Python Programming

Unit-wise PYQ – Step-by-Step Solutions

Codes: 1323203 / DI02032011 (Diploma Engineering – Sem 2)

Papers covered: Summer 2023 · Winter 2023 · Summer 2024 · Winter 2024 · Summer 2025 · Winter 2025

136 unique questions across 5 units

Contents

Unit 1: Problem Solving using Flowchart and Algorithm

Unit 1 | Problem Solving using Flowchart and Algorithm

1.1 Flowchart & Algorithm — 3-mark questions

1. Define flowchart and list any four symbols used in flowchart with their purpose. **[3 M]**

- *Appeared in:* W'24 – Q.1(a); Su'25 – Q.1(a)

Solution

Flowchart: A flowchart is a diagrammatic (graphical) representation of an algorithm using standardized symbols connected by arrows, showing the step-by-step logic to solve a problem.
Four symbols:

Symbol	Name	Purpose
Oval	Terminator	Represents the Start or End of flowchart
Rectangle	Process	Represents a computation or processing step
Diamond	Decision	Represents a conditional check (Yes/No branch)
Parallelogram	Input/Output	Represents reading input or displaying output

2. Define algorithm. What are the advantages of Algorithm? **[3 M]**

- *Appeared in:* Su'23 – Q.1(a)

Solution

Algorithm: An algorithm is a finite, ordered set of well-defined, unambiguous instructions that describes how to solve a given problem step by step.
Advantages:

- Easy to understand — written in simple English-like steps
- Language independent — can be converted to any programming language
- Easy to debug — errors can be traced step by step
- Serves as documentation and a blueprint before writing actual code
- Promotes logical and systematic problem solving

3. List the importance of flowchart and algorithm. **[3 M]**

- *Appeared in:* Su'24 – Q.1(a)

Solution

Importance of Flowchart and Algorithm:

- Provides a clear, visual step-by-step understanding of a problem
- Acts as a blueprint before writing actual code
- Helps detect and fix logical errors before coding begins
- Language independent — applicable to any programming language
- Facilitates team communication and project documentation
- Simplifies complex problems into small, manageable steps
- Reduces development time and effort

4. Define Algorithm. Write algorithm to find area of a circle. **[3 M]**

- *Appeared in:* W'25 – Q.1(a)

Solution

Algorithm: A finite set of step-by-step, unambiguous instructions used to solve a specific problem.

Algorithm to find area of a circle ($A = \pi r^2$):

Step 1: Start

Step 2: Read radius r

Step 3: Calculate area $= 3.14159 \times r \times r$

Step 4: Print area

Step 5: Stop

5. Write the difference between flowchart and algorithm. [3 M]

- *Appeared in:* Su'23 – Q.2(a) [OR]

Solution

Aspect	Algorithm	Flowchart
Definition	Textual step-by-step instructions	Graphical/diagrammatic representation
Form	Written in English-like numbered steps	Uses standard symbols connected by arrows
Visualization	Hard to visualize directly	Easy to visualize the flow
Modification	Easy to modify (just edit text)	Harder to modify once drawn
Tools	No special tools required	Requires drawing tools

1.2 Flowchart & Algorithm — 4-mark questions

6. State rules for problem solving using flowchart. Design flowchart to find simple interest. [4 M]

- *Appeared in:* Su'23 – Q.1(b); W'25 – Q.1(b)

Solution

Rules for Flowchart:

- Use standard symbols only (oval, rectangle, diamond, parallelogram)
- Every flowchart must have exactly one START and one STOP
- Flow direction is top-to-bottom and left-to-right
- Decision box must have exactly two branches: Yes and No
- All symbols must be connected by arrows showing direction of flow
- Flowlines must not cross; use connectors if needed

Flowchart for Simple Interest (SI = PRT/100):

1. **[START]** — Oval
2. **Input P, R, T** — Parallelogram
3. **SI = (P × R × T) / 100** — Rectangle (Process)
4. **Print SI** — Parallelogram
5. **[STOP]** — Oval

7. Define Algorithm. Design algorithm for finding maximum of three numbers. [4 M]

- *Appeared in:* W'23 – Q.1(b)

Solution

Algorithm: A finite, ordered sequence of unambiguous instructions used to solve a given problem.

Algorithm for maximum of three numbers A, B, C:

Step 1: Start

Step 2: Input A, B, C

Step 3: If A > B go to Step 4, else go to Step 5

Step 4: If A > C then Max = A, else Max = C; go to Step 6

Step 5: If B > C then Max = B, else Max = C

Step 6: Print Max

Step 7: Stop

8. Draw flowchart to find whether the entered number is even or odd. [4 M]

- *Appeared in:* Su'24 – Q.1(b)

Solution

Flowchart for Even/Odd check:

1. **[START]** — Oval
2. **Input N** — Parallelogram
3. **Is N % 2 == 0?** — Diamond (Decision)
   Yes → Step 4 No → Step 5
4. **Print "Even"** — Parallelogram → go to Step 6
5. **Print "Odd"** — Parallelogram
6. **[STOP]** — Oval

Equivalent Algorithm:
Step 1: Start Step 2: Input N Step 3: If N%2==0 print "Even" else print "Odd" Step 4: Stop

9. Prepare algorithms for the following: (1) To find maximum of two given numbers (2) To find sum of two given numbers. [4 M]

- *Appeared in:* Su'25 – Q.1(b)

Solution

Algorithm 1 – Maximum of two numbers:

Step 1: Start

Step 2: Input A, B

Step 3: If A > B then Max = A, else Max = B

Step 4: Print Max

Step 5: Stop

Algorithm 2 – Sum of two numbers:

Step 1: Start

Step 2: Input A, B

Step 3: Sum = A + B

Step 4: Print Sum

Step 5: Stop

1.3 Flowchart & Algorithm — 7-mark questions

10. Design flowchart to calculate sum of first twenty even natural numbers. [7 M]

- *Appeared in:* W'24 – Q.1(c)

Solution

Logic: Even natural numbers: 2, 4, 6, . . . , 40. Use a counter loop.
Flowchart:

1. **[START]** — Oval
2. **Set Sum = 0, Count = 0, Num = 2** — Rectangle
3. **Is Count < 20?** — Diamond
   Yes → Step 4 No → Step 7
4. **Sum = Sum + Num** — Rectangle
5. **Num = Num + 2; Count = Count + 1** — Rectangle
6. Go back to Step 3 (loop)
7. **Print Sum** — Parallelogram
8. **[STOP]** — Oval

Algorithm:

Step 1: Start
Step 2: Set Sum = 0, Count = 0, Num = 2
Step 3: If Count ≥ 20 go to Step 7
Step 4: Sum = Sum + Num
Step 5: Num = Num + 2; Count = Count + 1
Step 6: Go to Step 3
Step 7: Print Sum (Result: Sum = 420)
Step 8: Stop

11. Create algorithm to print odd numbers between 1 to 20. **[7 M]**

- *Appeared in:* W'24 – Q.1(c) [OR]

Solution

Algorithm to print odd numbers from 1 to 20:

Step 1: Start
Step 2: Set i = 1
Step 3: If i > 20 go to Step 7
Step 4: If i % 2 ≠ 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

Flowchart:

1. **[START]**
2. **Set i = 1**
3. **Is i ≤ 20?** — Yes → Step 4; No → Step 7
4. **Is i % 2 ≠ 0?** — Yes → Print i; No → Step 5
5. **i = i + 1** — go back to Step 3
6. (from print) go to Step 5

7. [STOP]

1.4 Pseudocode — 3-mark questions

12. Write pseudocode to check if the entered number is positive or negative. **[3 M]**

- *Appeared in:* W'23 – Q.1(a)

Solution

Pseudocode:

```
BEGIN
    INPUT num
    IF num > 0 THEN
        PRINT "Positive"
    ELSE IF num < 0 THEN
        PRINT "Negative"
    ELSE
        PRINT "Zero"
    END IF
END
```

13. Define pseudocode. Write pseudocode to find the smallest of two numbers. **[3 M]**

- *Appeared in:* Su'23 – Q.2(a)

Solution

Pseudocode: An informal, high-level description of a program or algorithm using plain English keywords and structured notation — easier to write than actual code and language independent.

Pseudocode for smallest of two numbers:

```
BEGIN
    INPUT a, b
    IF a < b THEN
        PRINT "Smallest is", a
    ELSE
        PRINT "Smallest is", b
    END IF
END
```

1.5 Pseudocode — 4-mark questions

14. Define pseudocode. Write pseudocode to find the largest of three numbers x, y, z. **[4 M]**

- *Appeared in:* Su'24 – Q.2(b)

Solution

Pseudocode: An informal, structured description of an algorithm using English-like keywords. It bridges the gap between the algorithm and actual program code. It is language independent and easy to understand.

