Unit 1: Problem Solving

Python Programming — DI02032011

Milav Dabgar

Lecturer, Dept of ICT
GP Palanpur

April 2, 2026

Unit 1 — Table of Contents

Syllabus Coverage

1.1 Introduction & Steps for Problem Solving
1.2 Algorithm: Definition, Characteristics & Examples
1.3 Flowchart: Symbols, Rules & Examples
1.4 Flowchart vs Algorithm
1.5 Pseudocode
1.6 GTU PYQ Practice

Weightage: ≈10% (10 marks) | Hours: 5

1.1 Introduction to Problem Solving

Steps, Algorithm & Flowchart

What is Problem Solving?

Definition

Problem solving is the process of identifying a problem, developing a plan to address it, and implementing that plan to arrive at a correct solution.

Steps for Problem Solving

1. **Understand** the problem clearly
2. **Identify** inputs, outputs and constraints
3. **Design** the solution (algorithm/flowchart)
4. **Code** the solution in a programming language
5. **Test & Debug** the program

Importance of Algorithm & Flowchart

Why use Algorithms?

- Clear step-by-step solution
- Language-independent
- Easy to convert to code
- Helps detect logical errors early
- Reusable and shareable

Why use Flowcharts?

- Visual representation of logic
- Easy to understand and follow
- Helps in program planning
- Useful for documentation
- Simplifies complex logic

GTU Tip

Both importance and differences between algorithm and flowchart appear frequently in 3-mark and 4-mark questions.

1.2 Algorithm

Definition, Characteristics & Examples

Algorithm — Definition & Characteristics

Definition (GTU Su'23, W'25 — 3M)

An **algorithm** is a finite, ordered sequence of well-defined instructions for solving a problem or performing a task.

Characteristics of a Good Algorithm

Input	Zero or more inputs
Output	At least one output
Definiteness	Each step is clear and unambiguous
Finiteness	Must terminate after finite steps
Effectiveness	Each step is basic and feasible

Algorithm Examples — Simple Programs

Algorithm: Find Area of Circle (W'25 — 3M)

Step 1: Start
Step 2: Read radius r
Step 3: Calculate area $= \pi \times r^2$
Step 4: Print area
Step 5: Stop

Algorithm: Sum of Two Numbers

Step 1: Start
Step 2: Read two numbers A and B
Step 3: Calculate Sum $= A + B$
Step 4: Print Sum
Step 5: Stop

Algorithm: Maximum of Two Numbers (W'23-4M)

Algorithm: Find Maximum of Two Numbers

Step 1: Start
Step 2: Read two numbers A and B
Step 3: If $A > B$ then
Step 3a: Print "A is greater"
Else
Step 3b: If $A < B$ then Print "B is greater"
Else Print "Both are equal"
Step 4: Stop

Algorithm: Maximum of Three Numbers

Step 1: Start
Step 2: Read A, B, C
Step 3: If $A \geq B$ and $A \geq C$: Print "A is max"
Step 4: Else if $B \geq C$: Print "B is max"
Step 5: Else: Print "C is max"
Step 6: Stop

Algorithm: Simple Interest (Su'23, W'25 — 4M)

Algorithm: Find Simple Interest

Step 1: Start
Step 2: Read Principal P, Rate R, Time T
Step 3: Calculate SI $= \dfrac{P \times R \times T}{100}$
Step 4: Print SI
Step 5: Stop

Algorithm: Positive or Negative Number (W'23 — 3M)

Step 1: Start
Step 2: Read number N
Step 3: If $N > 0$: Print "Positive"
Step 4: Else if $N < 0$: Print "Negative"
Step 5: Else: Print "Zero"
Step 6: Stop

Algorithm: Print Odd Numbers 1 to 20 (W'24-7M)

Algorithm: Print Odd Numbers Between 1 and 20

Step 1: Start
Step 2: Set $i = 1$
Step 3: While $i \leq 20$:
Step 3a: If $i \bmod 2 \neq 0$: Print i
Step 3b: Set $i = i + 1$
Step 4: Stop

Advantages of Algorithm

- Easy to understand — language-independent
- Can be directly converted to any programming language
- Helps to detect and fix logical errors before coding

Algorithm: Sum of First N Even Numbers (Su'25-4M)

Algorithm: Find Sum of Two & Max of Two Numbers

(1) Sum of two numbers:
Step 1: Start Step 2: Read A, B Step 3: Sum $= A + B$ Step 4: Print Sum
Step 5: Stop

(2) Maximum of two numbers:
Step 1: Start Step 2: Read A, B
Step 3: If $A > B$: Print "A is max" Else: Print "B is max"
Step 4: Stop

Algorithm: Even/Odd Check

Step 1: Start Step 2: Read N
Step 3: If $N \bmod 2 = 0$: Print "Even" Else: Print "Odd"
Step 4: Stop

1.3 Flowchart

Symbols, Rules & Examples

Flowchart — Definition & Symbols (W'24, Su'25 — 3M)

Definition

A **flowchart** is a diagrammatic representation of an algorithm using standard symbols to show the flow of control in a program.

Standard Flowchart Symbols

Symbol Shape	Purpose
Oval (Terminator)	Start / Stop of the program
Parallelogram	Input / Output operation
Rectangle (Process)	Processing / Calculation step
Diamond (Decision)	Yes/No decision (if/else)
Arrow (Flow line)	Direction of flow
Circle (Connector)	Connects different parts

Flowchart — Rules & Limitations (Su'23 — 4M)

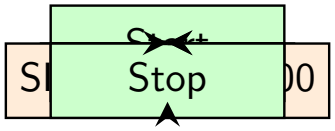
Rules for Drawing a Flowchart

1. Every flowchart must have a **Start** and **Stop** symbol
2. Use **standard symbols** only
3. Flow must go from **top to bottom** or left to right
4. Each symbol has **one entry** and **one exit** point
5. Decision box has **two exits**: Yes and No
6. Use **connectors** to avoid crossing lines

Limitations of Flowchart

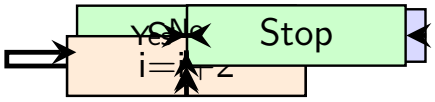
Complex programs $\Rightarrow$ very large and hard to read | Difficult to modify | No standard for detail level

Flowchart: Simple Interest (Su'23, W'25 — 4M)



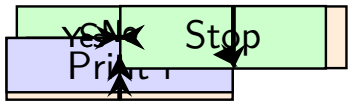
Flowchart: Even or Odd (Su'24 — 4M)

Flowchart: Sum of First 20 Even Numbers (W'24 — 7M)



Flowchart: Positive, Negative or Zero

Flowchart: Print Odd Numbers from 1 to 20 (W'24 OR — 7M)



1.4 Comparison

Flowchart vs Algorithm

Flowchart vs Algorithm (Su'23 OR — 3M)

Comparison Table

Parameter	Algorithm	Flowchart
Form	Written text (steps)	Graphical diagram
Symbols	Plain English / text	Standard shapes
Readability	Easy for programmers	Easy for all
Modification	Easy to edit	Difficult to redraw
Complexity	Can handle complex logic	Gets messy for complex logic
Tool needed	Pen/paper/editor	Drawing tool
Debugging	Harder to visualize	Easier to trace flow

1.5 Pseudocode

Definition & Examples

Pseudocode — Definition (W'23, Su'23 — 3M)

Definition

Pseudocode is an informal, high-level description of an algorithm using structured English-like syntax that is easy to read but not tied to any specific programming language.

Characteristics

- Not a real programming language — no strict syntax
- Uses keywords like `IF`, `ELSE`, `WHILE`, `FOR`, `PRINT`
- Closer to a programming language than plain English
- Easy to convert to actual code
- Bridges the gap between algorithm and code

Pseudocode Examples (W'23 — 3M; Su'23 — 3M)

Check Positive or Negative

```
BEGIN
READ N
IF N > 0 THEN
PRINT "Positive"
ELSE IF N < 0 THEN
PRINT "Negative"
ELSE
PRINT "Zero"
END IF
END
```

Pseudocode: Smallest of Two (Su'23 — 3M)

Find Smallest of Two Numbers

```
BEGIN
READ A, B
IF A < B THEN
PRINT "A is smallest"
ELSE IF B < A THEN
PRINT "B is smallest"
ELSE
PRINT "Both are equal"
END IF
END
```

Pseudocode: Largest of Three (Su'24 — 4M)

Find Largest of Three Numbers X, Y, Z

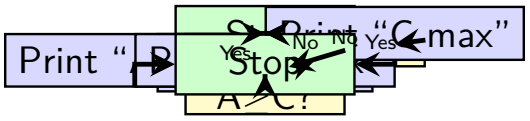
```
BEGIN
READ X, Y, Z
IF X >= Y AND X >= Z THEN
PRINT "X is largest"
ELSE IF Y >= Z THEN
PRINT "Y is largest"
ELSE
PRINT "Z is largest"
END IF
END
```

Pseudocode vs Algorithm vs Flowchart

Quick Comparison

Aspect	Algorithm	Flowchart	Pseudocode
Form	Numbered steps	Graphic diagram	Structured English
Language	Natural/English	Symbols	Semi-code
Coding ease	Good	Moderate	Very good
Modification	Easy	Hard	Easy
Audience	Programmer	Anyone	Programmer

Flowchart: Maximum of Three Numbers



1.6 GTU PYQ Practice

All exam questions from Su'23 – W'25

3-Mark PYQ Questions

Try These (3 marks each)

1. Define flowchart and list any four symbols used. *[W'24, Su'25]*
2. Define algorithm. What are the advantages of Algorithm? *[Su'23]*
3. List the importance of flowchart and algorithm. *[Su'24]*
4. Define Algorithm. Write algorithm to find area of a circle. *[W'25]*
5. Write the difference between flowchart and algorithm. *[Su'23 OR]*
6. Write pseudocode to check if a number is positive or negative. *[W'23]*
7. Define pseudocode. Write pseudocode to find smallest of two numbers. *[Su'23]*

4-Mark PYQ Questions

Try These (4 marks each)

1. State rules for problem solving using flowchart. Design flowchart for SI. *[Su'23, W'25]*
2. Define Algorithm. Design algorithm for max of three numbers. *[W'23]*
3. Draw flowchart to find whether entered number is even or odd. *[Su'24]*
4. Prepare algorithms: (1) max of two numbers (2) sum of two numbers. *[Su'25]*
5. Define pseudocode. Write pseudocode for largest of three X, Y, Z. *[Su'24]*

7-Mark PYQ Questions

Try These (7 marks each)

1. Design flowchart to calculate sum of first 20 even natural numbers. *[W'24]*
2. Create algorithm to print odd numbers between 1 to 20. *[W'24 OR]*

Exam Pattern Reminder

Q.1(a) = 3M **Q.1(b)** = 4M **Q.1(c) / Q.2(c)** = 7M each with an OR option.
Unit 1 questions appear mainly in **Q.1(a)**, **Q.1(b)**, **Q.1(c)**.

Worked: Algorithm + Flowchart — Area of Circle (W'25)

Algorithm

Step 1: Start
Step 2: Read radius r
Step 3: Set $\pi = 3.14159$
Step 4: Area $= \pi \times r^2$
Step 5: Print Area
Step 6: Stop

Key Notes

The algorithm has:
✓ Definite start/stop
✓ Clear input/output
✓ Finite steps
✓ One main calculation
✓ No branching needed

Worked: Algorithm — Odd Numbers 1 to 20 (W'24 OR)

Create algorithm to print odd numbers between 1 to 20

Step 1: Start
Step 2: Initialize $i = 1$
Step 3: If $i > 20$ go to Step 7
Step 4: If $i \bmod 2 \neq 0$ then Print i
Step 5: Set $i = i + 1$
Step 6: Go to Step 3
Step 7: Stop

Output: 1, 3, 5, 7, 9, 11, 13, 15, 17, 19

Practice: Design Tips for Flowcharts

How to Draw a Correct Flowchart in Exam

1. Start with the **Oval** labelled "Start"
2. Use **Parallelogram** for every Read/Print
3. Use **Rectangle** for every calculation
4. Use **Diamond** for every condition (if/loop check)
5. For loops: draw the **back-arrow** from last step to decision
6. End with **Oval** labelled "Stop"
7. Label all **Yes/No** exits from each diamond

Common Mistakes to Avoid

Missing Start/Stop | Unlabelled decision exits | Parallelogram used for calculations | Missing flow arrows

Algorithm Characteristics — Quick Recall

5 Essential Properties (Knuth)

Finiteness	Must terminate after finite number of steps
Definiteness	Each step must be precisely defined
Input	Zero or more inputs from outside
Output	At least one result/output produced
Effectiveness	All steps must be feasible/basic

GTU 3M Answer Template

"An **algorithm** is a finite sequence of well-defined instructions to solve a problem. Its advantages are: (1) language-independent, (2) easy to understand, (3) easy to convert to code, (4) helps detect errors early."

