test))
```

30. Fit `AdaBoostClassifier` and compare with Random Forest

```
from sklearn.ensemble import AdaBoostClassifier
adb = AdaBoostClassifier(n_estimators=100, random_state=42)
adb.fit(X_train_scaled, y_train)
print("AdaBoost test accuracy:", adb.score(X_test_scaled, y_test))
print("Random Forest test accuracy:", rf.score(X_test_scaled, y_test))
```

31. Perform PCA to reduce dimensions of the dataset

```
pca = PCA(n_components=2)
X_pca = pca.fit_transform(X_train_scaled)
print("Explained variance ratio:", pca.explained_variance_ratio_)
```

32. Visualize explained variance of PCA components

```
import matplotlib.pyplot as plt
plt.plot(range(1, len(pca.explained_variance_ratio_)+1),
 pca.explained_variance_ratio_, marker='o')
plt.xlabel('Principal Component')
plt.ylabel('Explained Variance Ratio')
plt.title('PCA Explained Variance')
plt.show()
```

33. Use `StandardScaler` + `PCA` + classifier in a pipeline

```
from sklearn.pipeline import Pipeline
pipeline = Pipeline([
    ('scaler', StandardScaler()),
    ('pca', PCA(n_components=2)),
    ('clf', LogisticRegression(max_iter=200))
])
pipeline.fit(X_train, y_train)
print("Pipeline test accuracy:", pipeline.score(X_test, y_test))
```

34. Use `PolynomialFeatures` + `Ridge` in a pipeline

```
pipeline_poly = Pipeline([
    ('poly', PolynomialFeatures(degree=2)),
    ('ridge', Ridge(alpha=1.0))
])
pipeline_poly.fit(X_lr, y_lr)
print("Predictions for first 5 samples:", pipeline_poly.predict(X_lr[:5]))
```

35. Fit `LogisticRegression` with L1 and L2 regularization

```
clf_l1 = LogisticRegression(penalty='l1', solver='liblinear', max_iter=200)
clf_l1.fit(X_train_scaled, y_train)
clf_l2 = LogisticRegression(penalty='l2', max_iter=200)
clf_l2.fit(X_train_scaled, y_train)
print("L1 test accuracy:", clf_l1.score(X_test_scaled, y_test))
print("L2 test accuracy:", clf_l2.score(X_test_scaled, y_test))
```

36. Compute confusion matrix for classification predictions

```
from sklearn.metrics import confusion_matrix
y_pred = clf.predict(X_test_scaled)
cm = confusion_matrix(y_test, y_pred)
print("Confusion matrix:\n", cm)
```

37. Plot ROC curve for a binary classifier

```
from sklearn.metrics import roc_curve, auc
from sklearn.preprocessing import label_binarize

# Binarize for class 0 vs rest
y_test_bin = label_binarize(y_test, classes=[0,1,2])[:,0]
y_score = clf.predict_proba(X_test_scaled)[:,0]
fpr, tpr, _ = roc_curve(y_test_bin, y_score)
roc_auc = auc(fpr, tpr)

plt.plot(fpr, tpr, label='ROC curve (area = {:.2f})'.format(roc_auc))
plt.plot([0,1],[0,1],'k--')
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('ROC Curve')
plt.legend()
plt.show()
```

38. Calculate AUC score for ROC curve

```
from sklearn.metrics import roc_auc_score
auc_score = roc_auc_score(y_test_bin, y_score)
print("AUC score:", auc_score)
```

39. Apply `SMOTE` to handle imbalanced dataset

```
from imblearn.over_sampling import SMOTE

# Example: simulate imbalance
y_imb = y.copy()
y_imb[y_imb==2] = 1  # make class 1 larger
```

```
sm = SMOTE(random_state=42)
X_res, y_res = sm.fit_resample(X, y_imb)
print("Original class distribution:", np.bincount(y_imb))
print("Resampled class distribution:", np.bincount(y_res))
```

40. Fit `DecisionTreeRegressor` and compute R² score

```
from sklearn.tree import DecisionTreeRegressor
dt_reg = DecisionTreeRegressor(random_state=42)
dt_reg.fit(X_lr, y_lr)
r2 = dt_reg.score(X_lr, y_lr)
print("Decision Tree R²:", r2)
```

41. Fit `RandomForestRegressor` and compute RMSE

```
from sklearn.ensemble import RandomForestRegressor
from sklearn.metrics import mean_squared_error
rf_reg = RandomForestRegressor(n_estimators=100, random_state=42)
rf_reg.fit(X_lr, y_lr)
y_pred = rf_reg.predict(X_lr)
rmse = mean_squared_error(y_lr, y_pred, squared=False)
print("Random Forest RMSE:", rmse)
```

42. Use `BaggingClassifier` on a small dataset

```
from sklearn.ensemble import BaggingClassifier
bag = BaggingClassifier(base_estimator=DecisionTreeClassifier(),
 n_estimators=10, random_state=42)
bag.fit(X_train_scaled, y_train)
print("Bagging test accuracy:", bag.score(X_test_scaled, y_test))
```

43. Use `StackingClassifier` combining multiple models

```
from sklearn.ensemble import StackingClassifier
estimators = [
    ('lr', LogisticRegression(max_iter=200)),
    ('rf', RandomForestClassifier(n_estimators=50, random_state=42))
]
stack = StackingClassifier(estimators=estimators,
 final_estimator=LogisticRegression())
stack.fit(X_train_scaled, y_train)
print("Stacking test accuracy:", stack.score(X_test_scaled, y_test))
```

44. Implement `Pipeline` to chain preprocessing and model

```
pipeline = Pipeline([
    ('scaler', StandardScaler()),
    ('clf', RandomForestClassifier(n_estimators=50, random_state=42))
])
pipeline.fit(X_train, y_train)
print("Pipeline test accuracy:", pipeline.score(X_test, y_test))
```

45. Apply `StandardScaler` inside a pipeline and fit model

```
pipeline = Pipeline([
    ('scaler', StandardScaler()),
    ('svc', SVC(kernel='linear'))
])
pipeline.fit(X_train, y_train)
print("Pipeline with SVC accuracy:", pipeline.score(X_test, y_test))
```

46. Fit `SGDClassifier` for large datasets

```
from sklearn.linear_model import SGDClassifier
sgd = SGDClassifier(max_iter=1000, tol=1e-3, random_state=42)
sgd.fit(X_train_scaled, y_train)
print("SGDClassifier test accuracy:", sgd.score(X_test_scaled, y_test))
```

47. Evaluate model using `cross_val_predict`

```
from sklearn.model_selection import cross_val_predict
y_pred_cv = cross_val_predict(clf, X_train_scaled, y_train, cv=5)
print("First 10 predictions using CV:", y_pred_cv[:10])
```

48. Apply `MinMaxScaler` and visualize feature distributions

```
X_scaled = MinMaxScaler().fit_transform(X)
import matplotlib.pyplot as plt
plt.boxplot(X_scaled)
plt.title("Boxplot of MinMax Scaled Features")
plt.show()
```

49. Fit `ElasticNet` regression and compare with Lasso/Ridge

```
from sklearn.linear_model import ElasticNet
en = ElasticNet(alpha=0.1, l1_ratio=0.5)
en.fit(X_lr, y_lr)
print("ElasticNet coefficients:", en.coef_)
```

50. Split dataset into train/validation/test sets manually

```
# 60% train, 20% val, 20% test
X_temp, X_test, y_temp, y_test = train_test_split(X, y, test_size=0.2,
random_state=42)
X_train, X_val, y_train, y_val = train_test_split
(X_temp, y_temp, test_size=0.25, random_state=42)
print("Train:", X_train.shape, "Val:", X_val.shape, "Test:", X_test.shape)
```

51. Apply feature selection using `SelectKBest`

```
from sklearn.feature_selection import SelectKBest, f_classif
```

```
selector = SelectKBest(score_func=f_classif, k=2)
X_new = selector.fit_transform(X_train_scaled, y_train)
print("Original shape:", X_train_scaled.shape)
print("Selected features shape:", X_new.shape)
```

Explanation: Selects top k features based on ANOVA F-test.

52. Apply recursive feature elimination (RFE) with estimator

```
from sklearn.feature_selection import RFE
estimator = LogisticRegression(max_iter=200)
rfe = RFE(estimator, n_features_to_select=2)
rfe.fit(X_train_scaled, y_train)
print("Selected features:", rfe.support_)
```

Explanation: Recursively eliminates least important features.

53. Apply `VarianceThreshold` to remove low variance features

```
from sklearn.feature_selection import VarianceThreshold
selector = VarianceThreshold(threshold=0.1)
X_var = selector.fit_transform(X_train_scaled)
print("Shape after removing low variance features:", X_var.shape)
```

Explanation: Removes features with variance below threshold.

54. Handle categorical features with `ColumnTransformer`

```
from sklearn.compose import ColumnTransformer
from sklearn.preprocessing import OneHotEncoder
import numpy as np

# Example dataset
X_cat = np.array([['Red', 1], ['Blue', 2], ['Green', 3]])
ct = ColumnTransformer([('onehot', OneHotEncoder(), [0])], remainder='passthrough')
X_trans = ct.fit_transform(X_cat)
print(X_trans)
```

Explanation: Applies transformations selectively to specific columns.

55. Encode categorical variables with `OneHotEncoder` inside pipeline

```
pipeline = Pipeline([
    ('preprocess', ColumnTransformer([('onehot', OneHotEncoder(), [0])],
 remainder='passthrough')),
    ('clf', LogisticRegression(max_iter=200))
])
y_dummy = np.array([0,1,0])
```

```
pipeline.fit(X_cat, y_dummy)
print("Pipeline fitted with one-hot encoder")
```

56. Perform nested cross-validation

```
from sklearn.model_selection import cross_val_score, GridSearchCV
param_grid = {'C':[0.1,1,10]}
grid = GridSearchCV(LogisticRegression(max_iter=200), param_grid, cv=3)
scores = cross_val_score(grid, X_train_scaled, y_train, cv=5)
print("Nested CV scores:", scores)
print("Mean nested CV accuracy:", scores.mean())
```

Explanation: Performs inner loop for hyperparameter tuning and outer loop for evaluation.

57. Fit `XGBoostClassifier` and compute feature importance

```
from xgboost import XGBClassifier
xgb = XGBClassifier(use_label_encoder=False, eval_metric='mlogloss', random_state=42)
xgb.fit(X_train_scaled, y_train)
importances = xgb.feature_importances_
print("XGBoost feature importances:", importances)
```

58. Apply `LightGBM` for classification task

```
from lightgbm import LGBMClassifier
lgb = LGBMClassifier(random_state=42)
lgb.fit(X_train_scaled, y_train)
print("LightGBM test accuracy:", lgb.score(X_test_scaled, y_test))
```

59. Perform hyperparameter tuning with Bayesian optimization (skopt)

```
from skopt import BayesSearchCV

opt = BayesSearchCV(LogisticRegression(max_iter=200),
                    {'C': (0.1, 10.0, 'log-uniform')}, cv=3)
opt.fit(X_train_scaled, y_train)
print("Best params (Bayesian):", opt.best_params_)
```

60. Implement custom scoring function in `cross_val_score`

```
from sklearn.metrics import make_scorer, f1_score
custom_scorer = make_scorer(f1_score, average='macro')
scores = cross_val_score(clf, X_train_scaled, y_train, cv=5, scoring=custom_scorer)
print("Custom F1-score CV:", scores)
```

61. Apply `StandardScaler` + `PCA` + `LogisticRegression` for pipeline

```
pipeline = Pipeline([
    ('scaler', StandardScaler()),
    ('pca', PCA(n_components=2)),
    ('clf', LogisticRegression(max_iter=200))
])
pipeline.fit(X_train, y_train)
print("Pipeline test accuracy:", pipeline.score(X_test, y_test))
```

62. Fit `KMeans` and compute silhouette score

```
from sklearn.metrics import silhouette_score
kmeans = KMeans(n_clusters=3, random_state=42)
labels = kmeans.fit_predict(X)
score = silhouette_score(X, labels)
print("Silhouette score:", score)
```

63. Use `DBSCAN` clustering on noisy dataset

```
from sklearn.cluster import DBSCAN
dbscan = DBSCAN(eps=0.5, min_samples=5)
labels = dbscan.fit_predict(X)
print("DBSCAN cluster labels:", labels[:10])
```

64. Fit `IsolationForest` for anomaly detection

```
from sklearn.ensemble import IsolationForest
iso = IsolationForest(contamination=0.1, random_state=42)
iso.fit(X_train_scaled)
pred = iso.predict(X_test_scaled)
print("Anomaly predictions (-1=outlier,1=inlier):", pred[:10])
```

65. Fit `OneClassSVM` for novelty detection

```
from sklearn.svm import OneClassSVM
ocsvm = OneClassSVM(gamma='auto', nu=0.1)
ocsvm.fit(X_train_scaled)
pred = ocsvm.predict(X_test_scaled)
print("OneClassSVM predictions:", pred[:10])
```

66. Apply ensemble stacking with multiple classifiers

```
stack = StackingClassifier(
    estimators=[('lr', LogisticRegression(max_iter=200)),
                ('rf', RandomForestClassifier(n_estimators=50, random_state=42))],
    final_estimator=LogisticRegression()
)
stack.fit(X_train_scaled, y_train)
print("StackingClassifier test accuracy:", stack.score(X_test_scaled, y_test))
```

67. Fit `HistGradientBoostingClassifier` and evaluate performance

```
from sklearn.ensemble import HistGradientBoostingClassifier
hgb = HistGradientBoostingClassifier(random_state=42)
hgb.fit(X_train_scaled, y_train)
```

```
print("HGBClassifier test accuracy:", hgb.score(X_test_scaled, y_test))
```

68. Perform multi-output regression using `MultiOutputRegressor`

```
from sklearn.multioutput import MultiOutputRegressor
from sklearn.linear_model import Ridge

y_multi = np.column_stack([y_lr, y_lr*2])
mor = MultiOutputRegressor(Ridge())
mor.fit(X_lr, y_multi)
print("Predictions for first 5 samples:\n", mor.predict(X_lr[:5]))
```

69. Use `GridSearchCV` with multiple scoring metrics

```
grid = GridSearchCV(LogisticRegression(max_iter=200), {'C':[0.1,1,10]}, cv=3,
 scoring=['accuracy','f1_macro'], refit='f1_macro')
grid.fit(X_train_scaled, y_train)
print("Best params:", grid.best_params_)
print("Best score (f1_macro):", grid.best_score_)
```

70. Fit `Pipeline` with imputer, scaler, PCA, and classifier

```
pipeline = Pipeline([
    ('imputer', SimpleImputer(strategy='mean')),
    ('scaler', StandardScaler()),
    ('pca', PCA(n_components=2)),
    ('clf', LogisticRegression(max_iter=200))
])
pipeline.fit(X_train, y_train)
print("Pipeline test accuracy:", pipeline.score(X_test, y_test))
```

71. Handle missing values using `IterativeImputer`

```
from sklearn.experimental import enable_iterative_imputer
from sklearn.impute import IterativeImputer

X_missing = X_train_scaled.copy()
X_missing[0,0] = np.nan
imputer = IterativeImputer(random_state=42)
X_imputed = imputer.fit_transform(X_missing)
print("First row after imputation:", X_imputed[0])
```

72. Fit `BaggingRegressor` on a regression dataset

```
from sklearn.ensemble import BaggingRegressor
bag_reg = BaggingRegressor(base_estimator=DecisionTreeRegressor(),
 n_estimators=10, random_state=42)
bag_reg.fit(X_lr, y_lr)
print("Predictions for first 5 samples:", bag_reg.predict(X_lr[:5]))
```

73. Fit `VotingClassifier` using hard and soft voting

```
from sklearn.ensemble import VotingClassifier
voting = VotingClassifier(
    estimators=[('lr', LogisticRegression(max_iter=200)),
('rf', RandomForestClassifier(n_estimators=50, random_state=42))],
    voting='soft'
)
voting.fit(X_train_scaled, y_train)
print("VotingClassifier test accuracy:", voting.score(X_test_scaled, y_test))
```

74. Apply time-series split for cross-validation

```
from sklearn.model_selection import TimeSeriesSplit
tscv = TimeSeriesSplit(n_splits=3)
for train_index, test_index in tscv.split(X_lr):
    print("Train indices:", train_index, "Test indices:", test_index)
```

75. Evaluate a regression model using `mean_absolute_percentage_error`

```
from sklearn.metrics import mean_absolute_percentage_error
y_pred = rf_reg.predict(X_lr)
mape = mean_absolute_percentage_error(y_lr, y_pred)
print("MAPE:", mape)
```

76. Implement custom transformer for preprocessing in pipeline

```
from sklearn.base import BaseEstimator, TransformerMixin

class LogTransformer(BaseEstimator, TransformerMixin):
    def fit(self, X, y=None):
        return self
    def transform(self, X):
        import numpy as np
        return np.log1p(X)  # log(1+x) transformation

pipeline = Pipeline([
    ('log', LogTransformer()),
    ('clf', LogisticRegression(max_iter=200))
])

# Using X_train_scaled for demonstration
pipeline.fit(X_train_scaled, y_train)
print("Pipeline with custom transformer test accuracy:",
pipeline.score(X_test_scaled, y_test))
```

Explanation: Custom transformer applies log transformation on all features.

77. Implement custom scoring function for model evaluation

```
from sklearn.metrics import make_scorer
```

```
def custom_score(y_true, y_pred):
    return (y_true == y_pred).mean() * 100  # accuracy in percentage

scorer = make_scorer(custom_score)
scores = cross_val_score(clf, X_train_scaled, y_train, cv=5, scoring=scorer)
print("Custom scoring (percentage accuracy) CV:", scores)
```

78. Apply stacking regressor with multiple base models

```
from sklearn.ensemble import StackingRegressor
from sklearn.linear_model import Ridge
from sklearn.tree import DecisionTreeRegressor
from sklearn.ensemble import RandomForestRegressor

stack_reg = StackingRegressor(
    estimators=[('ridge', Ridge()), ('dt', DecisionTreeRegressor())],
    final_estimator=RandomForestRegressor()
)
stack_reg.fit(X_lr, y_lr)
print("StackingRegressor predictions for first 5 samples:",
 stack_reg.predict(X_lr[:5]))
```

79. Apply feature importance analysis on Random Forest and plot top 10 features

```
import matplotlib.pyplot as plt
import pandas as pd

feat_importances = pd.Series(rf.feature_importances_, index=iris.feature_names)
top_features = feat_importances.sort_values(ascending=False)[:10]
top_features.plot(kind='barh', title='Top 10 Feature Importances')
plt.show()
```

80. Fit `CatBoostClassifier` and handle categorical features automatically

```
from catboost import CatBoostClassifier

# Example: Iris dataset has no categorical, just demonstration
cat = CatBoostClassifier(iterations=100, learning_rate=0.1, verbose=0,
random_state=42)
cat.fit(X_train, y_train)
print("CatBoostClassifier test accuracy:", cat.score(X_test, y_test))
```

81. Fit `LGBMRegressor` for regression task and evaluate RMSE

```
from lightgbm import LGBMRegressor
lgb_reg = LGBMRegressor(n_estimators=100, random_state=42)
lgb_reg.fit(X_lr, y_lr)
y_pred = lgb_reg.predict(X_lr)
rmse = mean_squared_error(y_lr, y_pred, squared=False)
print("LightGBM RMSE:", rmse)
```

82. Apply `FeatureUnion` to combine multiple feature sets

```
from sklearn.pipeline import FeatureUnion
from sklearn.preprocessing import PolynomialFeatures

union = FeatureUnion([
    ('poly', PolynomialFeatures(degree=2)),
    ('raw', 'passthrough')
])
X_combined = union.fit_transform(X_lr)
print("Shape after FeatureUnion:", X_combined.shape)
```

83. Handle high-cardinality categorical variables in a dataset

```
from sklearn.preprocessing import OneHotEncoder
import pandas as pd

# Example high-cardinality
df = pd.DataFrame({'color': ['Red','Blue','Green','Red','Blue','Green','Red']})
ohe = OneHotEncoder(handle_unknown='ignore')
X_encoded = ohe.fit_transform(df[['color']]).toarray()
print("Encoded shape:", X_encoded.shape)
```

84. Apply `Pipeline` with OneHotEncoder and KNNClassifier

```
pipeline = Pipeline([
    ('onehot', OneHotEncoder()),
    ('knn', KNeighborsClassifier(n_neighbors=3))
])
# Dummy small dataset
X_small = np.array([['Red'],['Blue'],['Green'],['Red']])
y_small = np.array([0,1,2,0])
pipeline.fit(X_small, y_small)
print("KNN predictions:", pipeline.predict(X_small))
```

85. Apply PCA on high-dimensional text embeddings (TF-IDF)

```
from sklearn.feature_extraction.text import TfidfVectorizer

texts = ["Machine learning is fun", "I love Python", "Deep learning and AI"]
tfidf = TfidfVectorizer()
X_text = tfidf.fit_transform(texts).toarray()
pca = PCA(n_components=2)
X_pca = pca.fit_transform(X_text)
print("PCA reduced shape:", X_pca.shape)
```

86. Fit `LogisticRegression` on imbalanced dataset using class_weight

