

List in Python

1. Write a Python program to input 8 numbers into a list and calculate the sum and average.

The program should:

- Accept 8 numeric values from the user.
- Store them in a list.
- Calculate total sum of all elements.
- Calculate average using:
 - $\text{Average} = \text{Sum} / \text{Number of Elements}$
- Display list, total sum, and average clearly.
- Demonstrate list processing and arithmetic operations.

2. Write a Python program to count how many even and odd numbers are present in a list.

The program should:

- Take multiple integer inputs from the user.
- Store numbers in a list.
- Traverse the list using a loop.
- Check each element:
 - Even $\rightarrow$ divisible by 2
 - Odd $\rightarrow$ not divisible by 2
- Count and display total even and odd numbers.
- Demonstrate conditional checking with list iteration.

3. Write a Python program to append, insert a new element into an existing list and remove a specific element from a list entered by user.

The program should:

- Create a list with initial values.
- Take a new element as input from the user, Use `append()` method to add the element.
- Take: New element value and position/index from user, Use `insert()` method to place element at desired position.
- Display updated list.
- Ask user for element to remove, Use `remove()` method to delete the specified item, Handle cases where element is not found.
- Display updated list after removal.

4. Write a Python program to sort list elements in ascending and descending order.

The program should:

- Take multiple numeric values as input.
- Store them in a list.
- Sort elements:
 - Ascending order
 - Descending order
- Use sorting techniques or built-in methods.
- Display sorted results clearly.
- Demonstrate ordering operations in lists.

5. Write a Python program to remove duplicate elements from a list and display only unique elements.

The program should:

- Take multiple elements as input from the user.
- Store them in a list.
- Identify repeated elements.
- Remove duplicate values while keeping unique elements.
- Display original list and updated unique list.
- Demonstrate duplicate filtering in Python lists.

6. Write a Python program to count the frequency of each element present in a list.

The program should:

- Take multiple values as input into a list.
- Traverse the list using loops.
- Count how many times each element appears.
- Display frequency of every element.
- Demonstrate counting and data analysis using lists.

7. Write a Python program to perform addition of two matrices using nested lists.

The program should:

- Take two matrices as input from the user.
- Store matrices using nested lists (2D lists).
- Ensure both matrices have the same dimensions.

- Perform element-wise addition.
- Display the resulting matrix in proper row-column format.
- Demonstrate matrix operations using lists.

8. Write a Python program to perform multiplication of two matrices using nested lists.

The program should:

- Accept two matrices from the user.
- Verify compatibility for multiplication (columns of first = rows of second).
- Use nested loops for multiplication logic.
- Store result in a new matrix (nested list).
- Display the final multiplied matrix.
- Demonstrate complex list-based computations.

9. Write a Python program to find all prime numbers stored in a list.

The program should:

- Take a list of numbers as input.
- Check each number for primality.
- Extract and display only prime numbers.
- Display final list of prime numbers.
- Demonstrate filtering and condition checking in lists.

10. Write a Python program to implement stack operations using lists.

The program should:

- Implement stack operations:
 - Push (insert element)
 - Pop (remove element)
 - Display stack elements
- Follow LIFO (Last In First Out) principle.
- Handle underflow conditions (empty stack).
- Display stack after each operation.
- Demonstrate real-world stack behavior using lists.

11. Write a Python program to implement queue operations using lists.

The program should:

- Implement queue operations:
 - Insert (enqueue)
 - Delete (dequeue)

- Display queue
- Follow FIFO (First In First Out) principle.
- Handle empty queue conditions.
- Display queue after each operation.
- Demonstrate real-world queue behavior using lists.

Tuple in Python

12. Write a Python program to calculate the sum and average of elements stored in a tuple.

The program should:

- Take multiple numeric values as input from the user.
- Store the values inside a tuple.
- Calculate the total sum of all tuple elements.
- Compute the average using:
 - $\text{Average} = \text{Sum} / \text{Number of Elements}$
- Display tuple elements, total sum, and average clearly.
- Demonstrate tuple traversal and arithmetic operations.

13. Write a Python program to sort tuple elements in ascending order.

The program should:

- Take multiple numeric values from the user.
- Store them in a tuple.
- Convert tuple into a list if required for sorting.
- Sort the elements in ascending order.
- Display original tuple and sorted tuple.
- Demonstrate tuple manipulation and sorting techniques.

14. Write a Python program to slice tuple elements using different indexes.

The program should:

- Create a tuple with multiple elements.
- Display slices using:
 - Starting index
 - Ending index
 - Step value
- Show different portions of the tuple.
- Demonstrate tuple slicing techniques in Python.

15. Write a Python program to create and manage student records using tuples.

The program should:

- Store student information such as:
 - Roll number
 - Name
 - Class
 - Marks
- Use tuples to maintain records.
- Display all student records in a formatted manner.
- Allow searching of records based on roll number or name.
- Demonstrate immutable data storage using tuples.