Worked: Two Algorithms (Su'25 — 4M)

(1) Algorithm: Max of Two Numbers

Step 1: Start Step 2: Read A, B
Step 3: IF A $>$ B: PRINT "A is max" ELSE: PRINT "B is max"
Step 4: Stop

(2) Algorithm: Sum of Two Numbers

Step 1: Start Step 2: Read A, B
Step 3: Sum $=$ A $+$ B
Step 4: PRINT Sum
Step 5: Stop

Importance of Flowchart & Algorithm (Su'24 — 3M)

Importance of Flowchart

- Provides a **visual overview** of the solution before coding
- Easier to **communicate** the logic to others
- Helps in **identifying bottlenecks** and logical errors
- Useful for **documentation** and maintenance
- Makes **debugging** easier

Importance of Algorithm

- **Language-independent** blueprint of the solution
- Can be **directly converted** to any programming language
- Promotes **step-by-step logical thinking**
- Helps in **estimating time complexity**
- Useful for **team collaboration**

Algorithm: Factorial of a Number (Su'24 OR, W'24 OR — 7M)

Algorithm: Factorial of N

Step 1: Start
Step 2: Read number N
Step 3: Set $fact \leftarrow 1$, $i \leftarrow 1$
Step 4: Repeat while $i \leq N$:
Step 4a: $fact \leftarrow fact \times i$
Step 4b: $i \leftarrow i + 1$
Step 5: Print $fact$
Step 6: Stop

Note: Factorial Formula

$N! = N \times (N-1) \times (N-2) \times \cdots \times 2 \times 1$

$0! = 1$ (by definition) $\quad 5! = 5 \times 4 \times 3 \times 2 \times 1 = 120$

Algorithm: Fibonacci Series (W'25 OR, Su'25 — 7M)

Algorithm: Print First N Fibonacci Numbers

Step 1: Start
Step 2: Read N
Step 3: Set $a \leftarrow 0$, $b \leftarrow 1$, $count \leftarrow 1$
Step 4: Print a
Step 5: Repeat while $count \leq N - 1$:
Step 5a: Print b
Step 5b: $temp \leftarrow a + b$
Step 5c: $a \leftarrow b$, $b \leftarrow temp$
Step 5d: $count \leftarrow count + 1$
Step 6: Stop

Fibonacci Sequence

0, 1, 1, 2, 3, 5, 8, 13, 21, 34, ... Each term = sum of previous two terms.

Algorithm: GCD of Two Numbers (Su'23 OR, W'23 — 7M)

Algorithm: GCD using Euclidean Method

Step 1: Start
Step 2: Read A and B
Step 3: Repeat while $B \neq 0$:
Step 3a: $temp \leftarrow B$
Step 3b: $B \leftarrow A \bmod B$
Step 3c: $A \leftarrow temp$
Step 4: Print A (GCD)
Step 5: Stop

Example

GCD(12, 8): $12 \bmod 8 = 4 \rightarrow$ GCD(8,4): $8 \bmod 4 = 0 \rightarrow$ GCD $= 4$

Algorithm: Swap Two Numbers (W'23 OR, Su'24 — 3M)

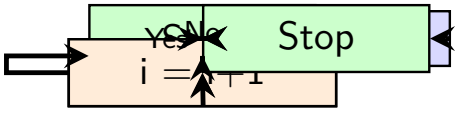
Using Temporary Variable

Step 1: Start
Step 2: Read A, B
Step 3: $temp \leftarrow A$
Step 4: $A \leftarrow B$
Step 5: $B \leftarrow temp$
Step 6: Print A, B
Step 7: Stop

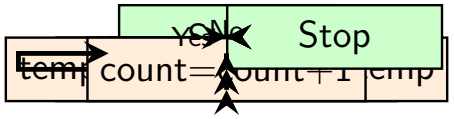
Without Temp Variable

Step 1: Start
Step 2: Read A, B
Step 3: $A \leftarrow A + B$
Step 4: $B \leftarrow A - B$
Step 5: $A \leftarrow A - B$
Step 6: Print A, B
Step 7: Stop

Flowchart: Factorial of N (Su'24 OR, W'24 OR — 7M)



Flowchart: Fibonacci Series (W'25 OR — 7M)



Pseudocode: Factorial (Su'24 OR, W'24 — 7M)

Pseudocode: Compute Factorial of N

BEGIN
INPUT N
SET $fact \leftarrow 1$
SET $i \leftarrow 1$
WHILE $i \leq N$ **DO**
$fact \leftarrow fact \times i$
$i \leftarrow i + 1$
ENDWHILE
PRINT $fact$
END

Trace: N=4

i	fact	i ≤ N?
1	1	Yes
2	2	Yes
3	6	Yes
4	24	Yes
5	—	No ⇒ Print 24

Pseudocode: Fibonacci Series (W'25, Su'25 — 7M)

Pseudocode: Print N Fibonacci Numbers

BEGIN
INPUT N
SET $a \leftarrow 0$, $b \leftarrow 1$, $count \leftarrow 1$
PRINT a
WHILE $count \leq N - 1$ **DO**
PRINT b
SET $temp \leftarrow a + b$
SET $a \leftarrow b$
SET $b \leftarrow temp$
$count \leftarrow count + 1$
ENDWHILE
END

Algorithm: Check Prime Number (W'24, Su'25 — 4M)

Algorithm: Prime Number Check

Step 1: Start
Step 2: Read N
Step 3: Set $flag \leftarrow 1$, $i \leftarrow 2$
Step 4: Repeat while $i \leq N/2$:
Step 4a: If $N \bmod i = 0$: Set $flag \leftarrow 0$; Break
Step 4b: $i \leftarrow i + 1$
Step 5: If $flag = 1$: Print "N is Prime"
Else: Print "N is Not Prime"
Step 6: Stop

Pseudocode: GCD of Two Numbers (Su'23 OR — 7M)

Pseudocode: GCD using Euclidean Method

BEGIN
INPUT A, B
WHILE $B \neq 0$ **DO**
SET $temp \leftarrow B$
SET $B \leftarrow A \bmod B$
SET $A \leftarrow temp$
ENDWHILE
PRINT A $\quad \{A$ is GCD$\}$
END

Trace: GCD(18, 12)

A	B	temp
18	12	12
12	6	6
6	0	$0 \Rightarrow$ Stop

GCD = 6

Pseudocode: Sum of Digits (W'24 OR — 4M)

Pseudocode

BEGIN
INPUT N
SET $sum \leftarrow 0$
WHILE $N > 0$ **DO**
$digit \leftarrow N \bmod 10$
$sum \leftarrow sum + digit$
$N \leftarrow N \div 10$
ENDWHILE
PRINT sum
END

Trace: N=1234

N	digit	sum
1234	4	4
123	3	7
12	2	9
1	1	10
0	–	Stop

Sum = 10

Types of Errors in Problem Solving (W'23 OR — 3M)

1. Syntax Error

Grammatical mistake in the program; detected by compiler/interpreter.
Example: Missing colon, wrong indentation, misspelled keyword.

2. Runtime Error

Error that occurs during execution; program crashes unexpectedly.
Example: Division by zero, accessing invalid index.

3. Logical Error

Program runs but produces wrong output; hardest to detect.
Example: Using + instead of − in a formula.

Steps in Problem Solving / Software Development

1. **Problem Analysis** — Understand input, output, and constraints.
2. **Algorithm Design** — Write step-by-step logical solution.
3. **Flowchart / Pseudocode** — Visualize or describe the logic.
4. **Coding** — Translate algorithm to a programming language.
5. **Testing & Debugging** — Run with sample inputs; fix errors.
6. **Documentation** — Record how the program works.
7. **Maintenance** — Update as requirements change.

Key Point

Algorithm and Flowchart are done **before** coding. They are **language-independent** tools.

Algorithm: Reverse a Number (Su'23, W'25 OR — 3M)

Algorithm: Reverse Digits of N

Step 1: Start
Step 2: Read N
Step 3: Set $rev \leftarrow 0$
Step 4: Repeat while $N > 0$:
Step 4a: $digit \leftarrow N \bmod 10$
Step 4b: $rev \leftarrow rev \times 10 + digit$
Step 4c: $N \leftarrow N \div 10$
Step 5: Print rev
Step 6: Stop

Trace: N=456

456 → digit=6, rev=6 | 45 → digit=5, rev=65 | 4 → digit=4, rev=654 | 0 → Stop.
Result: 654

Worked: Algorithm & Flowchart for Factorial (7M)

Algorithm

Step 1: Start
Step 2: Read N
Step 3: $fact = 1$, $i = 1$
Step 4: While $i \leq N$:
$fact = fact \times i$
$i = i + 1$
Step 5: Print $fact$
Step 6: Stop

Key Points

- Initialize before loop
- Update inside loop
- Print after loop

$4! = 4 \times 3 \times 2 \times 1 = 24$
$5! = 120$, $6! = 720$

Examiner mark scheme:
Algorithm: 4M
Flowchart: 3M

Worked: Algorithm & Pseudocode for GCD (7M)

Algorithm

Step 1: Start
Step 2: Read A, B
Step 3: While $B \neq 0$:
$temp = B$
$B = A \bmod B$
$A = temp$
Step 4: Print A (GCD)
Step 5: Stop

Trace: GCD(24,16)

A=24, B=16
$24 \bmod 16 = 8 \rightarrow$ A=16, B=8
$16 \bmod 8 = 0 \rightarrow$ A=8, B=0
B=0: Stop. GCD = **8**

LCM: $\text{LCM} = \dfrac{A \times B}{\text{GCD}}$

Worked: Algorithm + Flowchart for Fibonacci (7M)

Algorithm

Step 1: Start
Step 2: Read N
Step 3: $a = 0,\ b = 1,\ c = 1$
Step 4: Print $a,\ b$
Step 5: While $c \leq N - 2$:
$next = a + b$
Print $next$
$a = b,\ b = next$
$c = c + 1$
Step 6: Stop

First 8 Terms (N=8)

0, 1, 1, 2, 3, 5, 8, 13

Pattern:

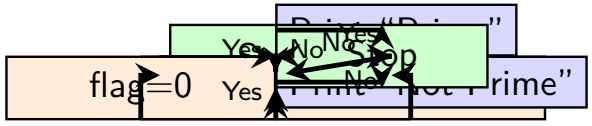
$F_n = F_{n-1} + F_{n-2}$

$F_0 = 0, F_1 = 1$
$F_2 = 1, F_3 = 2$
$F_4 = 3, F_5 = 5$
$F_6 = 8$

Algorithm Quick Reference Card

Problem	Key Steps in Algorithm	GTU
Factorial of N	Init fact=1,i=1; loop: fact*=i,i+=1	7M
Fibonacci N terms	a=0,b=1; print a; loop: print b, update	7M
GCD of A,B	Euclidean: while B≠0: A,B = B, A%B	7M
Swap A,B	temp=A; A=B; B=temp	3M
Largest of 3	if A≥B and A≥C: max=A	4M
Sum of digits	while N>0: sum+=N%10; N//=10	4M
Reverse number	while N>0: rev=rev*10+N%10; N//=10	3M
Count digits	while N>0: count+=1; N//=10	3M
Prime check	flag=1; for i=2 to N/2: if N%i=0: flag=0	4M
SI	SI = P*R*T/100 (direct formula)	4M

Flowchart: Check Prime Number (W'24, Su'25 — 4M)



Common Mistakes & Exam Tips for Unit 1

Common Mistakes

× Forgetting **Start/Stop** in flowchart
× Using wrong symbol (rectangle for decision)
× Not labelling Yes/No on diamond
× Algorithm without step numbers
× Not initializing variables
× Infinite loop (no update in loop)
× Writing code instead of pseudocode

Exam Tips

✓ 7M: Write both algorithm + flowchart
✓ 4M: Algorithm or flowchart (as asked)
✓ 3M: Definition + comparison table
✓ Use proper notation: $\leftarrow$ or $=$
✓ Show variable trace for verification
✓ Neat flowchart with arrows
✓ Number all algorithm steps

Unit 1 — Summary

Key Formulas & Concepts

Algorithm	Finite ordered steps; language-independent
Flowchart	Graphical using standard symbols
Pseudocode	Structured English; between algo and code
Symbols	Oval=Start/Stop, Parallelogram=I/O, Rectangle=Process, Diamond=Decision
Rules	Start/Stop, top-bottom, label Yes/No

Exam Weightage (Unit 1 — 10%)

3-mark $\times$ 1–2 | 4-mark $\times$ 1 | 7-mark $\times$ 0–1

Unit 2: Introduction to Python

Python Programming — DI02032011

Milav Dabgar

Lecturer, Dept of ICT
GP Palanpur

April 2, 2026

Unit 2 — Table of Contents

Syllabus Coverage

2.1 Introduction, Features & Applications of Python
2.2 Python Installation & Basic Program Structure
2.3 Keywords, Identifiers & Variables
2.4 Data Types
2.5 Operators
2.6 Type Conversion
2.7 GTU PYQ Practice

Weightage: ≈14% | Hours: 6

2.1 Introduction to Python

Features & Applications

What is Python?

Definition

Python is a **high-level, interpreted, general-purpose** programming language known for its simple syntax and readability. Created by **Guido van Rossum** (released 1991).

Quick Facts

Paradigm	Object-Oriented, Functional, Procedural
Typing	Dynamically typed
Extension	`.py`
Interpreter	CPython (default), PyPy, Jython
Version	Python 3.x (current)
License	Open-source (PSF License)

Features of Python (Su'25, W'25 — 3M)

Key Features of Python

Simple & Readable	English-like syntax, beginner-friendly
Interpreted	Executes line-by-line, no compilation needed
Dynamically Typed	No need to declare variable types
Object-Oriented	Supports classes, objects, inheritance
Large Library	Rich standard library (math, os, re, etc.)
Platform-Independent	Runs on Windows, Linux, macOS
Free & Open-Source	Available at python.org for free
High-Level	Handles memory management automatically

Applications of Python (Su'25 OR, W'25 OR — 3M)

Where is Python Used?