```
clf = LogisticRegression(class_weight='balanced', max_iter=200)
y_imb = y.copy()
y_imb[y_imb==2] = 1
clf.fit(X_train_scaled, y_imb)
print("Test accuracy on imbalanced dataset:", clf.score(X_test_scaled, y_test))
```

87. Apply iterative hyperparameter tuning with `RandomizedSearchCV`

```
param_dist = {'C': np.logspace(-3,3,10)}
rand_search = RandomizedSearchCV(LogisticRegression(max_iter=200), param_dist,
n_iter=5, cv=3)
rand_search.fit(X_train_scaled, y_train)
print("Best hyperparameter C:", rand_search.best_params_)
```

88. Apply nested cross-validation for unbiased model evaluation

```
# Already demonstrated in 56, for clarity:
nested_scores = cross_val_score(rand_search, X_train_scaled, y_train, cv=5)
print("Nested CV scores:", nested_scores)
```

89. Evaluate models with `precision`, `recall`, and `f1-score`

```
from sklearn.metrics import precision_score, recall_score, f1_score

y_pred = clf.predict(X_test_scaled)
print("Precision:", precision_score(y_test, y_pred, average='macro'))
print("Recall:", recall_score(y_test, y_pred, average='macro'))
print("F1-score:", f1_score(y_test, y_pred, average='macro'))
```

90. Fit `SGDRegressor` on large datasets

```
from sklearn.linear_model import SGDRegressor
sgd_reg = SGDRegressor(max_iter=1000, tol=1e-3, random_state=42)
sgd_reg.fit(X_lr, y_lr)
print("SGDRegressor predictions:", sgd_reg.predict(X_lr[:5]))
```

91. Apply multi-class ROC curves

```
from sklearn.preprocessing import label_binarize
from sklearn.metrics import roc_curve, auc

y_bin = label_binarize(y_test, classes=[0,1,2])
y_score = clf.predict_proba(X_test_scaled)
for i in range(3):
    fpr, tpr, _ = roc_curve(y_bin[:,i], y_score[:,i])
    roc_auc = auc(fpr, tpr)
    print(f"Class {i} AUC:", roc_auc)
```

92. Perform clustering evaluation with Davies-Bouldin score

```
from sklearn.metrics import davies_bouldin_score
kmeans = KMeans(n_clusters=3, random_state=42)
labels = kmeans.fit_predict(X)
db_score = davies_bouldin_score(X, labels)
print("Davies-Bouldin score:", db_score)
```

93. Fit `ExtraTreesClassifier` and extract feature importance

```
from sklearn.ensemble import ExtraTreesClassifier
etc = ExtraTreesClassifier(n_estimators=100, random_state=42)
etc.fit(X_train_scaled, y_train)
feat_importances = pd.Series(etc.feature_importances_, index=iris.feature_names)
print("ExtraTrees feature importances:\n", feat_importances)
```

94. Apply `Pipeline` with scaling, feature selection, and classifier

```
pipeline = Pipeline([
    ('scaler', StandardScaler()),
    ('select', SelectKBest(k=2)),
    ('clf', RandomForestClassifier(n_estimators=50, random_state=42))
])
pipeline.fit(X_train, y_train)
print("Pipeline test accuracy:", pipeline.score(X_test, y_test))
```

95. Fit `MLPClassifier` (neural network) for classification

```
from sklearn.neural_network import MLPClassifier
mlp = MLPClassifier(hidden_layer_sizes=(50,), max_iter=500, random_state=42)
mlp.fit(X_train_scaled, y_train)
print("MLPClassifier test accuracy:", mlp.score(X_test_scaled, y_test))
```

96. Apply dimensionality reduction using `TruncatedSVD` for sparse data

```
from sklearn.decomposition import TruncatedSVD
svd = TruncatedSVD(n_components=2)
X_svd = svd.fit_transform(X_text)
print("TruncatedSVD shape:", X_svd.shape)
```

97. Fit `RidgeClassifierCV` and compare with Logistic Regression

```
from sklearn.linear_model import RidgeClassifierCV
ridge_cv = RidgeClassifierCV(alphas=[0.1,1.0,10.0])
ridge_cv.fit(X_train_scaled, y_train)
print("RidgeClassifierCV test accuracy:", ridge_cv.score(X_test_scaled, y_test))
print("LogisticRegression test accuracy:", clf.score(X_test_scaled, y_test))
```

98. Apply out-of-fold predictions for stacking ensemble

```
from sklearn.model_selection import cross_val_predict
oof_pred = cross_val_predict(stack, X_train_scaled, y_train, cv=5, method='predict')
print("Out-of-fold predictions:", oof_pred[:10])
```

99. Fit multiple regression models and select best using cross-validation

```
from sklearn.model_selection import cross_val_score
models = [Ridge(), Lasso(alpha=0.1), ElasticNet(alpha=0.1)]
```

```
for model in models:
    scores = cross_val_score(model, X_lr, y_lr, cv=5, scoring='r2')
    print(f"{model.__class__.__name__} mean R²:", scores.mean())
```

100. Implement end-to-end ML project pipeline

```
pipeline = Pipeline([
    ('imputer', SimpleImputer(strategy='mean')),
    ('scaler', StandardScaler()),
    ('pca', PCA(n_components=2)),
    ('clf', RandomForestClassifier(n_estimators=100, random_state=42))
])
pipeline.fit(X_train, y_train)
y_pred = pipeline.predict(X_test)
from sklearn.metrics import classification_report
print(classification_report(y_test, y_pred))
```

Explanation: This is an **industry-level pipeline**, including missing value imputation, scaling, dimensionality reduction, model fitting, and evaluation.

101. Create a simple line plot

```
import matplotlib.pyplot as plt

x = [1, 2, 3, 4, 5]
y = [2, 3, 5, 7, 11]

plt.plot(x, y)
plt.xlabel("X-axis")
plt.ylabel("Y-axis")
plt.title("Simple Line Plot")
plt.show()
```

Explanation: Basic line plot with axis labels and title.

102. Plot multiple lines on the same graph

```
y2 = [1, 4, 6, 8, 10]

plt.plot(x, y, label='Line 1')
plt.plot(x, y2, label='Line 2', linestyle='--', color='red')
plt.xlabel("X-axis")
plt.ylabel("Y-axis")
plt.title("Multiple Lines")
plt.legend()
plt.show()
```

Explanation: Multiple lines with labels, colors, and legends.

103. Create a bar chart

```
categories = ['A', 'B', 'C', 'D']
values = [10, 15, 7, 12]

plt.bar(categories, values, color='skyblue')
plt.xlabel("Categories")
plt.ylabel("Values")
plt.title("Bar Chart")
plt.show()
```

104. Create a horizontal bar chart

```
plt.barh(categories, values, color='green')
plt.xlabel("Values")
plt.ylabel("Categories")
plt.title("Horizontal Bar Chart")
plt.show()
```

105. Create a stacked bar chart

```
values2 = [5, 7, 3, 8]
plt.bar(categories, values, label='Set 1')
plt.bar(categories, values2, bottom=values, label='Set 2', color='orange')
plt.title("Stacked Bar Chart")
plt.legend()
plt.show()
```

106. Create a scatter plot

```
x = [5, 7, 8, 7, 2, 17]
y = [99, 86, 87, 88, 100, 86]

plt.scatter(x, y, color='red', marker='o')
plt.xlabel("X-axis")
plt.ylabel("Y-axis")
plt.title("Scatter Plot")
plt.show()
```

107. Customize marker style and size in scatter plot

```
plt.scatter(x, y, color='blue', marker='^', s=100)  # s=size
plt.xlabel("X-axis")
plt.ylabel("Y-axis")
plt.title("Customized Scatter Plot")
plt.show()
```

108. Create a histogram

```
data = [22, 87, 5, 43, 56, 73, 55, 54, 11, 20, 51, 5]
plt.hist(data, bins=5, color='purple', edgecolor='black')
plt.xlabel("Bins")
plt.ylabel("Frequency")
plt.title("Histogram")
plt.show()
```

109. Create a pie chart

```
sizes = [15, 30, 45, 10]
labels = ['A', 'B', 'C', 'D']
plt.pie(sizes, labels=labels, autopct='%1.1f%%', startangle=90)
plt.title("Pie Chart")
plt.show()
```

110. Create an exploded pie chart

```
explode = (0, 0.1, 0, 0)  # explode second slice
plt.pie(sizes, labels=labels, autopct='%1.1f%%', startangle=90, explode=explode)
plt.title("Exploded Pie Chart")
plt.show()
```

111. Plot sine and cosine waves

```
import numpy as np

x = np.linspace(0, 2*np.pi, 100)
y_sin = np.sin(x)
y_cos = np.cos(x)

plt.plot(x, y_sin, label='Sine')
plt.plot(x, y_cos, label='Cosine')
plt.title("Sine and Cosine Waves")
plt.xlabel("Angle [rad]")
plt.ylabel("Value")
plt.legend()
plt.show()
```

112. Add grid lines to a plot

```
plt.plot(x, y_sin, color='green')
plt.grid(True)
plt.title("Plot with Grid")
plt.show()
```

113. Add annotations to a plot

```
plt.plot(x, y_sin, label='Sine')
plt.annotate('Peak', xy=(np.pi/2,1), xytext=(2, 0.5),
             arrowprops=dict(facecolor='black', shrink=0.05))
plt.title("Annotated Plot")
plt.show()
```

114. Subplots: 2 plots in one figure (1 row, 2 columns)

```
plt.subplot(1, 2, 1)
plt.plot(x, y_sin, color='blue')
plt.title("Sine")
plt.subplot(1, 2, 2)
plt.plot(x, y_cos, color='red')
plt.title("Cosine")

plt.tight_layout()
plt.show()
```

115. Subplots: 2 rows, 1 column

```
plt.subplot(2, 1, 1)
plt.plot(x, y_sin)
plt.title("Sine Wave")

plt.subplot(2, 1, 2)
plt.plot(x, y_cos)
plt.title("Cosine Wave")

plt.tight_layout()
plt.show()
```

116. Change figure size

```
plt.figure(figsize=(10,5))
plt.plot(x, y_sin)
plt.title("Large Figure")
plt.show()
```

117. Customize line style, color, and width

```
plt.plot(x, y_sin, color='red', linestyle='--', linewidth=3)
plt.title("Custom Line Style")
plt.show()
```

118. Add multiple legends for multiple lines

```
plt.plot(x, y_sin, label='Sine', color='blue')
plt.plot(x, y_cos, label='Cosine', color='green')
plt.title("Multiple Legends")
plt.legend(loc='upper right')
plt.show()
```

119. Plot bar chart with error bars

```
values = [10, 15, 7, 12]
errors = [1, 2, 1, 2]
plt.bar(categories, values, yerr=errors, capsize=5, color='orange')
plt.title("Bar Chart with Error Bars")
plt.show()
```

120. Save a figure to file

```
plt.plot(x, y_sin)
plt.title("Save Figure Example")
plt.savefig("sine_plot.png", dpi=300)
plt.show()
```

121. Plot multiple subplots with shared axes

```
fig, (ax1, ax2) = plt.subplots(2, 1, sharex=True)
ax1.plot(x, y_sin, color='blue')
ax1.set_title("Sine Wave")
ax2.plot(x, y_cos, color='green')
ax2.set_title("Cosine Wave")
plt.xlabel("Angle [rad]")
plt.tight_layout()
plt.show()
```

Explanation: Shares x-axis between subplots for better comparison.

122. Add a secondary y-axis

```
fig, ax1 = plt.subplots()
ax2 = ax1.twinx()  # create second y-axis
ax1.plot(x, y_sin, color='blue', label='Sine')
ax2.plot(x, y_cos, color='red', label='Cosine')
ax1.set_ylabel("Sine")
ax2.set_ylabel("Cosine")
plt.title("Dual Y-axis Plot")
plt.show()
```

123. Customize tick marks and labels

```
plt.plot(x, y_sin)
plt.xticks([0, np.pi/2, np.pi, 3*np.pi/2, 2*np.pi], ['0','π/2','π','3π/2','2π'])
plt.yticks([-1, 0, 1])
plt.title("Custom Tick Marks")
plt.show()
```

124. Use logarithmic scale on axes

```
x_log = np.logspace(0.1, 2, 100)
y_log = x_log ** 2
plt.plot(x_log, y_log)
plt.xscale('log')
plt.yscale('log')
plt.title("Log-Log Scale Plot")
plt.show()
```

125. Plot a stacked area chart

```
y1 = [3, 4, 5, 6, 7]
y2 = [1, 2, 3, 4, 5]
plt.stackplot(x, y1, y2, colors=['skyblue','orange'], labels=['Y1','Y2'])
plt.legend(loc='upper left')
plt.title("Stacked Area Chart")
plt.show()
```

126. Plot error bars in a line plot

```
y = [2, 3, 5, 7, 11]
y_err = [0.5, 0.4, 0.3, 0.6, 0.2]
plt.errorbar(x, y, yerr=y_err, fmt='-o', capsize=5, color='green')
plt.title("Line Plot with Error Bars")
plt.show()
```

127. Fill area under the curve

```
plt.plot(x, y_sin, color='blue')
plt.fill_between(x, y_sin, color='skyblue', alpha=0.4)
plt.title("Filled Area Under Curve")
plt.show()
```

128. Create a polar plot

```
theta = np.linspace(0, 2*np.pi, 100)
r = np.abs(np.sin(2*theta) * np.cos(2*theta))
plt.polar(theta, r, color='red')
plt.title("Polar Plot")
plt.show()
```

129. Plot multiple lines with different markers and styles

```
plt.plot(x, y_sin, marker='o', linestyle='-', color='blue', label='Sine')
plt.plot(x, y_cos, marker='s', linestyle='--', color='red', label='Cosine')
plt.title("Multiple Lines with Markers")
plt.legend()
plt.show()
```

130. Plot horizontal and vertical lines

```
plt.plot(x, y_sin)
plt.axhline(y=0, color='black', linestyle='--')
plt.axvline(x=np.pi, color='red', linestyle='--')
plt.title("Horizontal and Vertical Lines")
plt.show()
```

131. Plot a stem plot

```
plt.stem(x, y_sin, linefmt='green', markerfmt='ro', basefmt='blue')
plt.title("Stem Plot")
plt.show()
```

132. Plot a step plot

```
plt.step(x, y_sin, where='mid', color='purple')
plt.title("Step Plot")
plt.show()
```

133. Plot a bar chart with gradient color

```
colors = plt.cm.viridis(np.linspace(0,1,len(values)))
plt.bar(categories, values, color=colors)
plt.title("Bar Chart with Gradient")
plt.show()
```

134. Create twin axes with different scales

```
fig, ax1 = plt.subplots()
ax2 = ax1.twinx()
ax1.plot(x, y_sin, 'g-', label='Sine')
ax2.plot(x, y_cos*50, 'b--', label='Cosine x50')
ax1.set_ylabel('Sine')
ax2.set_ylabel('Cosine x50')
plt.title("Twin Axes with Different Scales")
plt.show()
```

135. Plot a histogram with density curve

```
import seaborn as sns
data = np.random.normal(0, 1, 1000)
plt.hist(data, bins=30, density=True, alpha=0.5, color='orange')
sns.kdeplot(data, color='blue')
plt.title("Histogram with Density Curve")
plt.show()
```

136. Add text annotation at multiple points

```
plt.plot(x, y_sin)
for i, j in zip(x, y_sin):
    plt.text(i, j, f'({i},{j:.2f})')
plt.title("Multiple Annotations")
plt.show()
```

137. Create figure with multiple subplots (2x2)

```
fig, axes = plt.subplots(2, 2, figsize=(10,6))
axes[0,0].plot(x, y_sin, color='blue'); axes[0,0].set_title('Sine')
axes[0,1].plot(x, y_cos, color='red'); axes[0,1].set_title('Cosine')
axes[1,0].bar(categories, values, color='green'); axes[1,0].set_title('Bar')
axes[1,1].scatter(x, y_sin, color='purple'); axes[1,1].set_title('Scatter')
plt.tight_layout()
plt.show()
```

138. Create a contour plot

```
X, Y = np.meshgrid(np.linspace(-3,3,100), np.linspace(-3,3,100))
Z = np.sin(np.sqrt(X**2 + Y**2))
plt.contour(X, Y, Z, cmap='viridis')
plt.title("Contour Plot")
plt.show()
```

139. Create a filled contour plot

```
plt.contourf(X, Y, Z, cmap='plasma')
plt.colorbar()
plt.title("Filled Contour Plot")
plt.show()
```

140. Customize figure background and axes color

```
fig, ax = plt.subplots()
ax.plot(x, y_sin, color='yellow')
fig.patch.set_facecolor('black')
ax.set_facecolor('gray')
ax.set_title("Custom Figure and Axes Colors", color='white')
plt.show()
```

141. 3D Line Plot

```
from mpl_toolkits.mplot3d import Axes3D

fig = plt.figure()
ax = fig.add_subplot(111, projection='3d')
z = np.linspace(0, 1, 100)
x = z * np.sin(25*z)
y = z * np.cos(25*z)
ax.plot(x, y, z, label='3D Line')
ax.set_title("3D Line Plot")
plt.show()
```

Explanation: Creates a 3D line in space using `Axes3D`.

142. 3D Scatter Plot

```
fig = plt.figure()
ax = fig.add_subplot(111, projection='3d')
ax.scatter(x, y, z, c=z, cmap='viridis')
ax.set_title("3D Scatter Plot")
plt.show()
```

143. 3D Surface Plot

```
X, Y = np.meshgrid(np.linspace(-3,3,50), np.linspace(-3,3,50))
```

```
Z = np.sin(np.sqrt(X**2 + Y**2))
fig = plt.figure()
ax = fig.add_subplot(111, projection='3d')
surf = ax.plot_surface(X, Y, Z, cmap='plasma')
fig.colorbar(surf)
ax.set_title("3D Surface Plot")
plt.show()
```

144. 3D Wireframe Plot

```
fig = plt.figure()
ax = fig.add_subplot(111, projection='3d')
ax.plot_wireframe(X, Y, Z, color='green')
ax.set_title("3D Wireframe Plot")
plt.show()
```

145. 3D Contour Plot

```
fig = plt.figure()
ax = fig.add_subplot(111, projection='3d')
ax.contour3D(X, Y, Z, 50, cmap='coolwarm')
ax.set_title("3D Contour Plot")
plt.show()
```

146. Animated line plot (basic animation)

```
from matplotlib.animation import FuncAnimation

fig, ax = plt.subplots()
line, = ax.plot([], [], lw=2)
ax.set_xlim(0, 2*np.pi)
ax.set_ylim(-1, 1)
x = np.linspace(0, 2*np.pi, 100)
y = np.sin(x)

def animate(i):
    line.set_data(x[:i], y[:i])
    return line,

ani = FuncAnimation(fig, animate, frames=len(x), interval=50, blit=True)
plt.show()
```

Explanation: Animates a sine wave drawing over time.

147. Custom colormap for scatter plot

```
colors = np.linspace(0,1,len(x))
plt.scatter(x, y_sin, c=colors, cmap='plasma', s=100)
plt.colorbar(label='Value')
plt.title("Scatter Plot with Custom Colormap")
plt.show()
```

148. Mixed plots in one figure (line + scatter + bar)

```
fig, ax = plt.subplots()
ax.plot(x, y_sin, color='blue', label='Line')
ax.scatter(x, y_cos, color='red', label='Scatter')
ax.bar(categories, values, color='green', alpha=0.3, label='Bar')
ax.set_title("Mixed Plots")
ax.legend()
plt.show()
```

149. Customize legends outside plot area

```
plt.plot(x, y_sin, label='Sine')
plt.plot(x, y_cos, label='Cosine')
plt.title("Legend Outside")
plt.legend(loc='center left', bbox_to_anchor=(1, 0.5))
plt.tight_layout()
plt.show()
```

150. Heatmap using `imshow` with annotations

```
matrix = np.random.rand(5,5)
plt.imshow(matrix, cmap='viridis', interpolation='nearest')
for i in range(5):
    for j in range(5):
        plt.text(j, i, f"{matrix[i,j]:.2f}", ha='center', va='center',
color='white')
plt.title("Heatmap with Annotations")
plt.colorbar()
plt.show()
```

151. Simple scatter plot with Seaborn

```
import seaborn as sns
import matplotlib.pyplot as plt

# Load dataset
tips = sns.load_dataset('tips')

sns.scatterplot(x='total_bill', y='tip', data=tips)
plt.title("Scatter Plot: Total Bill vs Tip")
plt.show()
```

Explanation: Basic scatter plot using Seaborn with `tips` dataset.

152. Scatter plot with hue for categories

```
sns.scatterplot(x='total_bill', y='tip', hue='smoker', data=tips)
plt.title("Scatter Plot with Hue")
plt.show()
```

Explanation: Different colors for smoker vs non-smoker.

153. Scatter plot with size and style