Pseudocode for largest of three numbers x, y, z:

```
BEGIN
    INPUT x, y, z
    IF x > y AND x > z THEN
        largest = x
    ELSE IF y > z THEN
        largest = y
    ELSE
        largest = z
    END IF
```

```
    PRINT "Largest is", largest
END
```

Unit 2: Introduction to Python

Unit 2 | Introduction to Python

2.1 Python Features & Applications — 3-mark questions

1. List the features of Python. **[3 M]**

- *Appeared in:* Su'25 – Q.2(a); W'25 – Q.2(a)

Solution

Features of Python:

- **Simple and Easy to Learn** — clean, readable syntax similar to English
- **Interpreted** — code is executed line by line; no compilation needed
- **Dynamically Typed** — variable types are determined at runtime
- **Object-Oriented** — supports classes, objects, inheritance
- **Free and Open Source** — freely available and open source
- **Extensive Standard Library** — rich set of built-in modules
- **Platform Independent** — runs on Windows, Linux, macOS
- **High-Level Language** — programmer-friendly, hides machine-level details

2. List the applications of Python. **[3 M]**

- *Appeared in:* Su'25 – Q.2(a) [OR]; W'25 – Q.2(a) [OR]

Solution

Applications of Python:

- **Web Development** — frameworks like Django, Flask
- **Data Science & Machine Learning** — libraries like NumPy, Pandas, TensorFlow
- **Artificial Intelligence** — natural language processing, computer vision
- **Automation and Scripting** — system automation, task scheduling
- **Game Development** — libraries like Pygame
- **Scientific Computing** — simulations, data analysis
- **Desktop GUI Applications** — Tkinter, PyQt

2.2 Data Types, Variables & Operators — 3-mark questions

3. Describe data types in Python with examples. **[3 M]**

- *Appeared in:* W'23 – Q.2(a)

Solution

Data Types in Python:

Data Type	Description	Example
`int`	Integer (whole number)	`x = 10`
`float`	Decimal number	`y = 3.14`
`str`	String (text)	`s = "Hello"`
`bool`	Boolean (True/False)	`b = True`
`list`	Ordered, mutable collection	`lst = [1, 2, 3]`
`tuple`	Ordered, immutable collection	`t = (1, 2, 3)`
`dict`	Key-value pairs	`d = {"a": 1}`
`set`	Unordered, unique elements	`st = {1, 2, 3}`

4. List out rules for defining variables in Python. **[3 M]**

- *Appeared in:* W'23 – Q.2(a) [OR]

Solution

Rules for Defining Variables in Python:

- Variable names must start with a letter (a–z, A–Z) or underscore (_)
- Can contain letters, digits (0–9), and underscores; **no spaces or special characters**
- **Case sensitive** — `Name` and `name` are different variables
- Cannot use Python reserved keywords (e.g., `if`, `for`, `while`, `class`)
- No length limit (but keep names meaningful and concise)
- Assignment is done using the `=` operator: `x = 5`

5. Discuss membership operator of Python with example. **[3 M]**

- *Appeared in:* W'24 – Q.2(a)

Solution

Membership Operators: Used to test whether a value is a member of a sequence (list, tuple, string, set).

Operator	Description	Example
`in`	Returns `True` if value found in sequence	`3 in [1,2,3]` → `True`
`not in`	Returns `True` if value NOT found in sequence	`5 not in [1,2,3]` → `True`

Examples: `"a" in "apple"` → `True` `"z" not in "hello"` → `True`

6. Write a short note on assignment operator in Python. **[3 M]**

- *Appeared in:* W'24 – Q.2(a) [OR]

Solution

Assignment Operators: Used to assign values to variables.

Operator	Example	Equivalent	Meaning
`=`	`x = 5`	—	Simple assignment
`+=`	`x += 3`	`x = x + 3`	Add and assign
`-=`	`x -= 2`	`x = x - 2`	Subtract and assign
`*=`	`x *= 4`	`x = x * 4`	Multiply and assign
`/=`	`x /= 2`	`x = x / 2`	Divide and assign
`%=`	`x %= 3`	`x = x % 3`	Modulo and assign
`**=`	`x **= 2`	`x = x ** 2`	Power and assign
`//=`	`x //= 3`	`x = x // 3`	Floor div and assign

2.3 Data Types & Operators — 4-mark questions

7. List various data types in Python and explain any three with example. **[4 M]**

- *Appeared in:* W'24 – Q.1(b)

Solution

Data Types in Python: int, float, str, bool, list, tuple, dict, set, NoneType.

Explanation of three data types:

1. `int` (Integer): Stores whole numbers (positive, negative, or zero).

`age = 20; marks = -5; print(type(age))` → `<class 'int'>`

2. `float`: Stores decimal (floating-point) numbers.

`price = 99.99; pi = 3.14159; print(type(price))` → `<class 'float'>`

3. `str` (String): Stores a sequence of characters enclosed in quotes.

`name = "Alice"; print(name[0])` → `A`

8. Develop Python code to read three numbers and find their average. **[4 M]**

- *Appeared in:* Su'23 – Q.2(b)

Solution

Python Program:

```
# Read three numbers and find average
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
```

9. Determine the output of the following Python code snippet involving arithmetic and comparison operators (e.g. x*y, xy, x//y, x%y, x>y).** [4 M]

- *Appeared in:* Su'23 – Q.2(b) [OR]; Su'24 – Q.2(b) [OR]

Solution

Representative Code Snippet:

```
x = 10
y = 3
print(x * y)     # Multiplication
print(x ** y)    # Exponentiation (power)
print(x // y)    # Floor division (integer quotient)
print(x % y)     # Modulo (remainder)
print(x > y)     # Comparison
```

Output:

```
30
1000
3
1
True
```

Explanation:

- x * y $= 10 \times 3 = 30$
- x ** y $= 10^3 = 1000$
- x // y $= \lfloor 10/3 \rfloor = 3$
- x % y $= 10 \bmod 3 = 1$
- x > y $= 10 > 3 =$ True

2.4 Programs using Operators & Data Types — 7-mark questions

10. List assignment operators in Python and build Python code to demonstrate any three. [7 M]

- *Appeared in:* Su'23 – Q.1(c)

Solution

Assignment Operators: =, +=, -=, *=, /=, %=, **=, //=

Python Program demonstrating three operators:

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

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

x *= 2        # x = x * 2
```

```
print("After x *= 2:", x)    # 30

x //= 4       # x = x // 4
print("After x //= 4:", x)   # 7
```

Output:

```
Initial x = 10
After x += 5:  15
After x *= 2:  30
After x //= 4:  7
```

11. List data types in Python and develop a program to identify any three. **[7 M]**

- *Appeared in:* Su'23 – Q.1(c) [OR]

Solution

Data Types in Python: `int`, `float`, `str`, `bool`, `list`, `tuple`, `dict`, `set`

Python Program to identify data types:

```
# Identifying data types using type()
a = 42
b = 3.14
c = "Hello"

print("Value:", a, "  Type:", type(a))
print("Value:", b, "  Type:", type(b))
print("Value:", c, "  Type:", type(c))

# Using isinstance()
print("\nIs 'a' integer?", isinstance(a, int))
print("Is 'b' float?", isinstance(b, float))
print("Is 'c' string?", isinstance(c, str))
```

Output:

```
Value:  42  Type:  <class 'int'>
Value:  3.14  Type:  <class 'float'>
Value:  Hello  Type:  <class 'str'>
Is 'a' integer?  True
Is 'b' float?  True
Is 'c' string?  True
```

12. Develop Python code to convert temperature from Celsius to Fahrenheit. **[7 M]**

- *Appeared in:* W'23 – Q.1(c)

Solution

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

Python Program:

```
# Temperature conversion: Celsius to Fahrenheit
celsius = float(input("Enter temperature in Celsius: "))
fahrenheit = (celsius * 9/5) + 32
print(f"Temperature in Fahrenheit: {fahrenheit:.2f} F")
```

Sample Output:

```
Enter temperature in Celsius:  100
Temperature in Fahrenheit:  212.00 F
```

Verification: $C = 0 \rightarrow F = 32$; $C = 100 \rightarrow F = 212$ ✓

13. List all comparison operators and explain each with Python code example. **[7 M]**

- *Appeared in:* W'23 – Q.1(c) [OR]

Solution

Comparison (Relational) Operators: Return `True` or `False`.

Operator	Name	Example	Result
`==`	Equal to	`5 == 5`	`True`
`!=`	Not equal to	`5 != 3`	`True`
`>`	Greater than	`7 > 3`	`True`
`<`	Less than	`2 < 5`	`True`
`>=`	Greater or equal	`5 >= 5`	`True`
`<=`	Less or equal	`3 <= 4`	`True`

```
a = 10; b = 20
print(a == b)    # False
print(a != b)    # True
print(a > b)     # False
print(a < b)     # True
print(a >= 10)   # True
print(b <= 20)   # True
```

14. List all logical operators and explain each with Python code example. **[7 M]**

- *Appeared in:* Su'24 – Q.1(c)

Solution

Logical Operators: Used to combine conditional statements.

Operator	Description	Example	Result
`and`	True if **both** conditions are True	`True and False`	`False`
`or`	True if **at least one** is True	`True or False`	`True`
`not`	Reverses the boolean value	`not True`	`False`

```
x = 10
print(x > 5 and x < 20)    # True (both true)
print(x < 5 or x > 8)      # True (second is true)
print(not (x > 5))          # False (reverses True)

# Practical example: age validation
age = 20
if age >= 18 and age <= 60:
    print("Eligible to vote")
```

15. Develop a program to calculate simple interest and compound interest. **[7 M]**

- *Appeared in:* Su'24 – Q.1(c) [OR]

Solution

Formulae: $\text{SI} = \dfrac{P \times R \times T}{100}$; $\text{CI} = P \times \left(1 + \dfrac{R}{100}\right)^{T} - P$