Dictionary in Python

16. Write a Python program to calculate the sum of all values stored in a dictionary.

The program should:

- Create a dictionary containing numeric values.
- Traverse all dictionary values using loops or methods.
- Add all values together.
- Display dictionary contents and total sum.
- Demonstrate dictionary traversal and arithmetic operations.

17. Write a Python program to create a dictionary from two separate lists.

The program should:

- Take one list containing keys.
- Take another list containing corresponding values.
- Combine both lists to form a dictionary.
- Display generated dictionary clearly.
- Demonstrate mapping of keys and values using dictionaries.

18. Write a Python program to create a student mark-sheet using nested dictionaries.

The program should:

- Store details of multiple students.
- Use nested dictionaries to store:
 - Student name
 - Subject names
 - Marks
- Calculate total marks and percentage if required.
- Display formatted student marksheet.
- Demonstrate nested dictionary structures in Python.

19. Write a Python program to create a dictionary using dictionary comprehension.

The program should:

- Generate dictionary items dynamically using comprehension syntax.
- Create keys and values using loops or mathematical expressions.
- Display resulting dictionary.
- Demonstrate concise dictionary creation techniques in Python.

20. Write a Python program to create an employee management system using dictionaries.

The program should:

- Store employee details such as:
 - Employee ID
 - Name
 - Department
 - Salary
- Use dictionaries to organize employee records.
- Allow operations such as:
 - Add employee
 - Update employee details // `student[key] = value, data.update({"b": 2})`
 - Delete employee //delete by key: `del student["age"]`
 - Search employee
 - Display all employee records
- Use loops and conditional statements for menu-driven functionality.
- Demonstrate practical use of dictionaries for record management systems.

21. Write a Python program to create a phonebook application using dictionaries.

The program should:

- Store contact details such as:
 - Person name
 - Phone number
 - Address (optional)
- Use dictionary keys for contact names.
- Provide menu options to:
 - Add contact
 - Search contact
 - Update contact
 - Delete contact
 - Display all contacts
- Demonstrate real-world use of dictionaries for contact management.

Set in Python

22. Write a Python program to add elements into a set and display the updated set.

The program should:

- Create a set with initial elements.
- Take a new element as input from the user.
- Add the element using the `add()` method.
- Display the updated set after insertion.
- Demonstrate dynamic modification of sets.

23. Write a Python program to remove elements from a set.

The program should:

- Create a set with user-defined values.
- Ask the user for the element to remove.
- Remove the element using:
 - `remove()` or `discard()` method
- Display the updated set.
- Handle cases where the element does not exist.
- Demonstrate deletion operations in sets.

24. Write a Python program to check whether an element exists in a set.

The program should:

- Create a set using user input.
- Ask the user for an element to search.
- Use membership operators (`in` / `not in`) to check existence.
- Display appropriate messages:
 - “Element found”
 - or “Element not found”

25. Write a Python program to perform the union operation between two sets.

The program should:

- Create two sets with user-defined elements.
- Perform union operation using:
 - `union()` method or `|` operator
- Display:
 - First set
 - Second set
 - Resulting union set
- Demonstrate combining elements from multiple sets.

26. Write a Python program to perform the intersection operation between two sets.

The program should:

- Create two sets using user input.
- Find common elements using:
 - `intersection()` method or `&` operator
- Display the resulting intersection set.
- Demonstrate common element identification using sets.

27. Write a Python program to perform the difference operation between two sets.

The program should:

- Create two sets with user-defined values.
- Find elements present in one set but not in the other.
- Use:
 - `difference()` method or `-` operator

- Display resulting difference set.
- Demonstrate exclusion operations using sets.

28. Write a Python program to remove duplicate values from a list using a set.

The program should:

- Take multiple elements from the user and store them in a list.
- Convert the list into a set.
- Display:
 - Original list with duplicates
 - Updated collection without duplicates
- Demonstrate practical use of sets for duplicate removal.

29. Write a Python program to perform the symmetric difference operation between two sets.

The program should:

- Create or accept two sets from the user.
- Find elements that are present in either set but not in both.
- Use symmetric difference operation (`^` or `symmetric_difference()`).
- Display:
 - Original sets
 - Resulting symmetric difference set
- Demonstrate exclusive set operations in Python.

30. Write a Python program to create a frozen set and demonstrate its properties.

The program should:

- Create a frozen set using `frozenset()`.
- Display elements of the frozen set.
- Attempt operations such as add/remove and observe behavior.
- Explain immutability property of frozen sets.
- Demonstrate difference between normal sets and frozen sets.

31. Write a Python program to perform mathematical set operations interactively.

The program should:

- Accept two sets from the user.
- Display menu options:

- Union
 - Intersection
 - Difference
 - Symmetric Difference
 - Subset Check
- Perform operations repeatedly using loops.
- Display results after every operation.
- Demonstrate interactive mathematical set processing.

32. Write a Python program to identify common and unique characters between two strings using sets.

The program should:

- Take two strings as input.
- Convert characters into sets.
- Display:
 - Common characters
 - Unique characters from first string
 - Unique characters from second string
- Demonstrate string comparison using set operations.