```
sns.scatterplot(x='total_bill', y='tip', hue='sex', style='smoker',
size='size', data=tips)
plt.title("Scatter Plot with Size and Style")
plt.show()
```

154. Line plot with Seaborn

```
sns.lineplot(x='total_bill', y='tip', data=tips)
plt.title("Line Plot: Total Bill vs Tip")
plt.show()
```

155. Line plot with multiple lines using hue

```
sns.lineplot(x='size', y='tip', hue='sex', data=tips)
plt.title("Line Plot by Sex")
plt.show()
```

156. Histogram using Seaborn

```
sns.histplot(tips['total_bill'], bins=20, kde=True, color='green')
plt.title("Histogram of Total Bill with KDE")
plt.show()
```

157. Distribution plot using `displot`

```
sns.displot(tips['tip'], kde=True, bins=15, color='orange')
plt.title("Distribution of Tip")
plt.show()
```

158. Box plot to show quartiles

```
sns.boxplot(x='day', y='total_bill', data=tips)
plt.title("Box Plot: Total Bill by Day")
plt.show()
```

159. Box plot with hue

```
sns.boxplot(x='day', y='total_bill', hue='smoker', data=tips)
plt.title("Box Plot by Day and Smoker")
plt.show()
```

160. Violin plot to show distribution

```
sns.violinplot(x='day', y='total_bill', data=tips)
plt.title("Violin Plot: Total Bill by Day")
plt.show()
```

161. Violin plot with hue and split

```
sns.violinplot(x='day', y='total_bill', hue='sex', split=True, data=tips)
plt.title("Violin Plot Split by Sex")
plt.show()
```

162. Count plot to show frequency

```
sns.countplot(x='day', data=tips, palette='Set2')
plt.title("Count of Records by Day")
plt.show()
```

163. Bar plot with confidence intervals

```
sns.barplot(x='day', y='total_bill', data=tips, ci=95)
plt.title("Bar Plot with CI")
plt.show()
```

164. Bar plot with hue

```
sns.barplot(x='day', y='total_bill', hue='sex', data=tips, palette='cool')
plt.title("Bar Plot by Sex")
plt.show()
```

165. Strip plot (jittered scatter plot)

```
sns.stripplot(x='day', y='total_bill', data=tips, jitter=True, color='red')
plt.title("Strip Plot with Jitter")
plt.show()
```

166. Swarm plot to avoid overlapping points

```
sns.swarmplot(x='day', y='total_bill', hue='sex', data=tips)
plt.title("Swarm Plot by Sex")
plt.show()
```

167. Pair plot to show pairwise relationships

```
sns.pairplot(tips, hue='sex')
plt.suptitle("Pair Plot by Sex", y=1.02)
plt.show()
```

168. Correlation heatmap

```
corr = tips.corr()
sns.heatmap(corr, annot=True, cmap='coolwarm')
plt.title("Correlation Heatmap")
plt.show()
```

169. Joint plot with scatter and regression line

```
sns.jointplot(x='total_bill', y='tip', data=tips, kind='reg', height=6)
plt.suptitle("Joint Plot with Regression", y=1.02)
plt.show()
```

170. Facet grid to plot multiple subsets

```
g = sns.FacetGrid(tips, col='time', row='smoker')
g.map(sns.scatterplot, 'total_bill', 'tip')
g.add_legend()
plt.suptitle("Facet Grid: Time and Smoker", y=1.05)
plt.show()
```

171. Advanced FacetGrid with multiple variables

```
g = sns.FacetGrid(tips, col='time', row='smoker', hue='sex', margin_titles=True)
g.map(sns.scatterplot, 'total_bill', 'tip')
g.add_legend()
plt.suptitle("Advanced FacetGrid: Multi-variable Analysis", y=1.05)
plt.show()
```

Explanation: Shows relationship across multiple categorical variables.

172. Pair plot with KDE diagonal and hue

```
sns.pairplot(tips, hue='sex', diag_kind='kde', palette='coolwarm')
plt.suptitle("Pair Plot with KDE Diagonal", y=1.02)
plt.show()
```

173. Heatmap with hierarchical clustering

```
flights = sns.load_dataset('flights')
flights_pivot = flights.pivot('month','year','passengers')
sns.clustermap(flights_pivot, cmap='YlGnBu', standard_scale=1)
plt.title("Clustered Heatmap of Flights", pad=100)
plt.show()
```

Explanation: Clustered heatmap for detecting seasonal patterns.

174. Regression plot with confidence interval

```
sns.regplot(x='total_bill', y='tip', data=tips, ci=95)
plt.title("Regression Plot with 95% CI")
plt.show()
```

175. Residual plot to check model fit

```
sns.residplot(x='total_bill', y='tip', data=tips)
plt.title("Residual Plot")
plt.show()
```

176. Polynomial regression using `order` parameter

```
sns.regplot(x='total_bill', y='tip', data=tips, order=2)
plt.title("Polynomial Regression (Order 2)")
plt.show()
```

177. Lowess smoothing for scatter plot

```
sns.regplot(x='total_bill', y='tip', data=tips, lowess=True, line_kws={'color':'red'})
plt.title("LOWESS Smoothing")
plt.show()
```

178. Categorical scatter plot with swarm and violin combined

```
sns.violinplot(x='day', y='total_bill', data=tips, inner=None, color='lightgray')
sns.swarmplot(x='day', y='total_bill', data=tips, hue='sex', dodge=True)
plt.title("Violin + Swarm Plot")
plt.show()
```

179. Boxen plot for large datasets

```
sns.boxenplot(x='day', y='total_bill', data=tips, hue='sex')
plt.title("Boxen Plot for Large Dataset")
plt.show()
```

180. KDE plot with multiple distributions

```
sns.kdeplot(tips[tips['sex']=='Male']['total_bill'], shade=True, label='Male')
sns.kdeplot(tips[tips['sex']=='Female']['total_bill'], shade=True, label='Female')
plt.title("KDE Plot by Sex")
plt.legend()
plt.show()
```

181. Joint KDE plot for two variables

```
sns.jointplot(x='total_bill', y='tip', data=tips, kind='kde', fill=True, cmap='Reds')
```

```
plt.suptitle("Joint KDE Plot", y=1.02)
plt.show()
```

182. Pair grid with different plots on upper and lower triangles

```
g = sns.PairGrid(tips, hue='sex')
g.map_upper(sns.scatterplot)
g.map_lower(sns.kdeplot)
g.map_diag(sns.histplot, kde=True)
g.add_legend()
plt.suptitle("PairGrid with Different Plots", y=1.02)
plt.show()
```

183. FacetGrid with histogram on multiple columns

```
g = sns.FacetGrid(tips, col='day', hue='sex')
g.map(sns.histplot, 'total_bill', bins=10, alpha=0.6)
g.add_legend()
plt.suptitle("FacetGrid Histograms by Day and Sex", y=1.05)
plt.show()
```

184. Visualizing correlation with `pairplot` and regression lines

```
sns.pairplot(tips, kind='reg', hue='smoker')
plt.suptitle("Pairplot with Regression by Smoker", y=1.02)
plt.show()
```

185. Multi-level categorical bar plot using `catplot`

```
sns.catplot(x='day', y='total_bill', hue='sex', col='time', kind='bar', data=tips)
plt.suptitle("Categorical Bar Plot by Time, Day, and Sex", y=1.05)
plt.show()
```

186. Box plot with log scale

```
sns.boxplot(x='day', y='total_bill', data=tips)
plt.yscale('log')
plt.title("Box Plot with Log Scale")
plt.show()
```

187. Heatmap with annotations and custom color palette

```
corr = tips.corr()
sns.heatmap(corr, annot=True, cmap='coolwarm', linewidths=0.5, linecolor='black')
plt.title("Annotated Correlation Heatmap")
plt.show()
```

188. Pairplot with categorical markers

```
sns.pairplot(tips, hue='sex', markers=["o","s"])
```

```
plt.suptitle("Pairplot with Custom Markers", y=1.02)
plt.show()
```

189. Overlay multiple KDEs on the same plot

```
sns.kdeplot(data=tips, x='total_bill', hue='smoker', multiple='stack')
plt.title("Stacked KDE Plot by Smoker")
plt.show()
```

190. Relational plot combining scatter and line plots

```
sns.relplot(x='total_bill', y='tip', hue='sex', kind='line', data=tips)
plt.title("Relational Line Plot by Sex")
plt.show()
```

191. PairGrid with scatter and KDE with custom palette

```
g = sns.PairGrid(tips, hue='sex', palette='Set2')
g.map_upper(sns.scatterplot)
g.map_lower(sns.kdeplot, fill=True)
g.map_diag(sns.kdeplot, fill=True)
g.add_legend()
plt.suptitle("Advanced PairGrid", y=1.02)
plt.show()
```

192. Swarm plot with dodging and hue palette

```
sns.swarmplot(x='day', y='total_bill', hue='sex', dodge=True, palette='Set1',
 data=tips)
plt.title("Swarm Plot with Hue and Dodge")
plt.show()
```

193. FacetGrid with KDE for multiple categories

```
g = sns.FacetGrid(tips, col='sex', row='smoker', margin_titles=True)
g.map(sns.kdeplot, 'total_bill', fill=True)
plt.suptitle("FacetGrid KDE by Sex and Smoker", y=1.05)
plt.show()
```

194. Categorical scatter with jitter and hue palette

```
sns.stripplot(x='day', y='tip', hue='sex', data=tips, jitter=True, dodge=True,
palette='Set2')
plt.title("Strip Plot with Hue and Jitter")
plt.show()
```

195. Box plot with split violins for comparison

```
sns.violinplot(x='day', y='total_bill', hue='sex', split=True, data=tips)
plt.title("Split Violin Plot")
plt.show()
```

196. Bar plot with nested categories using catplot

```
sns.catplot(x='day', y='total_bill', hue='sex', col='time', kind='bar', data=tips)
plt.suptitle("Nested Bar Plot by Time, Day, and Sex", y=1.05)
plt.show()
```

197. Joint plot with hexbin style for density visualization

```
sns.jointplot(x='total_bill', y='tip', data=tips, kind='hex', cmap='coolwarm')
plt.suptitle("Hexbin Joint Plot", y=1.02)
plt.show()
```

198. Heatmap with masked upper triangle for cleaner visualization

```
mask = np.triu(np.ones_like(corr, dtype=bool))
sns.heatmap(corr, mask=mask, annot=True, cmap='RdBu', vmin=-1, vmax=1)
plt.title("Masked Heatmap for Correlation")
plt.show()
```

199. Regression plot with categorical split using `hue` and `markers`

```
sns.lmplot(x='total_bill', y='tip', hue='sex', data=tips, markers=["o","s"], ci=None)
plt.title("Regression Plot by Sex")
plt.show()
```

200. End-to-end project-level visualization: multiple Seaborn plots combined

```
fig, axes = plt.subplots(2, 2, figsize=(12,10))

sns.scatterplot(x='total_bill', y='tip', hue='sex', data=tips, ax=axes[0,0])
axes[0,0].set_title("Scatter Plot")

sns.boxplot(x='day', y='total_bill', hue='smoker', data=tips, ax=axes[0,1])
axes[0,1].set_title("Box Plot")

sns.violinplot(x='day', y='tip', hue='sex', split=True, data=tips, ax=axes[1,0])
axes[1,0].set_title("Violin Plot")

sns.heatmap(tips.corr(), annot=True, cmap='coolwarm', ax=axes[1,1])
axes[1,1].set_title("Correlation Heatmap")

plt.tight_layout()
plt.show()
```

Explanation: Combines scatter, box, violin, and heatmap in a single figure –

industry-level dashboard style visualization.

UNIT-16

1. Connect to SQLite and create a new database

```
import sqlite3

conn = sqlite3.connect('school.db')
print("Database created and connected successfully!")
conn.close()
```

Output:

```
Database created and connected successfully!
```

2. Create a table (id, name, age, email)

```
import sqlite3

conn = sqlite3.connect('school.db')
cursor = conn.cursor()
cursor.execute('''CREATE TABLE IF NOT EXISTS students (
                    id INTEGER PRIMARY KEY,
                    name TEXT,
                    age INTEGER,
                    email TEXT)''')
conn.commit()
print("Table created successfully!")
conn.close()
```

Output:

```
Table created successfully!
```

3. Insert a single row

```
import sqlite3

conn = sqlite3.connect('school.db')
cursor = conn.cursor()
cursor.execute("INSERT INTO students (name, age, email) VALUES (?, ?, ?)",
               ("John Doe", 21, "john@example.com"))
conn.commit()
print("Record inserted successfully!")
conn.close()
```

4. Insert multiple rows using `executemany()`

```
import sqlite3

data = [
    ("Alice", 22, "alice@example.com"),
    ("Bob", 24, "bob@example.com"),
    ("Charlie", 23, "charlie@example.com")
]

conn = sqlite3.connect('school.db')
cursor = conn.cursor()
cursor.executemany("INSERT INTO students (name, age, email) VALUES (?, ?, ?)", data)
conn.commit()
print("Multiple records inserted successfully!")
conn.close()
```

5. Fetch all rows

```
import sqlite3

conn = sqlite3.connect('school.db')
cursor = conn.cursor()
cursor.execute("SELECT * FROM students")
rows = cursor.fetchall()
for row in rows:
    print(row)
conn.close()
```

Output:

```
(1, 'John Doe', 21, 'john@example.com')
(2, 'Alice', 22, 'alice@example.com')
...
```

6. Fetch a single row with a condition

```
import sqlite3

conn = sqlite3.connect('school.db')
cursor = conn.cursor()
cursor.execute("SELECT * FROM students WHERE id = ?", (2,))
row = cursor.fetchone()
print(row)
conn.close()
```

7. Update a record

```
import sqlite3

conn = sqlite3.connect('school.db')
cursor = conn.cursor()
cursor.execute("UPDATE students SET name = ? WHERE id = ?", ("Alicia", 2))
conn.commit()
print("Record updated successfully!")
conn.close()
```

8. Delete a record

```
import sqlite3

conn = sqlite3.connect('school.db')
cursor = conn.cursor()
cursor.execute("DELETE FROM students WHERE id = ?", (3,))
conn.commit()
print("Record deleted successfully!")
conn.close()
```

9. Drop table if exists

```
import sqlite3

conn = sqlite3.connect('school.db')
cursor = conn.cursor()
cursor.execute("DROP TABLE IF EXISTS temp_table")
conn.commit()
print("Table dropped successfully!")
conn.close()
```

10. Count total number of records

```
import sqlite3

conn = sqlite3.connect('school.db')
cursor = conn.cursor()
cursor.execute("SELECT COUNT(*) FROM students")
count = cursor.fetchone()[0]
print("Total records:", count)
conn.close()
```

11. Select rows where age > 25

```
conn = sqlite3.connect('school.db')
cursor = conn.cursor()
cursor.execute("SELECT * FROM students WHERE age > 25")
print(cursor.fetchall())
conn.close()
```

12. Sort records by name (ascending)

```
conn = sqlite3.connect('school.db')
cursor = conn.cursor()
cursor.execute("SELECT * FROM students ORDER BY name ASC")
print(cursor.fetchall())
conn.close()
```

13. Use LIKE operator

```
conn = sqlite3.connect('school.db')
cursor = conn.cursor()
cursor.execute("SELECT * FROM students WHERE name LIKE 'A%'")
print(cursor.fetchall())
conn.close()
```

14. Fetch MAX, MIN, and AVG age

```
conn = sqlite3.connect('school.db')
cursor = conn.cursor()
cursor.execute("SELECT MAX(age), MIN(age), AVG(age) FROM students")
print(cursor.fetchone())
conn.close()
```

15. Parameterized query

```
name = input("Enter name: ")
conn = sqlite3.connect('school.db')
cursor = conn.cursor()
cursor.execute("SELECT * FROM students WHERE name = ?", (name,))
print(cursor.fetchall())
conn.close()
```

16. Join two tables

```
conn = sqlite3.connect('school.db')
cursor = conn.cursor()

cursor.execute('''CREATE TABLE IF NOT EXISTS courses (
                    id INTEGER PRIMARY KEY,
                    student_id INTEGER,
                    course_name TEXT)''')
```

```
cursor.executemany("INSERT INTO courses (student_id, course_name) VALUES (?, ?)",
                   [(1, 'Python'), (2, 'SQL'), (1, 'AI')])

cursor.execute('''SELECT students.name, courses.course_name
                  FROM students JOIN courses
                  ON students.id = courses.student_id''')
print(cursor.fetchall())
conn.close()
```

17. Fetch distinct values

```
conn = sqlite3.connect('school.db')
cursor = conn.cursor()
cursor.execute("SELECT DISTINCT age FROM students")
print(cursor.fetchall())
conn.close()
```

18. Search dynamically using user input

```
search_email = input("Enter email to search: ")
conn = sqlite3.connect('school.db')
cursor = conn.cursor()
cursor.execute("SELECT * FROM students WHERE email = ?", (search_email,))
print(cursor.fetchall())
conn.close()
```

19. Implement transaction (commit & rollback)

```
import sqlite3

conn = sqlite3.connect('school.db')
cursor = conn.cursor()
try:
    cursor.execute("UPDATE students SET age = age + 1")
    conn.commit()
    print("Transaction committed!")
except:
    conn.rollback()
    print("Transaction rolled back!")
finally:
    conn.close()
```

20. Fetch records in chunks (pagination)

```
conn = sqlite3.connect('school.db')
cursor = conn.cursor()
cursor.execute("SELECT * FROM students")
while True:
    rows = cursor.fetchmany(2)
    if not rows:
        break
    print(rows)
conn.close()
```

21. Export data to CSV

```
import sqlite3, csv

conn = sqlite3.connect('school.db')
cursor = conn.cursor()
cursor.execute("SELECT * FROM students")
rows = cursor.fetchall()
```

```
with open('students_export.csv', 'w', newline='') as f:
    writer = csv.writer(f)
    writer.writerow(['ID', 'Name', 'Age', 'Email'])
    writer.writerows(rows)

print("Data exported to students_export.csv")
conn.close()
```

22. Import CSV data into table

```
import sqlite3, csv

conn = sqlite3.connect('school.db')
cursor = conn.cursor()

with open('students_export.csv', 'r') as f:
    reader = csv.reader(f)
    next(reader)
    cursor.executemany("INSERT INTO students (id, name, age, email) VALUES (?, ?, ?,

conn.commit()
print("Data imported successfully!")
conn.close()
```

23. Create a view and fetch data

```
conn = sqlite3.connect('school.db')
cursor = conn.cursor()
cursor.execute("CREATE VIEW IF NOT EXISTS student_view AS SELECT name, email FROM stu
cursor.execute("SELECT * FROM student_view")
print(cursor.fetchall())
conn.close()
```

24. Handle database exceptions

```
import sqlite3

try:
    conn = sqlite3.connect('school.db')
    cursor = conn.cursor()
    cursor.execute("SELECT * FROM nonexistent_table")
except sqlite3.Error as e:
    print("Database error:", e)
finally:
    conn.close()
```

25. Connect to MySQL/PostgreSQL (example)

```
# For MySQL
import mysql.connector

try:
    conn = mysql.connector.connect(
        host='localhost',
        user='root',
        password='password',
        database='testdb'
    )
    cursor = conn.cursor()
    cursor.execute("SELECT DATABASE()")
    print("Connected to:", cursor.fetchone())
finally:
    cursor.close()
```

```
conn.close()
```

Students now we are going to learn basic applications on python. For that here are some basic tutorial on some topics along with basics to advanced programs with proper explanation.

Topic 1: Tkinter

UNIT 1 – Introduction to Tkinter

What is Tkinter?

Tkinter is Python's standard library for creating GUI (windows, buttons, text boxes, etc.).
It comes pre-installed with Python — no need to install separately.

Create Your First Window

```
import tkinter as tk

# Create main window
root = tk.Tk()

# Set window title
root.title("My First Tkinter App")

# Set window size
root.geometry("400x300")

# Run the GUI loop
root.mainloop()
```

Explanation:

- `Tk()` → Creates the main application window.
- `title()` → Sets window title.
- `geometry()` → Defines size (width x height).
- `mainloop()` → Keeps window open until you close it.

UNIT 2 – Basic Widgets

1.Label

```
import tkinter as tk

root = tk.Tk()
root.title("Label Example")

label = tk.Label(root, text="Hello, Tkinter!", font=("Arial", 18))
label.pack(pady=20)

root.mainloop()
```

2.Button

```
import tkinter as tk

def say_hello():
    print("Hello Button Clicked!")

root = tk.Tk()
root.title("Button Example")

button = tk.Button(root, text="Click Me", command=say_hello)
button.pack(pady=20)

root.mainloop()
```

3.Entry (Text Input)

```
import tkinter as tk

def show_text():
    print(entry.get())

root = tk.Tk()
root.title("Entry Example")

entry = tk.Entry(root, width=30)
entry.pack(pady=10)

button = tk.Button(root, text="Get Text", command=show_text)
button.pack(pady=10)

root.mainloop()
```

UNIT 3 – Layout Managers

There are 3 layout systems in Tkinter:

Layout	Description
`.pack()`	Places widgets automatically
`.grid()`	Places widgets in a table-like grid
`.place()`	Places widgets at specific coordinates

Example (Grid Layout):

```
import tkinter as tk

root = tk.Tk()
root.title("Grid Layout")

tk.Label(root, text="Name").grid(row=0, column=0)
tk.Entry(root).grid(row=0, column=1)

tk.Label(root, text="Email").grid(row=1, column=0)
tk.Entry(root).grid(row=1, column=1)

tk.Button(root, text="Submit").grid(row=2, column=1)

root.mainloop()
```

UNIT 4 – Useful Widgets

Widget	Purpose
`Label`	Display text
`Button`	Perform action
`Entry`	Single-line input
`Text`	Multi-line text box
`Checkbutton`	Checkbox
`Radiobutton`	Select one option
`Listbox`	Display list of items
`Frame`	Container for grouping widgets

Example (Checkbox + Radio):

```
import tkinter as tk

root = tk.Tk()

# Checkbutton
var1 = tk.BooleanVar()
tk.Checkbutton(root, text="Python", variable=var1).pack()

# Radiobutton
lang = tk.StringVar(value="None")
tk.Radiobutton(root, text="Male", variable=lang, value="Male").pack()
tk.Radiobutton(root, text="Female", variable=lang, value="Female").pack()

root.mainloop()
```

UNIT 5 – MessageBox, Menu, and File Dialog

MessageBox Example

```
import tkinter as tk
from tkinter import messagebox

def greet():
    messagebox.showinfo("Greeting", "Hello from Tkinter!")

root = tk.Tk()
tk.Button(root, text="Say Hello", command=greet).pack(pady=20)
root.mainloop()
```

File Dialog Example

```
import tkinter as tk
from tkinter import filedialog

def open_file():
    file_path = filedialog.askopenfilename()
    print("Selected:", file_path)

root = tk.Tk()
tk.Button(root, text="Open File", command=open_file).pack(pady=20)
root.mainloop()
```

UNIT 6 – Mini Project Example

Simple Login GUI

```
import tkinter as tk
from tkinter import messagebox

def login():
    username = user_entry.get()
    password = pass_entry.get()
    if username == "admin" and password == "1234":
        messagebox.showinfo("Login", "Login Successful!")
    else:
        messagebox.showerror("Login", "Invalid Credentials")

root = tk.Tk()
root.title("Login Form")
root.geometry("300x200")

tk.Label(root, text="Username").pack(pady=5)
user_entry = tk.Entry(root)
user_entry.pack()

tk.Label(root, text="Password").pack(pady=5)
pass_entry = tk.Entry(root, show="*")
pass_entry.pack()

tk.Button(root, text="Login", command=login).pack(pady=10)

root.mainloop()
```

UNIT 7 – Advanced Widgets

- `Canvas` → Draw shapes, lines, and images
- `Frame` → For grouping widgets
- `Scrollbar`, `Spinbox`, `Scale`
- `ttk` (Themed Tkinter) → Modern look widgets

UNIT 8 – Frames, Canvas & ttk Styling

1.Using Frames (for Layout Management)

A **Frame** acts like a container — helps organize widgets neatly.

