Unit 5: List, Tuple, Set, Dictionary & String

Python Programming (DI02032011)

Milav Dabgar

Lecturer, Dept of ICT
GP Palanpur

April 2, 2026

Unit 5 — Contents

List — Introduction

Definition

A **list** is an ordered, mutable (changeable) collection that can hold items of different data types. Created using `[ ]`.

Key properties:

- **Ordered** — items maintain insertion order
- **Mutable** — can add, remove, or change items
- **Allows duplicates** — same value can appear multiple times
- **Heterogeneous** — can mix different data types

Creating lists:

- `fruits = ["apple", "banana", "cherry"]`
- `nums = [1, 2, 3, 4, 5]`
- `mixed = [1, "hello", 3.14, True]`
- `empty = []`

List — Indexing & Slicing (W'23 Q.5b OR, W'25 Q.5c OR)

```
nums = [10, 20, 30, 40, 50]
# Index:   0   1   2   3   4
# Neg:    -5  -4  -3  -2  -1

# Indexing
print(nums[0])     # 10
print(nums[-1])    # 50  (last element)

# Slicing: [start : stop : step]
print(nums[1:4])   # [20, 30, 40]
print(nums[:3])    # [10, 20, 30]
print(nums[::2])   # [10, 30, 50]
print(nums[::-1]) # [50, 40, 30, 20, 10]
```

List — Built-in Functions (W'25 Q.4a, W'24 Q.4b OR)

```python
nums = [3, 1, 4, 1, 5, 9, 2, 6]

print(len(nums))          # 8
print(max(nums))          # 9
print(min(nums))          # 1
print(sum(nums))          # 31
print(sum(nums)/len(nums)) # avg: 3.875
nums.sort()
print(nums)   # [1,1,2,3,4,5,6,9]
print(nums.index(4))    # 3
print(nums.count(1))    # 2
nums.reverse()
print(nums)   # [9,6,5,4,3,2,1,1]
```

List — Methods (Su'24 Q.4c, Su'25 Q.4c)

```
nums = [1, 2, 3]

nums.append(4)          # add to end
print(nums)             # [1, 2, 3, 4]

nums.insert(1, 99)      # insert at index 1
print(nums)             # [1, 99, 2, 3, 4]

nums.remove(99)         # remove first 99
print(nums)             # [1, 2, 3, 4]

popped = nums.pop()     # remove last
print(popped)           # 4
print(nums)             # [1, 2, 3]

nums.extend([5, 6])     # add multiple
print(nums)             # [1, 2, 3, 5, 6]
```

Nested List (W'23 Q.5b, W'23 Q.3b OR)

List inside a List

A **nested list** is a list that contains other lists as elements.

```
matrix = [[1, 2, 3],
          [4, 5, 6],
          [7, 8, 9]]

print(matrix[0])        # [1, 2, 3]
print(matrix[1][2])     # 6 (row 1, col 2)

# Display all elements
for row in matrix:
    for item in row:
        print(item, end=" ")
    print()
```

List Programs (Su'23 Q.5a, W'23 Q.5a)

Swap two elements

```
lst = [10, 20, 30, 40]
# Swap first and last
lst[0], lst[-1] = lst[-1], lst[0]
print(lst)  # [40, 20, 30, 10]
```

Sum of all elements

```
lst = [1, 2, 3, 4, 5]
total = 0
for num in lst:
    total += num
print("Sum:", total)    # Sum: 15
# Or: print(sum(lst))
```

List: Smallest Number (Su'25 Q.5a)

Find smallest in: [21

]

```
List1 = [21, 6, 3, 18, 12, 15]

# Method 1: using min()
print("Smallest:", min(List1))  # 3

# Method 2: using sort
List1.sort()
print("Smallest:", List1[0])    # 3

# Method 3: manual
smallest = List1[0]
for num in List1:
    if num < smallest:
        smallest = num
print("Smallest:", smallest)    # 3
```

List: Prime and Non-Prime (W'24 Q.5c OR)

Lists of primes and non-primes 1 to 50

```
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

primes = []
non_primes = []
for i in range(1, 51):
    if is_prime(i):
        primes.append(i)
    else:
        non_primes.append(i)

print("Primes:", primes)
print("Non-primes:", non_primes)
```

String — Introduction & Indexing (Su'25 Q.4a OR)

String in Python

A **string** is an ordered, immutable sequence of characters. Created using ' ' or " ".

```
s = "Python"
# Index:  0  1  2  3  4  5
# Neg:   -6 -5 -4 -3 -2 -1

print(s[0])       # P
print(s[-1])      # n
print(s[1:4])     # yth
print(s[::-1])    # nohtyP  (reverse)
print(len(s))     # 6
```