Web Development	Django, Flask frameworks
Data Science	NumPy, Pandas, Matplotlib
Machine Learning	TensorFlow, PyTorch, scikit-learn
Automation/Scripting	Automate repetitive tasks
Scientific Computing	SciPy for research and analysis
Game Development	Pygame library
Desktop Applications	Tkinter, PyQt
Cybersecurity	Penetration testing, scripting

2.2 Program Structure, Keywords

Identifiers & Variables

Basic Python Program Structure

Structure of a Python Program

```
# This is a comment
# 1. Import statements (optional)
import math

# 2. Variable declarations
name = "GTU Student"
marks = 85

# 3. Main logic
print("Name:", name)
print("Marks:", marks)
print("Square root:", math.sqrt(marks))
```

Python Keywords

Definition

Keywords are **reserved words** with predefined meanings. They cannot be used as variable names.

Python Keywords (Selected)

`False`	`True`	`None`	`and`	`or`	`not`
`if`	`elif`	`else`	`for`	`while`	`break`
`continue`	`def`	`return`	`class`	`import`	`from`
`in`	`is`	`lambda`	`pass`	`try`	`except`
`global`	`del`	`with`	`as`	`raise`	`yield`

Identifiers & Variables (W'23 OR — 3M)

Identifier

Name given to a variable, function, class or object.

Rules for Identifiers / Variables

1. Must start with a **letter** (a–z, A–Z) or **underscore _**
2. Can contain letters, digits (0–9) and underscores
3. **Cannot** start with a digit
4. **Case-sensitive**: `age` ≠ `Age` ≠ `AGE`
5. Cannot be a keyword (`for`, `if`, etc.)
6. No spaces or special characters allowed

Examples

Valid: `name`, `_count`, `total_marks`, `x1`
Invalid: `2name`, `for`, `my-var`, `hello world`

2.3 Data Types

int, float, str, bool, complex

Python Data Types (W'23, W'24, Su'25, W'25 — 3/7M)

Built-in Data Types

Type	Example	Description
int	`x = 10`	Whole numbers, no decimal
float	`y = 3.14`	Decimal/real numbers
str	`s = "Hello"`	Text / sequence of characters
bool	`b = True`	True or False values
complex	`c = 3+4j`	Real + imaginary part
list	`l = [1,2,3]`	Ordered, mutable collection
tuple	`t = (1,2,3)`	Ordered, immutable collection
dict	`d = {"a":1}`	Key-value pairs
set	`s = {1,2,3}`	Unordered, unique elements

Identifying Data Types

```
x = 10              # int
y = 3.14            # float
name = "GTU"        # str
flag = True         # bool
c = 2 + 3j          # complex

print(type(x))    # <class 'int'>
print(type(y))    # <class 'float'>
print(type(name))# <class 'str'>
print(type(flag))# <class 'bool'>
```

Type Conversion (Curriculum — 2.4)

Implicit (Automatic)

Python converts type automatically when safe.

```
x = 5       # int
y = 2.0     # float
z = x + y   # 7.0 (float)
print(type(z))
# <class 'float'>
```

Explicit (Manual)

Use `int()`, `float()`, `str()` etc.

```
a = "123"
b = int(a)    # 123
c = float(a) # 123.0
d = str(456) # "456"
e = bool(0)   # False
e = bool(5)   # True
```

Python Operators — Overview (W'24, Su'24 — 3/7M)

Types of Operators

Type	Examples
Arithmetic	`+`, `-`, `*`, `/`, `//`, `%`, `**`
Comparison	`==`, `!=`, `>`, `<`, `>=`, `<=`
Logical	`and`, `or`, `not`
Assignment	`=`, `+=`, `-=`, `*=`, `/=`
Bitwise	`&`, `|`, `^`, `~`, `<<`, `>>`
Membership	`in`, `not in`
Identity	`is`, `is not`

2.4 Operators in Detail

Arithmetic, Comparison, Logical, Assignment, Membership

Arithmetic Operators (Su'23 OR, Su'24 OR — 4M)

Arithmetic Operators with Examples

Op	Name	Example	Result
+	Addition	5 + 3	8
-	Subtraction	5 - 3	2
*	Multiplication	5 * 3	15
/	Division	7 / 2	3.5
//	Floor Division	7 // 2	3
%	Modulus	7 % 2	1
**	Exponentiation	2 ** 3	8

Comparison Operators (W'23 — 7M; W'25 OR — 7M)

Comparison (Relational) Operators

Op	Meaning	Example	Result
==	Equal to	5 == 5	True
!=	Not equal to	5 != 3	True
>	Greater than	5 > 3	True
<	Less than	3 < 5	True
>=	Greater or equal	5 >= 5	True
<=	Less or equal	3 <= 5	True

```
x, y = 10, 20
print(x > y)    # False
print(x == 10) # True
print(x != y)   # True
```

Logical Operators (Su'24 — 7M)

Logical Operators — and, or, not

Op	Meaning	Example	Result
`and`	Both True	`T and F`	False
`or`	Either True	`T or F`	True
`not`	Negation	`not T`	False

```
x, y = 10, 20
print(x > 5 and y > 5)    # True
print(x > 15 or y > 15)   # True
print(not(x > 5))          # False
```

Truth Table (and)

T and T = T T and F = F F and T = F F and F = F

Assignment Operators (W'24 OR — 3M; Su'23 — 7M)

Assignment Operators

Op	Equivalent to	Example	After (x=10)
=	Assign value	`x = 10`	x = 10
+=	x = x + val	`x += 5`	x = 15
-=	x = x - val	`x -= 3`	x = 7
*=	x = x * val	`x *= 2`	x = 20
/=	x = x / val	`x /= 2`	x = 5.0
//=	x = x // val	`x //= 3`	x = 3
%=	x = x % val	`x %= 3`	x = 1
**=	x = x ** val	`x **= 2`	x = 100

Membership Operators (W'24 — 3M)

Definition

Membership operators test whether a value exists in a sequence (string, list, tuple, set, dict).

in and not in

Op	Description	Returns
`in`	Returns True if value is in sequence	True / False
`not in`	Returns True if value not in sequence	True / False

```
fruits = ["apple","mango","banana"]
print("mango" in fruits)      # True
print("grape" not in fruits) # True
name = "Python"
print("Py" in name)           # True
```

Program: Average of Three Numbers (Su'23 — 4M)

Read three numbers and find average

```
# Program to find average of 3 numbers
a = float(input("Enter first number: "))
b = float(input("Enter second number: "))
c = float(input("Enter third number: "))

average = (a + b + c) / 3
print("Average =", average)
```

Sample Output

```
Enter first number: 10
Enter second number: 20
Enter third number: 30
Average = 20.0
```

Program: Celsius to Fahrenheit (W'23 — 7M)

Temperature Conversion

```
# Celsius to Fahrenheit conversion
celsius = float(input("Enter temperature in Celsius: "))

fahrenheit = (celsius * 9/5) + 32

print("Temperature in Fahrenheit =", fahrenheit)
```

Formula & Sample

$$F = \frac{C \times 9}{5} + 32$$

```
Enter temperature in Celsius: 100
Temperature in Fahrenheit = 212.0
```

Program: Simple & Compound Interest (Su'24 OR — 7M)

SI and CI Program

```
p = float(input("Principal: "))
r = float(input("Rate (%): "))
t = float(input("Time (years): "))

si = (p * r * t) / 100
ci = p * ((1 + r/100) ** t) - p

print("Simple Interest =", round(si, 2))
print("Compound Interest =", round(ci, 2))
```

Program: Total & Average Marks (W'24 — 7M)

Calculate total and average of 4 subject marks

```
s1 = float(input("Subject 1 marks: "))
s2 = float(input("Subject 2 marks: "))
s3 = float(input("Subject 3 marks: "))
s4 = float(input("Subject 4 marks: "))

total   = s1 + s2 + s3 + s4
average = total / 4

print("Total Marks   =", total)
print("Average Marks =", average)
```

Program: Square & Cube (W'24 OR — 7M)

Find square and cube of a number

```
num = float(input("Enter a number: "))

square = num ** 2
cube   = num ** 3

print("Square of", num, "=", square)
print("Cube of",   num, "=", cube)
```

Sample Output

```
Enter a number: 5
Square of 5.0 = 25.0
Cube of 5.0 = 125.0
```

Output Prediction — Operators (Su'23 OR, Su'24 OR — 4M)

Predict the Output

```
x = 10
y = 3
print(x * y)    # ?
print(x ** y)   # ?
print(x // y)   # ?
print(x % y)    # ?
print(x > y)    # ?
```

Answers

`x * y = 30` `x ** y = 1000` `x // y = 3` `x % y = 1` `x > y = True`

Comparison Operators — Full Program (Su'23 — 7M)

List all comparison operators with examples

```
a, b = 15, 10
print("a == b :", a == b)   # False
print("a != b :", a != b)   # True
print("a  > b :", a >  b)   # True
print("a  < b :", a <  b)   # False
print("a >= b :", a >= b)   # True
print("a <= b :", a <= b)   # False
```

Logical Operators — Full Program (Su'24 — 7M)

Demonstrate all logical operators

```
x, y = 5, 10

# and operator
print(x > 3 and y > 3)   # True
print(x > 3 and y > 15)  # False

# or operator
print(x > 3 or y > 15)   # True
print(x > 8 or y > 15)   # False

# not operator
print(not(x > 3))          # False
print(not(x > 8))          # True
```

Assignment Operators — Program (Su'23 — 7M)

Demonstrate three assignment operators

```
x = 10
print("Initial x =", x)    # 10

x += 5
print("After x+=5:", x)    # 15

x -= 3
print("After x-=3:", x)    # 12

x *= 2
print("After x*=2:", x)    # 24

x //= 4
print("After x//=4:", x)   # 6
```

2.5 Programs: Data Types

3-mark, 4-mark and 7-mark solutions

Data Types Program (W'24 — 4M)

List data types in Python and explain any three

```
# int type
age = 25
print("age =", age, "| type:", type(age))

# float type
price = 99.99
print("price =", price, "| type:", type(price))

# str type
name = "GTU"
print("name =", name, "| type:", type(name))

# bool type
flag = True
print("flag =", flag, "| type:", type(flag))
```

All Data Types — 7M Program (Su'25, W'25 — 7M)

List different data types with examples

```
x   = 10                # int
y   = 3.14              # float
s   = "Hello Python"    # str
b   = True              # bool
c   = 2 + 3j            # complex
lst = [1, 2, 3]         # list
tup = (4, 5, 6)         # tuple
dct = {"a": 1}          # dict
st  = {7, 8, 9}         # set

for v in [x,y,s,b,c,lst,tup,dct,st]:
    print(v, "->", type(v).__name__)
```

Types of Operators — 7M (Su'25 OR, W'25 — 7M)

Types of Operators in Python

Arithmetic	+, -, *, /, //, %, **
Comparison	==, !=, >, <, >=, <=
Logical	and, or, not
Assignment	=, +=, -=, *=, /=, //=, %=
Membership	in, not in
Identity	is, is not
Bitwise	&, |, ^, ~, «, »

Two types demonstrated

```
# Arithmetic + Relational
x, y = 12, 5
print(x + y, x // y, x % y) # 17 2 2
print(x > y, x == y)              # True False
```

input() and print() Functions

input() Function

Reads input from user as **string**.

```
name = input("Enter name: ")
age  = int(input("Age: "))
```

print() Function

Displays output to the screen.

```
print("Hello")          # basic
print("Age:", age)      # multi
print(3 + 4)            # expr
print("A", end=" ")     # end
print("B")              # A B
```

Type Safety

Always convert `input()` when numeric:
`x = int(input("..."))` or `x = float(input("..."))`

2.6 GTU PYQ Practice

Su'23 – W'25

3-Mark PYQ Questions

Try These (3 marks)

1. List the features of Python. *[Su'25, W'25]*
2. List the applications of Python. *[Su'25 OR, W'25 OR]*
3. Describe data types in Python with examples. *[W'23]*
4. List out rules for defining variables in Python. *[W'23 OR]*
5. Discuss membership operator of Python with example. *[W'24]*
6. Write a short note on assignment operator in Python. *[W'24 OR]*

4-Mark PYQ Questions

Try These (4 marks)

1. List various data types in Python and explain any three with example. *[W'24]*
2. Develop Python code to read three numbers and find their average. *[Su'23]*
3. Determine the output: `x=10, y=3; x*y, x**y, x//y, x%y, x>y.` *[Su'23 OR, Su'24 OR]*

7-Mark PYQ Questions

Try These (7 marks)

1. List assignment operators; build code to demonstrate any three. *[Su'23]*
2. Develop code to convert temperature from Celsius to Fahrenheit. *[W'23]*
3. List all comparison operators and explain each with Python code. *[W'23 OR]*
4. List all logical operators and explain each with Python code. *[Su'24]*
5. Calculate SI and CI. *[Su'24 OR]*
6. Calculate total and average marks of 4 subject marks. *[W'24]*
7. Find square and cube of a given number. *[W'24 OR]*
8. List different data types and explain all with examples. *[Su'25, W'25]*

Bitwise Operators (W'25 OR — 3M)

Bitwise Operators in Python

Op	Name	Example (a=10, b=6)	Result
&	AND	`10 & 6`	2 (1010 & 0110 = 0010)
|	OR	`10 | 6`	14 (1010 | 0110 = 1110)
^	XOR	`10 ^ 6`	12 (1010 ^ 0110 = 1100)
~	NOT	`~10`	-11 (inverts bits)
<<	Left Shift	`10 << 1`	20 (shift bits left)
>>	Right Shift	`10 >> 1`	5 (shift bits right)

Note

Bitwise operators work on integer binary representations. 10 in binary = 1010; 6 = 0110.