```
import tkinter as tk

root = tk.Tk()
root.title("Frame Example")

top_frame = tk.Frame(root, bg="lightblue", pady=10)
top_frame.pack(fill="x")

bottom_frame = tk.Frame(root, bg="lightgreen", pady=10)
bottom_frame.pack(fill="x")

tk.Label(top_frame, text="Top Section").pack()
tk.Label(bottom_frame, text="Bottom Section").pack()

root.mainloop()
```

2.Canvas (Draw Shapes, Lines, Text, Images)

```
import tkinter as tk

root = tk.Tk()
root.title("Canvas Example")

canvas = tk.Canvas(root, width=400, height=300, bg="white")
canvas.pack()

# Draw shapes
canvas.create_rectangle(50, 50, 150, 150, fill="lightblue")
canvas.create_oval(200, 50, 300, 150, fill="pink")
canvas.create_line(50, 200, 350, 200, width=3)
canvas.create_text(200, 250, text="Tkinter Canvas Demo", font=("Arial",
14))

root.mainloop()
```

3.ttk Styling (Modern Look)

`ttk` = themed widgets (modern version of Tkinter widgets)

```
import tkinter as tk
from tkinter import ttk

root = tk.Tk()
root.title("ttk Example")

style = ttk.Style()
style.configure("TButton", font=("Arial", 14), padding=10)

ttk.Label(root, text="Hello from ttk!", font=("Arial", 16)).pack(pady=10)
ttk.Button(root, text="Styled Button").pack(pady=10)

root.mainloop()
```

UNIT 9 – Events and Binding

You can trigger functions when certain **events** occur — like key press, mouse click, etc.

```
import tkinter as tk

def on_key(event):
    print(f"You pressed: {event.char}")

def on_click(event):
    print(f"Mouse clicked at: {event.x}, {event.y}")

root = tk.Tk()
root.title("Event Binding")

root.bind("<Key>", on_key)
root.bind("<Button-1>", on_click)
```

```
tk.Label(root, text="Click or Type Something").pack(pady=20)

root.mainloop()
```

UNIT 10 – Menu Bar

```
import tkinter as tk

def new_file():
    print("New file created!")

root = tk.Tk()
root.title("Menu Example")

menu_bar = tk.Menu(root)

# File Menu
file_menu = tk.Menu(menu_bar, tearoff=0)
file_menu.add_command(label="New", command=new_file)
file_menu.add_command(label="Exit", command=root.quit)
menu_bar.add_cascade(label="File", menu=file_menu)

root.config(menu=menu_bar)
root.mainloop()
```

UNIT 11 – Scrollbar, Listbox & Text Widget

Scrollbar with Text

```
import tkinter as tk

root = tk.Tk()
root.title("Scrollbar Example")

text = tk.Text(root, wrap="word", height=10, width=40)
text.pack(side="left", fill="y")

scroll = tk.Scrollbar(root, command=text.yview)
scroll.pack(side="right", fill="y")

text.config(yscrollcommand=scroll.set)

root.mainloop()
```

Basic Tkinter applications with programming:

Q1. Create a Tkinter window with a Label and a Button that changes the Label text when clicked.

Code:

```
import tkinter as tk

def change_text():
    label.config(text="Button Clicked!")

root = tk.Tk()
root.title("Change Label Text")

label = tk.Label(root, text="Hello, Tkinter!", font=("Arial", 16))
label.pack(pady=10)

button = tk.Button(root, text="Click Me", command=change_text)
button.pack(pady=10)

root.mainloop()
```

Explanation:

- `label.config()` dynamically changes label text.
- `command` in the button calls `change_text()` when clicked.

Q2. Create a simple Login Form with username and password validation.

Code:

```
import tkinter as tk
from tkinter import messagebox

def login():
    username = user_entry.get()
    password = pass_entry.get()
    if username == "admin" and password == "1234":
        messagebox.showinfo("Login", "Login Successful!")
    else:
        messagebox.showerror("Login", "Invalid Credentials")

root = tk.Tk()
root.title("Login Form")
root.geometry("300x200")

tk.Label(root, text="Username").pack(pady=5)
user_entry = tk.Entry(root)
user_entry.pack()

tk.Label(root, text="Password").pack(pady=5)
pass_entry = tk.Entry(root, show="*")
pass_entry.pack()
```

```
tk.Button(root, text="Login", command=login).pack(pady=10)

root.mainloop()
```

Explanation:

- `Entry` widget is used for text input.
- `show="*"` hides password characters.
- `messagebox` displays pop-up results.

Q3. Create a program that takes user input from an Entry widget and displays it in a Label.

Code:

```
import tkinter as tk

def display():
    label.config(text=f"You entered: {entry.get()}")

root = tk.Tk()
root.title("Display Input")

entry = tk.Entry(root, width=30)
entry.pack(pady=10)

button = tk.Button(root, text="Show", command=display)
button.pack()

label = tk.Label(root, text="", font=("Arial", 12))
label.pack(pady=10)

root.mainloop()
```

Explanation:

- `entry.get()` fetches input from Entry.
- `label.config()` updates label text dynamically.

Q4. Create a program with three buttons — Red, Green, Blue — that change the window background color.

Code:

```
import tkinter as tk

def change_color(color):
    root.config(bg=color)

root = tk.Tk()
root.title("Change Background")

tk.Button(root, text="Red", bg="red", command=lambda:
change_color("red")).pack(fill="x")
tk.Button(root, text="Green", bg="green", command=lambda:
change_color("green")).pack(fill="x")
```

```
tk.Button(root, text="Blue", bg="blue", command=lambda:
change_color("blue")).pack(fill="x")

root.mainloop()
```

Explanation:

- `lambda` allows sending arguments through button commands.
- `root.config(bg=...)` changes background color.

Q5. Create a simple Calculator using Tkinter.

Code:

```
import tkinter as tk
from tkinter import messagebox

def calculate():
    try:
        result = eval(entry.get())
        label.config(text=f"Result: {result}")
    except:
        messagebox.showerror("Error", "Invalid Expression")

root = tk.Tk()
root.title("Simple Calculator")

entry = tk.Entry(root, width=25)
entry.pack(pady=10)

tk.Button(root, text="Calculate", command=calculate).pack(pady=5)
label = tk.Label(root, text="Result: ")
label.pack()

root.mainloop()
```

Explanation:

- `eval()` evaluates mathematical expressions.
- Use `try-except` to catch invalid input errors.

Q6. Create a program to display selected gender using Radiobuttons.

Code:

```
import tkinter as tk

def show_gender():
    label.config(text=f"Selected: {gender.get()}")

root = tk.Tk()
root.title("Radiobutton Example")

gender = tk.StringVar(value="None")

tk.Radiobutton(root, text="Male", variable=gender, value="Male").pack()
tk.Radiobutton(root, text="Female", variable=gender,
value="Female").pack()
tk.Radiobutton(root, text="Other", variable=gender, value="Other").pack()

tk.Button(root, text="Show", command=show_gender).pack(pady=10)
label = tk.Label(root, text="")
label.pack()

root.mainloop()
```

Explanation:

- `StringVar()` stores selected Radio button value.
- All buttons share the same `variable` for mutual exclusivity.

Q7. Create a Text Editor with Scrollbar.

Code:

```
import tkinter as tk

root = tk.Tk()
root.title("Text Editor")

text = tk.Text(root, wrap="word", height=10, width=40)
text.pack(side="left", fill="y")

scroll = tk.Scrollbar(root, command=text.yview)
scroll.pack(side="right", fill="y")

text.config(yscrollcommand=scroll.set)

root.mainloop()
```

Explanation:

- `Text()` widget supports multiline text.
- `Scrollbar` is connected to `text.yview` for scrolling.

Q8. Create a program to open a file and display its contents in a Text box.

Code:

```
import tkinter as tk
from tkinter import filedialog

def open_file():
    file = filedialog.askopenfilename(filetypes=[("Text Files", "*.txt")])
    if file:
        text.delete(1.0, tk.END)
        with open(file, "r") as f:
            text.insert(tk.END, f.read())

root = tk.Tk()
root.title("File Reader")

tk.Button(root, text="Open File", command=open_file).pack(pady=5)
```

```
text = tk.Text(root, wrap="word")
text.pack(fill="both", expand=True)

root.mainloop()
```

Explanation:

- `filedialog.askopenfilename()` opens file selector.
- `text.delete()` clears old text; `text.insert()` adds new text.

Q9. Create a digital clock using Tkinter.

Code:

```
import tkinter as tk
import time

def update_time():
    current_time = time.strftime("%H:%M:%S")
    label.config(text=current_time)
    label.after(1000, update_time)

root = tk.Tk()
root.title("Digital Clock")

label = tk.Label(root, font=("Arial", 40))
label.pack(pady=20)

update_time()
root.mainloop()
```

Explanation:

- `time.strftime()` formats current time.
- `label.after(1000, func)` updates every second.

Q10. Create a To-Do List App.

Code:

```
import tkinter as tk
from tkinter import messagebox

def add_task():
    task = entry.get()
    if task != "":
        listbox.insert(tk.END, task)
        entry.delete(0, tk.END)
    else:
        messagebox.showwarning("Warning", "Please enter a task")

def delete_task():
    try:
        index = listbox.curselection()[0]
        listbox.delete(index)
    except:
        messagebox.showerror("Error", "Select a task to delete")

root = tk.Tk()
```

```
root.title("To-Do List")

entry = tk.Entry(root, width=35)
entry.pack(pady=10)

tk.Button(root, text="Add Task", command=add_task).pack()
tk.Button(root, text="Delete Task", command=delete_task).pack(pady=5)

listbox = tk.Listbox(root, width=45, height=10)
listbox.pack(pady=10)

root.mainloop()
```

Explanation:

- `Listbox` stores tasks.
- `curselection()` gets the index of selected task.
- Buttons add and delete tasks dynamically.

10 Tkinter mini-projects

PROJECT 1 – Digital Clock

Description:

Displays live time updated every second.

```
import tkinter as tk
import time

def update_time():
    current = time.strftime("%H:%M:%S")
    clock.config(text=current)
    clock.after(1000, update_time)

root = tk.Tk()
root.title("Digital Clock")

clock = tk.Label(root, font=("Arial", 40), fg="black")
clock.pack(pady=30)

update_time()
root.mainloop()
```

Concepts: `Label`, `after()`, `time.strftime()`

PROJECT 2 – Simple Calculator

Description:

Perform basic math (+, -, *, /).

```
import tkinter as tk

def click(event):
    text = event.widget.cget("text")
    if text == "=":
        try:
            value = eval(str(screen.get()))
            screen.set(value)
        except:
            screen.set("Error")
    elif text == "C":
        screen.set("")
    else:
        screen.set(screen.get() + text)

root = tk.Tk()
root.title("Calculator")

screen = tk.StringVar()
entry = tk.Entry(root, textvar=screen, font="Arial 20", justify="right")
entry.pack(fill="x", ipadx=8, pady=10, padx=10)
```

```
buttons = [
    "7","8","9","/",
    "4","5","6","*",
    "1","2","3","-",
    "0",".","C","+","="
]

f = tk.Frame(root)
f.pack()

for i, b in enumerate(buttons):
    btn = tk.Button(f, text=b, padx=20, pady=20, font="Arial 14")
    btn.grid(row=i//4, column=i%4)
    btn.bind("<Button-1>", click)

root.mainloop()
```

Concepts: `Button`, `StringVar`, `grid()`, `bind()`

PROJECT 3 – To-Do List App

Description:

Add and delete daily tasks.

```
import tkinter as tk
from tkinter import messagebox

def add_task():
    task = entry.get()
    if task:
        listbox.insert(tk.END, task)
        entry.delete(0, tk.END)
    else:
        messagebox.showwarning("Warning", "Enter a task!")

def delete_task():
    try:
        selected = listbox.curselection()[0]
        listbox.delete(selected)
    except:
        messagebox.showerror("Error", "No task selected")

root = tk.Tk()
root.title("To-Do List")

entry = tk.Entry(root, width=35)
entry.pack(pady=10)
tk.Button(root, text="Add", command=add_task).pack()
tk.Button(root, text="Delete", command=delete_task).pack(pady=5)

listbox = tk.Listbox(root, width=45, height=10)
listbox.pack(pady=10)

root.mainloop()
```

Concepts: `Listbox`, `curselection()`, `messagebox`

PROJECT 4 – Temperature Converter

Description:

Convert °C ↔ °F.

```
import tkinter as tk

def convert():
    temp = float(entry.get())
    if var.get() == 1:
        result = (temp * 9/5) + 32
        label_result.config(text=f"{result:.2f} °F")
    else:
        result = (temp - 32) * 5/9
        label_result.config(text=f"{result:.2f} °C")

root = tk.Tk()
root.title("Temperature Converter")

tk.Label(root, text="Enter Temperature:").pack()
entry = tk.Entry(root)
entry.pack(pady=5)

var = tk.IntVar(value=1)
tk.Radiobutton(root, text="C → F", variable=var, value=1).pack()
tk.Radiobutton(root, text="F → C", variable=var, value=2).pack()

tk.Button(root, text="Convert", command=convert).pack(pady=5)
label_result = tk.Label(root, text="")
label_result.pack(pady=10)

root.mainloop()
```

Concepts: `Radiobutton`, `IntVar`, math logic

PROJECT 5 – Basic Quiz App

Description:

Multiple choice quiz with score.

```
import tkinter as tk
from tkinter import messagebox

questions = [
    ["What is capital of India?", "Delhi", "Mumbai", "Kolkata", "Delhi"],
    ["2 + 5 = ?", "5", "7", "9", "7"],
    ["Which is programming language?", "Python", "Car", "Apple", "Python"]
]
index = 0
score = 0

def next_q():
    global index, score
    if var.get() == questions[index][4]:
        score += 1
    index += 1
    if index < len(questions):
        show_question()
    else:
        messagebox.showinfo("Result", f"Your Score:
{score}/{len(questions)}")
        root.destroy()

def show_question():
    q, a, b, c, ans = questions[index]
    question.config(text=q)
    var.set(None)
    for i, opt in enumerate([a,b,c]):
        buttons[i].config(text=opt, value=opt)

root = tk.Tk()
root.title("Quiz App")

question = tk.Label(root, font=("Arial", 16))
question.pack(pady=10)

var = tk.StringVar()
buttons = []
for i in range(3):
    rb = tk.Radiobutton(root, variable=var, font=("Arial", 14))
    rb.pack(anchor="w", padx=50)
    buttons.append(rb)

tk.Button(root, text="Next", command=next_q).pack(pady=10)
show_question()
root.mainloop()
```

Concepts: `Radiobutton`, `StringVar`, dynamic question loading

PROJECT 6 – Random Password Generator

Description:

Generate secure passwords.

```
import tkinter as tk
import random, string

def generate():
    length = int(entry.get())
    chars = string.ascii_letters + string.digits + string.punctuation
    pwd = "".join(random.choice(chars) for i in range(length))
    label.config(text=pwd)

root = tk.Tk()
root.title("Password Generator")

tk.Label(root, text="Password Length:").pack(pady=5)
entry = tk.Entry(root)
entry.pack(pady=5)
tk.Button(root, text="Generate", command=generate).pack(pady=5)
label = tk.Label(root, text="", font=("Arial", 14))
label.pack(pady=10)
```

```
root.mainloop()
```

Concepts: `random.choice()`, `string` module

PROJECT 7 – Currency Converter

Description:

Convert INR ↔ USD (fixed rate).

```
import tkinter as tk

def convert():
    amount = float(entry.get())
    if var.get() == 1:
        result = amount / 83.0
        label_result.config(text=f"{result:.2f} USD")
    else:
        result = amount * 83.0
        label_result.config(text=f"{result:.2f} INR")

root = tk.Tk()
root.title("Currency Converter")

tk.Label(root, text="Enter Amount:").pack()
entry = tk.Entry(root)
entry.pack(pady=5)

var = tk.IntVar(value=1)
tk.Radiobutton(root, text="INR → USD", variable=var, value=1).pack()
tk.Radiobutton(root, text="USD → INR", variable=var, value=2).pack()

tk.Button(root, text="Convert", command=convert).pack(pady=5)
label_result = tk.Label(root, text="")
label_result.pack(pady=10)

root.mainloop()
```

Concepts: `Radiobutton`, conversion logic

PROJECT 8 – Notepad

Description:

Open, save, and edit text files.

```
import tkinter as tk
from tkinter import filedialog

def open_file():
    file = filedialog.askopenfilename()
    if file:
        text.delete(1.0, tk.END)
        with open(file, "r") as f:
            text.insert(tk.END, f.read())
```

```
def save_file():
    file = filedialog.asksaveasfilename(defaultextension=".txt")
    if file:
        with open(file, "w") as f:
            f.write(text.get(1.0, tk.END))

root = tk.Tk()
root.title("Mini Notepad")

menu = tk.Menu(root)
file_menu = tk.Menu(menu, tearoff=0)
file_menu.add_command(label="Open", command=open_file)
file_menu.add_command(label="Save", command=save_file)
menu.add_cascade(label="File", menu=file_menu)
root.config(menu=menu)

text = tk.Text(root, wrap="word")
text.pack(expand=True, fill="both")

root.mainloop()
```

Concepts: `Menu`, `filedialog`, `Text`

PROJECT 9 – Drawing App

Description:

Draw freely with mouse.

```
import tkinter as tk

def paint(event):
    x1, y1 = (event.x - 2), (event.y - 2)
    x2, y2 = (event.x + 2), (event.y + 2)
    canvas.create_oval(x1, y1, x2, y2, fill=color.get(),
outline=color.get())

root = tk.Tk()
root.title("Drawing App")

color = tk.StringVar(value="black")
tk.Radiobutton(root, text="Black", variable=color,
value="black").pack(side="left")
tk.Radiobutton(root, text="Red", variable=color,
value="red").pack(side="left")
tk.Radiobutton(root, text="Blue", variable=color,
value="blue").pack(side="left")

canvas = tk.Canvas(root, bg="white", width=500, height=400)
canvas.pack()
canvas.bind("<B1-Motion>", paint)

root.mainloop()
```

Concepts: `Canvas`, `bind()`, `Radiobutton`

PROJECT 10 – Student Registration Form

Description:

Collect and print student info.