String — Operations (Su'23 Q.4c, W'23 Q.5c)

```
a = "Hello"
b = "World"

# Concatenation
print(a + " " + b)    # Hello World

# Repetition
print(a * 3)          # HelloHelloHello

# Membership
print("ell" in a)     # True
print("xyz" in a)     # False

# Comparison
print("apple" < "banana")  # True
```

String — Built-in Methods (W'24 Q.5a, W'25 Q.3a)

```
s = "Hello World"

print(s.upper())           # HELLO WORLD
print(s.lower())           # hello world
print(s.replace("World","Python")) # Hello Python
print(s.count("l"))        # 3
print(s.split())           # ['Hello', 'World']
print(s.strip())           # removes whitespace
print(s.startswith("He"))# True
print(s.endswith("ld"))    # True
print(s.find("World"))     # 6
print(s.isdigit())         # False
print("abc".isalpha())     # True
```

String Program: Vowel Count (Su'24 Q.4c OR, W'25 Q.4a OR)

Count vowels

```
s = input("Enter string: ")
vowels = consonants = upper = lower = 0

for ch in s:
    if ch in "aeiouAEIOU":
        vowels += 1
    elif ch.isalpha():
        consonants += 1
    if ch.isupper():
        upper += 1
    elif ch.islower():
        lower += 1

print("Vowels:", vowels)
print("Consonants:", consonants)
print("Uppercase:", upper)
print("Lowercase:", lower)
```

String Program: Substring Check (Su'24 Q.5b)

Check if substring is present

```
main_str = input("Enter main string: ")
sub_str  = input("Enter substring: ")

# Method 1: using 'in'
if sub_str in main_str:
    print("Substring found!")
else:
    print("Substring not found.")

# Method 2: using find()
pos = main_str.find(sub_str)
if pos != -1:
    print("Found at index:", pos)
else:
    print("Not found")
```

Tuple — Introduction & Operations (W'24 Q.5b)

What is a Tuple?

A **tuple** is an ordered, **immutable** (unchangeable) sequence. Created using ( ).

```
t = (10, 20, 30, 40, 50)

print(t[0])         # 10
print(t[-1])        # 50
print(t[1:3])       # (20, 30)
print(t + (60,))    # concat: (10,20,...,60)
print(t * 2)        # repeat
print(len(t))       # 5
print(max(t))       # 50
print(min(t))       # 10
print(20 in t)      # True
```

Nested Tuple (Su'25 Q.4a)

Nested Tuple

A **nested tuple** is a tuple that contains other tuples as elements.

```
nested = ((1, 2), (3, 4), (5, 6))

print(nested[0])        # (1, 2)
print(nested[1][0])     # 3
print(nested[2][1])     # 6

# Traverse
for t in nested:
    for val in t:
        print(val, end=" ")
    print()
```

Tuple — Methods & Functions Table (Su'23 Q.5c, Su'24 Q.5c)

Function/Method	Description
`len(t)`	Number of elements
`max(t)`	Largest element
`min(t)`	Smallest element
`sum(t)`	Sum of elements
`t.count(x)`	Count occurrences of x
`t.index(x)`	Index of first x
`sorted(t)`	Returns sorted list
`x in t`	Membership test

Note: Tuples are *immutable* — no `append()`, `remove()` or `sort()` in place.

List vs Tuple

List	Tuple
Mutable (can change)	Immutable (cannot change)
Created with `[ ]`	Created with `( )`
Slower than tuple	Faster than list
More memory usage	Less memory usage
`append()`, `remove()` etc.	Only `count()`, `index()`
Used for dynamic data	Used for fixed data

Set — Introduction & Operations (Su'23 Q.5c OR, Su'25 Q.5c)

What is a Set?

A **set** is an unordered, mutable collection with **no duplicate** elements. Created using `{ }` or `set()`.

```
s = {1, 2, 3, 4, 5}
print(s)                    # unordered output
s.add(6)                    # add element
s.remove(3)                 # remove element
s.discard(10)               # remove (no error if absent)

# Set operations
A = {1, 2, 3, 4}
B = {3, 4, 5, 6}
print(A | B)    # union: {1,2,3,4,5,6}
print(A & B)    # intersection: {3,4}
print(A - B)    # difference: {1,2}
print(A ^ B)    # sym.diff: {1,2,5,6}
```

Set — Functions & Properties

Method/Operation	Description
`s.add(x)`	Add element x
`s.remove(x)`	Remove x (error if absent)
`s.discard(x)`	Remove x (no error)
`s.clear()`	Remove all elements
`s.pop()`	Remove and return arbitrary item
`A | B`	Union
`A & B`	Intersection
`A - B`	Difference
`A ^ B`	Symmetric difference
`A.issubset(B)`	Is A a subset of B?