```
# Calculate Simple Interest and Compound Interest
P = float(input("Enter Principal amount: "))
R = float(input("Enter Rate of interest (%): "))
T = float(input("Enter Time (years): "))

SI = (P * R * T) / 100
CI = P * ((1 + R/100) ** T) - P

print(f"\nSimple Interest   = Rs. {SI:.2f}")
```

```
print(f"Compound Interest = Rs. {CI:.2f}")
```

Sample Output:

```
Enter Principal amount:  1000
Enter Rate of interest (%):  10
Enter Time (years):  2
Simple Interest = Rs.  200.00
Compound Interest = Rs.  210.00
```

16. Create a program to calculate total and average marks of a student based on four subject marks. [7 M]

- *Appeared in:* W'24 – Q.2(c)

Solution

```
# Calculate total and average of 4 subject marks
name = input("Enter student name: ")
m1 = float(input("Enter marks of Subject 1: "))
m2 = float(input("Enter marks of Subject 2: "))
m3 = float(input("Enter marks of Subject 3: "))
m4 = float(input("Enter marks of Subject 4: "))

total   = m1 + m2 + m3 + m4
average = total / 4

print(f"\nStudent: {name}")
print(f"Total Marks  : {total}")
print(f"Average Marks: {average:.2f}")
```

Sample Output:

```
Enter student name:  Ravi
Marks of Subject 1:  80 2:  75 3:  90 4:  85
Student:  Ravi  Total:  330.0  Average:  82.50
```

17. Develop code to find square and cube of a given number input by user. [7 M]

- *Appeared in:* W'24 – Q.2(c) [OR]

Solution

```
# Square and cube of a number
num = float(input("Enter a number: "))
square = num ** 2
cube   = num ** 3
print(f"Number : {num}")
print(f"Square : {square}")
print(f"Cube   : {cube}")
```

Sample Output:

```
Enter a number:  5
Number :  5.0
Square :  25.0
Cube :  125.0
```

18. List different data types in Python and explain all data types with examples. [7 M]

- *Appeared in:* Su'25 – Q.2(c)

Solution

Python Data Types (with examples):

1. `int` — Integer; `x = 10, y = -5`

2. `float` — Decimal number; `pi = 3.14, price = 99.5`

3. `complex` — Complex number; `c = 3 + 4j`
4. `str` — String; `name = "Python"`; supports indexing, slicing
5. `bool` — Boolean; `flag = True / False`; subclass of int
6. `list` — Ordered, mutable; `lst = [1, 2.5, "hi"]`
7. `tuple` — Ordered, immutable; `t = (1, 2, 3)`
8. `dict` — Key-value pairs, mutable; `d = {"name": "Alice"}`
9. `set` — Unordered, no duplicates; `s = {1, 2, 3}`
10. `NoneType` — Represents absence of value; `x = None`

Use `type(var)` to check the type of any variable.

19. List different types of operators in Python and explain any two types with example. [7 M]

- *Appeared in:* Su'25 – Q.2(c) [OR]

Solution

Types of Operators in Python: Arithmetic, Relational (Comparison), Assignment, Logical, Bitwise, Membership, Identity.

1. Arithmetic Operators: Perform mathematical operations.

Operator	Operation	Example (x=10, y=3)
`+`	Addition	`x+y = 13`
`-`	Subtraction	`x-y = 7`
`*`	Multiplication	`x*y = 30`
`/`	Division	`x/y = 3.33`
`//`	Floor Division	`x//y = 3`
`%`	Modulo	`x%y = 1`
`**`	Exponentiation	`x**y = 1000`

2. Logical Operators: Combine boolean conditions.
`and` — True if both True `or` — True if any True `not` — reverses boolean
Example: `x > 5 and x < 20` → `True`

20. List different data types in Python. Explain any two data types with example. [7 M]

- *Appeared in:* W'25 – Q.1(c)

Solution

Data Types: int, float, complex, str, bool, list, tuple, dict, set, NoneType.

1. `list` — An ordered, mutable (changeable) sequence of items.
Items can be of different types. Created using square brackets `[ ]`.

```
fruits = ["apple", "banana", "mango"]
numbers = [1, 2.5, 3]
```

`fruits[0]` → `"apple"` `fruits.append("grape")` adds an item.

2. `dict` (Dictionary) — An ordered (Python 3.7+) collection of key–value pairs, enclosed in `{ }`.

```
student = {"name": "Raj", "age": 20, "marks": 85}
```

`student["name"]` → `"Raj"` Keys must be unique; values can be any type.

21. Explain Arithmetic operators and Relational operators with Python code example. [7 M]

- *Appeared in:* W'25 – Q.1(c) [OR]

Solution

Arithmetic Operators: Perform mathematical calculations.

Op	Meaning	Example (a=15, b=4)
`+`	Addition	`a+b = 19`
`-`	Subtraction	`a-b = 11`
`*`	Multiplication	`a*b = 60`
`/`	True Division	`a/b = 3.75`
`//`	Floor Division	`a//b = 3`
`%`	Modulo	`a%b = 3`
`**`	Power	`a**b = 50625`

Relational (Comparison) Operators: Compare two values; return `True/False`.

Op	Meaning	Example (a=15, b=4)
`==`	Equal to	`a==b` → `False`
`!=`	Not equal	`a!=b` → `True`
`>`	Greater than	`a>b` → `True`
`<`	Less than	`a<b` → `False`
`>=`	Greater or equal	`a>=15` → `True`
`<=`	Less or equal	`b<=4` → `True`

Unit 3: Flow of Control

Unit 3 | Flow of Control

3.1 Jump Statements — 3-mark questions

1. Describe the use of `break` statement with necessary examples. [3 M]

- *Appeared in:* Su'23 – Q.3(a)

Solution

`break` Statement: Terminates (exits) the nearest enclosing loop immediately when encountered.

```
# break exits the loop when i == 5
for i in range(1, 11):
    if i == 5:
        break
    print(i, end=" ")
# Output: 1 2 3 4
```

Use case: Exit a search loop as soon as the target is found, avoiding unnecessary iterations.

2. Describe the use of `continue` statement with necessary examples. [3 M]

- *Appeared in:* Su'23 – Q.3(a) [OR]; Su'24 – Q.2(a) [OR]

Solution

`continue` Statement: Skips the rest of the current loop iteration and jumps to the next iteration.

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

Difference from `break`: `break` exits the loop; `continue` only skips the current iteration.

3.2 Selection Statements — 3-mark questions

3. Develop Python code to identify if the scanned number is even or odd. [3 M]

- *Appeared in:* Su'23 – Q.4(a)

Solution

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

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

4. Create Python code to check if a number is positive or negative. **[3 M]**

- *Appeared in:* Su'23 – Q.4(a) [OR]

Solution

```
n = float(input("Enter a number: "))
if n > 0:
    print(n, "is Positive")
elif n < 0:
    print(n, "is Negative")
else:
    print("The number is Zero")
```

5. Explain if-elif-else statement with flowchart and suitable example. **[3 M]**

- *Appeared in:* W'24 – Q.3(a)

Solution

`if-elif-else`: Checks multiple conditions in sequence; executes the first block whose condition is `True`; the `else` block runs if none match.

Syntax:

`if cond1: block1 elif cond2: block2 else: block3`

Flowchart: cond1? Yes→block1; No→cond2? Yes→block2; No→block3.

Example:

```
marks = 75
if marks >= 70:  print("Distinction")
elif marks >= 60:  print("First Class")
else:  print("Pass")
```

Output: `Distinction`

3.3 Loop Statements — 3-mark questions

6. Develop Python program to print numbers 1 to 10 using loops. **[3 M]**

- *Appeared in:* W'23 – Q.3(a)

Solution

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

7. Develop Python code to find odd and even numbers from 1 to N using loops. **[3 M]**

- *Appeared in:* W'23 – Q.3(a) [OR]

Solution

```
N = int(input("Enter N: "))
print("Even:", end=" ")
for i in range(2, N+1, 2):
    print(i, end=" ")
```

```
print("\nOdd:", end=" ")
for i in range(1, N+1, 2):
    print(i, end=" ")
```

8. Write Python program to print odd numbers between 1 to 20 using loops. **[3 M]**

- *Appeared in:* Su'24 – Q.3(a)

Solution

```
for i in range(1, 21, 2):
    print(i, end=" ")
# Output: 1 3 5 7 9 11 13 15 17 19
```

9. Write Python program to find sum of numbers from 1 to 100. **[3 M]**

- *Appeared in:* Su'24 – Q.3(a) [OR]

Solution

```
total = 0
for i in range(1, 101):
    total += i
print("Sum of 1 to 100 =", total)
# Output: Sum of 1 to 100 = 5050
```

Verification: $\frac{n(n+1)}{2} = \frac{100 \times 101}{2} = 5050$ ✓

10. Explain nested loop with a suitable example. **[3 M]**

- *Appeared in:* W'24 – Q.3(a) [OR]

Solution

Nested Loop: A loop inside another loop. The inner loop completes all iterations for every single iteration of the outer loop.