Identity Operators (W'25, Su'25 OR — 3M)

Identity Operators

Operator	Meaning	Returns True if
`is`	Identity check	Both point to **same object** in memory
`is not`	Not Identity	Both point to **different objects**

```
a = [1, 2, 3]
b = [1, 2, 3]
c = a
print(a is c)      # True  (same object)
print(a is b)      # False (different objects)
print(a == b)      # True  (same value)
print(a is not b)  # True
```

print() Function — Formatting (W'25 OR — 3M)

```python
# sep parameter: separator between values
print("A", "B", "C", sep="-")     # A-B-C
print(1, 2, 3, sep=", ")          # 1, 2, 3

# end parameter: what to print at end
print("Hello", end=" ")
print("World")                    # Hello World

# f-string formatting (Python 3.6+)
name = "Raj"
marks = 95.5
print(f"Name: {name}, Marks: {marks:.1f}")
# Name: Raj, Marks: 95.5

# format() method
print("Value = {:.2f}".format(3.14159))  # 3.14
```

Escape Sequences (Su'23 OR — 3M)

Common Escape Sequences

Sequence	Meaning	Example Output
\n	Newline	Moves to next line
\t	Tab	Horizontal tab space
\\	Backslash	Prints \
\"	Double quote	Prints "
\'	Single quote	Prints '
\r	Carriage return	Returns cursor to line start

```
print("Line1\nLine2")      # on two lines
print("Name\tMarks")       # Name    Marks
print("C:\\Python")        # C:\Python
print("He said \"Hi\"")    # He said "Hi"
```

Multiple Assignment & Operator Precedence (W'24 — 3M)

```
# Multiple assignment
a = b = c = 10          # all get 10
x, y, z = 1, 2, 3      # unpacking
a, b = b, a             # swap (Pythonic)

# Augmented assignment
x = 5
x += 3    # x = 8
x -= 2    # x = 6
x *= 4    # x = 24
x //= 5   # x = 4
x **= 2   # x = 16
```

Operator Precedence (High to Low)

** > ~+/- (unary) > * / // % > + - > << >> > & ^ | > == != < > <= >= > not > and > or

input() Function & Type Conversion (W'25 — 3M)

```python
# input() always returns a string
name = input("Enter name: ")      # string
age  = int(input("Enter age: ")) # integer
gpa  = float(input("GPA: "))      # float

# Type conversion functions
print(int("42"))        # 42
print(float("3.14"))    # 3.14
print(str(100))         # '100'
print(bool(0))          # False
print(bool(5))          # True

# Multiple inputs on one line
a, b = map(int, input("a b: ").split())
```

Program: Area of Circle & Triangle (W'25, Su'24 OR — 4M)

```python
import math

# Area of Circle
r = float(input("Enter radius: "))
area = math.pi * r * r
print(f"Area of circle = {area:.2f}")

# Area of Triangle (Heron's Formula)
a = float(input("Side a: "))
b = float(input("Side b: "))
c = float(input("Side c: "))
s = (a + b + c) / 2
area = math.sqrt(s*(s-a)*(s-b)*(s-c))
print(f"Area of triangle = {area:.2f}")
```

Program: BMI Calculator (W'25 OR — 4M)

```
# BMI = weight / (height^2)
weight = float(input("Weight (kg): "))
height = float(input("Height (m): "))

bmi = weight / (height ** 2)
print(f"BMI = {bmi:.2f}")

if bmi < 18.5:
    print("Underweight")
elif bmi < 25:
    print("Normal Weight")
elif bmi < 30:
    print("Overweight")
else:
    print("Obese")
```

Program: Hypotenuse & Power Without ** (Su'25 — 4M)

```python
import math

# Hypotenuse of right triangle
a = float(input("Base: "))
b = float(input("Height: "))
hyp = math.sqrt(a**2 + b**2)
print(f"Hypotenuse = {hyp:.2f}")

# Power without ** operator
base = int(input("Base: "))
exp  = int(input("Exponent: "))
result = 1
for i in range(exp):
    result *= base
print(f"{base}^{exp} = {result}")
```

Number Systems & Conversions (W'25 OR, Su'25 OR — 3M)

Literals

System	Prefix	Example
Binary	`0b` or `0B`	`0b1010` = 10
Octal	`0o` or `0O`	`0o12` = 10
Hexadecimal	`0x` or `0X`	`0xA` = 10

```
# Conversion functions
print(bin(10))    # 0b1010
print(oct(10))    # 0o12
print(hex(10))    # 0xa
print(int('1010', 2))   # binary to int = 10
print(int('12', 8))     # octal to int  = 10
print(int('A', 16))     # hex to int    = 10
```

Comments in Python (Su'23 — 3M)

Types of Comments

Type	Syntax
Single-line comment	`# This is a comment`
Multi-line string (docstring)	`'''...'''` or `"""..."""`

```python
# Single-line comment
x = 10    # inline comment

'''
This is a
multi-line comment
(docstring)
'''

def greet():
    """This function prints a greeting."""
    print("Hello!")
```

Output Prediction Practice (Su'24, W'24 — 4M)

```
# Predict output
a = 7
b = 3
print(a // b)      # ?
print(a % b)       # ?
print(a ** b)      # ?
print(a > b and b > 0)   # ?
print(not (a == 7))      # ?
```

Answers

a // b = 2
a % b = 1
a ** b = 343
a>b and b>0 = True
not (a==7) = False

Remember:
// = floor division
% = remainder
** = power

Program: Distance, Volume, Perimeter (Su'25 — 4M)

```python
# Distance using speed and time
speed = float(input("Speed (km/h): "))
time  = float(input("Time (h): "))
dist  = speed * time
print(f"Distance = {dist:.2f} km")

# Volume of cylinder
import math
r = float(input("Radius: "))
h = float(input("Height: "))
vol = math.pi * r * r * h
print(f"Volume = {vol:.2f}")

# Perimeter of rectangle
l = float(input("Length: "))
w = float(input("Width: "))
print(f"Perimeter = {2*(l+w):.2f}")
```

String Basics (W'23 OR, Su'25 — 3M)

```
s = "Python"
print(len(s))          # 6
print(s[0])            # P    (indexing)
print(s[-1])           # n    (last char)
print(s[0:3])          # Pyt  (slicing)
print(s * 2)           # PythonPython
print("Py" in s)       # True

# String is immutable
# s[0] = "p"           # Error!
name = input("Name: ")
print("Hello, " + name + "!")   # concatenation
```

Worked: All Operator Types Program (W'25, Su'25 — 7M)

```
a, b = 15, 4
# Arithmetic
print(a+b, a-b, a*b, a/b, a//b, a%b, a**b)
# 19 11 60 3.75 3 3 50625

# Comparison
print(a>b, a<b, a==b, a!=b)    # T F F T

# Logical
print(a>5 and b>0)   # True
print(a>20 or b>2)   # True
print(not(a>5))      # False

# Assignment (augmented)
x = 10; x += 5; print(x)   # 15

# Membership
print(15 in [10,15,20])     # True
print("t" in "Python")      # True
```

Chained Comparison & Ternary Expression (W'25 OR — 3M)

```python
# Chained comparison (Pythonic)
x = 15
print(10 < x < 20)    # True
print(1 <= x <= 100) # True

# Ternary (conditional) expression
a, b = 10, 20
max_val = a if a > b else b
print("Max:", max_val)    # Max: 20

# type() and isinstance()
n = 42
print(type(n))              # <class 'int'>
print(isinstance(n, int))   # True
print(isinstance(n, float))# False
```

Quick Revision: Data Types & Variables

Data Type	Example	type()	Mutable
int	`42, -5`	`int`	No
float	`3.14`	`float`	No
str	`"Hi"`	`str`	No
bool	`True`	`bool`	No
list	`[1,2]`	`list`	**Yes**
tuple	`(1,2)`	`tuple`	No
dict	`{k:v}`	`dict`	**Yes**
set	`{1,2}`	`set`	**Yes**

Variable Rules

- Start with letter or _
- Case-sensitive: `Name` $\neq$ `name`
- No spaces or special chars (except _)
- Cannot use Python keywords (`if`, `for`, `while`…)

Worked: Largest of Three Numbers (Su'24, W'23 — 4M)

```python
# Method 1: if-elif-else
a = int(input("a: "))
b = int(input("b: "))
c = int(input("c: "))

if a >= b and a >= c:
    print("Largest:", a)
elif b >= c:
    print("Largest:", b)
else:
    print("Largest:", c)

# Method 2: using max()
print("Largest:", max(a, b, c))
```

Worked: Multiple Python Short Programs (W'25 — 4M)

```python
# Swap without temp (Pythonic)
a, b = 10, 20
a, b = b, a
print(f"a={a}, b={b}")  # a=20, b=10

# Count even numbers in list
nums = [1,2,3,4,5,6,7,8,9,10]
even = sum(1 for n in nums if n%2==0)
print("Even count:", even)   # 5

# Print ASCII value of a character
ch = input("Character: ")
print(f"ASCII of {ch} = {ord(ch)}")
print(f"Char of 65 = {chr(65)}")

# Check if number is single digit
n = int(input("Number: "))
if 0 <= n <= 9:
    print("Single digit")
else:
    print("Multiple digits")
```

Unit 2 — Rapid Revision Flash Sheet

Topic	Key Fact
Python features	Interpreted, dynamic, OOP, open-source
Program structure	`import`, variables, statements, functions
Identifiers	Start with letter/_; case-sensitive
Data types	int, float, str, bool, complex, list, tuple, dict, set
Explicit conversion	`int()`, `float()`, `str()`, `bool()`
Arithmetic ops	`+ - * / // % **` (7 operators)
Comparison ops	`== != > < >= <=` (return bool)
Logical ops	`and or not`
Assignment ops	`= += -= *= /= //= %= **=`
Membership ops	`in`, `not in`
Identity ops	`is`, `is not` (check object identity)
Bitwise ops	`& | ^ ~ << >>`
`input()`	Always returns **string**; use `int()` to convert
`print()`	`sep=`, `end=`, f-strings

Unit 2 — Summary

Key Concepts

Features	Simple, interpreted, OOP, open-source, cross-platform
Data Types	int, float, str, bool, complex, list, tuple, dict, set
Identifiers	Start with letter/_, case-sensitive, no keywords
Operators	Arithmetic, Comparison, Logical, Assignment, Membership
Type Conv.	int(), float(), str(), bool() for explicit conversion
I/O	`input()` returns str; `print()` displays output

Exam Weightage (Unit 2 — 14%)

3-mark × 1–2 | 4-mark × 1 | 7-mark × 1–2

Unit 3: Flow of Control

Python Programming (DI02032011)

Milav Dabgar

Lecturer, Dept of ICT
GP Palanpur

April 2, 2026

Unit 3 — Contents

What is Flow of Control?

Definition

Flow of Control refers to the order in which individual statements, instructions, or function calls are executed or evaluated in a program.

Types of Flow of Control:

- **Sequential** — Statements execute one after another (default)
- **Selection / Decision** — Execute different code based on a condition
- **Iteration / Loop** — Repeat a block of code multiple times
- **Jump** — Skip or break out of a block (`break`, `continue`)

Selection Statements — Overview

Selection / Decision Statements

Used to **choose** a path of execution based on a condition.

Statement	Description
`if`	Execute block only if condition is True
`if-else`	One block if True, another if False
`if-elif-else`	Multiple conditions / ladder
`nested if`	`if` inside another `if`

The if Statement

Syntax:

```
if condition:
    # block executed if condition is True
```

Example — check positive number:

```
n = int(input("Enter number: "))
if n > 0:
    print("Number is positive")
print("End of program")
```

Note: Indentation defines the block. If condition is False, the block is skipped.

The if-else Statement (Su'23 Q.3b, Su'25 Q.2b)

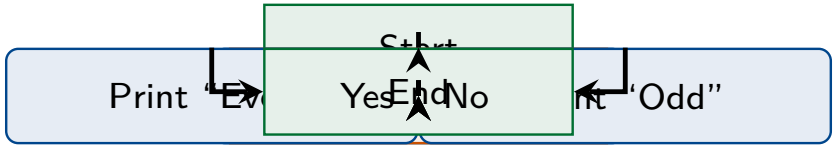
Syntax:

```
if condition:
    # block if True
else:
    # block if False
```

Example — even or odd:

```
n = int(input("Enter number: "))
if n % 2 == 0:
    print(n, "is Even")
else:
    print(n, "is Odd")
```

Flowchart — `if-else`



Nested `if` Statement (W'23 Q.2b, Su'24 Q.3b)

Syntax:

```
if condition1:
    if condition2:
        # inner block
    else:
        # inner else
else:
    # outer else
```

Example — positive, negative or zero:

```
n = int(input("Enter number: "))
if n >= 0:
    if n == 0:
        print("Zero")
    else:
        print("Positive")
else:
```

The `if-elif-else` Ladder (W'24 Q.3a)

Syntax:

```
if condition1:
    # block 1
elif condition2:
    # block 2
elif condition3:
    # block 3
else:
    # default block
```

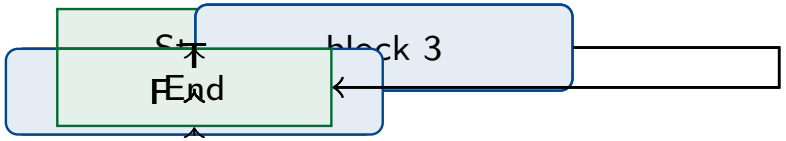
- Conditions are checked **top to bottom**
- First True condition executes its block
- `else` is optional — runs if none match
- Used when there are **multiple exclusive** conditions

if-elif-else — Grade Example (Su'24 Q.2c OR)

Grade Calculator

```
marks = int(input("Enter marks: "))
if marks >= 70:
    print("Grade: Distinction")
elif marks >= 60:
    print("Grade: First Class")
elif marks >= 50:
    print("Grade: Second Class")
elif marks >= 35:
    print("Grade: Pass Class")
else:
    print("Grade: Fail")
```

if-elif-else — Flowchart



Selection Statements — Comparison Table

Statement	When to use	Branches
`if`	Single optional action	1 (or skip)
`if-else`	Two exclusive paths	2
`if-elif-else`	Multiple exclusive conditions	3 or more
`nested if`	Conditions within conditions	Multiple levels

Loops — Overview

What is a Loop?