Dictionary — Introduction (W'24 Q.5b OR)

What is a Dictionary?

A **dictionary** stores data as **key-value pairs**. Keys must be unique and immutable. Created using `{ key: value }`.

```
student = {"name": "Alice",
           "roll": 101,
           "marks": 85}

print(student["name"])          # Alice
print(student.get("marks"))     # 85

student["marks"] = 90           # update
student["city"] = "Surat"       # add new key

del student["city"]             # delete key
print(len(student))             # 3
```

Dictionary — Methods (W'24 Q.5b OR)

```
d = {"a": 1, "b": 2, "c": 3}

print(d.keys())     # dict_keys(['a','b','c'])
print(d.values())   # dict_values([1, 2, 3])
print(d.items())    # dict_items([('a',1),...])

# Traverse
for key, val in d.items():
    print(key, "->", val)

# Check membership
print("a" in d)     # True (checks keys)

d.update({"d": 4}) # add/update
d.pop("b")          # remove key 'b'
print(d)            # {'a':1,'c':3,'d':4}
```

Dictionary Program: Students Above 70 (W'25 Q.5a OR)

Dictionary of students; display those with marks > 70

```
n = int(input("Enter number of students: "))
students = {}

for i in range(n):
    roll = int(input("Roll: "))
    name = input("Name: ")
    marks = int(input("Marks: "))
    students[roll] = {"name": name,
                      "marks": marks}

print("\nStudents with marks > 70:")
for roll, info in students.items():
    if info["marks"] > 70:
        print(info["name"], "-", info["marks"])
```

PYQ Practice — 3-Mark Questions

3-Mark PYQ Bank

1. Write Python code to swap two elements in a list *(Su'23, W'23, Su'24, Su'25 OR)*
2. Write Python code to find sum of all elements in a list *(Su'23 OR, W'23 OR, Su'24 OR)*
3. Define list and explain how it is created in Python *(W'24 OR)*
4. Find the smallest number in `[21, 6, 3, 18, 12, 15]` *(Su'25)*
5. List any six built-in functions of List *(W'25)*
6. Develop Python program to find multiplication of all elements in a list *(W'25)*
7. List any six built-in functions of string *(W'25)*
8. Explain any one String Operation with example *(W'25 OR)*
9. Explain string methods: `count()`, `upper()`, `replace()` *(W'24)*
10. Explain indexing in string with example *(Su'25 OR)*
11. Develop Python program to count number of vowels in a string *(W'25 OR)*
12. Explain concept of nested tuple with example *(Su'25)*

PYQ Practice — 4-Mark Questions

4-Mark PYQ Bank

1. Develop code to create a nested list and display its elements *(W'23 OR)*
2. Explain nested list by giving example *(W'23)*
3. Explain indexing and slicing operations in Python list *(W'23 OR)*
4. Discuss list functions: `len()`, `sum()`, `sort()`, `index()` *(W'24 OR)*
5. Explain slicing of string with suitable example *(W'24)*
6. Write Python program to check if substring is present in a string *(Su'24)*
7. Develop program to demonstrate any 4 built-in functions for string *(Su'25)*
8. Explain tuple operations with example *(W'24)*
9. Develop program to demonstrate any 4 built-in functions for tuple *(Su'25 OR)*
10. Write program to demonstrate set functions and operations *(Su'24 OR)*
11. Explain dictionary built-in functions and methods *(W'24 OR)*
12. Create dictionary with roll, name, marks; display names with marks $>$ 70 *(W'25 OR)*

PYQ Practice — 7-Mark Questions

7-Mark PYQ Bank

1. List various list operations and explain any three *(Su'23 OR)*
2. Explain List Methods and its built-in Functions *(Su'24)*
3. Explain List operations with examples *(Su'25)*
4. Develop Python code to create list of prime and non-prime numbers 1–50 *(W'24 OR)*
5. Explain Indexing and Slicing in List with example *(W'25 OR)*
6. List various string operations and explain any three *(Su'23)*
7. Explain string operations with examples *(W'23)*
8. Count and display vowels, consonants, uppercase and lowercase in string *(Su'24 OR)*
9. Demonstrate tuple functions and operations *(Su'23)*
10. Explain tuple in brief with necessary example *(W'23 OR)*
11. Explain tuple Operations, Functions and Methods *(Su'24)*
12. List set functions and operations; develop program demonstrating them *(Su'23 OR, Su'25, W'25)*
13. Explain Dictionary operations and methods with examples *(W'23, W'24, Su'25, W'25)*

Worked: String Built-in Functions (Su'25 Q.5b)