```
# 3x3 Multiplication table
for i in range(1, 4):          # outer loop
    for j in range(1, 4):    # inner loop
        print(i*j, end="\t")
    print()
```

Output:

```
1   2   3
2   4   6
3   6   9
```

3.4 Selection Statements — 4-mark questions

11. Explain if...else statement with suitable example. **[4 M]**

- *Appeared in:* Su'23 – Q.3(b); Su'25 – Q.2(b); W'25 – Q.2(b)

Solution

if-else: Executes one of two blocks based on condition.

Syntax:

```
if condition:
    # executed when True
else:
    # executed when False
```

Example:

```
age = int(input("Enter your age: "))
if age >= 18:
    print("You are eligible to vote.")
else:
```

```
    print("You are not eligible to vote.")
```

Flowchart: condition? Yes→True-block; No→False-block; both→STOP.

12. Explain Nested if statement in Python with python code example. [4 M]

- *Appeared in:* W'23 – Q.2(b); Su'24 – Q.3(b); W'25 – Q.2(b) [OR]

Solution

Nested `if`: An `if` statement placed inside another `if` or `else` block to check additional conditions.

```
# Check if positive and even/odd
n = int(input("Enter a number: "))
if n > 0:
    print("Positive number")
    if n % 2 == 0:
        print("and it is Even")
    else:
        print("and it is Odd")
elif n < 0:
    print("Negative number")
else:
    print("Zero")
```

Output: Enter 8 → `Positive number / and it is Even`

3.5 Loop Statements — 4-mark questions

13. Explain `for` loop statement with example. [4 M]

- *Appeared in:* Su'23 – Q.3(b) [OR]; W'23 – Q.2(b) [OR]; W'24 – Q.2(b) [OR]; Su'25 – Q.3(b)

Solution

`for` Loop: Iterates over a sequence or range for a known number of times.

Syntax:

```
for variable in sequence:    # sequence can be range/list/string
    # loop body
```

Example:

```
for i in range(1, 6):
    print(f"Square of {i} = {i**2}")
# Output: Square of 1=1 ... Square of 5=25
```

`range(start, stop, step)`: Generates numbers from `start` to `stop-1`.

14. Explain the need for `continue` and `break` statements in Python. [4 M]

- *Appeared in:* W'24 – Q.2(b)

Solution

`break` allows early exit when a target is found or an error occurs:

```
nums = [3, 7, 1, 9, 4]
for n in nums:
    if n == 9:
        print("Found 9!"); break
```

`continue` skips invalid/unwanted iterations without stopping the loop:

```
for d in [5, -3, 8, -1, 6]:
    if d < 0:
        continue          # skip negatives
    print(d, end=" ")     # Output: 5 8 6
```

Key: Without these statements, loops are rigid. They add flexibility and control.

15. Explain `while` loop with example. [4 M]

- *Appeared in:* Su'25 – Q.2(b) [OR]

Solution

`while` Loop: Repeats a block as long as its condition is `True`.

```
i = 1
while i <= 5:
    print(i, end=" ")
    i += 1            # prevents infinite loop
# Output: 1 2 3 4 5
```

Note: If the condition is initially `False`, the loop body never executes.

16. Explain `break` statement with example. [4 M]

- *Appeared in:* Su'25 – Q.3(b) [OR]

Solution

`break`: Immediately exits the innermost loop.

```
# Find first multiple of 7
for i in range(1, 51):
    if i % 7 == 0:
        print("First multiple of 7:", i)
        break
# Output: First multiple of 7: 7
```

Control continues at the statement immediately after the loop.

17. Give differences between `for` loop and `while` loop. Develop a Python program to print numbers 1 to 25 using `for` loop. [4 M]

- *Appeared in:* W'25 – Q.3(b)

Solution

Aspect	`for` loop	`while` loop
Use case	Known number of iterations	Condition-based / unknown count
Iteration	Over a sequence/range	Until condition becomes False
Init	In loop header	Before the loop
Risk	No infinite loop risk	Infinite loop if update missing

```
for i in range(1, 26):
    print(i, end=" ")
# Output: 1 2 3 4 ... 25
```

18. Develop Python program to find if the entered number is prime or not. [4 M]

- *Appeared in:* W'25 – Q.3(b) [OR]

Solution

```
n = int(input("Enter a number: "))
if n <= 1:
    print(n, "is neither prime nor composite")
else:
    is_prime = True
    for i in range(2, int(n**0.5) + 1):
        if n % i == 0:
            is_prime = False
            break
    if is_prime:
        print(n, "is a Prime number")
    else:
        print(n, "is NOT a Prime number")
```

3.6 Pattern Programs — 4-mark questions

19. Develop Python program to print the following star pattern: * / * * / * * * / * * * * / * * * * * [4 M]

- *Appeared in:* W'23 – Q.3(b)

Solution

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

Output:

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

20. Write Python program to print the following number triangle pattern: 1 / 2 3 / 4 5 6 / 7 8 9 10 [4 M]

- *Appeared in:* Su'24 – Q.3(b) [OR]

Solution

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

Output: 1 / 2 3 / 4 5 6 / 7 8 9 10

3.7 Selection, Loop & Pattern Programs — 7-mark questions

21. Write Python code to show whether the entered number is prime or not. [7 M]

- *Appeared in:* Su'23 – Q.2(c)

Solution

```
n = int(input("Enter a number: "))
if n <= 1:
    print(n, "is neither prime nor composite")
else:
    is_prime = True
    for i in range(2, int(n**0.5) + 1):
        if n % i == 0:
            is_prime = False
            break
    if is_prime:
        print(n, "is a Prime number")
    else:
        print(n, "is NOT a Prime number",
              "(divisible by", i, ")")
```

Outputs: 29 → Prime; 15 → NOT Prime (div by 3)

22. Write Python code to display the following patterns: (A) number triangle (B) inverted star triangle. [7 M]

- *Appeared in:* Su'23 – Q.2(c) [OR]

Solution

(A) Number triangle (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()
```

(B) Inverted star triangle (* * * * * / ... / *):

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

23. Explain different types of selection/decision making flow-of-control structures with example. [7 M]

- *Appeared in:* W'23 – Q.2(c)

Solution

1. Simple `if`:

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

2. `if-else`:

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

3. `if-elif-else` ladder:

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

4. Nested `if`:

```
n = int(input("Enter: "))
if n > 0:
    if n % 2 == 0: print("Positive Even")
    else:          print("Positive Odd")
else:
    print("Non-positive")
```

24. Describe `break` and `continue` statements in Python in brief. [7 M]

- *Appeared in:* W'23 – Q.2(c) [OR]

Solution

`break`: Exits the enclosing loop immediately.

```
for i in range(1, 11):
    if i == 6:
        break
    print(i, end=" ")
# Output: 1 2 3 4 5
```

`continue`: Skips current iteration; moves to next.

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

Differences:

- `break` — exits the loop entirely
- `continue` — skips current iteration; loop continues
- Both work with `for` and `while`

25. Explain `while` loop with syntax, flowchart and Python code example. [7 M]

- *Appeared in:* Su'24 – Q.2(c)

Solution

Syntax:

```
# initialize variable
while condition:
    # body
    # update variable (avoid infinite loop)
```

Flowchart: START → init → [condition?] False→exit; True→body→update→back to condition.

Example – Sum of digits:

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

Example – Print powers of 2:

```
p = 1
while p <= 512:
    print(p, end=" ")
    p *= 2
# Output: 1 2 4 8 16 32 64 128 256 512
```

26. Explain if-elif-else ladder with syntax, flowchart and Python code. [7 M]

- *Appeared in:* Su'24 – Q.2(c) [OR]

Solution

Syntax:

```
if condition1:
    block1
elif condition2:
    block2
elif condition3:
    block3
else:
    default_block
```

Flowchart: cond1? Yes→block1; No→cond2? Yes→block2; ... No→else.

Example – Grade Calculator:

```
marks = int(input("Enter marks (0-100): "))
if marks >= 80:
    grade = "A (Distinction)"
elif marks >= 70:
    grade = "B (First Class)"
elif marks >= 60:
    grade = "C (Higher Second)"
elif marks >= 50:
    grade = "D (Second Class)"
elif marks >= 35:
    grade = "E (Pass)"
else:
    grade = "F (Fail)"
print("Grade:", grade)
```

Key: Only the first matching block executes; all others are skipped.