```
import tkinter as tk
from tkinter import messagebox

def submit():
    data = f"Name: {name.get()}\nAge: {age.get()}\nGender:
{gender.get()}\nCourse: {course.get()}"
    messagebox.showinfo("Registration Details", data)

root = tk.Tk()
root.title("Student Registration Form")

tk.Label(root, text="Name").grid(row=0, column=0)
tk.Label(root, text="Age").grid(row=1, column=0)
tk.Label(root, text="Gender").grid(row=2, column=0)
tk.Label(root, text="Course").grid(row=3, column=0)

name = tk.Entry(root); name.grid(row=0, column=1)
age = tk.Entry(root); age.grid(row=1, column=1)
gender = tk.StringVar(value="Male")
tk.Radiobutton(root, text="Male", variable=gender,
value="Male").grid(row=2, column=1)
tk.Radiobutton(root, text="Female", variable=gender,
value="Female").grid(row=2, column=2)
course = tk.Entry(root); course.grid(row=3, column=1)

tk.Button(root, text="Submit", command=submit).grid(row=4, column=1,
pady=10)

root.mainloop()
```

5 industry-level advanced Tkinter project in a single file
"""
Top5_Tkinter_Advanced_Apps.py

Contains five industry-level Tkinter apps in one file:
1) Student Management System (students.db)
2) Expense Tracker (expenses.db)
3) Login & Signup System (users.db)
4) Payroll System (payroll.db)
5) Library Management System (library.db)

Run the file and follow the text-menu that appears in console to open the GUI you want.
Dependencies: standard library only (tkinter, sqlite3, hashlib, datetime)

Save this file and run: python Top5_Tkinter_Advanced_Apps.py

How to run

1. Download or copy the file contents from the canvas into `Top5_Tkinter_Advanced_Apps.py`.
2. Run in your terminal / command prompt:
3. `python Top5_Tkinter_Advanced_Apps.py`
4. You'll see a simple text menu in the console. Enter `1–5` to launch the corresponding GUI app.

Quick notes & tips

- All apps use **SQLite** (no external DB required). Database files are created automatically in the working directory (e.g., `students.db`, `expenses.db`, etc.).
- The **Login/Signup** uses SHA-256 hashing for stored passwords (`hashlib`), so raw passwords are not saved.
- Each GUI is self-contained and demonstrates **CRUD** operations, validation, and user feedback via `messagebox`.
- The file is fully commented so you can study or modify each app (add fields, export CSV, print reports, etc.).

```
"""


import tkinter as tk
from tkinter import messagebox, simpledialog, filedialog
import sqlite3
import hashlib
from datetime import datetime
import os

# ----------------------------
# Utility functions
# ----------------------------

def ensure_dir(path):
    d = os.path.dirname(path)
    if d and not os.path.exists(d):
        os.makedirs(d)


def hash_password(pwd: str) -> str:
    return hashlib.sha256(pwd.encode()).hexdigest()


# -----------------------------
# 1) Student Management System
# -----------------------------
class StudentManagementApp:
    DB = 'students.db'

    def __init__(self):
        self.conn = sqlite3.connect(self.DB)
        self.cur = self.conn.cursor()
        self.cur.execute('''CREATE TABLE IF NOT EXISTS student (
                    id INTEGER PRIMARY KEY AUTOINCREMENT,
                    name TEXT NOT NULL,
                    age INTEGER,
                    course TEXT)''')
        self.conn.commit()
        self.root = tk.Tk()
        self.root.title('Student Management System')
        self.build_ui()
        self.load_students()
        self.root.mainloop()

    def build_ui(self):
        frame = tk.Frame(self.root, padx=10, pady=10)
        frame.pack()

        tk.Label(frame, text='Name').grid(row=0, column=0)
        tk.Label(frame, text='Age').grid(row=1, column=0)
        tk.Label(frame, text='Course').grid(row=2, column=0)

        self.name_e = tk.Entry(frame)
        self.age_e = tk.Entry(frame)
        self.course_e = tk.Entry(frame)
        self.name_e.grid(row=0, column=1)
        self.age_e.grid(row=1, column=1)
        self.course_e.grid(row=2, column=1)

        tk.Button(frame, text='Add', width=12, command=self.add_student).grid(row=3,
column=0, pady=5)
        tk.Button(frame, text='Update', width=12, command=self.update_student).grid(row=3,
column=1)
```

```
        tk.Button(frame, text='Delete', width=12, command=self.delete_student).grid(row=3,
column=2)

        self.listbox = tk.Listbox(self.root, width=60)
        self.listbox.pack(padx=10, pady=10)
        self.listbox.bind('<<ListboxSelect>>', self.on_select)

    def add_student(self):
        name = self.name_e.get().strip()
        age = self.age_e.get().strip()
        course = self.course_e.get().strip()
        if not name:
            messagebox.showwarning('Validation', 'Name is required')
            return
        try:
            age_val = int(age) if age else None
        except ValueError:
            messagebox.showwarning('Validation', 'Age must be an integer')
            return
        self.cur.execute('INSERT INTO student (name, age, course) VALUES (?, ?, ?)', (name,
age_val, course))
        self.conn.commit()
        messagebox.showinfo('Success', 'Student added')
        self.clear_entries()
        self.load_students()

    def load_students(self):
        self.listbox.delete(0, tk.END)
        for row in self.cur.execute('SELECT id, name, age, course FROM student'):
            self.listbox.insert(tk.END, row)

    def on_select(self, event):
        try:
            sel = self.listbox.get(self.listbox.curselection())
        except Exception:
            return
        sid, name, age, course = sel
        self.name_e.delete(0, tk.END); self.name_e.insert(0, name)
        self.age_e.delete(0, tk.END); self.age_e.insert(0, age if age is not None else '')
        self.course_e.delete(0, tk.END); self.course_e.insert(0, course)

    def clear_entries(self):
        self.name_e.delete(0, tk.END)
        self.age_e.delete(0, tk.END)
        self.course_e.delete(0, tk.END)

    def update_student(self):
        try:
            sel = self.listbox.get(self.listbox.curselection())
        except Exception:
            messagebox.showwarning('Select', 'Select a student to update')
            return
        sid = sel[0]
        name = self.name_e.get().strip(); age = self.age_e.get().strip(); course =
self.course_e.get().strip()
        try:
            age_val = int(age) if age else None
        except ValueError:
            messagebox.showwarning('Validation', 'Age must be an integer')
            return
        self.cur.execute('UPDATE student SET name=?, age=?, course=? WHERE id=?', (name,
age_val, course, sid))
        self.conn.commit()
        messagebox.showinfo('Updated', 'Student updated')
        self.load_students()

    def delete_student(self):
        try:
            sel = self.listbox.get(self.listbox.curselection())
        except Exception:
            messagebox.showwarning('Select', 'Select a student to delete')
            return
        sid = sel[0]
        if messagebox.askyesno('Confirm', 'Delete selected student?'):
            self.cur.execute('DELETE FROM student WHERE id=?', (sid,))
            self.conn.commit()
            self.load_students()


# ----------------------------
# 2) Expense Tracker
# ----------------------------
class ExpenseTrackerApp:
    DB = 'expenses.db'

    def __init__(self):
        self.conn = sqlite3.connect(self.DB)
        self.cur = self.conn.cursor()
        self.cur.execute('''CREATE TABLE IF NOT EXISTS expense (
                            id INTEGER PRIMARY KEY AUTOINCREMENT,
                            item TEXT,
                            amount REAL,
                            date TEXT)''')
        self.conn.commit()
        self.root = tk.Tk()
        self.root.title('Expense Tracker')
        self.build()
        self.load()
        self.root.mainloop()

    def build(self):
        f = tk.Frame(self.root, padx=10, pady=10)
        f.pack()
        tk.Label(f, text='Item').grid(row=0, column=0)
        tk.Label(f, text='Amount').grid(row=1, column=0)
        self.item_e = tk.Entry(f); self.item_e.grid(row=0, column=1)
        self.amount_e = tk.Entry(f); self.amount_e.grid(row=1, column=1)
        tk.Button(f, text='Add', command=self.add).grid(row=2, column=0, columnspan=2,
pady=5)
        self.listbox = tk.Listbox(self.root, width=60)
        self.listbox.pack(padx=10, pady=10)
        tk.Button(self.root, text='Delete Selected', command=self.delete).pack()
        self.total_label = tk.Label(self.root, text='Total: ₹0')
        self.total_label.pack(pady=5)

    def add(self):
        item = self.item_e.get().strip(); amt = self.amount_e.get().strip()
        if not item or not amt:
            messagebox.showwarning('Validation', 'Both item & amount required')
            return
        try:
            amtv = float(amt)
        except ValueError:
            messagebox.showwarning('Validation', 'Amount must be numeric')
            return
        date = datetime.now().strftime('%d-%m-%Y')
        self.cur.execute('INSERT INTO expense (item, amount, date) VALUES (?, ?, ?)', (item,
amtv, date))
        self.conn.commit()
        self.item_e.delete(0, tk.END); self.amount_e.delete(0, tk.END)
        self.load()

    def load(self):
        self.listbox.delete(0, tk.END)
        total = 0.0
        for row in self.cur.execute('SELECT id, item, amount, date FROM expense'):
            self.listbox.insert(tk.END, row)
            total += float(row[2])
        self.total_label.config(text=f'Total: ₹{total:.2f}')

    def delete(self):
        try:
            sel = self.listbox.get(self.listbox.curselection())
        except Exception:
            messagebox.showwarning('Select', 'Select an expense to delete')
            return
        eid = sel[0]
        if messagebox.askyesno('Confirm', 'Delete selected expense?'):
            self.cur.execute('DELETE FROM expense WHERE id=?', (eid,))
            self.conn.commit()
            self.load()


# -----------------------------
# 3) Login & Signup System
# -----------------------------
class LoginSignupApp:
    DB = 'users.db'

    def __init__(self):
        self.conn = sqlite3.connect(self.DB)
        self.cur = self.conn.cursor()
        self.cur.execute('CREATE TABLE IF NOT EXISTS user (username TEXT PRIMARY KEY,
password TEXT)')
        self.conn.commit()
        self.root = tk.Tk()
        self.root.title('Login & Signup')
        self.build()
        self.root.mainloop()

    def build(self):
        f = tk.Frame(self.root, padx=10, pady=10)
        f.pack()
        tk.Label(f, text='Username').grid(row=0, column=0)
        tk.Label(f, text='Password').grid(row=1, column=0)
        self.user_e = tk.Entry(f); self.user_e.grid(row=0, column=1)
        self.pwd_e = tk.Entry(f, show='*'); self.pwd_e.grid(row=1, column=1)
        tk.Button(f, text='Login', command=self.login).grid(row=2, column=0, pady=5)
        tk.Button(f, text='Signup', command=self.signup).grid(row=2, column=1)

    def signup(self):
        username = self.user_e.get().strip(); pwd = self.pwd_e.get().strip()
        if not username or not pwd:
            messagebox.showwarning('Validation', 'Provide username & password')
            return
        hashed = hash_password(pwd)
        try:
            self.cur.execute('INSERT INTO user (username, password) VALUES (?, ?)', (username,
hashed))
            self.conn.commit()
            messagebox.showinfo('Success', 'Signup successful. You can login now.')
        except sqlite3.IntegrityError:
            messagebox.showerror('Error', 'Username already exists')

    def login(self):
        username = self.user_e.get().strip(); pwd = self.pwd_e.get().strip()
        if not username or not pwd:
            messagebox.showwarning('Validation', 'Provide username & password')
            return
        hashed = hash_password(pwd)
        self.cur.execute('SELECT * FROM user WHERE username=? AND password=?',
(username, hashed))
        if self.cur.fetchone():
            messagebox.showinfo('Welcome', f'Welcome, {username}!')
        else:
            messagebox.showerror('Error', 'Invalid credentials')



# -----------------------------
# 4) Payroll System
# -----------------------------
class PayrollApp:
    DB = 'payroll.db'

    def __init__(self):
        self.conn = sqlite3.connect(self.DB)
        self.cur = self.conn.cursor()
        self.cur.execute('''CREATE TABLE IF NOT EXISTS employee (
                            id INTEGER PRIMARY KEY AUTOINCREMENT,
                            name TEXT,
                            salary REAL)''')
        self.conn.commit()
        self.root = tk.Tk()
        self.root.title('Payroll System')
        self.build()
        self.load()
        self.root.mainloop()

    def build(self):
        f = tk.Frame(self.root, padx=10, pady=10)
        f.pack()
        tk.Label(f, text='Name').grid(row=0, column=0)
        tk.Label(f, text='Salary').grid(row=1, column=0)
        self.name_e = tk.Entry(f); self.name_e.grid(row=0, column=1)
        self.salary_e = tk.Entry(f); self.salary_e.grid(row=1, column=1)
        tk.Button(f, text='Add', command=self.add).grid(row=2, column=0)
        tk.Button(f, text='Delete', command=self.delete).grid(row=2, column=1)
        tk.Button(f, text='Total Payroll', command=self.total_payroll).grid(row=2, column=2)
        self.listbox = tk.Listbox(self.root, width=60)
        self.listbox.pack(padx=10, pady=10)
    def add(self):
        name = self.name_e.get().strip(); sal = self.salary_e.get().strip()
        if not name or not sal:
            messagebox.showwarning('Validation', 'Provide name & salary')
            return
        try:
            sal_v = float(sal)
        except ValueError:
            messagebox.showwarning('Validation', 'Salary must be numeric')
            return
        self.cur.execute('INSERT INTO employee (name, salary) VALUES (?, ?)', (name, sal_v))
        self.conn.commit()
        self.name_e.delete(0, tk.END); self.salary_e.delete(0, tk.END)
        self.load()

    def load(self):
        self.listbox.delete(0, tk.END)
        for row in self.cur.execute('SELECT id, name, salary FROM employee'):
            self.listbox.insert(tk.END, row)

    def delete(self):
        try:
            sel = self.listbox.get(self.listbox.curselection())
        except Exception:
            messagebox.showwarning('Select', 'Select an employee to delete')
            return
        eid = sel[0]
        if messagebox.askyesno('Confirm', 'Delete selected employee?'):
            self.cur.execute('DELETE FROM employee WHERE id=?', (eid,))
            self.conn.commit()
            self.load()

    def total_payroll(self):
        self.cur.execute('SELECT SUM(salary) FROM employee')
        total = self.cur.fetchone()[0] or 0.0
        messagebox.showinfo('Total Payroll', f'Total Salary Payout: ₹{total:.2f}')


# ----------------------------
# 5) Library Management System
# ----------------------------
class LibraryApp:
    DB = 'library.db'

    def __init__(self):
        self.conn = sqlite3.connect(self.DB)
        self.cur = self.conn.cursor()
        self.cur.execute('''CREATE TABLE IF NOT EXISTS books (
                        id INTEGER PRIMARY KEY AUTOINCREMENT,
                        title TEXT,
                        author TEXT,
                        copies INTEGER DEFAULT 1)''')
        self.cur.execute('''CREATE TABLE IF NOT EXISTS transactions (
                        id INTEGER PRIMARY KEY AUTOINCREMENT,
                        book_id INTEGER,
                        action TEXT,
                        date TEXT)''')
        self.conn.commit()
        self.root = tk.Tk()
        self.root.title('Library Management')
        self.build()
        self.load()
        self.root.mainloop()

    def build(self):
        f = tk.Frame(self.root, padx=10, pady=10)
        f.pack()
        tk.Label(f, text='Title').grid(row=0, column=0)
        tk.Label(f, text='Author').grid(row=1, column=0)
        tk.Label(f, text='Copies').grid(row=2, column=0)
        self.title_e = tk.Entry(f); self.title_e.grid(row=0, column=1)
        self.author_e = tk.Entry(f); self.author_e.grid(row=1, column=1)
        self.copies_e = tk.Entry(f); self.copies_e.grid(row=2, column=1)
        tk.Button(f, text='Add Book', command=self.add_book).grid(row=3, column=0)
        tk.Button(f, text='Issue Book', command=self.issue_book).grid(row=3, column=1)
        tk.Button(f, text='Return Book', command=self.return_book).grid(row=3, column=2)
        tk.Button(f, text='Delete', command=self.delete_book).grid(row=3, column=3)
        self.listbox = tk.Listbox(self.root, width=80)
        self.listbox.pack(padx=10, pady=10)

    def add_book(self):
        title = self.title_e.get().strip(); author = self.author_e.get().strip(); copies =
self.copies_e.get().strip()
        if not title:
            messagebox.showwarning('Validation', 'Title required')
            return
        try:
            copies_v = int(copies) if copies else 1
        except ValueError:
            messagebox.showwarning('Validation', 'Copies must be integer')
            return
        self.cur.execute('INSERT INTO books (title, author, copies) VALUES (?, ?, ?)', (title,
author, copies_v))
        self.conn.commit()
        self.title_e.delete(0, tk.END); self.author_e.delete(0, tk.END); self.copies_e.delete(0,
tk.END)
        self.load()

    def load(self):
        self.listbox.delete(0, tk.END)
        for row in self.cur.execute('SELECT id, title, author, copies FROM books'):
            self.listbox.insert(tk.END, row)

    def issue_book(self):
        try:
            sel = self.listbox.get(self.listbox.curselection())
        except Exception:
            messagebox.showwarning('Select', 'Select a book to issue')
            return
        bid, title, author, copies = sel
        if copies <= 0:
            messagebox.showwarning('Unavailable', 'No copies available')
            return
        self.cur.execute('UPDATE books SET copies=copies-1 WHERE id=?', (bid,))
        self.cur.execute('INSERT INTO transactions (book_id, action, date) VALUES (?, ?, ?)',
(bid, 'issue', datetime.now().strftime('%d-%m-%Y')))
        self.conn.commit()
        self.load()
        messagebox.showinfo('Issued', f'Book issued: {title}')

    def return_book(self):
        try:
            sel = self.listbox.get(self.listbox.curselection())
        except Exception:
            messagebox.showwarning('Select', 'Select a book to return')
            return
        bid, title, author, copies = sel
        self.cur.execute('UPDATE books SET copies=copies+1 WHERE id=?', (bid,))
        self.cur.execute('INSERT INTO transactions (book_id, action, date) VALUES (?, ?, ?)',
(bid, 'return', datetime.now().strftime('%d-%m-%Y')))
        self.conn.commit()
        self.load()
        messagebox.showinfo('Returned', f'Book returned: {title}')

    def delete_book(self):
        try:
            sel = self.listbox.get(self.listbox.curselection())
        except Exception:
            messagebox.showwarning('Select', 'Select a book to delete')
            return
        bid = sel[0]
        if messagebox.askyesno('Confirm', 'Delete selected book?'):
            self.cur.execute('DELETE FROM books WHERE id=?', (bid,))
        self.conn.commit()
        self.load()


# ----------------------------
# Launcher CLI
# ----------------------------
if __name__ == '__main__':
    print('Top 5 Tkinter Advanced Apps')
    print('1 - Student Management System')
    print('2 - Expense Tracker')
    print('3 - Login & Signup System')
    print('4 - Payroll System')
    print('5 - Library Management System')
    choice = input('Enter number to launch (1-5): ').strip()
    if choice == '1':
        StudentManagementApp()
    elif choice == '2':
        ExpenseTrackerApp()
    elif choice == '3':
        LoginSignupApp()
    elif choice == '4':
        PayrollApp()
    elif choice == '5':
        LibraryApp()
    else:
        print('Invalid choice. Exiting.')
```

PROJECT : Student Management System using GUI (Tkinter)

"""

Student Management System - GUI (Tkinter)

Features:

```
- SQLite backend (student_system_gui.db)
- Create table on startup
- Add, View, Update, Delete students
- Search by name/email
- Sort, Count
- Export to CSV, Import from CSV
- Create SQL view and display
- Transaction demo (commit/rollback)
- Uses tkinter + ttk Treeview for tabular display

Run:
   python Student Management System - GUI
(Tkinter).py

No external packages required (only stdlib).
"""


import sqlite3
import csv
import os
import tkinter as tk
from tkinter import ttk, messagebox, filedialog

DB_FILE = 'student_system_gui.db'

# -------------------- Database helpers --------------------

def connect_db():
    return sqlite3.connect(DB_FILE)
```

```
def initialize_db():
    conn = connect_db()
    cur = conn.cursor()
    cur.execute('''
        CREATE TABLE IF NOT EXISTS students (
            id INTEGER PRIMARY KEY AUTOINCREMENT,
            name TEXT NOT NULL,
            age INTEGER,
            email TEXT UNIQUE,
            course TEXT
        )
    ''')
    conn.commit()
    conn.close()