Demonstrate 4 built-in string functions

```
s = "  Hello, Python World!  "

# 1. strip() - remove leading/trailing spaces
print(s.strip())            # Hello, Python World!

# 2. upper() / lower()
print(s.strip().upper())  # HELLO, PYTHON WORLD!

# 3. split() - split into list
words = s.strip().split()
print(words)  # ['Hello,', 'Python', 'World!']

# 4. replace() - replace substring
print(s.replace("World", "Class"))
```

Worked: Dictionary Operations (W'23 Q.5c, Su'25 Q.5c)

Dictionary creation and operations

```
# Create dictionary
marks = {"Maths": 85, "Science": 90,
         "English": 78}

# Access and update
print(marks["Maths"])       # 85
marks["Science"] = 95       # update
marks["Hindi"] = 80         # add

# Keys, values, items
print(list(marks.keys()))
print(list(marks.values()))

# Iterate
for sub, m in marks.items():
    print(sub, ":", m)

# Delete and clear
marks.pop("English")
```

Worked: Set Operations (W'25 Q.4c)

Demonstrate set operations and functions

```
A = {1, 2, 3, 4, 5}
B = {4, 5, 6, 7, 8}

print("Union:", A | B)
# {1, 2, 3, 4, 5, 6, 7, 8}

print("Intersection:", A & B)
# {4, 5}

print("Difference A-B:", A - B)
# {1, 2, 3}

print("Sym.Diff:", A ^ B)
# {1, 2, 3, 6, 7, 8}

A.add(10)
A.remove(1)
print("Updated A:", A)
print("Is B subset of A?", B.issubset(A))
```

List Comprehension (W'25, Su'25 — 4M)

Syntax

```
[expression for item in iterable if condition]
```

```python
# Squares of 1-10
squares = [x**2 for x in range(1, 11)]
print(squares)    # [1,4,9,16,25,36,49,64,81,100]

# Even numbers 1-20
evens = [x for x in range(1, 21) if x % 2 == 0]
print(evens)      # [2, 4, 6, ..., 20]

# Uppercase each word
words = ["python", "java", "c", "html"]
upper = [w.upper() for w in words]
print(upper)      # ['PYTHON', 'JAVA', 'C', 'HTML']

# Filter marks above 60
marks = [55, 72, 41, 88, 35, 65, 90]
passed = [m for m in marks if m >= 40]
print(passed)     # [55, 72, 41, 88, 65, 90]
```

Dictionary Comprehension (W'25 OR — 4M)

```python
# Create dict from list
nums = [1, 2, 3, 4, 5]
squares_dict = {n: n**2 for n in nums}
print(squares_dict)
# {1: 1, 2: 4, 3: 9, 4: 16, 5: 25}

# Swap keys and values
original = {'a': 1, 'b': 2, 'c': 3}
inverted = {v: k for k, v in original.items()}
print(inverted)   # {1: 'a', 2: 'b', 3: 'c'}

# Filter: only subjects passed
marks = {'Math':85, 'Sci':38, 'Eng':72, 'CS':91}
passed = {sub: m for sub, m in marks.items() if m>=40}
print(passed)   # {'Math': 85, 'Eng': 72, 'CS': 91}
```

List: Sorting, Searching & Copying (W'24, Su'25 — 4M)

```
nums = [64, 25, 12, 22, 11]

# sort() --- modifies in place
nums.sort()
print(nums)          # [11, 12, 22, 25, 64]
nums.sort(reverse=True)
print(nums)          # [64, 25, 22, 12, 11]

# sorted() --- returns new list
lst = [5, 2, 8, 1, 9]
new = sorted(lst)  # original unchanged

# Linear search
def linear_search(lst, target):
    for i, val in enumerate(lst):
        if val == target:
            return i    # index found
    return -1           # not found

idx = linear_search([10,20,30,40], 30)
print("Found at index:", idx)    # 2

# List copy
a = [1, 2, 3]
```

2D List — Matrix Operations (W'25, Su'24 — 7M)

```python
# Create 3x3 matrix
matrix = [[1,2,3],[4,5,6],[7,8,9]]

# Access element: matrix[row][col]
print(matrix[1][2])      # 6

# Print matrix
for row in matrix:
    for val in row:
        print(val, end="\t")
    print()

# Matrix addition
A = [[1,2],[3,4]]
B = [[5,6],[7,8]]
C = [[A[i][j]+B[i][j] for j in range(2)] for i in range(2)]
print(C)     # [[6,8],[10,12]]

# Transpose
T = [[A[j][i] for j in range(2)] for i in range(2)]
print(T)     # [[1,3],[2,4]]
```