27. Create program to display the following patterns: (A) & character triangle (B) number triangle (1 / 1 2 / 1 2 3 / 1 2 3 4 / 1 2 3 4 5). [7 M]

- *Appeared in:* Su'25 – Q.1(c)

Solution

(A) `&` character triangle:

```
for i in range(1, 6):
    print("& " * i)
```

Output: & / & & / & & & / & & & & / & & & & &

(B) Number triangle:

```
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

28. Develop Python programs: (1) Check if entered number is odd or even (2) Find average of 1 to n. [7 M]

- *Appeared in:* Su'25 – Q.1(c) [OR]

Solution

Program 1:

```
num = int(input("Enter a number: "))
print("Even" if num % 2 == 0 else "Odd")
```

Program 2:

```
n = int(input("Enter value of n: "))
total = sum(range(1, n+1))
average = total / n
print(f"Sum={total}, Average={average:.2f}")
```

Output (n=10): `Sum=55, Average=5.50`

29. Create program to display: (A) `@` inverted triangle (B) squared number series (1 / 1 4 / 1 4 9 / 1 4 9 16). [7 M]

- *Appeared in:* W'25 – Q.2(c)

Solution

(A) `@` inverted triangle:

```
for i in range(5, 0, -1):
    print("@ " * i)
```

Output: @ @ @ @ @ / @ @ @ @ / @ @ @ / @ @ / @

(B) Squared number series:

```
for i in range(1, 5):
    for j in range(1, i+1):
        print(j**2, end=" ")
    print()
```

Output: 1 / 1 4 / 1 4 9 / 1 4 9 16

30. Explain `for` loop in Python with its syntax, flowchart and Python code example. [7 M]

- *Appeared in:* W'25 – Q.2(c) [OR]

Solution

Syntax:

```
for variable in sequence:
    # body (executed for each item)
[else:
    # runs when loop ends normally (no break)]
```

Flowchart: START → get next item → items left? No→exit; Yes→body→repeat.

Example – Factorial:

```
n = int(input("Enter n: "))
```

```
fact = 1
for i in range(1, n + 1):
    fact *= i
print(f"{n}! = {fact}")
```

Example – Iterating string:

```
for ch in "Python":
    print(ch, end="-")
# Output: P-y-t-h-o-n-
```

31. Create program to display the following pattern: A / AB / ABC / ABCD / ABCDE. **[7 M]**

- *Appeared in:* W'24 – Q.3(c) [OR]

Solution

```
rows = 5
for i in range(1, rows + 1):
    for j in range(i):
        print(chr(65 + j), end="")   # 65 = ord('A')
    print()
```

Output:

```
A
AB
ABC
ABCD
ABCDE
```

Explanation: `chr(65)`=A, `chr(66)`=B, etc. Row i prints i letters starting from A.

Unit 4: Functions

Unit 4 | Functions

4.1 Functions & Scope of Variables — 3-mark questions

1. Explain local variable with example. **[3 M]**

- *Appeared in:* Su'25 – Q.3(a)

Solution

Local Variable: A variable declared **inside** a function. It exists only within that function and cannot be accessed outside.

```
def greet():
    msg = "Hello!"      # local variable
    print(msg)

greet()                 # Output: Hello!
# print(msg)            # Error: 'msg' not defined
```

Scope: Created when function is called; destroyed when function returns.

2. Explain global variable with example. **[3 M]**

- *Appeared in:* Su'25 – Q.3(a) [OR]

Solution

Global Variable: A variable declared **outside** all functions. Accessible from anywhere in the program.

```
count = 0           # global variable

def increment():
    global count    # declare intention to modify global
```

```
    count += 1
    print("Count:", count)

increment()          # Output: Count: 1
increment()          # Output: Count: 2
print(count)         # Output: 2
```

Key: Use `global` keyword inside function to modify a global variable.

3. List Python standard library mathematical functions. **[3 M]**

- *Appeared in:* W'23 – Q.4(a); Su'24 – Q.4(a) [OR]

Solution

Math Module Functions (`import math`):

Function	Description	Example
`math.sqrt(x)`	Square root	`math.sqrt(16)` = 4.0
`math.pow(x,y)`	x to the power y	`math.pow(2,3)` = 8.0
`math.floor(x)`	Floor value	`math.floor(3.7)` = 3
`math.ceil(x)`	Ceiling value	`math.ceil(3.2)` = 4
`math.fabs(x)`	Absolute value	`math.fabs(-5)` = 5.0
`math.log(x)`	Natural log	`math.log(math.e)` = 1.0
`math.pi`	Value of π	3.14159...
`math.e`	Euler's number	2.71828...

4. Describe Python math module with proper Python code example. **[3 M]**

- *Appeared in:* Su'24 – Q.4(a)

Solution

Math Module: Built-in module providing mathematical functions. Import with `import math`.

```
import math

print("sqrt(25) =", math.sqrt(25))     # 5.0
print("pi       =", math.pi)           # 3.14159...
print("ceil(4.3)=", math.ceil(4.3))    # 5
print("floor(4.9)=",math.floor(4.9))   # 4
print("log(e)   =", math.log(math.e)) # 1.0
```

5. Describe built-in functions `max()`, `input()`, `pow()` with example. **[3 M]**

- *Appeared in:* W'24 – Q.4(a)

Solution

```
# max(): returns the largest value
print(max(3, 7, 1, 9, 4))      # Output: 9
print(max([10, 5, 8]))          # Output: 10

# input(): reads user input as string
name = input("Enter name: ")   # waits for user
print("Hello,", name)

# pow(x, y): returns x raised to power y
print(pow(2, 8))                # Output: 256
print(pow(3, 3))                # Output: 27
```

6. Explain random module with various functions. **[3 M]**

- *Appeared in:* W'24 – Q.4(a) [OR]

Solution

Random Module: Provides functions to generate pseudo-random numbers.

```
import random

# random(): float in [0.0, 1.0)
print(random.random())                # e.g. 0.4873

# randint(a,b): integer in [a, b] inclusive
print(random.randint(1, 6))           # e.g. 4 (dice roll)

# choice(seq): random element from sequence
print(random.choice(["A","B","C"])) # e.g. B

# shuffle(lst): shuffle list in place
lst = [1, 2, 3, 4, 5]
random.shuffle(lst)
print(lst)                            # e.g. [3,1,5,2,4]

# randrange(start,stop,step)
print(random.randrange(0, 101, 5))   # multiple of 5
```

7. Explain built-in functions in Python. [3 M]

- *Appeared in:* W'23 – Q.4(a) [OR]; Su'24 – Q.4(b) [OR]

Solution

Built-in Functions: Functions available without importing any module.

Function	Purpose	Example
`print()`	Display output	`print("Hi")`
`input()`	Read user input	`x = input("Enter:")`
`int()`	Convert to integer	`int("5")` = 5
`float()`	Convert to float	`float("3.14")` = 3.14
`len()`	Length of sequence	`len([1,2,3])` = 3
`range()`	Generate range	`range(1,6)`
`type()`	Type of object	`type(10)` = int
`abs()`	Absolute value	`abs(-5)` = 5
`max()`	Maximum value	`max(3,1,4)` = 4
`min()`	Minimum value	`min(3,1,4)` = 1
`sum()`	Sum of iterable	`sum([1,2,3])` = 6

4.2 User-defined Functions — 4-mark questions

8. Define function. Explain user-defined function with suitable example. [4 M]

- *Appeared in:* Su'23 – Q.4(b)

Solution

Function: A named, reusable block of code that performs a specific task. Called by its name.
User-defined Function: Created by the programmer using `def` keyword.

```
# Defining a function
def greet(name):
    print("Hello,", name, "! Welcome.")

# Calling the function
greet("Raj")      # Output: Hello, Raj ! Welcome.
greet("Priya")    # Output: Hello, Priya ! Welcome.
```

Advantages: Code reusability, modularity, easier debugging, readability.

9. Explain how to define and call a user-defined function with suitable example. [4 M]

- *Appeared in:* W'24 – Q.3(b)

Solution

Defining a Function: Use `def` keyword followed by function name, parameters in parentheses, and colon.

Calling a Function: Use function name followed by arguments in parentheses.

```
# Define function to add two numbers
def add(a, b):              # a, b are parameters
    result = a + b
    return result           # return value to caller

# Call the function
x = add(10, 20)             # 10, 20 are arguments
print("Sum =", x)           # Output: Sum = 30

# Can call multiple times
print(add(5, 3))            # Output: 8
```

10. Explain return statement in function handling. [4 M]

- *Appeared in:* W'24 – Q.3(b) [OR]

Solution

`return` Statement: Exits a function and optionally sends a value back to the caller.

```
def square(n):
    return n * n            # returns computed value

result = square(7)
print("Square =", result)    # Output: Square = 49

# Function can return multiple values
def min_max(lst):
    return min(lst), max(lst)

lo, hi = min_max([3, 1, 4, 1, 5, 9])
print("Min:", lo, "Max:", hi)    # Min: 1  Max: 9
```

Notes: A function with no `return` returns `None`. `return` without a value also returns `None`.

11. Develop Python program using user-defined function to find factorial of the number taken from user. [4 M]

- *Appeared in:* Su'25 – Q.4(b); W'25 – Q.4(b) [OR]

Solution

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

num = int(input("Enter a positive integer: "))
if num < 0:
    print("Factorial undefined for negative numbers")
else:
    print(f"{num}! = {factorial(num)}")
```

Output: Enter 5 → `5! = 120`

12. Develop Python program using user-defined function to check if the number taken from user is prime. [4 M]

- *Appeared in:* Su'25 – Q.4(b) [OR]

Solution

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

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

13. Create a user-defined function to print the Fibonacci series of 0 to N numbers (N as argument). [4 M]

- *Appeared in:* W'25 – Q.4(b)

Solution

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

N = int(input("Enter N: "))
fibonacci(N)
```

Output (N=50): 0 1 1 2 3 5 8 13 21 34

4.3 Scope of Variables — 4-mark questions

14. Explain local and global variables with suitable examples. [4 M]

- *Appeared in:* Su'23 – Q.4(b) [OR]

Solution

Local Variable: Defined inside a function; accessible only within that function.

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

demo()
# print(x)            # Error: x not defined here
```

Global Variable: Defined outside functions; accessible everywhere.

```
total = 100             # global variable

def show():
    print("Global total =", total)    # reads global

def update():
    global total
    total += 50         # modifies global

show()     # Global total = 100
update()
show()     # Global total = 150
```

15. Write Python program that explains the scope of a variable. [4 M]

- *Appeared in:* Su'24 – Q.4(b)

Solution

```
# Demonstrating variable scope
x = "global"            # global scope

def outer():
    x = "outer"         # local to outer()

    def inner():
        x = "inner"     # local to inner()
        print("inner x:", x)

    inner()
    print("outer x:", x)

outer()
print("global x:", x)
```

Output:

```
inner x:  inner
outer x:  outer
global x:  global
```

LEGB Rule: Python searches scopes in order: **L**ocal → **E**nclosing → **G**lobal → **B**uilt-in.