A loop **repeats** a block of code as long as a condition is True (or for each item in a sequence).

Loop	Description
`for`	Iterates over a sequence or range
`while`	Repeats while condition is True
`nested loop`	A loop inside another loop

Jump statements inside loops:

- `break` — exit the loop immediately
- `continue` — skip to next iteration

The for Loop (Su'23 Q.3b OR, W'23 Q.2b OR)

Syntax:

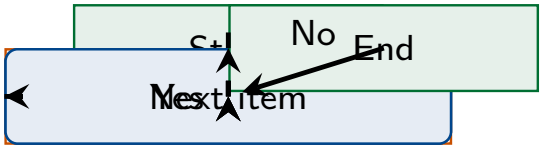
```
for variable in sequence:
    # loop body
```

Common uses with range():

```
# Print 1 to 10
for i in range(1, 11):
    print(i, end=" ")
# Output: 1 2 3 4 5 6 7 8 9 10

# range(start, stop, step)
for i in range(1, 20, 2):
    print(i, end=" ")  # odd numbers
```

`for` Loop — Flowchart



The while Loop (Su'24 Q.2c)

Syntax:

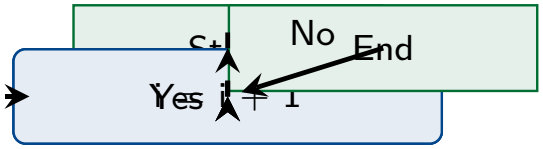
```
while condition:
    # loop body
    # update condition variable
```

Example — print 1 to 10:

```
i = 1
while i <= 10:
    print(i, end=" ")
    i = i + 1
# Output: 1 2 3 4 5 6 7 8 9 10
```

Warning: Forgetting to update the condition variable causes an **infinite loop**!

`while` Loop — Flowchart (Su'24 Q.2c)



`for` vs `while` — Comparison (W'25 Q.3b)

for loop	while loop
Fixed number of iterations	Indefinite iterations
Iterates over a sequence	Runs while condition True
Loop variable managed by loop	Programmer manages variable
Compact syntax with `range()`	More flexible for input loops
Best for lists, strings, ranges	Best for user-input validation

Nested Loops (W'24 Q.3a OR)

Definition

A loop inside another loop is called a **nested loop**. The inner loop runs **completely** for every iteration of the outer loop.

Example:

```
for i in range(1, 4):          # outer
    for j in range(1, 4):      # inner
        print(i * j, end="   ")
    print()
# Output:
# 1   2   3
# 2   4   6
# 3   6   9
```

Program: Numbers 1–10 & Odd/Even 1–N (W'23 Q.3a)

Print 1 to 10 using for loop

```
for i in range(1, 11):
    print(i, end=" ")
```

Odd and Even from 1 to N

```
n = int(input("Enter N: "))
for i in range(1, n + 1):
    if i % 2 == 0:
        print(i, "is Even")
    else:
        print(i, "is Odd")
```

Program: Sum 1 to 100 (Su'24 Q.3a OR)

Sum of numbers from 1 to 100

```
total = 0
for i in range(1, 101):
    total = total + i
print("Sum =", total)
# Sum = 5050
```

Using while loop

```
total = 0
i = 1
while i <= 100:
    total += i
    i += 1
print("Sum =", total)
```

`break` Statement (Su'23 Q.3a)

Definition

`break` **immediately exits** the loop, regardless of the loop condition.

Syntax:

```
for/while ...:
    if condition:
        break
```

Example — find first even number:

```
for i in range(1, 20):
    if i % 2 == 0:
        print("First even:", i)
        break
```

`continue` Statement (Su'23 Q.3a OR)

Definition

`continue` **skips** the rest of the current iteration and moves to the next one.

Example — skip even numbers:

```
for i in range(1, 11):
    if i % 2 == 0:
        continue        # skip even
    print(i, end=" ")
# Output: 1 3 5 7 9
```

Key difference:

- `break` — exits the loop entirely
- `continue` — skips current iteration, loop continues

Star Triangle Pattern (W'23 Q.3b)

Print star triangle (5 rows):

```
for i in range(1, 6):
    for j in range(1, i + 1):
        print("*", end=" ")
    print()
```

Output:

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

Inverted Star Triangle Pattern (Su'23 Q.2c OR)

Print inverted star triangle (5 rows):

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

Output:

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

Number Triangle Pattern (Su'23 Q.2c, Su'24 Q.3b OR)

Problem: Print `1 / 1 2 / 1 2 3 / 1 2 3 4 / 1 2 3 4 5`

```
for i in range(1, 6):
    for j in range(1, i + 1):
        print(j, end=" ")
    print()
```

Output:

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

& Character Triangle Pattern (Su'25 Q.1c)

Print & triangle (5 rows):

```
for i in range(1, 6):
    for j in range(1, i + 1):
        print("&", end=" ")
    print()
```

Inverted @ triangle (W'25 Q.2c):

```
for i in range(5, 0, -1):
    for j in range(1, i + 1):
        print("@", end=" ")
    print()
```

Alphabet Triangle Pattern (W'24 Q.3c OR)

Print A / AB / ABC / ABCD / ABCDE:

```
for i in range(1, 6):
    for j in range(i):
        print(chr(65 + j), end="")
    print()
```

Output:

```
A
AB
ABC
ABCD
ABCDE
```

Note: chr(65) = 'A', chr(66) = 'B', etc.

Number Series: Row Numbers (Su'23 Q.2c)

Print: 1 2 3 / 4 5 6 / 7 8 9 10 (triangular number series):

```
n = 1
for i in range(1, 5):
    for j in range(1, i + 1):
        print(n, end=" ")
        n += 1
    print()
```

Output:

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

Squared Number Series (W'25 Q.2c)

Print: 1 / 1 4 / 1 4 9 / 1 4 9 16:

```
for i in range(1, 5):
    for j in range(1, i + 1):
        print(j * j, end=" ")
    print()
```

Output:

```
1
1 4
1 4 9
1 4 9 16
```

Program: Prime Number Check (Su'23 Q.2c, W'25 Q.3b OR)

Check if number is prime

```
n = int(input("Enter number: "))
is_prime = True
if n < 2:
    is_prime = False
else:
    for i in range(2, int(n**0.5) + 1):
        if n % i == 0:
            is_prime = False
            break
if is_prime:
    print(n, "is Prime")
else:
    print(n, "is Not Prime")
```

Program: Odd/Even + Average 1–n (Su'25 Q.1c OR)

Check odd or even

```
n = int(input("Enter number: "))
if n % 2 == 0:
    print(n, "is Even")
else:
    print(n, "is Odd")
```

Average of numbers 1 to n

```
n = int(input("Enter n: "))
total = 0
for i in range(1, n + 1):
    total += i
print("Average =", total / n)
```

Program: Print 1–25 with `for` Loop (W'25 Q.3b)

Print numbers 1 to 25 using for loop

```
for i in range(1, 26):
    print(i, end=" ")
print()
```

Print odd numbers 1 to 20 using while loop

```
i = 1
while i <= 20:
    if i % 2 != 0:
        print(i, end=" ")
    i += 1
```

PYQ Practice — 3-Mark Questions

3-Mark PYQ Bank

1. Describe the use of `break` statement with examples *(Su'23)*
2. Describe the use of `continue` statement with examples *(Su'23 OR, Su'24 OR)*
3. Develop Python code to identify if number is even or odd *(Su'23)*
4. Create Python code to check if number is positive or negative *(Su'23 OR)*
5. Explain if-elif-else statement with flowchart and example *(W'24)*
6. Develop Python program to print numbers 1 to 10 using loops *(W'23)*
7. Develop Python code to find odd and even numbers from 1 to N *(W'23 OR)*
8. Write Python program to print odd numbers between 1 to 20 *(Su'24)*
9. Write Python program to find sum of numbers from 1 to 100 *(Su'24 OR)*
10. Explain nested loop with a suitable example *(W'24 OR)*

PYQ Practice — 4-Mark Questions

4-Mark PYQ Bank

1. Explain if...else statement with suitable example *(Su'23, Su'25, W'25)*
2. Explain Nested if statement with Python code example *(W'23, Su'24, W'25 OR)*
3. Explain `for` loop statement with example *(Su'23 OR, W'23 OR, W'24 OR, Su'25)*
4. Explain the need for `continue` and `break` statements *(W'24)*
5. Explain `while` loop with example *(Su'25 OR)*
6. Explain `break` statement with example *(Su'25 OR)*
7. Differences between `for` and `while` loop; print 1–25 using `for` *(W'25)*
8. Develop Python program to find if the entered number is prime or not *(W'25 OR)*
9. Print star pattern (5 rows) *(W'23)*
10. Print number triangle: 1 2 3 / 4 5 6 / 7 8 9 10 *(Su'24 OR)*

PYQ Practice — 7-Mark Questions

7-Mark PYQ Bank

1. Write Python code to check whether entered number is prime *(Su'23)*
2. Display patterns: (A) number triangle (B) inverted star triangle *(Su'23 OR)*
3. Explain different types of selection/decision making structures *(W'23)*
4. Describe `break` and `continue` statements in Python *(W'23 OR)*
5. Explain `while` loop with syntax, flowchart and Python code *(Su'24)*
6. Explain if-elif-else ladder with syntax, flowchart and Python code *(Su'24 OR)*
7. Patterns: (A) `&` character triangle (B) number triangle 1/12/123 *(Su'25)*
8. Programs: (1) odd or even (2) average of 1 to n *(Su'25 OR)*
9. Patterns: (A) `@` inverted triangle (B) squared number series *(W'25)*
10. Explain `for` loop with syntax, flowchart and Python code *(W'25 OR)*
11. Pattern: A / AB / ABC / ABCD / ABCDE *(W'24 OR)*

Worked Solution: `break` & `continue` (W'23 Q.2c OR)

break statement

```
# break: exit loop when 5 is found
for i in range(1, 11):
    if i == 5:
        print("Found 5, breaking!")
        break
    print(i, end=" ")
# Output: 1 2 3 4 Found 5, breaking!
```

continue statement

```
# continue: skip multiples of 3
for i in range(1, 11):
    if i % 3 == 0:
        continue
    print(i, end=" ")
# Output: 1 2 4 5 7 8 10
```

Worked Solution: Selection Structures (W'23 Q.2c)

All three selection types in one program

```
# if statement
x = 10
if x > 0:
    print("Positive")

# if-else
if x % 2 == 0:
    print("Even")
else:
    print("Odd")

# if-elif-else
if x >= 90:
    print("A")
elif x >= 75:
    print("B")
elif x >= 50:
    print("C")
else:
    print("F")
```

Worked: `while` Loop — Full Solution (Su'24 Q.2c)

`while` loop syntax:

```
while condition:
    # body
    # update
```

Example — sum of digits:

```
n = int(input("Enter number: "))
total = 0
while n != 0:
    digit = n % 10
    total += digit
    n = n // 10
print("Sum of digits:", total)
```

Worked: `for` Loop — Full Solution (W'25 Q.2c OR)

Syntax and examples:

```
# Syntax
for variable in sequence:
    body

# Iterate over a list
fruits = ["apple", "banana", "cherry"]
for f in fruits:
    print(f)

# Iterate over range
for i in range(1, 6):
    print(i ** 2, end=" ")
# Output: 1 4 9 16 25

# Iterate over string
for ch in "PYTHON":
    print(ch, end="-")
```

The pass Statement (Su'25 OR, W'25 — 3M)

What is pass?

pass is a **null operation** — it does nothing. Used as a placeholder when a statement is syntactically required but no code is needed.

```
# Placeholder in if
x = 10
if x > 5:
    pass            # do nothing for now
else:
    print("small")

# Placeholder in loop
for i in range(5):
    pass            # empty loop body

# Placeholder in function
def future_function():
    pass            # to be implemented later
```

range() Function (Su'24, W'24 OR — 3M)

Syntax of range()

`range(stop)` | `range(start, stop)` | `range(start, stop, step)`

```
for i in range(5):          # 0 1 2 3 4
    print(i, end=" ")

for i in range(1, 6):       # 1 2 3 4 5
    print(i, end=" ")

for i in range(0, 11, 2): # 0 2 4 6 8 10
    print(i, end=" ")

for i in range(10, 0, -2):# 10 8 6 4 2
    print(i, end=" ")

# Countdown
for i in range(5, 0, -1): # 5 4 3 2 1
    print(i, end=" ")
```

`else` Clause with Loops (Su'25 OR, W'25 OR — 3M)

Loop-else Concept

The `else` block after a loop executes when the loop completes **normally** (not interrupted by `break`).

```
# for-else: search example
nums = [1, 3, 5, 7, 9]
target = 6
for n in nums:
    if n == target:
        print("Found!")
        break
else:
    print("Not found")    # prints if no break

# while-else
i = 0
while i < 3:
    print(i)
    i += 1
else:
    print("Loop done")    # runs when i==3
```

Program: Multiplication Table (W'23, Su'24 — 4M)

```python
# Multiplication table of N
n = int(input("Enter number: "))
print(f"Multiplication Table of {n}:")
for i in range(1, 11):
    print(f"{n} x {i:2d} = {n*i:3d}")

# Output for n=5:
# 5 x  1 =   5
# 5 x  2 =  10
# ...
# 5 x 10 =  50

# Nested: All tables from 1 to 5
for n in range(1, 6):
    for i in range(1, 11):
        print(f"{n}x{i}={n*i}", end="  ")
    print()
```