String Programs (Su'25, W'25 — 4M)

```python
# Reverse a string
s = "Python"
print(s[::-1])              # nohtyP

# Count words in sentence
sentence = "Hello how are you"
words = sentence.split()
print(len(words))           # 4

# Count occurrences of character
s = "mississippi"
print(s.count("s"))         # 4

# Remove duplicate characters
s = "programming"
seen = []
result = ""
for c in s:
    if c not in seen:
        result += c; seen.append(c)
print(result)  # progamin
```

String: Palindrome & Anagram (Su'24, W'24 — 4M)

```python
# Palindrome check
def is_palindrome(s):
    s = s.lower().replace(" ", "")
    return s == s[::-1]

print(is_palindrome("racecar"))  # True
print(is_palindrome("madam"))    # True
print(is_palindrome("hello"))    # False

# Anagram check
def is_anagram(s1, s2):
    return sorted(s1.lower()) == sorted(s2.lower())

print(is_anagram("listen", "silent"))  # True
print(is_anagram("hello", "world"))    # False

# Title case conversion
s = "python programming language"
print(s.title())   # Python Programming Language
```

String: Caesar Cipher & Formatting (Su'25 OR — 4M)

```python
# Caesar cipher (shift=3)
def caesar(text, shift=3):
    result = ""
    for c in text:
        if c.isalpha():
            base = ord('A') if c.isupper() else ord('a')
            result += chr((ord(c)-base+shift)%26+base)
        else:
            result += c
    return result

print(caesar("Hello Python"))    # Khoor Sbwkrq
print(caesar("Khoor Sbwkrq", -3)) # Hello Python

# String alignment
name = "Raj"; marks = 95
print(f"{'Name':<10} {'Marks':>5}")
print(f"{name:<10} {marks:>5}")
```

Nested Dictionary (W'25 OR, Su'25 — 4M)

```
# Student database - nested dict
students = {
    "S001": {"name": "Raj",   "marks": 85, "grade": "A"},
    "S002": {"name": "Priya", "marks": 72, "grade": "B"},
    "S003": {"name": "Sam",   "marks": 91, "grade": "A+"},
}

# Access nested values
print(students["S001"]["name"])    # Raj
print(students["S002"]["marks"])   # 72

# Iterate nested dict
for roll, info in students.items():
    print(f"{roll}: {info['name']} - {info['marks']}")

# Add new key to nested
students["S001"]["phone"] = "9876543210"
print(students["S001"])
```

Dictionary: Frequency Counter (W'24, Su'25 — 4M)

```python
# Count character frequency
sentence = "hello world"
freq = {}
for c in sentence:
    if c != ' ':
        freq[c] = freq.get(c, 0) + 1
print(freq)
# {'h':1,'e':1,'l':3,'o':2,'w':1,'r':1,'d':1}

# Count word frequency
text = "one two one three two one"
words = text.split()
word_freq = {}
for w in words:
    word_freq[w] = word_freq.get(w, 0) + 1
print(word_freq)
# {'one':3, 'two':2, 'three':1}

# Most frequent word
most_common = max(word_freq, key=word_freq.get)
print("Most common:", most_common)   # one
```

Dictionary: Merge, Update & iterate (W'25, Su'24 OR — 4M)

```python
d1 = {'a': 1, 'b': 2, 'c': 3}
d2 = {'b': 20, 'd': 4, 'e': 5}

# update(): merges d2 into d1 (overwrite duplicates)
d1.update(d2)
print(d1)   # {'a':1,'b':20,'c':3,'d':4,'e':5}

# Merge using | (Python 3.9+)
d3 = {'x': 1} | {'y': 2}   # {'x':1,'y':2}

# Delete key
del d1['a']
popped = d1.pop('b', 0)   # remove and return
print(popped)   # 20

# Check if key exists
print('c' in d1)      # True
print(d1.get('z', 0))# 0 (default)

# All keys, values, items
print(list(d1.keys()))
print(list(d1.values()))
```

Set: Advanced Operations (W'24, Su'24 — 4M)

```
A = {1, 2, 3, 4, 5}
B = {3, 4, 5, 6, 7}

# All set operations
print(A | B)     # Union:     {1,2,3,4,5,6,7}
print(A & B)     # Intersection: {3,4,5}
print(A - B)     # Difference:   {1,2}
print(A ^ B)     # Symmetric Diff:{1,2,6,7}

# Subsets and supersets
C = {3, 4}
print(C.issubset(A))       # True  (C  A)
print(A.issuperset(C))     # True  (A  C)
print(A.isdisjoint({8,9}))# True  (no common)

# Add, remove, discard
A.add(10)
A.remove(1)           # raises KeyError if missing
A.discard(99)         # no error if missing
print(len(A))
```