4.4 Modules & Built-in Functions — 4-mark questions

16. Explain math module and random module with example. [4 M]

- *Appeared in:* Su'23 – Q.5(b)

Solution

Math Module:

```
import math
print(math.sqrt(49))    # 7.0
print(math.pi)          # 3.14159...
print(math.factorial(5))  # 120
```

Random Module:

```
import random
print(random.random())          # float [0.0, 1.0)
print(random.randint(1, 100))   # integer 1-100
lst = [10, 20, 30, 40]
random.shuffle(lst)
print(lst)                      # shuffled list
```

17. Explain usage of built-in functions: `print()`, `min()`, `sum()`, `input()`. [4 M]

- *Appeared in:* Su'23 – Q.5(b) [OR]

Solution

```
# print(): display output with optional sep, end
print("Hello", "World", sep="-", end="!\n")
# Output: Hello-World!

# input(): read user input (returns string)
name = input("Enter name: ")
print("Hello,", name)

# min(): return minimum value
print(min(5, 2, 8, 1))          # 1
print(min([10, 3, 7]))          # 3

# sum(): return sum of iterable
print(sum([1, 2, 3, 4, 5]))     # 15
```

```python
print(sum(range(1, 11)))        # 55
```

18. Explain Module in Python with python code example. [4 M]

- *Appeared in:* W'23 – Q.4(b)

Solution

Module: A Python file (`.py`) containing functions, classes, and variables that can be imported and reused in other programs.
Types: Built-in (`math`, `random`, `os`), Third-party (`numpy`, `pandas`), User-defined.
Importing a module:

```python
import math                 # import entire module
print(math.sqrt(25))        # 5.0

from math import pi, ceil   # import specific items
print(pi)                   # 3.14159...
print(ceil(4.2))            # 5

import math as m            # alias
print(m.floor(7.9))         # 7
```

19. Describe Python math module with python code example. [4 M]

- *Appeared in:* W'23 – Q.4(b) [OR]

Solution

Math Module: Provides access to mathematical functions. Must be imported with `import math`.

```python
import math
print("sqrt(144)  =", math.sqrt(144))    # 12.0
print("pow(2,10)  =", math.pow(2,10))    # 1024.0
print("floor(3.9) =", math.floor(3.9))   # 3
print("ceil(3.1)  =", math.ceil(3.1))    # 4
print("pi         =", math.pi)           # 3.14159...
print("e          =", math.e)            # 2.71828...
print("factorial(6)=",math.factorial(6))# 720
print("log10(1000)=", math.log10(1000)) # 3.0
```

20. Explain Random module in Python. [4 M]

- *Appeared in:* W'25 – Q.5(b)

Solution

Random Module: Generates pseudo-random numbers. Import with `import random`.

```python
import random
print(random.random())          # float [0.0,1.0)
print(random.uniform(1, 10))    # float [1.0,10.0]
print(random.randint(1, 6))     # int [1,6] (dice)
print(random.randrange(0,100,2))# even number 0-98
print(random.choice("ABCDE"))   # random char
lst = [1,2,3,4,5]
random.shuffle(lst)
print(lst)                      # shuffled
print(random.sample(lst, 3))    # 3 unique random items
```

21. Explain statistics module in Python. [4 M]

- *Appeared in:* W'25 – Q.5(b) [OR]

Solution

Statistics Module: Provides functions for statistical calculations.

```
import statistics as st

data = [10, 20, 30, 20, 40, 30, 20]

print("Mean    =", st.mean(data))      # 24.28...
print("Median  =", st.median(data))    # 20
print("Mode    =", st.mode(data))      # 20
print("StDev   =", st.stdev(data))     # 10.19...
print("Variance=", st.variance(data)) # 103.9...
```

Key functions: `mean()`, `median()`, `mode()`, `stdev()`, `variance()`, `median_low()`, `median_high()`.

4.5 User-defined Functions & Programs — 7-mark questions

22. Create user-defined function to print Fibonacci series 0 to N. [7 M]

- *Appeared in:* Su'23 – Q.3(c); W'24 – Q.4(c) [OR]

Solution

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

N = int(input("Enter N: "))
fibonacci(N)
```

Output (N=100): `0 1 1 2 3 5 8 13 21 34 55 89`

Logic: Start with a=0, b=1. Each step: print a, then update a=b, b=a+b (previous a).

23. Write Python code to determine if a number is Armstrong or palindrome using user-defined function. [7 M]

- *Appeared in:* Su'23 – Q.3(c) [OR]

Solution

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

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

num = int(input("Enter a number: "))

if is_armstrong(num):
    print(num, "is an Armstrong number")
else:
    print(num, "is NOT an Armstrong number")

if is_palindrome(num):
    print(num, "is a Palindrome")
else:
    print(num, "is NOT a Palindrome")
```

Note: Armstrong (narcissistic): $153 = 1^3 + 5^3 + 3^3$; Palindrome: 121 = "121" reversed.

24. Create user-defined function to find factorial of a given number. [7 M]

- *Appeared in:* W'23 – Q.3(c); W'24 – Q.3(c)

Solution

```
def factorial(n):
    """Return factorial of n using iteration."""
    if n < 0:
        return None           # undefined for negatives
    if n == 0 or n == 1:
        return 1
    result = 1
    for i in range(2, n + 1):
        result *= i
    return result

# Test the function
num = int(input("Enter a non-negative integer: "))
result = factorial(num)
if result is None:
    print("Factorial is undefined for negative numbers")
else:
    print(f"{num}! = {result}")
```

Outputs: 0!=1; 5!=120; 10!=3628800

25. Explain local and global variables with examples; explain the concept of scope of variable. [7 M]

- *Appeared in:* W'23 – Q.3(c) [OR]; W'23 – Q.4(c) [OR]; W'25 – Q.3(c)

Solution

Local Variable: Exists only inside the function where it is defined.

```
def func():
    local_var = "I am local"
    print(local_var)

func()
# print(local_var)   # NameError!
```

Global Variable: Defined outside all functions; accessible everywhere.

```
global_var = "I am global"

def func():
    print(global_var)     # can read global
    global global_var
    global_var = "Modified"

func()
print(global_var)         # "Modified"
```

Scope Concept (LEGB Rule): Python resolves names in this order: **L**ocal → **E**nclosing (nested function) → **G**lobal → **B**uilt-in.

```
x = "global"
def outer():
    x = "enclosing"
    def inner():
        print(x)     # finds 'x' in enclosing scope
    inner()
outer()              # Output: enclosing
```

26. Write Python program to determine if a number is Armstrong or palindrome using user-defined function (number passed as argument). [7 M]

- *Appeared in:* W'23 – Q.4(c); Su'24 – Q.3(c)

Solution

```
def is_armstrong(num):
    s = str(num)
    n = len(s)
    return num == sum(int(d)**n for d in s)

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

def check_number(num):
    print(f"\nChecking number: {num}")
    if is_armstrong(num):
        print(f"{num} is an Armstrong number")
    else:
        print(f"{num} is NOT an Armstrong number")
    if is_palindrome(num):
        print(f"{num} is a Palindrome")
    else:
        print(f"{num} is NOT a Palindrome")

# Test cases
for n in [153, 121, 370, 123]:
    check_number(n)
```

Outputs: 153: Armstrong+Palindrome? No; 121: Not Armstrong, Palindrome

27. Write program using a function that reverses the entered value. [7 M]

- *Appeared in:* Su'24 – Q.3(c) [OR]

Solution

```
def reverse_number(n):
    """Reverse an integer without converting to string."""
    reversed_num = 0
    sign = -1 if n < 0 else 1
    n = abs(n)
    while n > 0:
        reversed_num = reversed_num * 10 + n % 10
        n //= 10
    return sign * reversed_num

def reverse_string(s):
    return s[::-1]

# Test number reversal
num = int(input("Enter an integer: "))
print("Reversed number:", reverse_number(num))

# Test string reversal
text = input("Enter a string: ")
print("Reversed string:", reverse_string(text))
```

Outputs: 12345 → 54321; "Python" → "nohtyP"

28. Create user-defined function which prints cube of all odd numbers between 1 to 7. [7 M]

- *Appeared in:* W'24 – Q.4(c)