# -------------------- CRUD operations --------------------

def insert_student(name, age, email, course):
    try:
        conn = connect_db()
        cur = conn.cursor()
        cur.execute("INSERT INTO students (name, age, email, course) VALUES
(?, ?, ?, ?)",
                    (name, age, email, course))
        conn.commit()
        return cur.lastrowid
    except sqlite3.IntegrityError as e:
        raise
    finally:
        conn.close()



def fetch_all_students(order_by=None, limit=None, offset=None):
    conn = connect_db()
    cur = conn.cursor()
    query = "SELECT id, name, age, email, course FROM students"
    if order_by:
        query += f" ORDER BY {order_by}"
    if limit is not None:
        query += f" LIMIT {limit}"
        if offset is not None:
            query += f" OFFSET {offset}"
    cur.execute(query)
    rows = cur.fetchall()
    conn.close()
    return rows


def search_students(term):
    conn = connect_db()
    cur = conn.cursor()
    cur.execute("SELECT id, name, age, email, course FROM students WHERE 
name LIKE ? OR email LIKE ?",
                ('%'+term+'%', '%'+term+'%'))
    rows = cur.fetchall()
    conn.close()
    return rows


def update_student_record(student_id, name, age, email, course):
    conn = connect_db()
    cur = conn.cursor()
    cur.execute("UPDATE students SET name=?, age=?, email=?, course=? 
WHERE id=?",
                (name, age, email, course, student_id))
    conn.commit()
    conn.close()


def delete_student_record(student_id):
    conn = connect_db()
    cur = conn.cursor()
    cur.execute("DELETE FROM students WHERE id=?", (student_id,))
    conn.commit()
    conn.close()

# -------------------- CSV Import/Export --------------------

def export_to_csv(filepath):
    try:
        rows = fetch_all_students()
        with open(filepath, 'w', newline='', encoding='utf-8') as f:
            writer = csv.writer(f)
            writer.writerow(['id', 'name', 'age', 'email', 'course'])
            writer.writerows(rows)
        return True
    except Exception as e:
        raise



def import_from_csv(filepath):
    try:
        conn = connect_db()
        cur = conn.cursor()
        with open(filepath, 'r', newline='', encoding='utf-8') as f:
            reader = csv.reader(f)
            headers = next(reader, None)
            rows = [tuple(r) for r in reader if r]
            # assume CSV has columns name,age,email,course or
id,name,age,email,course
            # We'll try to insert rows that have 4 or 5 fields
            for r in rows:
                if len(r) == 4:
                    cur.execute("INSERT INTO students (name, age, email, course)
VALUES (?, ?, ?, ?)", r)
```

```
            elif len(r) == 5:
                # ignore id column and insert
                cur.execute("INSERT INTO students (name, age, email, course)
VALUES (?, ?, ?, ?)", (r[1], r[2], r[3], r[4]))
        conn.commit()
        conn.close()
        return True
    except sqlite3.IntegrityError as e:
        conn.rollback()
        conn.close()
        raise
    except Exception as e:
        conn.close()
        raise

# -------------------- Views & Transactions --------------------

def create_student_view():
    conn = connect_db()
    cur = conn.cursor()
    cur.execute("CREATE VIEW IF NOT EXISTS student_view AS SELECT
name, email, course FROM students")
    conn.commit()
    cur.execute("SELECT name, email, course FROM student_view")
    rows = cur.fetchall()
    conn.close()
    return rows



def transaction_increment_ages(commit=True):
    conn = connect_db()
    cur = conn.cursor()
    try:
        cur.execute("UPDATE students SET age = age + 1")
        if commit:
            conn.commit()
            return 'committed'
        else:
            conn.rollback()
            return 'rolled back'
    finally:
        conn.close()

# -------------------- Utility --------------------

def count_students():
    conn = connect_db()
    cur = conn.cursor()
    cur.execute("SELECT COUNT(*) FROM students")
    total = cur.fetchone()[0]
    conn.close()
    return total

# -------------------- GUI --------------------

class StudentApp(tk.Tk):
    def __init__(self):
        super().__init__()
        self.title('Student Management System - GUI')
        self.geometry('900x600')
        self.resizable(False, False)

        self.selected_student_id = None

        self.create_widgets()
        self.refresh_tree()

    def create_widgets(self):
        # Top frame for form
        frm = ttk.Frame(self, padding=10)
        frm.pack(fill='x')

        ttk.Label(frm, text='Name:').grid(row=0, column=0, sticky='w')
        self.name_var = tk.StringVar()
        ttk.Entry(frm, textvariable=self.name_var, width=25).grid(row=0,
column=1, padx=5)

        ttk.Label(frm, text='Age:').grid(row=0, column=2, sticky='w')
        self.age_var = tk.StringVar()
        ttk.Entry(frm, textvariable=self.age_var, width=10).grid(row=0,
column=3, padx=5)

        ttk.Label(frm, text='Email:').grid(row=1, column=0, sticky='w')
        self.email_var = tk.StringVar()
        ttk.Entry(frm, textvariable=self.email_var, width=25).grid(row=1,
column=1, padx=5)

        ttk.Label(frm, text='Course:').grid(row=1, column=2, sticky='w')
        self.course_var = tk.StringVar()
        ttk.Entry(frm, textvariable=self.course_var, width=20).grid(row=1,
column=3, padx=5)

        ttk.Button(frm, text='Add', command=self.add_student).grid(row=0,
column=4, padx=10)
        ttk.Button(frm, text='Update', command=self.update_student).grid(row=1,
column=4, padx=10)
        ttk.Button(frm, text='Delete', command=self.delete_student).grid(row=2,
column=4, padx=10)

        # Search and utility
        util_frm = ttk.Frame(self, padding=10)
        util_frm.pack(fill='x')
        ttk.Label(util_frm, text='Search:').grid(row=0, column=0)
        self.search_var = tk.StringVar()
        ttk.Entry(util_frm, textvariable=self.search_var, width=30).grid(row=0,
column=1, padx=5)
        ttk.Button(util_frm, text='Search', command=self.search).grid(row=0,
column=2, padx=5)
        ttk.Button(util_frm, text='Refresh',
command=self.refresh_tree).grid(row=0, column=3, padx=5)
        ttk.Button(util_frm, text='Count', command=self.show_count).grid(row=0,
column=4, padx=5)
        ttk.Button(util_frm, text='Sort by Name',
command=self.sort_by_name).grid(row=0, column=5, padx=5)

        # CSV / view / transaction buttons
        ops_frm = ttk.Frame(self, padding=10)
        ops_frm.pack(fill='x')
        ttk.Button(ops_frm, text='Export CSV',
command=self.export_csv).grid(row=0, column=0, padx=5)
        ttk.Button(ops_frm, text='Import CSV',
command=self.import_csv).grid(row=0, column=1, padx=5)
        ttk.Button(ops_frm, text='Create View',
command=self.create_view).grid(row=0, column=2, padx=5)
        ttk.Button(ops_frm, text='Transaction Demo',
command=self.transaction_demo).grid(row=0, column=3, padx=5)

        # Treeview to show students
        tree_frm = ttk.Frame(self, padding=10)
        tree_frm.pack(fill='both', expand=True)

        columns = ('id', 'name', 'age', 'email', 'course')
        self.tree = ttk.Treeview(tree_frm, columns=columns, show='headings',
height=15)
        for col in columns:
            self.tree.heading(col, text=col.title())
            # set width
            if col == 'email':
                self.tree.column(col, width=220)
            elif col == 'name':
                self.tree.column(col, width=140)
            else:
```

```
            self.tree.column(col, width=100)

        vsb = ttk.Scrollbar(tree_frm, orient='vertical', command=self.tree.yview)
        hsb = ttk.Scrollbar(tree_frm, orient='horizontal',
command=self.tree.xview)
        self.tree.configure(yscroll=vsb.set, xscroll=hsb.set)
        self.tree.bind('<<TreeviewSelect>>', self.on_tree_select)

        self.tree.grid(row=0, column=0, sticky='nsew')
        vsb.grid(row=0, column=1, sticky='ns')
        hsb.grid(row=1, column=0, sticky='ew')
        tree_frm.rowconfigure(0, weight=1)
        tree_frm.columnconfigure(0, weight=1)

    # ---------------- GUI Actions ----------------
    def add_student(self):
        name = self.name_var.get().strip()
        age = self.age_var.get().strip()
        email = self.email_var.get().strip()
        course = self.course_var.get().strip()
        if not name or not email:
            messagebox.showwarning('Validation', 'Name and Email are required')
            return
        try:
            student_id = insert_student(name, int(age) if age else None, email,
course)
            messagebox.showinfo('Success', f'Added student with ID {student_id}')
            self.clear_form()
            self.refresh_tree()
        except sqlite3.IntegrityError:
            messagebox.showerror('Error', 'Email must be unique')
        except Exception as e:
            messagebox.showerror('Error', str(e))

    def update_student(self):
        if not self.selected_student_id:
            messagebox.showwarning('Selection', 'Select a student to update')
            return
        name = self.name_var.get().strip()
        age = self.age_var.get().strip()
        email = self.email_var.get().strip()
        course = self.course_var.get().strip()
        try:
            update_student_record(self.selected_student_id, name, int(age) if age
else None, email, course)
            messagebox.showinfo('Success', 'Record updated')
            self.clear_form()
            self.refresh_tree()
        except sqlite3.IntegrityError:
            messagebox.showerror('Error', 'Email must be unique')
        except Exception as e:
            messagebox.showerror('Error', str(e))

    def delete_student(self):
        if not self.selected_student_id:
            messagebox.showwarning('Selection', 'Select a student to delete')
            return
        if messagebox.askyesno('Confirm', 'Are you sure you want to delete this
student?'):
            delete_student_record(self.selected_student_id)
            messagebox.showinfo('Deleted', 'Record deleted')
            self.clear_form()
            self.refresh_tree()

    def search(self):
        term = self.search_var.get().strip()
        rows = search_students(term)
        self.populate_tree(rows)

    def refresh_tree(self):
        rows = fetch_all_students(order_by='name ASC')
        self.populate_tree(rows)

    def populate_tree(self, rows):
        for i in self.tree.get_children():
            self.tree.delete(i)
        for row in rows:
            self.tree.insert('', 'end', values=row)

    def on_tree_select(self, event):
        sel = self.tree.selection()
        if sel:
            vals = self.tree.item(sel[0])['values']
            self.selected_student_id = vals[0]
            self.name_var.set(vals[1])
            self.age_var.set(vals[2])
            self.email_var.set(vals[3])
            self.course_var.set(vals[4])

    def clear_form(self):
        self.name_var.set('')
        self.age_var.set('')
        self.email_var.set('')
        self.course_var.set('')
        self.selected_student_id = None

    def export_csv(self):
        fp = filedialog.asksaveasfilename(defaultextension='.csv',
filetypes=[('CSV files','*.csv')])
        if not fp:
            return
        try:
            export_to_csv(fp)
            messagebox.showinfo('Export', f'Data exported to {fp}')
        except Exception as e:
            messagebox.showerror('Error', str(e))

    def import_csv(self):
        fp = filedialog.askopenfilename(filetypes=[('CSV files','*.csv')])
        if not fp:
            return
        try:
            import_from_csv(fp)
            messagebox.showinfo('Import', 'Data imported successfully')
            self.refresh_tree()
        except sqlite3.IntegrityError as e:
            messagebox.showerror('Integrity Error', str(e))
        except Exception as e:
            messagebox.showerror('Error', str(e))

    def create_view(self):
        try:
            rows = create_student_view()
            # Show view rows in a popup
            top = tk.Toplevel(self)
            top.title('Student View')
            txt = tk.Text(top, width=80, height=20)
            txt.pack(padx=10, pady=10)
            for r in rows:
                txt.insert('end', f"{r}\n")
        except Exception as e:
            messagebox.showerror('Error', str(e))

    def transaction_demo(self):
        if messagebox.askyesno('Transaction', 'Commit increment ages?
(Yes=commit, No=rollback)'):
            res = transaction_increment_ages(commit=True)
        else:
            res = transaction_increment_ages(commit=False)
        messagebox.showinfo('Transaction', f'Transaction {res}')
        self.refresh_tree()

    def show_count(self):
        total = count_students()
        messagebox.showinfo('Count', f'Total students: {total}')

    def sort_by_name(self):
        rows = fetch_all_students(order_by='name ASC')
        self.populate_tree(rows)

# -------------------- Run App --------------------

if __name__ == '__main__':
    initialize_db()
    app = StudentApp()
    app.mainloop()
```

TOPIC 2:Kivy tutorial

Kivy is a **Python framework** for building **cross-platform GUI/mobile apps**.

Kivy Programs with Explanation and Output

Program 1 – Hello World

```
from kivy.app import App
from kivy.uix.label import Label

class HelloApp(App):
    def build(self):
        return Label(text="Hello Kivy!")

HelloApp().run()
```

Explanation:

- `App` is the main class for all Kivy applications.
- `build()` returns the root widget (`Label` in this case).
- `Label` displays simple text.

Output:

- A window opens with **“Hello Kivy!”** displayed in the center.

Program 2 – Button Click

```
from kivy.app import App
from kivy.uix.button import Button

class ButtonApp(App):
    def build(self):
        btn = Button(text="Click Me!")
        btn.bind(on_press=self.on_click)  # Bind click event
        return btn

    def on_click(self, instance):
        instance.text = "Clicked!"

ButtonApp().run()
```

Explanation:

- `Button` is a clickable widget.
- `bind(on_press=...)` links a function to the button press.
- The `instance` parameter refers to the button clicked.
- Clicking changes the text to “Clicked!”.

Output:

- Button displayed. When clicked, the text changes.

Program 3 – Text Input and Label

```
from kivy.app import App
from kivy.uix.boxlayout import BoxLayout
from kivy.uix.textinput import TextInput
from kivy.uix.label import Label
from kivy.uix.button import Button

class InputApp(App):
    def build(self):
        layout = BoxLayout(orientation='vertical')
        self.input = TextInput(text="")
        self.label = Label(text="Enter your name")
        btn = Button(text="Submit")
        btn.bind(on_press=self.show_text)
        layout.add_widget(self.input)
        layout.add_widget(btn)
        layout.add_widget(self.label)
        return layout

    def show_text(self, instance):
        self.label.text = f"Hello {self.input.text}!"

InputApp().run()
```

Explanation:

- `TextInput` allows user input.
- `BoxLayout(orientation='vertical')` stacks widgets vertically.
- Clicking “Submit” runs `show_text()` which updates the label.

Output:

- Input box + button + label.
- Enter “John”, click submit → label shows “Hello John!”.

Program 4 – Simple Calculator

```
from kivy.app import App
from kivy.uix.boxlayout import BoxLayout
from kivy.uix.textinput import TextInput
from kivy.uix.button import Button
from kivy.uix.label import Label

class CalculatorApp(App):
    def build(self):
        self.layout = BoxLayout(orientation='vertical')
        self.input = TextInput(text="", multiline=False)
        self.label = Label(text="Result:")
        btn = Button(text="Calculate")
        btn.bind(on_press=self.calculate)
        self.layout.add_widget(self.input)
        self.layout.add_widget(btn)
        self.layout.add_widget(self.label)
        return self.layout

    def calculate(self, instance):
        try:
            self.label.text = f"Result: {eval(self.input.text)}"
        except:
            self.label.text = "Invalid Input"

CalculatorApp().run()
```

Explanation:

- `eval()` evaluates math expressions entered by the user.
- Updates the label with the result.
- Handles invalid input with a `try/except`.

Output:

- Enter `5+7`, click calculate → label shows `Result: 12`.

Program 5 – Change Background Color

```
from kivy.app import App
from kivy.uix.button import Button
from kivy.uix.floatlayout import FloatLayout
from kivy.core.window import Window
import random

class BGApp(App):
    def build(self):
        layout = FloatLayout()
        btn = Button(text="Change BG Color", size_hint=(.3,.1),
pos_hint={'x':.35, 'y':.45})
        btn.bind(on_press=self.change_color)
        layout.add_widget(btn)
        return layout

    def change_color(self, instance):
        Window.clearcolor = (random.random(), random.random(),
random.random(), 1)

BGApp().run()
```

Explanation:

- `FloatLayout` allows absolute positioning.
- `Window.clearcolor` changes the background.
- `random.random()` gives a random RGB value.

Output:

- Button in the middle. Click → background changes to random color.

Program 6 – Slider Demo

```
from kivy.app import App
from kivy.uix.slider import Slider
from kivy.uix.label import Label
from kivy.uix.boxlayout import BoxLayout

class SliderApp(App):
    def build(self):
        layout = BoxLayout(orientation='vertical')
        self.label = Label(text="Value: 0")
        self.slider = Slider(min=0, max=100, value=0)
        self.slider.bind(value=self.on_value)
        layout.add_widget(self.slider)
        layout.add_widget(self.label)
        return layout

    def on_value(self, instance, value):
        self.label.text = f"Value: {int(value)}"

SliderApp().run()
```

Explanation:

- Slider lets user select a number.
- `bind(value=...)` updates the label dynamically.

Output:

- Move slider → label shows current value.

Program 7 – CheckBox Demo

```
from kivy.app import App
from kivy.uix.boxlayout import BoxLayout
from kivy.uix.checkbox import CheckBox
from kivy.uix.label import Label

class CheckBoxApp(App):
    def build(self):
        layout = BoxLayout(orientation='vertical')
        self.label = Label(text="Select Option")
        self.checkbox = CheckBox()
        self.checkbox.bind(active=self.on_active)
        layout.add_widget(self.checkbox)
        layout.add_widget(self.label)
        return layout

    def on_active(self, instance, value):
        self.label.text = "Checked!" if value else "Unchecked!"

CheckBoxApp().run()
```

Explanation:

- `CheckBox` can be toggled.
- Label shows “Checked!” or “Unchecked!” based on state.

Output:

- Checkbox. Toggle → label updates.

Program 8 – Switch Widget Demo

```
from kivy.app import App
from kivy.uix.boxlayout import BoxLayout
from kivy.uix.switch import Switch
from kivy.uix.label import Label

class SwitchApp(App):
    def build(self):
        layout = BoxLayout(orientation='vertical')
        self.label = Label(text="Switch OFF")
        self.sw = Switch()
        self.sw.bind(active=self.switch_toggle)
        layout.add_widget(self.sw)
        layout.add_widget(self.label)
        return layout

    def switch_toggle(self, instance, value):
        self.label.text = "Switch ON" if value else "Switch OFF"

SwitchApp().run()
```

Explanation:

- Similar to checkbox, but looks like a toggle switch.
- Updates label dynamically.

Output:

- Toggle switch → label updates.

Program 9 – Image Display

```
from kivy.app import App
from kivy.uix.image import Image

class ImageApp(App):
    def build(self):
        return Image(source='example.png')  # put an image in same folder

ImageApp().run()
```

Explanation:

- Displays an image file in the window.
- File must be in the same directory.

Output:

- Window shows the image.

Program 10 – Grid of Buttons

```
from kivy.app import App
from kivy.uix.gridlayout import GridLayout
from kivy.uix.button import Button

class GridApp(App):
    def build(self):
        layout = GridLayout(cols=3)
        for i in range(1, 10):
            layout.add_widget(Button(text=f"Button {i}"))
        return layout

GridApp().run()
```

Explanation:

- `GridLayout(cols=3)` → 3 columns, auto rows.

- Buttons fill the grid.

Output:

- 3x3 grid of buttons labeled Button 1…9.

Program 11 – Countdown Timer

```
from kivy.app import App
from kivy.uix.label import Label
from kivy.clock import Clock

class TimerApp(App):
    def build(self):
        self.count = 10
        self.label = Label(text=str(self.count), font_size=50)
        Clock.schedule_interval(self.update, 1)  # call every 1 second
        return self.label

    def update(self, dt):
        self.count -= 1
        if self.count >= 0:
            self.label.text = str(self.count)
        else:
            self.label.text = "Time's Up!"