Tuple: Practical Usage (Su'24, W'25 OR — 3M)

```
# Tuple packing / unpacking
point = (3, 4)              # 2D coordinate (tuple)
x, y = point                # unpacking
print(x, y)                 # 3 4

# Return multiple values from function
def min_max(lst):
    return min(lst), max(lst)   # returns tuple

lo, hi = min_max([5, 2, 8, 1, 9])
print(f"Min={lo}, Max={hi}")    # Min=1, Max=9

# Named access
student = ("Raj", 20, "CSE")
name, age, branch = student

# Tuple in a set / as dict key (immutable)
locations = {(23.0, 72.5): "Gandhinagar",
             (19.0, 72.8): "Mumbai"}
print(locations[(23.0, 72.5)])   # Gandhinagar
```

Data Structures — Comprehensive Comparison (W'25 — 7M)

Feature	List	String	Tuple	Set	Dict
Syntax	`[...]`	`"..."`	`(...)`	`{...}`	`{k:v}`
Ordered	Yes	Yes	Yes	No	Yes(3.7+)
Mutable	Yes	No	No	Yes	Yes
Duplicates	Yes	Yes	Yes	No	Keys: No
Indexing	Yes	Yes	Yes	No	By key
Use case	General	Text	Fixed data	Unique items	Key-value
Methods	Many	Many	Few	Union/inter	get/update

Quick Access Syntax

```
lst[0]   s[1:4]   tup[-1]   d["key"]   d.get("k",default)
```

Program: Student Marks Management (W'25 — 7M)

```python
# Store and process marks using dictionary
n = int(input("Number of students: "))
students = {}

for i in range(n):
    name = input(f"Student {i+1} name: ")
    marks = float(input(f"Marks for {name}: "))
    students[name] = marks

# Display all with grade
print("\nResult:")
for name, m in students.items():
    if m >= 90:   grade = "A+"
    elif m >= 75: grade = "A"
    elif m >= 60: grade = "B"
    elif m >= 40: grade = "C"
    else:         grade = "F"
    print(f"{name:<12}: {m:5.1f} ({grade})")

avg = sum(students.values()) / len(students)
topper = max(students, key=students.get)
print(f"\nAverage: {avg:.1f}")
print(f"Topper:  {topper} ({students[topper]})")
```

Program: Remove Duplicates & Intersect (Su'25, W'24 — 4M)

```python
# Remove duplicates from list using set
nums = [1, 2, 3, 2, 4, 1, 5, 3]
unique = list(set(nums))
print("Unique:", sorted(unique))  # [1,2,3,4,5]

# Preserve order while removing duplicates
seen = []
result = []
for n in nums:
    if n not in seen:
        result.append(n); seen.append(n)
print("Order preserved:", result)  # [1,2,3,4,5]

# Common elements of two lists (intersection)
A = [1, 2, 3, 4, 5]
B = [3, 4, 5, 6, 7]
common = list(set(A) & set(B))
print("Common:", common)   # [3, 4, 5]

# Elements only in A (difference)
only_A = list(set(A) - set(B))
print("Only in A:", only_A)  # [1, 2]
```

Program: Element Frequency in List (Su'25, W'25 OR — 4M)

```python
# Find frequency of each element
nums = [1, 2, 3, 2, 1, 4, 1, 3, 5]

# Method 1: Dictionary
freq = {}
for n in nums:
    freq[n] = freq.get(n, 0) + 1
for k, v in sorted(freq.items()):
    print(f"{k}: {'*'*v} ({v})")

# Method 2: Using count()
unique_nums = set(nums)
for n in sorted(unique_nums):
    print(f"{n} appears {nums.count(n)} times")

# Most frequent element
most_freq = max(freq, key=freq.get)
print(f"Most frequent: {most_freq} ({freq[most_freq]} times)")
```

Worked: Venn Diagram with Sets (W'24, Su'23 — 4M)

```
# Students who like Math, Science, both, or neither
math = {"Raj", "Priya", "Sam", "Tom", "Eve"}
sci  = {"Sam", "Tom", "Ana", "Priya", "Bob"}

both   = math & sci
only_m = math - sci
only_s = sci - math
either = math | sci
neither_count = 30 - len(either)   # class of 30

print(f"Both Math & Science:   {both}")
print(f"Only Math:             {only_m}")
print(f"Only Science:          {only_s}")
print(f"Either Math or Sci:    {len(either)}")
print(f"Neither:               {neither_count}")
print(f"Math XOR Sci:          {math ^ sci}")
```

Worked: List Sort & Stats Without Library (Su'25 — 7M)