Solution

```
def cube_of_odds():
    """Print cube of odd numbers between 1 and 7."""
    print("Odd numbers and their cubes (1 to 7):")
    for i in range(1, 8):       # 1 to 7 inclusive
```

```
        if i % 2 != 0:          # check if odd
            print(f"  {i}^3 = {i**3}")

cube_of_odds()
```

Output:

```
1^3 = 1
3^3 = 27
5^3 = 125
7^3 = 343
```

29. List functions of math module in Python and develop a Python program demonstrating math module functions. [7 M]

- *Appeared in:* Su'25 – Q.3(c)

Solution

Key Math Module Functions: `sqrt()`, `pow()`, `floor()`, `ceil()`, `fabs()`, `log()`, `log10()`, `sin()`, `cos()`, `tan()`, `pi`, `e`, `factorial()`, `gcd()`, `trunc()`, `exp()`.

```
import math

print("--- Math Module Demo ---")
print("sqrt(64)   =", math.sqrt(64))       # 8.0
print("pow(3,4)   =", math.pow(3,4))       # 81.0
print("floor(7.8) =", math.floor(7.8))     # 7
print("ceil(7.2)  =", math.ceil(7.2))      # 8
print("fabs(-15)  =", math.fabs(-15))      # 15.0
print("log(e)     =", math.log(math.e))    # 1.0
print("log10(100) =", math.log10(100))     # 2.0
print("sin(pi/2)  =", math.sin(math.pi/2))# 1.0
print("factorial(7)=",math.factorial(7))   # 5040
print("gcd(12,8)  =", math.gcd(12,8))     # 4
print("pi         =", math.pi)
print("e          =", math.e)
```

30. List functions of statistics module in Python and develop a Python program demonstrating statistics module functions. [7 M]

- *Appeared in:* Su'25 – Q.3(c) [OR]

Solution

Key Statistics Module Functions: `mean()`, `median()`, `mode()`, `stdev()`, `variance()`, `median_low()`, `median_high()`, `geometric_mean()`, `harmonic_mean()`.

```
import statistics as st

marks = [72, 85, 68, 90, 72, 78, 85, 72, 88]

print("--- Statistics Demo ---")
print("Data   :", marks)
print("Mean   :", round(st.mean(marks), 2))      # avg
print("Median :", st.median(marks))              # middle
print("Mode   :", st.mode(marks))                # most freq
print("Std Dev:", round(st.stdev(marks), 2))     # spread
print("Variance:", round(st.variance(marks), 2)) # spread^2
print("Med Low:", st.median_low(marks))
print("Med Hi :", st.median_high(marks))
```

31. List Python standard library mathematical functions and explain any three in brief. [7 M]

- *Appeared in:* W'25 – Q.3(c) [OR]

Solution

Standard Library Math Functions: `sqrt()`, `pow()`, `floor()`, `ceil()`, `fabs()`, `log()`, `log2()`, `log10()`, `sin()`, `cos()`, `tan()`, `factorial()`, `gcd()`, `trunc()`, `exp()`, `hypot()`, `pi`, `e`, `inf`, `nan`

Explanation of three functions:

1. `math.sqrt(x)`: Returns the square root of x.

```
import math
print(math.sqrt(100))  # 10.0
print(math.sqrt(2))    # 1.4142...
```

2. `math.factorial(n)`: Returns n! (product of 1 to n).

```
print(math.factorial(0))  # 1
print(math.factorial(5))  # 120
print(math.factorial(10)) # 3628800
```

3. `math.gcd(a, b)`: Returns Greatest Common Divisor of a and b.

```
print(math.gcd(48, 36))  # 12
print(math.gcd(100, 75)) # 25
```

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

Unit 5 | Dictionary, List, Set, String and Tuple

5.1 List — 3-mark questions

1. Write Python code to swap two elements in a list. **[3 M]**

- *Appeared in:* Su'23 – Q.5(a); W'23 – Q.5(a); Su'24 – Q.5(a); Su'25 – Q.5(a) [OR]

Solution

```
lst = [10, 20, 30, 40, 50]
print("Before swap:", lst)

# Swap elements at index 1 and 3
lst[1], lst[3] = lst[3], lst[1]
print("After swap:", lst)
```

Output:

```
Before swap:  [10, 20, 30, 40, 50]
After swap:  [10, 40, 30, 20, 50]
```

2. Write Python code to find sum of all elements in a list. **[3 M]**

- *Appeared in:* Su'23 – Q.5(a) [OR]; W'23 – Q.5(a) [OR]; Su'24 – Q.5(a) [OR]

Solution

```
lst = [5, 10, 15, 20, 25]

# Method 1: Using built-in sum()
print("Sum (built-in):", sum(lst))      # 75

# Method 2: Using loop
total = 0
for element in lst:
    total += element
print("Sum (loop):", total)             # 75
```

3. Define list and explain how it is created in Python. **[3 M]**

- *Appeared in:* W'24 – Q.5(a) [OR]

Solution

List: An ordered, mutable (changeable) collection of items enclosed in square brackets `[ ]`. Items can be of different data types.

Creating a List:

- Empty list: `lst = []`
- Integer list: `nums = [1, 2, 3, 4, 5]`
- Mixed list: `mixed = [10, "hello", 3.14, True]`
- Using `list()`: `lst = list(range(1, 6))` → `[1,2,3,4,5]`
- Nested list: `matrix = [[1,2],[3,4],[5,6]]`

Access elements: `nums[0]` = 1 (first), `nums[-1]` = 5 (last).

4. Find the smallest number in the given list: `List1 = [21, 6, 3, 18, 12, 15]`. [3 M]

- *Appeared in:* Su'25 – Q.5(a)

Solution

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

# Method 1: Built-in min()
print("Smallest:", min(List1))     # 3

# Method 2: Using loop
smallest = List1[0]
for num in List1:
    if num < smallest:
        smallest = num
print("Smallest:", smallest)       # 3
```

5. List any six built-in functions of List. [3 M]

- *Appeared in:* W'25 – Q.4(a)

Solution

Function/Method	Purpose	Example
`len(lst)`	Length of list	`len([1,2,3])` = 3
`lst.append(x)`	Add item at end	`[1,2].append(3)`
`lst.insert(i,x)`	Insert at index i	`[1,3].insert(1,2)`
`lst.remove(x)`	Remove first x	`[1,2,3].remove(2)`
`lst.sort()`	Sort in place	`[3,1,2].sort()`
`lst.reverse()`	Reverse in place	`[1,2,3].reverse()`
`lst.index(x)`	Index of first x	`[1,2,3].index(2)` = 1
`lst.count(x)`	Count occurrences	`[1,2,2].count(2)` = 2

6. Develop Python program to find multiplication of all elements in the list. [3 M]

- *Appeared in:* W'25 – Q.5(a)

Solution

```
lst = [2, 3, 4, 5]
product = 1
for num in lst:
    product *= num
print("Product of all elements:", product)  # 120
```

5.1b List — 4-mark questions

7. Develop code to create a nested list and display its elements. [4 M]

- *Appeared in:* W'23 – Q.3(b) [OR]

Solution

```
# Create a 3x3 matrix as nested list
matrix = [[1, 2, 3],
          [4, 5, 6],
          [7, 8, 9]]

print("Nested list (matrix):")
for row in matrix:
    for element in row:
        print(element, end="\t")
    print()

# Access specific element
print("\nElement at row 1, col 2:", matrix[1][2])  # 6
```

8. Explain nested list by giving example. [4 M]

- *Appeared in:* W'23 – Q.5(b)

Solution

Nested List: A list that contains other lists as its elements. Used to represent 2D data like matrices, tables.

Example:

`students = [["Raj", 85], ["Priya", 92], ["Sam", 78]]`

Access: `students[0]` = `["Raj", 85]` `students[1][0]` = `"Priya"`

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

`matrix[1][2]` = 6 (row 1, column 2)

Iteration: Use nested `for` loops to access all elements.

9. Explain indexing and slicing operations in Python list. [4 M]

- *Appeared in:* W'23 – Q.5(b) [OR]

Solution

Indexing: Access individual elements using index (0-based from front; -1 from end).

`lst = [10, 20, 30, 40, 50]`

`lst[0] = 10; lst[2] = 30; lst[-1] = 50; lst[-2] = 40`

Slicing: Extract a sub-list using `lst[start:stop:step]`.

`lst[1:4]` = `[20, 30, 40]` (index 1 to 3)

`lst[:3]` = `[10, 20, 30]` (first 3)

`lst[2:]` = `[30, 40, 50]` (from index 2)

`lst[::2]` = `[10, 30, 50]` (every 2nd)

`lst[::-1]` = `[50, 40, 30, 20, 10]` (reversed)

10. Discuss list functions: `len()`, `sum()`, `sort()`, `index()`. [4 M]

- *Appeared in:* W'24 – Q.4(b) [OR]

Solution

`lst = [30, 10, 50, 20, 40]`

1. `len(lst)`: Returns number of elements. `len(lst)` = 5.

2. `sum(lst)`: Returns sum of all numeric elements. `sum(lst)` = 150.

3. `lst.sort()`: Sorts list in ascending order (in place).

After `lst.sort()`: `[10, 20, 30, 40, 50]`.

`lst.sort(reverse=True)`: descending order.

4. `lst.index(x)`: Returns index of first occurrence of x.

`lst.index(30)` = 0 (after sort, 30 is at index 2).

Raises `ValueError` if x not in list.

5.2 String — 3-mark questions

11. List any six built-in functions of string. **[3 M]**

- *Appeared in:* W'25 – Q.3(a)

Solution

Method	Purpose	Example
`upper()`	Convert to uppercase	`"hello".upper()` = `"HELLO"`
`lower()`	Convert to lowercase	`"HELLO".lower()` = `"hello"`
`len()`	Length of string	`len("Python")` = 6
`strip()`	Remove leading/trailing spaces	`" hi ".strip()` = `"hi"`
`split()`	Split into list	`"a b".split()` = `["a","b"]`
`replace()`	Replace substring	`"cat".replace("c","b")` = `"bat"`
`find()`	Index of substring	`"hello".find("ll")` = 2
`count()`	Count occurrences	`"aabc".count("a")` = 2

12. Explain any one String Operation with example. **[3 M]**

- *Appeared in:* W'25 – Q.3(a) [OR]

Solution

String Concatenation: Joining two or more strings using the `+` operator.