Program: Factorial & Sum of Digits (W'24, Su'25 — 4M)

```python
# Factorial using while loop
n = int(input("N: "))
fact, i = 1, 1
while i <= n:
    fact *= i
    i += 1
print(f"{n}! = {fact}")

# Sum of digits
num = int(input("Number: "))
total = 0
while num > 0:
    total += num % 10
    num //= 10
print("Sum of digits =", total)
```

Program: Fibonacci Series (W'25, Su'25 — 7M)

```python
# Fibonacci series up to N terms
n = int(input("How many terms? "))
a, b = 0, 1
count = 0
print("Fibonacci series:")
while count < n:
    print(a, end=" ")
    a, b = b, a + b
    count += 1

# Alternative: for loop version
a, b = 0, 1
for _ in range(n):
    print(a, end=" ")
    a, b = b, a + b
# Output (n=8): 0 1 1 2 3 5 8 13
```

Program: Reverse Number & Palindrome (Su'24 OR, W'23 — 4M)

```python
# Reverse a number
n = int(input("Number: "))
original = n
rev = 0
while n > 0:
    rev = rev * 10 + n % 10
    n //= 10
print("Reversed:", rev)
if original == rev:
    print(f"{original} is a Palindrome")
else:
    print(f"{original} is NOT a Palindrome")
```

Program: Sum of Series (W'25, Su'24 — 7M)

```python
# Series: 1 + 1/2 + 1/3 + ... + 1/N
n = int(input("N: "))
total = 0.0
for i in range(1, n+1):
    total += 1/i
print(f"Sum = {total:.4f}")

# Series: 1^2 + 2^2 + ... + N^2
total2 = 0
for i in range(1, n+1):
    total2 += i*i
print(f"Sum of squares = {total2}")

# Series: 1 + 3 + 5 + ... (N odd numbers)
total3, i, cnt = 0, 1, 0
while cnt < n:
    total3 += i; i += 2; cnt += 1
print(f"Sum of {n} odd numbers = {total3}")
```

Diamond Star Pattern (W'25 OR, Su'25 — 7M)

```
n = int(input("Rows: "))    # e.g. n=4
# Upper half (including middle)
for i in range(1, n+1):
    print(" "*(n-i) + "*"*(2*i-1))
# Lower half
for i in range(n-1, 0, -1):
    print(" "*(n-i) + "*"*(2*i-1))

# Output (n=4):
#     *
#    ***
#   *****
#  *******
#   *****
#    ***
#     *
```

Floyd's Triangle & Right Angle (Su'24 OR, W'25 — 4M)

```python
# Floyd's Triangle (consecutive numbers)
n = 5
num = 1
for i in range(1, n+1):
    for j in range(i):
        print(num, end=" ")
        num += 1
    print()
# 1
# 2 3
# 4 5 6
# 7 8 9 10
# 11 12 13 14 15

# Right-angle with row numbers
for i in range(1, n+1):
    for j in range(1, i+1):
        print(j, end=" ")
    print()
# 1  / 1 2  / 1 2 3 ...
```

Nested Loop: Multiplication Table Grid (W'25 — 7M)

```python
# Print 5x5 multiplication table grid
n = 5
print("    ", end="")
for j in range(1, n+1):
    print(f"{j:4d}", end="")
print()
print("-" * (4*n+5))
for i in range(1, n+1):
    print(f"{i:3d}|", end="")
    for j in range(1, n+1):
        print(f"{i*j:4d}", end="")
    print()
# Output:
#        1   2   3   4   5
# ----------------------
#   1|   1   2   3   4   5
#   2|   2   4   6   8  10
```

Program: Count Digits & Armstrong Number (Su'25, W'24 — 4M)

```python
# Count digits in a number
n = int(input("Number: "))
count = 0
temp = n
while temp != 0:
    temp //= 10
    count += 1
print(f"Digits: {count}")

# Armstrong Number (3-digit: sum of cubes)
n = int(input("Number: "))
digits = [int(d) for d in str(n)]
p = len(digits)
arm_sum = sum(d**p for d in digits)
if arm_sum == n:
    print(f"{n} is Armstrong")
else:
    print(f"{n} is NOT Armstrong")
```

Worked: break, continue, pass (W'24 OR — 7M)

```python
# Find first multiple of 7 between 1-50
for i in range(1, 51):
    if i % 7 == 0:
        print(f"First multiple of 7: {i}")
        break       # exit loop immediately

# Print only even numbers 1-10 (skip odd)
for i in range(1, 11):
    if i % 2 != 0:
        continue    # skip rest of this iteration
    print(i, end=" ")  # 2 4 6 8 10

# pass: future implementation placeholder
for i in range(5):
    if i == 3:
        pass        # do nothing special for 3
    else:
        print(i, end=" ")  # 0 1 2 4
```

Worked: while Loop with Validation (W'25, Su'25 — 7M)

```
# Grade entry until 'done'
total, count = 0, 0
while True:
    grade = input("Grade (or 'done'): ")
    if grade == 'done':
        break
    score = float(grade)
    total += score
    count += 1
if count > 0:
    print(f"Average: {total/count:.2f}")

# Count positive/negative/zero
n = int(input("How many numbers? "))
pos = neg = zer = 0
for _ in range(n):
    x = int(input())
    if x > 0: pos += 1
    elif x < 0: neg += 1
    else: zer += 1
print(f"Pos={pos}, Neg={neg}, Zero={zer}")
```

Unit 3 — Rapid Revision Sheet

Statement	Key Points
`if-elif-else`	Check multiple conditions in sequence
`for` loop	Fixed iterations using `range()` or sequence
`while` loop	Repeat while condition is True
`break`	Exit loop immediately
`continue`	Skip current iteration, go to next
`pass`	Null statement; placeholder
`range(s,e,st)`	Generate sequence from s to e-1 step st
loop `else`	Runs when loop completes without `break`
Nested loops	Use for 2D patterns, tables, matrix
Indentation	Mandatory in Python (4 spaces or 1 tab)

GTU Pattern (Unit 3 — 14%)

3-mark: definitions, comparisons 4-mark: programs 7-mark: patterns + programs

Common Mistakes in Loops (Exam Guide)

Mistakes

× Infinite while (missing update)
× Off-by-one (`range(1,n)` vs `range(1,n+1)`)
× Wrong indentation
× Using = instead of == in `if`
× Modifying loop variable inside
× Missing `break` in when needed

Correct Patterns

`while i < n: …; i += 1`
`range(1, n+1)` for 1 to n
Always 4 spaces indent
`if x == 5:` not `if x = 5:`
Update counter outside body
Use `break` to exit search

Worked: Pattern Analysis Method (7M Exam Guide)

How to Design Any Pattern Loop

Step 1: Draw the pattern and number each row (starting from 1).
Step 2: Identify what is printed per row (spaces + characters).
Step 3: Find the formula for spaces and characters in terms of row i.
Step 4: Outer loop: `for i in range(1, n+1)`
Step 5: Inner loop: use formula from Step 3.

```
# Right-angle triangle (n=4)
# Row 1: *              (1 star)
# Row 2: * *            (2 stars)
# Row 3: * * *          (3 stars)
# Row 4: * * * *        (4 stars)
# Formula: row i has i stars
n = 4
for i in range(1, n+1):
    print("* " * i)
```

Worked: GCD & LCM Using Loop (Su'25, W'24 — 4M)

```
# GCD using Euclidean algorithm
a = int(input("A: "))
b = int(input("B: "))
original_a, original_b = a, b
while b != 0:
    a, b = b, a % b
gcd = a
print(f"GCD of {original_a} and {original_b} = {gcd}")

# LCM using GCD
lcm = (original_a * original_b) // gcd
print(f"LCM = {lcm}")

# Output for A=12, B=8:
# GCD = 4
# LCM = 24
```

Worked: Comprehensive Loop Programs (W'25 — 7M)

```python
# Sum of N terms: 1 - 1/2 + 1/3 - 1/4 ...
n = int(input("N: "))
s, sign = 0, 1
for i in range(1, n+1):
    s += sign * (1/i)
    sign *= -1
print(f"Sum = {s:.4f}")

# Check perfect number
n = int(input("Number: "))
total = sum(i for i in range(1, n) if n % i == 0)
if total == n:
    print(f"{n} is Perfect")
else:
    print(f"{n} is NOT Perfect")

# Count odd numbers between A and B
a, b = int(input("A: ")), int(input("B: "))
count = sum(1 for i in range(a, b+1) if i % 2 != 0)
print(f"Odd count between {a} and {b}: {count}")
```

Unit 3 — Summary

Selection

if — single condition
if-else — two branches
if-elif-else — ladder
nested if — multi-level

Jump

break — exit loop
continue — skip iteration

Loops

for — fixed iterations
while — condition-based
nested — loop inside loop

Patterns

Star / Number / Character
Triangle / Inverted / Series
Use nested `for` loops

Unit 4: Functions

Python Programming (DI02032011)

Milav Dabgar

Lecturer, Dept of ICT
GP Palanpur

April 2, 2026

Unit 4 — Contents

What is a Function?

Definition

A **function** is a named, reusable block of code that performs a specific task. It is defined once and can be called (invoked) multiple times.

Why use functions?

- **Reusability** — write once, use many times
- **Modularity** — break large programs into smaller pieces
- **Readability** — code is easier to understand
- **Maintainability** — fix bugs in one place

Types of functions in Python:

- **Built-in functions** — `print()`, `len()`, `input()`, `max()`, etc.
- **User-defined functions** — defined using `def` keyword
- **Module functions** — from `math`, `random`, `statistics`

Defining and Calling a Function (Su'23 Q.4b, W'24 Q.3b)

Syntax:

```
def function_name(parameters):
    """Docstring (optional)"""
    # function body
    return value    # optional
```

Example:

```
def greet(name):                # define
    print("Hello,", name)

greet("Alice")                  # call  --> Hello, Alice
greet("Bob")                    # call  --> Hello, Bob
```

Function with Return Value (W'24 Q.3b OR)

return statement

return sends a value back to the caller and exits the function.

```
def add(a, b):
    result = a + b
    return result

x = add(5, 3)       # x = 8
print(x)

def square(n):
    return n * n

print(square(4))    # 16
print(square(7))    # 49
```

Types of Arguments

Positional Arguments

```
def greet(name, age):
    print(name, "is", age, "years old")

greet("Alice", 20)    # positional
```

Default Arguments

```
def greet(name, msg="Good morning"):
    print(msg + ",", name)

greet("Alice")                  # uses default
greet("Bob", "Good night")   # override default
```

Function: Factorial (W'23 Q.3c, W'24 Q.3c, Su'25 Q.4b)

User-defined function to find factorial

```
def factorial(n):
    fact = 1
    for i in range(1, n + 1):
        fact = fact * i
    return fact

num = int(input("Enter number: "))
result = factorial(num)
print("Factorial of", num, "=", result)
```

Example: `factorial(5)` $= 5 \times 4 \times 3 \times 2 \times 1 = 120$

Function: Fibonacci Series (Su'23 Q.3c, W'24 Q.4c OR)

Fibonacci series 0 to N

```
def fibonacci(n):
    a, b = 0, 1
    print("Fibonacci series:")
    while a <= n:
        print(a, end=" ")
        a, b = b, a + b
    print()

n = int(input("Enter N: "))
fibonacci(n)
# For N=20: 0 1 1 2 3 5 8 13
```

Function: Armstrong & Palindrome (Su'23 Q.3c OR, W'23 Q.4c)

Check Armstrong number

```
def is_armstrong(n):
    digits = str(n)
    power = len(digits)
    total = sum(int(d)**power for d in digits)
    return total == n

print(is_armstrong(153))  # True (1^3+5^3+3^3=153)
```

Check palindrome

```
def is_palindrome(n):
    return str(n) == str(n)[::-1]

print(is_palindrome(121))   # True
print(is_palindrome(123))   # False
```

Function: Prime Check (Su'25 Q.4b OR)

Check prime using user-defined function

```
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

num = int(input("Enter number: "))
if is_prime(num):
    print(num, "is Prime")
else:
    print(num, "is Not Prime")
```

Function: Reverse & Cube of Odds (Su'24 Q.3c OR, W'24 Q.4c)

Reverse the entered value

```
def reverse_value(n):
    return str(n)[::-1]

val = input("Enter value: ")
print("Reversed:", reverse_value(val))
```

Cube of odd numbers 1 to 7

```
def cube_of_odds():
    for i in range(1, 8):
        if i % 2 != 0:
            print(f"Cube of {i} = {i**3}")

cube_of_odds()
```

Local Variables (Su'25 Q.3a)

Local Variable

A variable defined **inside a function** is local. It exists only within that function and cannot be accessed outside.

```
def my_func():
    x = 10          # local variable
    print("Inside:", x)

my_func()
# print(x)    # ERROR: x not defined outside
```

Global Variables (Su'25 Q.3a OR)

Global Variable

A variable defined **outside all functions** is global. Use the `global` keyword inside a function to modify it.

```
count = 0              # global variable

def increment():
    global count       # declare global
    count += 1

increment()
increment()
print("Count:", count)   # Count: 2
```

Scope of Variables — Full Example (W'23 Q.3c OR, W'25 Q.3c)

Local vs Global scope

```
x = 100              # global

def show():
    y = 200          # local
    print("Inside: x =", x, "y =", y)

show()
print("Outside: x =", x)
# print(y)  # ERROR: y not accessible here
```

Variable	Scope	Accessible
x	Global	Everywhere
y	Local (show)	Only inside `show()`

Built-in Functions (W'23 Q.4a OR, W'24 Q.4a)