TimerApp().run()
```

Explanation:

- `Clock.schedule_interval()` repeatedly calls a function every specified seconds.
- `update()` decreases the counter.
- Label updates dynamically with remaining time.

Output:

- Window displays a large number starting at 10, counting down each second.
- After 0 → label shows "Time's Up!".

Program 12 – Simple Login App

```
from kivy.app import App
from kivy.uix.boxlayout import BoxLayout
from kivy.uix.textinput import TextInput
from kivy.uix.button import Button
from kivy.uix.label import Label
from kivy.uix.popup import Popup

class LoginApp(App):
    def build(self):
        self.layout = BoxLayout(orientation='vertical', padding=10,
spacing=10)
        self.user = TextInput(hint_text="Username")
        self.pwd = TextInput(hint_text="Password", password=True)
        btn = Button(text="Login")
        btn.bind(on_press=self.login)
        self.layout.add_widget(self.user)
        self.layout.add_widget(self.pwd)
        self.layout.add_widget(btn)
        return self.layout

    def login(self, instance):
        if self.user.text == "admin" and self.pwd.text == "123":
            popup = Popup(title='Success', content=Label(text='Login
Successful!'), size_hint=(0.5,0.5))
        else:
            popup = Popup(title='Error', content=Label(text='Invalid
Credentials'), size_hint=(0.5,0.5))
        popup.open()

LoginApp().run()
```

Explanation:

- Uses `TextInput` for username/password.
- Password hidden with `password=True`.
- `Popup` shows login success/error.
- Checks credentials against hard-coded values.

Output:

- Enter `admin` / `123` → success popup.
- Any other input → error popup.

Program 13 – To-Do List

```
from kivy.app import App
from kivy.uix.boxlayout import BoxLayout
from kivy.uix.textinput import TextInput
from kivy.uix.button import Button
from kivy.uix.label import Label

class ToDoApp(App):
    def build(self):
        self.layout = BoxLayout(orientation='vertical')
        self.input = TextInput(hint_text="Enter task")
        self.label = Label(text="")
        btn = Button(text="Add Task")
        btn.bind(on_press=self.add_task)
        self.layout.add_widget(self.input)
        self.layout.add_widget(btn)
        self.layout.add_widget(self.label)
        self.tasks = []
        return self.layout

    def add_task(self, instance):
        task = self.input.text.strip()
        if task:
            self.tasks.append(task)
            self.label.text = "\n".join(self.tasks)
            self.input.text = ""

ToDoApp().run()
```

Explanation:

- User enters tasks in `TextInput`.
- Click button → task added to `tasks` list.
- Label shows all tasks separated by newlines.

Output:

- Type task → click "Add Task" → task appears below.
- Multiple tasks stack vertically.

Program 14 – Color Picker

```
from kivy.app import App
from kivy.uix.boxlayout import BoxLayout
from kivy.uix.label import Label
from kivy.uix.colorpicker import ColorPicker

class ColorPickerApp(App):
    def build(self):
        layout = BoxLayout(orientation='vertical')
        self.label = Label(text="Pick a color")
        self.cp = ColorPicker()
        self.cp.bind(color=self.on_color)
        layout.add_widget(self.cp)
        layout.add_widget(self.label)
        return layout

    def on_color(self, instance, value):
        self.label.text = f"Selected RGBA: {value}"

ColorPickerApp().run()
```

Explanation:

- `ColorPicker` provides an interactive color selector.
- `bind(color=...)` updates label whenever color changes.
- Label shows RGBA values of the selected color.

Output:

- Color selection area + label showing current RGBA values.
- Drag picker → label updates in real time.

Program 15 – ScrollView with Buttons

```
from kivy.app import App
from kivy.uix.scrollview import ScrollView
from kivy.uix.gridlayout import GridLayout
from kivy.uix.button import Button

class ScrollApp(App):
    def build(self):
        layout = ScrollView()
        grid = GridLayout(cols=1, size_hint_y=None)
        grid.bind(minimum_height=grid.setter('height'))
        for i in range(50):
            grid.add_widget(Button(text=f"Button {i+1}", size_hint_y=None, 
height=40))
        layout.add_widget(grid)
        return layout

ScrollApp().run()
```

Explanation:

- `ScrollView` allows vertical scrolling.
- `GridLayout` contains many buttons stacked vertically.
- `size_hint_y=None` + `bind(minimum_height=...)` ensures scroll works.

Output:

- Window displays a list of 50 buttons.
- Scrollbar allows moving up/down through all buttons.

Tkinter vs Kivy – Detailed Comparison

Feature	Tkinter	Kivy
Library Type	Standard Python GUI library	Open-source Python framework for multi-platform GUI
Platform Support	Mainly **Windows, macOS, Linux**	Cross-platform: **Windows, macOS, Linux, Android, iOS**
Installation	Comes pre-installed with Python	Must install via `pip install kivy`
Programming Style	Procedural / Object-Oriented	Object-Oriented, event-driven, supports KV language
Widgets Available	Basic widgets: `Label`, `Button`, `Entry/TextInput`, `Canvas`, `Checkbutton`, `Radiobutton`, `Scale`	Advanced widgets: `Label`, `Button`, `TextInput`, `Slider`, `Switch`, `ColorPicker`, `Video`, multi-touch gestures
Look & Feel (UI)	Native OS look; simple & classic	Modern, highly customizable, not native; supports graphics, animations
Layouts	Pack, Grid, Place	BoxLayout, GridLayout, FloatLayout, StackLayout, RelativeLayout, AnchorLayout
Graphics Support	Limited; canvas for drawing basic shapes	Strong graphics engine; supports OpenGL ES 2 for animations, touch gestures, custom graphics
Event Handling	Basic event binding (button clicks, key presses)	Advanced event handling: gestures, touch, multi-touch, animation events
Mobile App Support	Not supported	Fully supports Android and iOS apps (via buildozer or pyjnius)
Learning Curve	Easy, beginner-friendly	Moderate, more complex due to layouts, graphics, KV language
Performance	Lightweight; fast for small desktop apps	Heavier, optimized for graphics and mobile; better for complex apps
Community & Resources	Large community, tons of tutorials, stable	Growing community, less resources than Tkinter but modern apps oriented
Use Cases	Small desktop apps, calculators, text editors, CRUD apps	Mobile apps, games, interactive dashboards, apps with gestures/animations

Key Differences in Words

1. **Platform Support**:
   - Tkinter → Desktop only
   - Kivy → Desktop + Mobile (Android/iOS)
2. **UI Design**:
   - Tkinter → Simple, native OS look, easier to design.
   - Kivy → Modern, flexible, supports gestures, multi-touch, and animations.
3. **Ease of Learning**:
   - Tkinter → Very beginner-friendly.
   - Kivy → Moderate; you may need to learn **KV language** for complex layouts.
4. **Widgets & Graphics**:
   - Tkinter → Basic widgets, simple GUI.

- Kivy → Advanced widgets, touch-friendly, animations, scalable for games and apps.

5. **Best Use Case**:
    - Tkinter → Quick desktop apps, tools, GUI prototypes.
    - Kivy → Cross-platform apps, mobile apps, interactive and animated apps.

Example Comparison

Task	Tkinter	Kivy
Simple Calculator	Easy to implement	Can implement, but overkill for basic desktop app
Mobile App (Android)	✗ Cannot	Works
Animated Buttons	✗ Hard	Easy using `Animation`
Multi-touch Gestures	✗ Not supported	Supported
Small Desktop Utility	Ideal	Works but heavier

Summary:

- Use **Tkinter** if you want **desktop-only apps**, **quick prototyping**, **learning GUI basics**, or **small projects**.
- Use **Kivy** if you want **cross-platform apps**, **mobile support**, **modern UI**, **animations**, or **touch/multi-touch interactions**.

Now students are required to Study 5 projects in kivy which are completely solved for their help in doing other project.

1. Smart Home Control App

Description:
An app that allows users to control smart home devices like lights, fans, and AC remotely. It connects to IoT devices using MQTT or HTTP APIs.

Key Features:

- Dashboard showing all connected devices.
- Toggle switches for turning devices on/off.
- Status monitoring (online/offline).
- Scheduling feature for automation.

Implementation Highlights:

- Kivy **GridLayout** for device panels.
- Kivy **Switch** for on/off controls.
- Integration with **paho-mqtt** library for IoT communication.

Expected Output:
A clean, interactive dashboard with a list of devices and toggle switches. Users can turn devices on/off, and the state changes are reflected instantly.

2. Restaurant Ordering System

Description:
An app for restaurants where customers can place orders from a tablet or kiosk interface.

Key Features:

- Menu with categories (Starters, Main Course, Desserts).
- Cart system to add/remove items.
- Order summary with total price.
- Integration with a backend database (SQLite/MySQL) for order tracking.

Implementation Highlights:

- Kivy **ScreenManager** for menu, cart, and checkout screens.
- Kivy **RecycleView** for dynamic menus.
- Database connection for storing order history.

Expected Output:
The user selects items from the menu, adds them to the cart, and sees a summary with total price. The order is stored in the database for kitchen staff.

3. Fitness Tracker App

Description:
A mobile app for tracking workouts, steps, water intake, and calories.

Key Features:

- Step counter using device sensors.
- Daily workout log.
- Graphical display of progress.
- Notifications for hydration and exercise reminders.

Implementation Highlights:

- Kivy **Matplotlib** integration for charts.
- Kivy **MDDatePicker** for logging activities.
- Kivy **Local Storage** for user data.

Expected Output:
Users can input their daily workouts and see graphical charts of their progress over time. Push notifications remind users to stay on track.

4. Real-Time Chat Application

Description:
A cross-platform chat app for businesses or communities.

Key Features:

- User authentication (login/signup).
- Real-time messaging using WebSocket.
- Group and private chat functionality.
- Multimedia support (images, emojis).

Implementation Highlights:

- Kivy **TextInput** for chat messages.
- Kivy **ScrollView** for chat history.
- Async WebSocket communication for real-time updates.

Expected Output:
A smooth, mobile-friendly chat interface where messages appear instantly as they are sent. Users can switch between chat groups and private chats seamlessly.

5. E-Learning Mobile App

Description:
An interactive learning app for students with courses, quizzes, and progress tracking.

Key Features:

- Courses categorized by subjects.
- Video/audio lectures.
- Multiple-choice quizzes with automatic scoring.
- Progress dashboard with certificates on completion.

Implementation Highlights:

- Kivy **ScreenManager** for multiple course screens.
- **Kivy Video widget** for lecture playback.
- SQLite database for tracking student progress.

Expected Output:
Students can select a course, watch lectures, take quizzes, and view a progress dashboard. Completion certificates can be generated automatically.

Summary of Project Outputs:

Project	**Main Output**
Smart Home App	Dashboard with toggle switches for devices
Restaurant Ordering	Menu selection, cart, order summary
Fitness Tracker	Workout logs with charts and reminders
Real-Time Chat	Instant messaging interface with groups
E-Learning App	Courses, quizzes, progress tracking

Project 1: Smart Home Control App

Description:

A dashboard app to control smart home devices. Users can turn devices on/off and see their status.

Python Code (Kivy)

```
from kivy.app import App
from kivy.uix.gridlayout import GridLayout
from kivy.uix.label import Label
from kivy.uix.switch import Switch
from kivy.uix.boxlayout import BoxLayout

class DeviceControl(BoxLayout):
    def __init__(self, device_name, **kwargs):
        super().__init__(orientation='horizontal', **kwargs)
        self.device_name = device_name
        self.label = Label(text=self.device_name, size_hint=(0.7, 1))
        self.switch = Switch(active=False, size_hint=(0.3, 1))
        self.switch.bind(active=self.toggle_device)
        self.add_widget(self.label)
        self.add_widget(self.switch)

    def toggle_device(self, instance, value):
        if value:
            print(f"{self.device_name} turned ON")
        else:
            print(f"{self.device_name} turned OFF")

class SmartHomeApp(App):
    def build(self):
        layout = GridLayout(cols=1, padding=20, spacing=10)
        devices = ["Living Room Light", "AC", "Fan", "Bedroom Light"]
        for device in devices:
            layout.add_widget(DeviceControl(device))
        return layout

if __name__ == "__main__":
    SmartHomeApp().run()
```

Explanation:

- **GridLayout**: Used to list devices vertically.
- **BoxLayout**: Each device has a label and a switch.
- **Switch widget**: Toggles device ON/OFF and prints status to console.
- You can expand this by connecting the switches to real IoT devices via **MQTT**.

Expected Output:

- A vertical list of devices with ON/OFF switches.
- Clicking the switch prints status in the console, e.g.:

```
Living Room Light turned ON
AC turned OFF
```

Project 2: Restaurant Ordering System

Description:

A simple ordering app where customers select menu items, add to cart, and see total bill.

Python Code (Kivy)

```
from kivy.app import App
from kivy.uix.boxlayout import BoxLayout
from kivy.uix.button import Button
from kivy.uix.label import Label

class RestaurantApp(App):
    def build(self):
        self.cart_total = 0
        self.cart_label = Label(text="Cart Total: $0", size_hint=(1, 0.2))
        layout = BoxLayout(orientation='vertical', padding=20, spacing=10)

        menu_items = {
            "Burger": 5,
            "Pizza": 8,
            "Pasta": 6,
            "Coke": 2
        }

        layout.add_widget(Label(text="Menu", font_size=30, size_hint=(1,
0.2)))

        for item, price in menu_items.items():
            btn = Button(text=f"{item} - ${price}", size_hint=(1, 0.2))
            btn.bind(on_press=lambda x, i=item, p=price:
self.add_to_cart(i, p))
            layout.add_widget(btn)

        layout.add_widget(self.cart_label)
        return layout

    def add_to_cart(self, item, price):
        self.cart_total += price
        self.cart_label.text = f"Cart Total: ${self.cart_total}"
        print(f"Added {item} to cart. Total: ${self.cart_total}")

if __name__ == "__main__":
    RestaurantApp().run()
```

Explanation:

- **BoxLayout**: Vertical layout for menu and cart display.
- **Button**: Represents each menu item.
- Clicking a menu item updates the **cart total** label and prints the cart to console.
- Can be extended to **SQLite database** to store order history.

Expected Output:

- Buttons showing menu items with prices.
- Clicking a button updates cart total in real-time.
- Console prints:

```
Added Burger to cart. Total: $5
Added Pizza to cart. Total: $13
```

Project 3: Fitness Tracker App

Description:

A mobile app to log workouts, track steps, water intake, and calories. Users can see their progress in a simple dashboard.

Python Code (Kivy)

```
from kivy.app import App
from kivy.uix.boxlayout import BoxLayout
from kivy.uix.label import Label
from kivy.uix.button import Button
from kivy.uix.textinput import TextInput

class FitnessApp(App):
    def build(self):
        self.workout_log = []
        self.layout = BoxLayout(orientation='vertical', padding=20,
spacing=10)

        self.layout.add_widget(Label(text="Fitness Tracker", font_size=30,
size_hint=(1, 0.2)))

        self.input_workout = TextInput(hint_text="Enter Workout (e.g.,
Running 30 mins)", size_hint=(1, 0.2))
        self.layout.add_widget(self.input_workout)

        add_btn = Button(text="Add Workout", size_hint=(1, 0.2))
        add_btn.bind(on_press=self.add_workout)
        self.layout.add_widget(add_btn)

        self.log_label = Label(text="Workout Log:\n", size_hint=(1, 0.4))
        self.layout.add_widget(self.log_label)

        return self.layout

    def add_workout(self, instance):
        workout = self.input_workout.text
        if workout.strip() != "":
            self.workout_log.append(workout)
            self.log_label.text = "Workout Log:\n" +
"\n".join(self.workout_log)
            self.input_workout.text = ""
            print(f"Workout Added: {workout}")
        else:
            print("No workout entered!")

if __name__ == "__main__":
    FitnessApp().run()
```

Explanation:

- **BoxLayout**: Vertical stacking of widgets.
- **TextInput**: User enters workout details.
- **Button**: Adds workout to log.
- **Label**: Displays all workouts dynamically.
- Prints added workouts in the console.

- Can be extended to **store workouts in SQLite** and visualize calories using **Kivy + Matplotlib**.

Expected Output:

- Text input for workout entry.
- Button “Add Workout”.
- Workout log updates in real-time on the screen.
- Console shows:

```
Workout Added: Running 30 mins
Workout Added: Push-ups 20 reps
```

TOC QUESTIONS SOLUTION VIVA

Project 4: Real-Time Chat App (Simplified)

Description:

A real-time chat interface using **local simulation**. For full industry-level chat, this would integrate **WebSocket or Firebase** for real-time messaging.

Python Code (Kivy)

```
from kivy.app import App
from kivy.uix.boxlayout import BoxLayout
from kivy.uix.scrollview import ScrollView
from kivy.uix.textinput import TextInput
from kivy.uix.button import Button
from kivy.uix.label import Label

class ChatApp(App):
    def build(self):
        self.layout = BoxLayout(orientation='vertical', padding=10,
spacing=10)

        self.chat_log = BoxLayout(orientation='vertical',
size_hint_y=None)
        self.chat_log.bind(minimum_height=self.chat_log.setter('height'))

        scroll = ScrollView(size_hint=(1, 0.8))
        scroll.add_widget(self.chat_log)
        self.layout.add_widget(scroll)

        self.input_box = BoxLayout(size_hint=(1, 0.1))
        self.text_input = TextInput(multiline=False)
        send_btn = Button(text="Send", size_hint=(0.3, 1))
        send_btn.bind(on_press=self.send_message)

        self.input_box.add_widget(self.text_input)
        self.input_box.add_widget(send_btn)
        self.layout.add_widget(self.input_box)

        return self.layout

    def send_message(self, instance):
        message = self.text_input.text
        if message.strip() != "":
            lbl = Label(text=message, size_hint_y=None, height=30)
            self.chat_log.add_widget(lbl)
            self.text_input.text = ""
            print(f"Message Sent: {message}")

if __name__ == "__main__":
    ChatApp().run()
```

Explanation:

- **ScrollView**: Scrollable chat log for messages.
- **TextInput + Button**: Enter and send message.
- **Dynamic Labels**: Each message added as a Label to chat log.
- **Console print**: Shows message sent.
- Can be extended to **Firebase / WebSocket** for real-time multi-user messaging.

Expected Output:

- Scrollable chat window.
- Enter text and click “Send”, message appears in chat log.
- Console shows:

```
Message Sent: Hello!
Message Sent: How are you?
```

Project 5: E-Learning Mobile App

Description:

An interactive learning app where students can select courses, watch lessons, take quizzes, and track their progress.

Python Code (Kivy)

```
from kivy.app import App
from kivy.uix.screenmanager import ScreenManager, Screen
from kivy.uix.boxlayout import BoxLayout
from kivy.uix.button import Button
from kivy.uix.label import Label
from kivy.uix.textinput import TextInput

# ---------- Screens ----------

class HomeScreen(Screen):
    def __init__(self, **kwargs):
        super().__init__(**kwargs)
        layout = BoxLayout(orientation='vertical', padding=20, spacing=10)
        layout.add_widget(Label(text="E-Learning App", font_size=30,
size_hint=(1, 0.2)))

        courses = ["Math", "Science", "History"]
        for course in courses:
            btn = Button(text=course, size_hint=(1, 0.2))
            btn.bind(on_press=lambda x, c=course: self.go_to_course(c))
            layout.add_widget(btn)

        self.add_widget(layout)

    def go_to_course(self, course):
        self.manager.current = "course"
        self.manager.get_screen("course").set_course(course)

class CourseScreen(Screen):
    def __init__(self, **kwargs):
        super().__init__(**kwargs)
        self.course_name = ""
        self.layout = BoxLayout(orientation='vertical', padding=20,
spacing=10)
        self.label = Label(text="", font_size=25, size_hint=(1, 0.2))
        self.layout.add_widget(self.label)

        self.quiz_btn = Button(text="Take Quiz", size_hint=(1, 0.2))
        self.quiz_btn.bind(on_press=self.go_to_quiz)
        self.layout.add_widget(self.quiz_btn)

        self.back_btn = Button(text="Back to Home", size_hint=(1, 0.2))
        self.back_btn.bind(on_press=self.go_home)
        self.layout.add_widget(self.back_btn)

        self.add_widget(self.layout)

    def set_course(self, course):
        self.course_name = course
        self.label.text = f"Welcome to {course} Course!"

    def go_to_quiz(self, instance):
        self.manager.current = "quiz"
        self.manager.get_screen("quiz").set_course(self.course_name)

    def go_home(self, instance):
        self.manager.current = "home"

class QuizScreen(Screen):
    def __init__(self, **kwargs):
        super().__init__(**kwargs)
        self.course_name = ""
        self.layout = BoxLayout(orientation='vertical', padding=20,
spacing=10)
        self.question_label = Label(text="", font_size=20, size_hint=(1,
0.2))
        self.layout.add_widget(self.question_label)

        self.answer_input = TextInput(hint_text="Enter your answer",
size_hint=(1, 0.2))
        self.layout.add_widget(self.answer_input)

        self.submit_btn = Button(text="Submit Answer", size_hint=(1, 0.2))
        self.submit_btn.bind(on_press=self.check_answer)
        self.layout.add_widget(self.submit_btn)

        self.result_label = Label(text="", size_hint=(1, 0.2))
        self.layout.add_widget(self.result_label)

        self.back_btn = Button(text="Back to Course", size_hint=(1, 0.2))
        self.back_btn.bind(on_press=self.go_back)
        self.layout.add_widget(self.back_btn)

        self.add_widget(self.layout)

        self.quiz_data = {
            "Math": {"question": "2 + 2 = ?", "answer": "4"},
            "Science": {"question": "H2O is chemical formula of?",
"answer": "Water"},
            "History": {"question": "Who discovered America?", "answer":
"Columbus"}
        }

    def set_course(self, course):
        self.course_name = course
        self.question_label.text = self.quiz_data[course]["question"]
        self.result_label.text = ""
        self.answer_input.text = ""

    def check_answer(self, instance):
        user_answer = self.answer_input.text.strip()
        correct_answer = self.quiz_data[self.course_name]["answer"]
        if user_answer.lower() == correct_answer.lower():
            self.result_label.text = "Correct!"
        else:
            self.result_label.text = f"Incorrect! Correct answer:
{correct_answer}"
        print(f"Quiz Answer for {self.course_name}: {user_answer}")

    def go_back(self, instance):
        self.manager.current = "course"