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

def mean(lst):   return sum(lst) / len(lst)
def median(lst):
    s = sorted(lst); n = len(s)
    return s[n//2] if n%2 else (s[n//2-1]+s[n//2])/2
def mode(lst):   return max(set(lst), key=lst.count)

marks = [55, 72, 88, 40, 72, 65, 90, 72]
print("Sorted:", bubble_sort(marks[:]))
print(f"Mean: {mean(marks):.2f}")
print(f"Median: {median(marks)}")
print(f"Mode: {mode(marks)}")
```

Worked: String Tokenization Program (Su'25, W'25 — 4M)

```python
# Count vowels, consonants, digits, spaces
def analyze_string(s):
    vowels = "aeiouAEIOU"
    v = c = d = sp = 0
    for ch in s:
        if ch in vowels:    v += 1
        elif ch.isalpha():  c += 1
        elif ch.isdigit():  d += 1
        elif ch == ' ':     sp += 1
    return v, c, d, sp

text = "Hello World 2025!"
v, c, d, sp = analyze_string(text)
print(f"Vowels:     {v}")
print(f"Consonants: {c}")
print(f"Digits:     {d}")
print(f"Spaces:     {sp}")
# Vowels: 3, Consonants: 7, Digits: 4, Spaces: 2
```

Worked: Phone Book using Dictionary (W'25 OR — 7M)

```
phone_book = {}

def add_contact(name, number):
    phone_book[name] = number
    print(f"Added {name}")

def search_contact(name):
    if name in phone_book:
        print(f"{name}: {phone_book[name]}")
    else:
        print("Contact not found")

def delete_contact(name):
    if name in phone_book:
        del phone_book[name]; print(f"Deleted {name}")
    else:
        print("Contact not found")

def display_all():
    for name, num in sorted(phone_book.items()):
        print(f"{name:<15}: {num}")

add_contact("Raj",   "9876543210")
add_contact("Priya", "9123456789")
search_contact("Raj")
display_all()
delete_contact("Raj")
```

Worked: Matrix Addition & Transpose (W'25— 7M)

```
def input_matrix(r, c):
    m = []
    for i in range(r):
        row = [int(x) for x in input(f"Row {i+1}: ").split()]
        m.append(row)
    return m

def print_matrix(m):
    for row in m:
        print(" ".join(f"{v:4d}" for v in row))

def add_matrix(A, B, r, c):
    return [[A[i][j]+B[i][j] for j in range(c)] for i in range(r)]

def transpose(A, r, c):
    return [[A[i][j] for i in range(r)] for j in range(c)]

r, c = 2, 2
print("Matrix A:")
A = input_matrix(r, c)
print("Matrix B:")
B = input_matrix(r, c)
print("Sum:"); print_matrix(add_matrix(A, B, r, c))
print("Transpose of A:"); print_matrix(transpose(A, r, c))
```

Worked: List Comprehension Programs (W'25, Su'25 — 4M)

```python
# Generate multiplication-table-like list
table = [i*j for i in range(1,4) for j in range(1,4)]
print(table)  # [1,2,3,2,4,6,3,6,9]

# Flatten 2D list
matrix = [[1,2,3],[4,5,6],[7,8,9]]
flat = [n for row in matrix for n in row]
print(flat)   # [1,2,3,4,5,6,7,8,9]

# Conditional replacement
nums = [1,-2,3,-4,5,-6]
abs_vals = [x if x>=0 else -x for x in nums]
print(abs_vals)  # [1,2,3,4,5,6]

# Primes using list comp
primes = [n for n in range(2,50)
          if all(n%i!=0 for i in range(2,n))]
print(primes)   # [2,3,5,7,11,13,17,19,23...]
```

Worked: Score Board using Multiple DS (Su'25 — 7M)

```
# Use list, dict, set, tuple together
scores = {}   # dict: name -> list of scores

def add_score(name, score):
    scores.setdefault(name, []).append(score)

def get_stats(name):
    s = scores[name]
    return (min(s), max(s), sum(s)/len(s))  # tuple

add_score("Raj",   85); add_score("Raj",   92)
add_score("Priya", 78); add_score("Priya", 88)
add_score("Sam",   65); add_score("Sam",   71)

players = set(scores.keys())  # unique names
print("Players:", players)
for name in sorted(players):
    mn, mx, avg = get_stats(name)
    print(f"{name:<8}: min={mn}, max={mx}, avg={avg:.1f}")
```

Worked: String Manipulation Programs (W'25, Su'25 — 4M)

```python
# Most frequent character
s = "programming"
freq = {}
for c in s:
    freq[c] = freq.get(c, 0) + 1
most = max(freq, key=freq.get)
print(f"Most frequent: '{most}' ({freq[most]} times)")