Common built-in functions

```
# print(), input()
name = input("Name: ")
print("Hello,", name)

# len(), max(), min(), sum()
nums = [3, 7, 1, 9, 4]
print(len(nums))     # 5
print(max(nums))     # 9
print(min(nums))     # 1
print(sum(nums))     # 24

# pow(), abs(), round()
print(pow(2, 8))     # 256
print(abs(-15))      # 15
print(round(3.14159, 2))   # 3.14
```

Math Module (W'23 Q.4a, Su'24 Q.4a, Su'25 Q.3c)

Import and use math module

Use `import math` to access mathematical functions.

```
import math

print(math.sqrt(25))      # 5.0
print(math.pow(2, 10))    # 1024.0
print(math.floor(4.7))    # 4
print(math.ceil(4.2))     # 5
print(math.factorial(5)) # 120
print(math.pi)            # 3.14159...
print(math.log(100, 10)) # 2.0
print(math.sin(math.pi/2)) # 1.0
```

Math Module — Functions Table (W'23 Q.4a, W'25 Q.3c OR)

Function	Description
`math.sqrt(x)`	Square root of x
`math.pow(x, y)`	x raised to power y
`math.floor(x)`	Largest integer $\leq$ x
`math.ceil(x)`	Smallest integer $\geq$ x
`math.factorial(x)`	Factorial of x
`math.log(x, base)`	Logarithm of x to base
`math.sin(x)`	Sine of x (radians)
`math.cos(x)`	Cosine of x (radians)
`math.pi`	Value of $\pi \approx 3.14159$
`math.e`	Value of $e \approx 2.71828$

Random Module (W'24 Q.4a OR, W'25 Q.5b)

Common random module functions

```python
import random

# random float in [0.0, 1.0)
print(random.random())

# random integer: randint(a, b) inclusive
print(random.randint(1, 6))   # dice roll

# random float in [a, b]
print(random.uniform(1.0, 10.0))

# choose random item from list
colors = ["red", "green", "blue"]
print(random.choice(colors))

# shuffle list in place
nums = [1, 2, 3, 4, 5]
random.shuffle(nums)
print(nums)
```

Statistics Module (Su'25 Q.3c OR, W'25 Q.5b OR)

Statistics module functions

```
import statistics as st

data = [4, 8, 15, 16, 23, 42]

print(st.mean(data))       # 18.0 (average)
print(st.median(data))     # 15.5 (middle value)
print(st.mode([1,2,2,3]))# 2 (most frequent)
print(st.stdev(data))      # standard deviation
print(st.variance(data)) # variance
```

Function	Purpose
`mean()`	Arithmetic mean
`median()`	Middle value
`mode()`	Most frequent value
`stdev()`	Standard deviation

Module in Python (W'23 Q.4b, Su'23 Q.5b)

What is a Module?

A **module** is a file containing Python code (functions, variables, classes) that can be imported into other programs.

```python
# Import entire module
import math
print(math.sqrt(16))    # 4.0

# Import specific function
from math import sqrt, pi
print(sqrt(25))  # 5.0
print(pi)        # 3.14159...

# Import with alias
import statistics as st
data = [1, 2, 3, 4, 5]
print(st.mean(data))    # 3.0
```

PYQ Practice — 3-Mark Questions

3-Mark PYQ Bank

1. Explain local variable with example *(Su'25)*
2. Explain global variable with example *(Su'25 OR)*
3. List Python standard library mathematical functions *(W'23, Su'24 OR)*
4. Describe Python math module with Python code example *(Su'24)*
5. Describe built-in functions `max()`, `input()`, `pow()` with example *(W'24)*
6. Explain random module with various functions *(W'24 OR)*
7. Explain built-in functions in Python *(W'23 OR, Su'24 OR)*

PYQ Practice — 4-Mark Questions

4-Mark PYQ Bank

1. Define function. Explain user-defined function with suitable example *(Su'23)*
2. Explain how to define and call a user-defined function *(W'24)*
3. Explain return statement in function handling *(W'24 OR)*
4. User-defined function to find factorial of a number *(Su'25, W'25 OR)*
5. User-defined function to check if number is prime *(Su'25 OR)*
6. User-defined function to print Fibonacci series 0 to N *(W'25)*
7. Explain local and global variables with suitable examples *(Su'23 OR)*
8. Write Python program that explains scope of a variable *(Su'24)*
9. Explain math module and random module with example *(Su'23)*
10. Explain Module in Python with Python code example *(W'23)*
11. Explain Random module in Python *(W'25)*
12. Explain statistics module in Python *(W'25 OR)*

PYQ Practice — 7-Mark Questions

7-Mark PYQ Bank

1. Create user-defined function to print Fibonacci series 0 to N *(Su'23, W'24 OR)*
2. Determine if number is Armstrong or palindrome using function *(Su'23 OR, W'23, Su'24)*
3. Create user-defined function to find factorial *(W'23, W'24)*
4. Explain local and global variables; concept of scope *(W'23 OR ×2, W'25)*
5. Write program using function that reverses the entered value *(Su'24 OR)*
6. Create function that prints cube of all odd numbers 1–7 *(W'24)*
7. List math module functions; develop program demonstrating them *(Su'25)*
8. List statistics module functions; develop program demonstrating them *(Su'25 OR)*
9. List Python standard library mathematical functions; explain any three *(W'25 OR)*

Worked Solution: Factorial (W'23 Q.3c, W'24 Q.3c)

Complete factorial program

```
def factorial(n):
    """Returns factorial of n"""
    if n == 0 or n == 1:
        return 1
    fact = 1
    for i in range(2, n + 1):
        fact *= i
    return fact

# Main program
num = int(input("Enter a positive integer: "))
if num < 0:
    print("Factorial not defined for negative")
else:
    print(f"Factorial of {num} = {factorial(num)}")
```

Worked Solution: Math Module Program (Su'25 Q.3c)

Program demonstrating math module functions

```
import math

n = float(input("Enter a number: "))

print("Square root:", math.sqrt(n))
print("Floor:", math.floor(n))
print("Ceil:", math.ceil(n))
print("log base 10:", math.log10(n))
print("Pi:", math.pi)
print("Power (n^2):", math.pow(n, 2))

# Hypotenuse of right triangle
a = 3; b = 4
print("Hypotenuse:", math.hypot(a, b))   # 5.0
```

Lambda / Anonymous Functions (W'25, Su'25 OR — 4M)

Syntax

`lambda` *arguments* : *expression*
Lambda creates a small, unnamed (anonymous) function in a single line.

```
# Regular function
def add(a, b):
    return a + b

# Equivalent lambda
add = lambda a, b: a + b
print(add(3, 5))      # 8

square = lambda x: x ** 2
print(square(4))      # 16

is_even = lambda n: n % 2 == 0
print(is_even(7))     # False
print(is_even(8))     # True
```

Lambda with map() & filter() (W'25 — 4M)

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

# map(): apply function to all elements
squares = list(map(lambda x: x**2, nums))
print(squares)   # [1, 4, 9, 16, 25, 36]

doubled = list(map(lambda x: x*2, nums))
print(doubled)   # [2, 4, 6, 8, 10, 12]

# filter(): keep elements that match
evens = list(filter(lambda x: x%2==0, nums))
print(evens)     # [2, 4, 6]

positives = list(filter(lambda x: x>3, nums))
print(positives) # [4, 5, 6]
```

*args — Variable Positional Arguments (W'25 OR — 4M)

Concept

*args allows a function to accept **any number** of positional arguments. Collected as a **tuple** inside the function.

```
def total(*nums):
    s = 0
    for n in nums:
        s += n
    return s

print(total(1, 2, 3))           # 6
print(total(10, 20, 30, 40))    # 100
print(total(5))                 # 5

def display(*items):
    for item in items:
        print(item, end=" ")
    print()
```

**kwargs — Variable Keyword Arguments (W'25 OR — 4M)

Concept

kwargs accepts any number of **keyword arguments. Collected as a **dictionary** inside the function.

```
def student_info(**details):
    for key, value in details.items():
        print(f"{key}: {value}")

student_info(name="Raj", age=20, gpa=8.5)
# name: Raj
# age: 20
# gpa: 8.5

# Combining *args and **kwargs
def mixed(*args, **kwargs):
    print("Args:", args)
    print("Kwargs:", kwargs)

mixed(1, 2, 3, x=10, y=20)
```

Recursion — Concept (W'24, Su'25 — 4M)

What is Recursion?

A function that **calls itself** is called a recursive function. Every recursive function must have a **base case** to stop recursion.

Structure

```
def recursive_fn(n):
if base_case:
return value
return expression with recursive_fn(n-1)
```

Key Terms

Base Case: Condition to stop recursion (prevents infinite loop)
Recursive Case: The function calls itself with simpler input
Call Stack: Each call is pushed; returns pop in reverse order

Recursive Function: Factorial (W'24, Su'25 OR — 7M)

```python
def factorial(n):
    # Base case
    if n == 0 or n == 1:
        return 1
    # Recursive case
    return n * factorial(n - 1)

print(factorial(5))    # 120
print(factorial(0))    # 1
print(factorial(6))    # 720
```

Call Trace: factorial(4)

`factorial(4)` → `4 * factorial(3)`
`factorial(3)` → `3 * factorial(2)`
`factorial(2)` → `2 * factorial(1)`
`factorial(1)` → 1 (base case)
Back: $2 \times 1 = 2 \rightarrow 3 \times 2 = 6 \rightarrow 4 \times 6 = \mathbf{24}$

Recursive Function: Fibonacci (Su'24, W'25 OR — 7M)

```python
def fibonacci(n):
    if n == 0:
        return 0      # base case 1
    if n == 1:
        return 1      # base case 2
    return fibonacci(n-1) + fibonacci(n-2)

# Print first 10 terms
for i in range(10):
    print(fibonacci(i), end=" ")
# 0 1 1 2 3 5 8 13 21 34
```

Note

Recursive Fibonacci is elegant but **slow** for large n (repeated calls). Iterative version is more efficient for large inputs.

Recursion vs Iteration (W'24 OR, Su'25 — 4M)

Comparison Table

Feature	Recursion	Iteration
Approach	Function calls itself	Loops (`for`/`while`)
Base case	Mandatory (no infinite calls)	Loop condition
Memory	Uses call stack (more)	Less memory
Speed	Usually slower	Usually faster
Code size	Often shorter	Can be longer
Debugging	Harder to trace	Easier
Use case	Tree, factorial, Fibonacci	Counting, sum, search

Recursive: Sum of Digits & Power (Su'25 OR — 4M)

```python
# Recursive sum of digits
def sum_digits(n):
    if n == 0:
        return 0
    return (n % 10) + sum_digits(n // 10)

print(sum_digits(1234))    # 10
print(sum_digits(999))     # 27

# Recursive power: base^exp
def power(base, exp):
    if exp == 0:
        return 1
    return base * power(base, exp - 1)

print(power(2, 8))     # 256
print(power(3, 4))     # 81
```

String Module (W'24 OR, Su'25 — 3M)

```python
import string

print(string.ascii_letters)
# abcdefghijklmnopqrstuvwxyzABCDEFGHIJKLMNOPQRSTUVWXYZ

print(string.ascii_lowercase)  # a-z
print(string.ascii_uppercase)  # A-Z
print(string.digits)           # 0123456789
print(string.punctuation)      # !"#$%&'()*+,-./:;<=>?

# Practical use: password validation
def is_strong(pwd):
    return (any(c in string.ascii_uppercase for c in pwd)
        and any(c in string.digits for c in pwd)
        and len(pwd) >= 8)

print(is_strong("Hello123"))   # True
print(is_strong("hello"))      # False
```

os Module — File System Basics (W'24 — 3M)

```python
import os

# Current working directory
print(os.getcwd())          # /home/user/python

# Change directory
os.chdir("/tmp")

# List files and folders
print(os.listdir("."))      # ['file1.py', 'data/']

# Create and remove directory
os.mkdir("new_folder")
os.rmdir("new_folder")

# Check if path exists
print(os.path.exists("data.txt"))  # True/False

# Get file size
print(os.path.getsize("data.txt")) # bytes

# Join paths safely
path = os.path.join("folder", "file.txt")
```

datetime Module (Su'25 OR — 3M)

```python
from datetime import datetime, date, timedelta

# Current date and time
now = datetime.now()
print(now)
# 2025-06-15 10:30:45.123456

print(now.year, now.month, now.day)
print(now.hour, now.minute, now.second)

# Today's date
today = date.today()
print(today)                    # 2025-06-15

# Date arithmetic
tomorrow = today + timedelta(days=1)
deadline = today + timedelta(weeks=2)
print(tomorrow, deadline)

# Format date as string
print(now.strftime("%d/%m/%Y"))   # 15/06/2025
```

Function: GCD & LCM (Su'24, W'25 — 4M)

```python
# GCD using Euclidean algorithm
def gcd(a, b):
    while b != 0:
        a, b = b, a % b
    return a

# LCM using GCD
def lcm(a, b):
    return (a * b) // gcd(a, b)

x = int(input("First number: "))
y = int(input("Second number: "))
print(f"GCD of {x} and {y} = {gcd(x, y)}")
print(f"LCM of {x} and {y} = {lcm(x, y)}")
# For x=12, y=8:
# GCD = 4,  LCM = 24
```

Function: Perfect Number & Abundant (W'25, Su'25 — 4M)

```python
def is_perfect(n):
    s = sum(i for i in range(1, n) if n % i == 0)
    return s == n

# Check perfect numbers up to 1000
print("Perfect numbers up to 1000:")
for num in range(1, 1001):
    if is_perfect(num):
        print(num, end=" ")
# 6 28 496