# ---------- Screen Manager ----------
```

```
class ELearningApp(App):
    def build(self):
        sm = ScreenManager()
        sm.add_widget(HomeScreen(name="home"))
        sm.add_widget(CourseScreen(name="course"))
        sm.add_widget(QuizScreen(name="quiz"))
        return sm

if __name__ == "__main__":
    ELearningApp().run()
```

Explanation:

- **ScreenManager**: Manages multiple screens (Home → Course → Quiz).
- **HomeScreen**: Displays list of courses.
- **CourseScreen**: Displays course welcome message and button to take quiz.
- **QuizScreen**: Shows one question per course, checks answer, and displays feedback.
- **Dynamic Labels**: Show questions and results.
- **Console Print**: Logs submitted answers.
- Easily extendable: Add multiple questions per course, store progress in SQLite, or integrate video lectures.

Expected Output:

1. **Home Screen:** Buttons for courses: Math, Science, History.
2. **Course Screen:** Welcome message, “Take Quiz” button, “Back to Home” button.
3. **Quiz Screen:**
    - Shows question (e.g., “2 + 2 = ?”).
    - User enters answer, clicks “Submit Answer”.
    - Result displayed: Correct / Incorrect.
    - Back button returns to course screen.
4. **Console Output:**

```
Quiz Answer for Math: 4
Quiz Answer for Science: Water
```

VIVA VOICE QUESTION AND ANSWER

General Python Basics

1. **What is Python?**

   Python is a high-level, interpreted, general-purpose programming language known for readability and rapid development.

2. **What are key features of Python?**

   Dynamic typing, automatic memory management, extensive standard library, portability, and support for multiple paradigms (procedural, OOP, functional).

3. **How do you run a Python script?**

   Use `python script.py` (or `python3 script.py`), or run code in an interactive shell or IDE.

4. **Difference between Python 2 and Python 3?**

   Python 3 has improved Unicode support, `print()` function, integer division behavior, and many modern improvements. Python 2 is deprecated.

5. **What is PEP 8?**

   PEP 8 is Python's style guide recommending conventions for readable code (naming, indentation, line length).

6. **What is a virtual environment and why use it?**

   A virtual environment isolates project dependencies (`venv`, `virtualenv`) to avoid conflicts between projects.

7. **How do you install packages?**

   Use `pip install package_name`. Manage requirements via `pip freeze > requirements.txt`.

8. **What is an interpreter?**

   Software that executes Python bytecode line-by-line; CPython is the standard interpreter.

9. **What is bytecode?**

Intermediate, platform-independent representation compiled from source (`.pyc` files) executed by the Python virtual machine.

10. **What is GIL?**

    The Global Interpreter Lock (GIL) ensures only one native thread executes Python bytecode at a time in CPython, affecting CPU-bound multithreading.

Data Types & Variables

11. **Name built-in data types in Python.**

    `int`, `float`, `complex`, `bool`, `str`, `list`, `tuple`, `set`, `dict`, `bytes`.

12. **Difference between list and tuple.**

    Lists are mutable; tuples are immutable and typically used for fixed collections.

13. **How to create a dictionary?**

    Using `{}` or `dict()`; e.g., `d = {'a':1, 'b':2}`.

14. **What is a set?**

    An unordered collection of unique elements; supports membership tests and mathematical set operations.

15. **Explain mutable vs immutable with examples.**

    Mutable: `list`, `dict` (can change). Immutable: `int`, `str`, `tuple` (cannot change).

16. **How to swap two variables without a temp variable?**

    `a, b = b, a`.

17. **What is type casting?**

    Converting from one type to another using functions like `int()`, `float()`, `str()`.

18. **How to check an object's type?**

    Use `type(obj)` or `isinstance(obj, Type)`.

19. **What are list comprehensions?**

Compact way to build lists: `[x*x for x in range(5)]`.

20. **What is slicing?**

    Extracting subsequence using `start:stop:step` syntax, e.g., `s[1:5:2]`.

Control Flow

21. **Explain `if`, `elif`, `else`.**

    Conditional branching; `elif` is else-if; only one branch executes.

22. **How does a `for` loop work?**

    Iterates over items of an iterable (list, tuple, dict, string, range).

23. **What is `while` loop?**

    Repeats while a condition is true; use `break` and `continue` to control flow.

24. **How to loop over dictionary keys and values?**

    `for k, v in d.items():`

25. **What does `enumerate()` do?**

    Adds a counter to an iterable: `for i, x in enumerate(seq):`.

26. **What is the `zip()` function?**

    Combines multiple iterables into tuples: `zip(a, b)`.

27. **Explain `break` and `continue`.**

    `break` exits loop; `continue` skips to next iteration.

28. **What is `pass` used for?**

    Placeholder statement doing nothing; used to create empty blocks.

29. **How to write a one-line `if` statement?**

    `x = 1 if cond else 0` (ternary expression).

30. **How to loop through a file line-by-line?**

    `with open('file') as f: for line in f: ...`

Functions

31. **How to define a function?**

    `def func(arg1, arg2=default):` followed by indented block and `return`.

32. **What is a lambda function?**

    Anonymous short function: `lambda x: x*x`.

33. **Explain *args and **kwargs.
    - `*args` captures positional args as a tuple; `**kwargs` captures keyword args as a dict.
34. **What is a docstring?**
    - String literal placed at function/module/class start to document it, accessible via `.__doc__`.
35. **What is a generator?**
    - Function with `yield` returning an iterator that generates values lazily.
36. **Difference between `return` and `yield`.**
    - `return` exits and gives value; `yield` yields a value and preserves function state.
37. **What are higher-order functions?**
    - Functions that accept functions as arguments or return functions (e.g., `map`, `filter`).
38. **What is recursion?**
    - Function calling itself; requires base case to avoid infinite recursion.
39. **How to set default mutable arguments safely?**
    - Use `None` and inside function assign a new object: `if lst is None: lst = []`.
40. **What is function annotation?**
    - Optional metadata for parameters and return values: `def f(x: int) -> str:`.

Object-Oriented Programming

41. **What is a class?**
    - A blueprint for creating objects encapsulating data (attributes) and behavior (methods).
42. **What is an object?**
    - An instance of a class.
43. **What are instance and class variables?**
    - Instance vars are per-object (`self.x`); class vars are shared across instances.
44. **Explain `__init__`.**
    - Constructor method called when an object is created to initialize attributes.
45. **What is inheritance?**
    - Mechanism for a class to derive from another, inheriting properties and methods.

46. **What is method overriding?**
    - Redefining a method in a subclass to change behavior.
47. **Explain `super()`.**
    - Calls a method from the parent class, commonly `super().__init__()`.
48. **What are `@staticmethod` and `@classmethod`?**
    - `@staticmethod` has no `self`; `@classmethod` receives the class (`cls`) as first arg.
49. **What is encapsulation?**
    - Hiding internal state and requiring access through well-defined interfaces.
50. **What is polymorphism?**
    - Ability to use a unified interface for different underlying types (duck typing).

Special Methods & Dunder

51. **What is `__str__` vs `__repr__`?**
    - `__str__` is user-friendly string; `__repr__` is unambiguous developer representation.
52. **What is `__len__`?**
    - Special method to make an object respond to `len()`.
53. **How to make an object iterable?**
    - Define `__iter__()` returning an iterator and `__next__()` on the iterator.
54. **What is `__call__`?**
    - Makes an instance callable like a function.
55. **What is operator overloading?**
    - Implementing dunder methods like `__add__`, `__eq__` to define custom behavior.
56. **What is `__enter__` and `__exit__`?**
    - Implement context manager protocol to use `with` statement.
57. **What does `__slots__` do?**
    - Restricts allowed instance attributes and can save memory by avoiding `__dict__`.
58. **What is a descriptor?**
    - Object with `__get__`, `__set__`, `__delete__` used to customize attribute access.
59. **What is multiple inheritance?**
    - Class inherits from more than one parent; MRO (method resolution order) defines leaf-to-root search.
60. **How to prevent inheritance?**
    - Use composition or document as final; Python doesn't have a built-in `final` keyword (use `@final` typing hint).

Modules & Packages

61. **Difference between module and package.**
    - Module is a `.py` file; package is a directory with `__init__.py` (namespace packages possible).
62. **How to import a module?**

- `import module` or `from module import name`.

63. **What is `__name__ == '__main__'` for?**
    - Ensures code runs only when script executed directly, not when imported.
64. **How to create a package?**
    - Create a directory with `__init__.py` and place modules inside.
65. **What are namespace packages?**
    - Packages without `__init__.py` supported since PEP 420.
66. **Explain `pip` and `PyPI`.**
    - `pip` installs packages from Python Package Index (PyPI).
67. **How to create a module-level constant?**
    - Define uppercase variable at module top (convention only).
68. **What is `__all__`?**
    - List of module attributes to export when `from module import *` is used.
69. **How to reload a module?**
    - Use `importlib.reload(module)`.
70. **What is `pkg_resources` / `importlib.metadata` used for?**
    - Query package metadata and entry points.

Exceptions & Error Handling

71. **How to handle exceptions?**
    - Use `try/except` blocks, optionally `else` and `finally`.
72. **How to raise an exception?**
    - Use `raise ExceptionType('message')`.
73. **What is `finally` used for?**
    - Cleanup code that runs regardless of exceptions.
74. **What is `assert` statement?**
    - Used to check conditions during development; raises `AssertionError` if false.
75. **How to create custom exceptions?**
    - Subclass `Exception`: `class MyError(Exception): pass`.
76. **Difference between `Exception` and `BaseException`.**
    - `BaseException` is base for all exceptions; `SystemExit`, `KeyboardInterrupt` inherit from it; generally inherit from `Exception` for custom errors.
77. **What is exception chaining?**
    - Use `raise NewError(...) from original_error` to preserve context.
78. **When to use `except Exception:` vs specific exceptions?**
    - Prefer specific exceptions; broad catches can hide bugs.
79. **How to get traceback?**
    - Use `traceback` module or `exc_info()` in logging.
80. **What is `contextlib.suppress()`?**
    - Helper to suppress specified exceptions in a `with` block.

File Handling

81. **How to open a file?**
    - `open('file.txt', 'r')` or better `with open('file.txt') as f:`.

82. **How to write to a file?**
    - Open with `'w'` or `'a'` and use `f.write()` or `print(..., file=f)`.
83. **How to read all lines?**
    - `lines = f.readlines()` or iterate directly: `for line in f:`.
84. **What is binary mode?**
    - Modes `'rb'` and `'wb'` for reading/writing bytes.
85. **How to handle file paths portably?**
    - Use `pathlib.Path` for OS-independent paths.
86. **How to delete a file?**
    - `Path('file').unlink()` or `os.remove('file')`.
87. **How to read large files without loading into memory?**
    - Iterate line-by-line or read chunks.

File Handling (continued)

88. **How to use temporary files?**
    - Use `tempfile.TemporaryFile()` or `TemporaryDirectory()` for automatic cleanup.
89. **What is pickling?**
    - Serializing Python objects to byte streams using the `pickle` module.
90. **What is unpickling?**
    - Deserializing pickled data back into Python objects.
91. **Is pickle secure for untrusted sources?**
    - No, unpickling untrusted data can execute arbitrary code.
92. **What is JSON in Python?**
    - JavaScript Object Notation; used for lightweight data exchange using `json` module.
93. **How to read a JSON file?**
    - `import json; data = json.load(open('file.json'))`
94. **How to write a JSON file?**
    - `json.dump(data, open('file.json', 'w'), indent=2)`
95. **What is CSV?**
    - Comma-separated values; a common text-based format for tabular data.
96. **How to read a CSV file?**
    - Use `csv.reader()` or Pandas `pd.read_csv()`.
97. **How to write to a CSV file?**
    - Use `csv.writer()` or Pandas `DataFrame.to_csv()`.
98. **How to handle encoding issues in file reading?**
    - Specify encoding explicitly, e.g., `open('file.txt', 'r', encoding='utf-8')`.
99. **What is `os.path` module used for?**
    - For path manipulations like `join`, `split`, `exists`, `dirname`, `basename`.
100. **What is `pathlib`?**
    - Modern object-oriented module for path operations, replacing many `os.path` uses.

Standard Library

101. **What is `os` module used for?**

- To interact with the operating system: environment, directories, and processes.

102. **What is `sys` module used for?**
   - To interact with Python interpreter: arguments, path, exit, version.
103. **What is `math` module used for?**
   - Provides mathematical functions like `sqrt()`, `sin()`, `log()`, constants like `pi`.
104. **What is `random` module used for?**
   - To generate pseudo-random numbers, shuffle sequences, or sample items.
105. **What is `statistics` module?**
   - Provides functions for mean, median, mode, stdev, etc.
106. **What is `shutil` module used for?**
   - For high-level file operations like copying, moving, deleting directories.
107. **What is `glob` module?**
   - Finds file paths matching patterns (wildcards) in directories.
108. **What is `uuid` used for?**
   - To generate universally unique identifiers (UUIDs).
109. **What is `hashlib`?**
   - Provides secure hash and message digest algorithms (SHA256, MD5, etc.).
110. **What is `base64` module used for?**
   - Encoding binary data into printable ASCII and decoding it back.

Functional Programming

111. **What is a lambda function?**
   - An anonymous, single-expression function defined using `lambda` keyword.
112. **Difference between `map` and `filter`?**
   - `map` applies function to each element, `filter` selects elements satisfying a condition.
113. **What is `reduce` function?**
   - Applies a function cumulatively to sequence items to reduce them to a single value.
114. **What are decorators?**
   - Functions that modify behavior of other functions or methods.
115. **Example of a simple decorator.**

```
def deco(func):
    def wrapper():
        print('Before')
        func()
        print('After')
    return wrapper
```

116. **What is `functools.wraps` used for?**
   - To preserve metadata (name, docstring) of decorated functions.
117. **What is `partial` function?**
   - Fixes certain arguments of a function and returns a new callable.
118. **What is a closure?**
   - Inner function that remembers values from enclosing scopes.
119. **How to use `itertools.chain()`?**

- To combine multiple iterables: `chain(list1, list2)`.

120. **What is `itertools.groupby()` used for?**
    - Groups elements of an iterable based on a key function.

Web Development & Databases

121. **How to make an HTTP request in Python?**
    - Use `requests.get(url)` or `urllib.request.urlopen(url)`.
122. **How to parse HTML in Python?**
    - Use `BeautifulSoup` from `bs4` or `lxml` for structured parsing.
123. **What is Flask?**
    - A lightweight web framework for building APIs and web apps.
124. **How to define a route in Flask?**

```
@app.route('/')
def home():
    return 'Hello'
```

125. **What is Django?**
    - A high-level web framework following MVC pattern, includes ORM and admin interface.
126. **Difference between Flask and Django?**
    - Flask is minimal; Django is full-featured with built-in ORM and admin.
127. **What is ORM?**
    - Object Relational Mapping, mapping classes to database tables.
128. **What is SQLAlchemy?**
    - A Python ORM for working with relational databases.
129. **How to connect to SQLite in Python?**

```
import sqlite3
conn = sqlite3.connect('db.sqlite3')
```

130. **How to execute SQL in Python?**

```
c = conn.cursor(); c.execute('SELECT * FROM table')
```

131. **How to prevent SQL injection?**
    - Use parameterized queries: `c.execute('SELECT * FROM users WHERE id=?', (id,))`.
132. **How to fetch all rows from cursor?**
    - `rows = c.fetchall()`.
133. **How to commit changes in DB?**
    - `conn.commit()`.
134. **How to close connection?**
    - `conn.close()`.
135. **What is a NoSQL database?**
    - A non-relational database (e.g., MongoDB) storing data as documents, key-value pairs, etc.
136. **How to connect to MongoDB?**
    - Use `pymongo.MongoClient('mongodb://localhost:27017/')`.

137. **How to insert a document in MongoDB?**
   - `db.collection.insert_one({'name':'Safin'}).`
138. **How to find documents?**
   - `db.collection.find({'name':'Safin'}).`
139. **What is REST API?**
   - Architectural style for web services using HTTP verbs (GET, POST, PUT, DELETE).
140. **How to consume REST API in Python?**
   - Use `requests` module for HTTP calls.

Advanced Python Concepts

141. **What is multithreading?**
   - Running multiple threads for concurrent execution (I/O-bound tasks).
142. **What is multiprocessing?**
   - Running multiple processes for parallel execution (CPU-bound tasks).
143. **What is `asyncio`?**
   - Asynchronous I/O library for cooperative multitasking.
144. **Difference between threading and asyncio.**
   - Threading uses OS threads; asyncio uses event loop and coroutines.
145. **What are coroutines?**
   - Special generator-based functions paused and resumed with `await` and `async`.
146. **What is a context manager?**
   - Object implementing `__enter__` and `__exit__`, used in `with` statement.
147. **What is `concurrent.futures`?**
   - High-level API for threading and multiprocessing with executors.
148. **What is `queue.Queue` used for?**
   - Thread-safe FIFO queue for producer-consumer problems.
149. **What is memory management in Python?**
   - Automatic garbage collection using reference counting and cyclic GC.
150. **What is weak reference?**
   - Reference that doesn't increase object's reference count (`weakref` module).
151. **What are metaclasses?**
   - Classes of classes; they control class creation behavior.
152. **What is monkey patching?**
   - Dynamically modifying or extending code at runtime.
153. **What is duck typing?**
   - Type determined by behavior rather than explicit inheritance.
154. **What is `__slots__` used for?**
   - Restricts instance attributes, reduces memory usage.
155. **What is type hinting?**
   - Optional annotations indicating expected data types for better readability.
156. **How to enforce type hints?**
   - Use tools like `mypy` or runtime validation libraries like `pydantic`.
157. **What is data class?**
   - A class automatically generating init, repr, eq using `@dataclass` decorator.
158. **How to freeze dataclass?**
   - Use `@dataclass(frozen=True)` to make it immutable.
159. **What is `__annotations__`?**

- Stores type hint metadata as a dictionary.

160. **What is dependency injection?**
   - Supplying external resources to components instead of hardcoding them.

Data Science & Visualization

161. **What is NumPy?**
   - Library for numerical computing and multi-dimensional arrays.
162. **What is Pandas?**
   - Library for data manipulation and analysis using DataFrames.
163. **How to read CSV in Pandas?**
   - `pd.read_csv('file.csv')`.
164. **How to filter DataFrame rows?**
   - `df[df['col'] > 10]`.
165. **How to handle missing data?**
   - Use `df.fillna()` or `df.dropna()`.
166. **What is Matplotlib?**
   - Plotting library for data visualization.
167. **How to plot a line chart?**
   - `plt.plot(x, y)` then `plt.show()`.
168. **What is Seaborn?**
   - Visualization library built on top of Matplotlib for statistical plots.
169. **What is Scikit-learn?**
   - Machine learning library providing models and tools for preprocessing.
170. **How to split data into train and test sets?**
   - `train_test_split(X, y, test_size=0.2)`.
171. **How to scale data?**
   - Use `StandardScaler()` or `MinMaxScaler()`.
172. **What is TensorFlow?**
   - Open-source deep learning library from Google.
173. **What is PyTorch?**
   - Deep learning library from Facebook emphasizing dynamic computation graphs.
174. **What is overfitting?**
   - Model performs well on training but poorly on unseen data.
175. **How to prevent overfitting?**
   - Use regularization, dropout, cross-validation.
176. **What is cross-validation?**
   - Splitting data into multiple folds to test model generalization.
177. **What is confusion matrix?**
   - Table summarizing correct/incorrect classifications in supervised learning.
178. **What is accuracy?**
   - Ratio of correctly predicted samples to total samples.
179. **What is precision and recall?**
   - Precision: TP/(TP+FP); Recall: TP/(TP+FN).
180. **What is F1-score?**
   - Harmonic mean of precision and recall.

Miscellaneous & Tools

181. **What is virtualenv?**
    - Tool for creating isolated Python environments.
182. **What is pipenv?**
    - Combines package management and virtual environments.
183. **What is poetry?**
    - Modern dependency manager and packaging tool.
184. **What is Jupyter Notebook?**
    - Interactive environment for data analysis and visualization.
185. **What is IPython?**
    - Enhanced interactive Python shell.
186. **What is black formatter?**
    - Auto-formats Python code for consistency.
187. **What is pylint?**
    - Static code analysis tool for linting and enforcing coding standards.
188. **What is pytest?**
    - Testing framework supporting fixtures and assertions.
189. **How to mock functions during testing?**
    - Use `unittest.mock` module.
190. **What is CI/CD?**
    - Continuous integration/deployment pipelines automating build, test, release.
191. **What is Docker used for?**
    - Containerizing apps for consistent environments.
192. **What is Git?**
    - Distributed version control system.
193. **What is GitHub?**
    - Platform for hosting repositories and collaboration.
194. **How to create a virtual environment?**
    - `python -m venv env` then `source env/bin/activate`.
195. **What is requirements.txt?**
    - File listing project dependencies for reproducibility.
196. **How to freeze dependencies?**
    - `pip freeze > requirements.txt`.
197. **What is logging module used for?**
    - Captures runtime logs, supports multiple severity levels.
198. **How to measure execution time?**
    - Use `time.time()` or `timeit` module.
199. **How to profile code performance?**
    - Use `cProfile` or `line_profiler`.
200. **How to compile Python to executable?**
    - Use `pyinstaller script.py` or `cx_Freeze`.