# Check if two strings are rotations
def are_rotations(s1, s2):
    return len(s1)==len(s2) and s1 in s2+s2

print(are_rotations("abcd", "cdab"))  # True
print(are_rotations("abcd", "abdc"))  # False

# Remove all vowels
def remove_vowels(s):
    return "".join(c for c in s if c.lower() not in "aeiou")

print(remove_vowels("Hello World"))  # Hll Wrld
```

PYQ Practice — All Years Consolidated (Review)

3-Mark Questions:

1. What is a list? State 4 list methods with example. *[W'24, Su'25]*
2. Difference between list and tuple. *[Su'23, W'25]*
3. Write note on set operations in Python. *[W'25, Su'24 OR]*
4. Explain dictionary with example. *[W'24, Su'23]*

4-Mark Questions:

1. Write Python program to sort list in ascending order. *[W'24 OR, Su'25]*
2. Explain string methods with examples (5 methods). *[W'24, W'25]*
3. Write program to find all elements common to two lists. *[Su'24 OR]*
4. Write program to count word frequency in a sentence. *[Su'25 OR]*

7-Mark Questions:

1. Write program to implement a phone book using dictionary. *[W'25 OR]*
2. Write program to add, multiply two matrices. *[W'25]*
3. Explain list/tuple/set/dict with programs for each. *[Su'25, W'25]*

Common Mistakes in Data Structures

Mistakes

- × Modifying list while iterating
- × Shallow vs deep copy confusion
- × Tuple as mutable (it's not)
- × Set indexing (sets unordered)
- × Dict key not found (no `get()`)
- × String assignment (`s[0]="x"`)

Correct Patterns

- ✓ Use `lst[:]` or `lst.copy()`
- ✓ Tuple = fixed; use list to change
- ✓ `list(my_set)` to get items
- ✓ `d.get(key, 0)` safely
- ✓ Build new string with `join()`
- ✓ Convert: `set(lst)`, `list(s)`

Unit 5 — Rapid Revision Flash Sheet

Structure	Key Methods / Operations
List	`append, insert, remove, pop, sort, reverse, index, count, copy`
String	`upper, lower, split, join, strip, replace, find, count, format`
Tuple	`count, index`; immutable; use for fixed data
Set	`add, remove, union(|), intersection(&), difference(-), issubset`
Dictionary	`get, keys, values, items, update, pop, setdefault`
Comprehension	`[expr for x in seq if cond]`; also dict `{k:v ...}`

GTU Weightage (Unit 5 — 20%)

3-mark: operations/methods 4-mark: programs 7-mark: complete programs

Worked: Shopping Cart using Dict & List (Su'25 — 7M)

```
cart = {}    # item_name -> (price, quantity)

def add_item(name, price, qty=1):
    if name in cart:
        cart[name] = (price, cart[name][1] + qty)
    else:
        cart[name] = (price, qty)

def display_cart():
    print(f"{'Item':<12} {'Price':>8} {'Qty':>4} {'Subtotal':>10}")
    print("-" * 40)
    total = 0
    for item, (price, qty) in cart.items():
        sub = price * qty
        total += sub
        print(f"{item:<12} {price:>8.2f} {qty:>4} {sub:>10.2f}")
    print(f"\n{'Total':>26} {total:>10.2f}")

add_item("Apples",  20.0, 3)
add_item("Bread",   35.0, 2)
add_item("Milk",    25.0, 2)
display_cart()
```

Worked: List Operations — All in One (W'25 — 7M)

```python
# Enter N numbers and demonstrate all list ops
n = int(input("Count: "))
nums = [int(input(f"[{i+1}]: ")) for i in range(n)]
print("Original:     ", nums)
print("Sorted(asc): ", sorted(nums))
print("Sorted(desc):", sorted(nums, reverse=True))
print("Maximum:", max(nums))
print("Minimum:", min(nums))
print("Sum:     ", sum(nums))
print("Average:", sum(nums)/len(nums))

# Second largest
uniq = list(set(nums))
uniq.sort(reverse=True)
print("2nd largest:", uniq[1] if len(uniq)>1 else "N/A")

# Count even/odd
even = [x for x in nums if x%2==0]
odd  = [x for x in nums if x%2!=0]
print(f"Even={even}, Odd={odd}")
```

Unit 5 — Summary

List

Ordered, Mutable, Duplicates OK
`append`, `remove`, `sort`
Indexing & slicing
Nested lists

Set & Dictionary

Set: Unordered, No duplicates
Union, Intersection, Difference
Dict: Key-value pairs
Fast lookup by key

Tuple

Ordered, Immutable
Faster than list
`count()`, `index()` only
Nested tuples

String

Immutable sequence
`upper`, `lower`, `split`
`replace`, `find`, `strip`
Slicing & indexing