# Explanation:
# 6 = 1+2+3, 28 = 1+2+4+7+14
def divisor_sum(n):
    return sum(i for i in range(1, n) if n%i==0)

print(divisor_sum(6))     # 6  (perfect)
print(divisor_sum(12))    # 16 (abundant: sum > n)
```

Function: Digit Operations (W'25 OR — 4M)

```
def digit_sum(n):
    """Sum of all digits of n"""
    total = 0
    while n > 0:
        total += n % 10
        n //= 10
    return total

def digit_count(n):
    """Count digits in n"""
    count = 0
    while n != 0:
        n //= 10
        count += 1
    return count

def digit_product(n):
    """Product of digits"""
    prod = 1
    while n > 0:
        prod *= n % 10
        n //= 10
    return prod
```

Function: Leap Year & Temperature Conversion (Su'25 — 4M)

```python
def is_leap_year(year):
    return (year%4==0 and year%100!=0) or (year%400==0)

def celsius_to_fahrenheit(c):
    return (c * 9/5) + 32

def fahrenheit_to_celsius(f):
    return (f - 32) * 5/9

def celsius_to_kelvin(c):
    return c + 273.15

y = int(input("Year: "))
if is_leap_year(y):
    print(f"{y} is a Leap Year")
else:
    print(f"{y} is NOT a Leap Year")

c = float(input("Celsius: "))
print(f"Fahrenheit: {celsius_to_fahrenheit(c):.2f}")
print(f"Kelvin:     {celsius_to_kelvin(c):.2f}")
```

Creating & Using Custom Modules (W'23, Su'23 — 4M)

```python
# File: mymath.py (our custom module)
def add(a, b):    return a + b
def sub(a, b):    return a - b
def mul(a, b):    return a * b
def div(a, b):    return a / b if b != 0 else "Error"
PI = 3.14159

# In another file: main.py
import mymath
print(mymath.add(5, 3))          # 8
print(mymath.PI)                  # 3.14159

# Selective import
from mymath import mul, div
print(mul(4, 7))                  # 28

# Import with alias
import mymath as mm
print(mm.sub(10, 4))              # 6
```

Worked: Lambda, map, filter Program (W'25 — 4M)

```
# Problem: Given a list of marks, find
# (a) squares of all marks
# (b) marks above 40 (pass)
# (c) double the failing marks

marks = [35, 72, 48, 19, 85, 40, 61]

squares = list(map(lambda x: x**2, marks))
print("Squares:", squares)

passed = list(filter(lambda x: x >= 40, marks))
print("Passed:", passed)

failed = list(filter(lambda x: x < 40, marks))
doubled = list(map(lambda x: x*2, failed))
print("Failed doubled:", doubled)

# Sorted using lambda key
marks.sort(key=lambda x: -x)  # descending
print("Sorted:", marks)
```

Worked: Recursive GCD (Su'24, W'25 — 4M)

```
def gcd(a, b):
    if b == 0:          # base case
        return a
    return gcd(b, a % b)    # recursive case

def lcm(a, b):
    return (a * b) // gcd(a, b)

# Test
print(gcd(48, 18))      # 6
print(gcd(100, 75))     # 25
print(lcm(4, 6))        # 12
print(lcm(12, 18))      # 36
```

Call Trace: gcd(48, 18)

`gcd(48,18)` → `gcd(18,12)` → `gcd(12,6)` → `gcd(6,0)` → **6**

Worked: Armstrong Number using Function (Su'23, W'23 — 7M)

```python
def is_armstrong(n):
    digits = str(n)
    power = len(digits)
    total = sum(int(d)**power for d in digits)
    return total == n

# Print all Armstrong numbers up to 999
print("Armstrong numbers up to 999:")
for i in range(1, 1000):
    if is_armstrong(i):
        print(i, end=" ")
# 1 2 3 4 5 6 7 8 9 153 370 371 407

# Single check
n = int(input("Enter number: "))
if is_armstrong(n):
    print(f"{n} is an Armstrong number")
else:
    print(f"{n} is NOT an Armstrong number")
```

Worked: Default & Keyword Arguments (W'24, Su'25 — 4M)

```python
# Default arguments
def greet(name, msg="Hello"):
    print(f"{msg}, {name}!")

greet("Raj")                # Hello, Raj!
greet("Priya", "Hi")        # Hi, Priya!
greet("Sam", msg="Hey")     # Hey, Sam!

# Keyword arguments (named)
def rectangle(length, width, unit="cm"):
    area = length * width
    perimeter = 2 * (length + width)
    print(f"Area={area} {unit}^2")
    print(f"Perimeter={perimeter} {unit}")

rectangle(5, 3)
rectangle(width=4, length=7, unit="m")
```

Worked: Statistics Module Program (Su'25 OR, W'25 — 4M)

```python
import statistics

marks = [85, 72, 91, 68, 78, 85, 90, 65, 72, 80]

print(f"Mean:   {statistics.mean(marks):.2f}")
print(f"Median: {statistics.median(marks):.2f}")
print(f"Mode:   {statistics.mode(marks)}")
print(f"Stdev:  {statistics.stdev(marks):.2f}")
print(f"Variance: {statistics.variance(marks):.2f}")

# Manual mean (without library)
n = len(marks)
manual_mean = sum(marks) / n
print(f"Manual Mean: {manual_mean:.2f}")

# Count above average
above = [m for m in marks if m > manual_mean]
print(f"Above average: {len(above)} students")
```

Worked: Math Module Programs (W'23, Su'24, W'25 — 4M)

```python
import math

# Quadratic equation roots: ax^2 + bx + c = 0
a, b, c = map(float, input("a b c: ").split())
discriminant = b**2 - 4*a*c
if discriminant > 0:
    r1 = (-b + math.sqrt(discriminant)) / (2*a)
    r2 = (-b - math.sqrt(discriminant)) / (2*a)
    print(f"Roots: {r1:.2f}, {r2:.2f}")
elif discriminant == 0:
    r = -b / (2*a)
    print(f"Equal roots: {r:.2f}")
else:
    print("Complex roots (no real solution)")

# Useful math functions
print(math.gcd(48, 36))       # 12
print(math.factorial(6))      # 720
print(math.comb(5, 2))        # 10   (5C2)
```

Worked: Random Module — Guessing Game (W'24 OR, W'25 — 4M)

```python
import random

# Number guessing game
secret = random.randint(1, 100)
attempts = 0
print("Guess the number (1-100)!")

while True:
    guess = int(input("Your guess: "))
    attempts += 1
    if guess < secret:
        print("Too low!")
    elif guess > secret:
        print("Too high!")
    else:
        print(f"Correct! ({attempts} attempts)")
        break

# Other random uses
colors = ["red", "blue", "green", "yellow"]
print(random.choice(colors))     # random color
nums = list(range(1, 11))
```

Worked: Scope — LEGB Rule (W'25, Su'25 — 4M)

LEGB: Scope Resolution Order

Local → **E**nclosing → **G**lobal → **B**uilt-in

```
count = 0           # global variable

def increment():
    global count    # access global
    count += 1

def outer():
    x = 10          # enclosing scope
    def inner():
        nonlocal x   # modify enclosing
        x += 5
        print("inner x:", x)
    inner()
    print("outer x:", x)

increment(); increment()
print("count:", count)     # 2
outer()     # inner x: 15, outer x: 15
```

Quick Reference: Python Standard Modules

Module	Import	Key Functions
math	`import math`	`sqrt, pow, floor, ceil, pi, factorial, gcd, comb`
random	`import random`	`random(), randint(), choice(), shuffle(), seed()`
statistics	`import statistics`	`mean, median, mode, stdev, variance`
string	`import string`	`ascii_letters, digits, punctuation`
os	`import os`	`getcwd, listdir, mkdir, path.exists`
datetime	`from datetime import datetime, date`	`now(), today(), strftime()`

Import Styles

`import math` | `from math import sqrt, pi` | `import math as m`

Quick Reference: Function Types in Python

Type	Description & Example
Built-in	Pre-defined: `len()`, `print()`, `input()`, `type()`, `range()`
User-defined	Defined with `def`: `def greet(): ...`
Lambda	Anonymous: `square = lambda x: x**2`
Recursive	Calls itself: `def fact(n): return n*fact(n-1)`
Higher-order	Takes/returns functions: `map()`, `filter()`
Module functions	From libraries: `math.sqrt()`, `random.randint()`

Argument Types

Positional: `add(3, 5)` **Keyword:** `greet(name="Raj")` **Default:** `def f(x=10)`
***args:** tuple **kwargs:** dict

Common Mistakes in Functions (Exam Guide)

Mistakes

× No base case in recursion
× Modifying global without `global`
× Using variable before definition
× Returning inside loop prematurely
× Wrong number of arguments
× Calling function before defining

Correct Patterns

✓ Always have `if n==0: return 1`
✓ Use `global x` to modify global
✓ Define before calling
✓ Use `return` at end of function
✓ Match parameter count
✓ Put function defs at top of file

Unit 4 — Rapid Revision Flash Sheet

Concept	Key Point
Function definition	`def name(params): body`
Return value	`return expr` (or `None` if absent)
Default argument	`def f(x=10):` — must come last
Lambda	`lambda x: x**2` — single expression
*args	Variable positional → tuple inside
**kwargs	Variable keyword → dict inside
Recursion	Function calls itself; needs base case
Local scope	Variable inside function only
Global keyword	`global x` to modify global inside fn
math module	`sqrt, pow, floor, ceil, pi, factorial`
random module	`randint, choice, shuffle, random()`
statistics module	`mean, median, mode, stdev`

GTU Weightage (Unit 4 — 16%)

3-mark: definitions, modules 4-mark: programs 7-mark: full programs

Worked: Recursive Palindrome Check (W'23 OR, Su'24 — 4M)

```python
def is_palindrome(s):
    """Check if string is palindrome recursively"""
    s = s.lower()
    if len(s) <= 1:
        return True          # base case
    if s[0] != s[-1]:
        return False         # mismatch
    return is_palindrome(s[1:-1])  # recurse on middle

# Test
words = ["madam", "racecar", "hello", "level"]
for w in words:
    result = "Palindrome" if is_palindrome(w) else "Not"
    print(f"{w}: {result}")
# madam: Palindrome
# racecar: Palindrome
# hello: Not
# level: Palindrome
```

Worked: Prime Numbers — All Methods (Su'25, W'24 OR — 7M)

```python
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

# Check single number
n = int(input("Number: "))
print(f"{n} is {'Prime' if is_prime(n) else 'Not Prime'}")

# Print all primes up to N
n = int(input("Up to: "))
primes = [i for i in range(2, n+1) if is_prime(i)]
print("Primes:", primes)
print("Count:", len(primes))

# Twin primes (differ by 2)
twins = [(i,i+2) for i in range(2,n) if is_prime(i) and is_prime
    (i+2)]
print("Twin primes:", twins[:5])
```

Worked: Number Theory Programs (Su'25 OR — 7M)

```
def is_perfect(n):
    return n > 1 and sum(i for i in range(1,n) if n%i==0) == n

def is_abundant(n):
    return sum(i for i in range(1,n) if n%i==0) > n

def is_armstrong(n):
    d = str(n); p = len(d)
    return sum(int(c)**p for c in d) == n

def is_palindrome_num(n):
    return str(n) == str(n)[::-1]

# Display analysis for 1-500
for n in range(1, 31):
    tags = []
    if is_prime(n):     tags.append("Prime")
    if is_perfect(n):   tags.append("Perfect")
    if is_armstrong(n):tags.append("Armstrong")
    if tags:
        print(f"{n:3d}: {', '.join(tags)}")
```

Worked: Student Report Card Program (Su'25 — 7M)

```python
def calculate_grade(marks):
    avg = sum(marks) / len(marks)
    if avg >= 90: return "A+"
    elif avg >= 75: return "A"
    elif avg >= 60: return "B"
    elif avg >= 40: return "C"
    else: return "F (Fail)"

def display_report(name, marks):
    print(f"\n--- Report Card: {name} ---")
    subjects = ["Maths","Science","English","Python","GTU"]
    for sub, m in zip(subjects, marks):
        print(f"{sub:10s}: {m:3d}", end="")
        print(" PASS" if m >= 40 else " FAIL")
    avg = sum(marks)/len(marks)
    print(f"Average: {avg:.1f} | Grade: {calculate_grade(marks)}")

name = input("Student name: ")
marks = [int(input(f"Subject {i+1}: ")) for i in range(5)]
display_report(name, marks)
```

Worked: PYQ 7M — Fibonacci (Recursive + Iterative) (W'25 — 7M)

```python
# Recursive Fibonacci
def fib_recursive(n):
    if n <= 1:
        return n
    return fib_recursive(n-1) + fib_recursive(n-2)

# Iterative Fibonacci
def fib_iterative(n):
    a, b = 0, 1
    for _ in range(n):
        a, b = b, a + b
    return a

n = int(input("Terms: "))
print("Recursive:", [fib_recursive(i) for i in range(n)])
print("Iterative:", [fib_iterative(i) for i in range(n)])
# Both give: [0, 1, 1, 2, 3, 5, 8, 13, ...]
```

Examiner Tip

For 7M: Show both recursive + iterative versions with output. Mention base case explicitly.

Unit 4 — Summary

User-defined

def keyword
Parameters / Arguments
return statement
Default arguments

Scope

Local — inside function
Global — outside all functions
global keyword to modify

Modules

math — sqrt, pow, floor, ceil, pi
random — random, randint, choice
statistics — mean, median, mode

Key Programs

Factorial, Fibonacci
Prime, Armstrong
Palindrome, Reverse
Cube of odds

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)

```
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)

```python
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)

```
# 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)

```python
# 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)

```
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