```
s1 = "Hello"; s2 = " World"
s3 = s1 + s2 → "Hello World"
```

String Repetition: Using `*` to repeat a string.

```
"ab" * 3 → "ababab"
```

String Membership: Using `in` to check if substring exists.

```
"py" in "python" → True
```

13. Explain string methods: `count()`, `upper()`, `replace()` with examples. **[3 M]**

- *Appeared in:* W'24 – Q.5(a)

Solution

1. `count(sub)`: Counts occurrences of substring.

`"banana".count("a")` = 3 `"hello".count("l")` = 2

2. `upper()`: Converts all characters to uppercase.

`"python".upper()` = `"PYTHON"` `"hello123".upper()` = `"HELLO123"`

3. `replace(old, new)`: Replaces all occurrences of `old` with `new`.

`"I like cats".replace("cats","dogs")` = `"I like dogs"`

`"aabbcc".replace("b","x")` = `"aaxxcc"`

14. Explain indexing in string with example. **[3 M]**

- *Appeared in:* Su'25 – Q.4(a) [OR]

Solution

String Indexing: Access individual characters using index (0-based from start; -1 from end).

```
s = "Python"
```

Char:	P	y	t	h	o	n
Index:	0	1	2	3	4	5
Neg:	-6	-5	-4	-3	-2	-1

`s[0]` = 'P'; `s[3]` = 'h'; `s[-1]` = 'n'; `s[-2]` = 'o'

Strings are immutable: `s[0] = "J"` raises `TypeError`.

15. Develop Python program to count and display number of vowels in a string. **[3 M]**

- *Appeared in:* W'25 – Q.4(a) [OR]

Solution

```
s = input("Enter a string: ")
vowels = "aeiouAEIOU"
count = sum(1 for ch in s if ch in vowels)
print(f"Number of vowels in '{s}': {count}")
```

Output: Enter "Hello World" → `Number of vowels: 3`

5.2b String — 4-mark questions

16. Explain slicing of string with suitable example. [4 M]

- *Appeared in:* W'24 – Q.4(b)

Solution

String Slicing: Extract a substring using `s[start:stop:step]`.
`stop` is exclusive; omitting `start`/`stop` defaults to beginning/end.
`s = "Programming"`
`s[0:4]` = `"Prog"` (index 0 to 3)
`s[4:]` = `"ramming"` (from index 4 to end)
`s[:7]` = `"Program"` (first 7 chars)
`s[::2]` = `"Pormin"` (every 2nd char)
`s[::-1]` = `"gnimmargorP"` (reversed)
`s[2:8:2]` = `"ora"` (index 2,4,6)

17. Write Python program to check if a substring is present in a given string. [4 M]

- *Appeared in:* Su'24 – Q.5(b)

Solution

```
def check_substring(main, sub):
    if sub in main:
        print(f"'{sub}' IS present in '{main}'")
        print(f"Position: {main.find(sub)}")
    else:
        print(f"'{sub}' is NOT present in '{main}'")

main = input("Enter main string: ")
sub  = input("Enter substring to search: ")
check_substring(main, sub)
```

Output: main="Hello World", sub="World" → `'World' IS present, Position: 6`

18. Develop Python program to demonstrate any 4 built-in functions for string. [4 M]

- *Appeared in:* Su'25 – Q.5(b)

Solution

```
s = "Hello, Python World!"
print("Original:", s)
print("upper()  :", s.upper())
print("lower()  :", s.lower())
print("replace():", s.replace("Python","Java"))
print("split()  :", s.split())
print("find()   :", s.find("Python"))    # 7
print("count(l) :", s.count("l"))        # 3
print("strip()  :", "  spaces  ".strip())
print("startswith:", s.startswith("Hello"))  # True
```

5.3 Tuple — 3 & 4-mark questions

19. Explain concept of nested tuple with example. [3 M]

- *Appeared in:* Su'25 – Q.4(a)

Solution

Nested Tuple: A tuple that contains other tuples as elements.

`student = ("Raj", (85, 90, 78), "Engineering")`

`student[0]` = `"Raj"`

`student[1]` = `(85, 90, 78)` (inner tuple)

`student[1][0]` = 85 (first mark)

`matrix = ((1,2,3), (4,5,6), (7,8,9))`

`matrix[1][2]` = 6 (row 1, col 2)

Key: Tuples are immutable — nested tuples cannot be modified after creation.

20. Explain tuple operations with example. [4 M]

- *Appeared in:* W'24 – Q.5(b)

Solution

`t1 = (1, 2, 3); t2 = (4, 5, 6)`

1. **Concatenation (+):** `t1 + t2` = `(1,2,3,4,5,6)`
2. **Repetition (*):** `t1 * 2` = `(1,2,3,1,2,3)`
3. **Indexing:** `t1[0]` = 1; `t1[-1]` = 3
4. **Slicing:** `t1[1:3]` = `(2, 3)`
5. **Membership:** `2 in t1` = `True`
6. **Length:** `len(t1)` = 3
7. **Iteration:** `for x in t1: print(x)`
8. **Comparison:** `(1,2) < (1,3)` = `True` (element-wise)
9. **Min/Max:** `min(t1)` = 1; `max(t1)` = 3

21. Develop Python program to demonstrate any 4 built-in functions for tuple. [4 M]

- *Appeared in:* Su'25 – Q.5(b) [OR]

Solution

```
t = (30, 10, 50, 20, 40, 10, 20, 10)
print("Tuple:", t)
print("len()   :", len(t))          # 8
print("min()   :", min(t))          # 10
print("max()   :", max(t))          # 50
print("sum()   :", sum(t))          # 190
print("count(10):", t.count(10))    # 3
print("index(50):", t.index(50))    # 2
print("sorted():", sorted(t))       # sorted list
```

5.4 Set & Dictionary — 4-mark questions

22. Write program to demonstrate set functions and operations. [4 M]

- *Appeared in:* Su'24 – Q.5(b) [OR]

Solution

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

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

```
A.remove(1); print("remove(1):", A)
print("len:", len(A))
print("3 in A:", 3 in A)
```

23. Explain dictionary built-in functions and methods. [4 M]

- *Appeared in:* W'24 – Q.5(b) [OR]

Solution

```
d = {"name":"Raj","age":20,"marks":85}
```

Method	Purpose	Example
`d.keys()`	All keys	`dict_keys(['name','age','marks'])`
`d.values()`	All values	`dict_values(['Raj',20,85])`
`d.items()`	Key-value pairs	`dict_items([...])`
`d.get(k)`	Value for key k	`d.get("age")` = 20
`d.update()`	Update with dict	`d.update({"city":"Surat"})`
`d.pop(k)`	Remove key k	`d.pop("age")`
`len(d)`	Number of pairs	`len(d)` = 3
`"name" in d`	Key membership	`True`

24. Develop a Python program to create a Dictionary with the roll number, name and marks of n students; display the names of students who have scored marks above 70. [4 M]

- *Appeared in:* W'25 – Q.5(a) [OR]

Solution

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

for i in range(n):
    roll = int(input(f"Enter roll number of student {i+1}: "))
    name = input("Enter name: ")
    marks = float(input("Enter marks: "))
    students[roll] = {"name": name, "marks": marks}

print("\nStudents with marks above 70:")
for roll, info in students.items():
    if info["marks"] > 70:
        print(f"  Roll: {roll}, Name: {info['name']}, Marks: {info['
            marks']}")
```

5.5 List — 7-mark questions

25. List various list operations and explain any three with example. [7 M]

- *Appeared in:* Su'23 – Q.4(c) [OR]

Solution

List Operations: Concatenation, Repetition, Indexing, Slicing, Membership, Iteration, Length, Comparison, Deletion.

1. Concatenation (+): Joins two lists into a new list.

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

2. Slicing (`lst[i:j:k]`): Extract sub-list.

`lst = [10,20,30,40,50]`

`lst[1:4]` = `[20,30,40]`; `lst[::-1]` = `[50,40,30,20,10]`

3. List Methods:

`lst.append(x)` – add at end; `lst.sort()` – sort in place; `lst.remove(x)` – remove first x; `lst.pop()` – remove and return last item.

26. Explain List Methods and its built-in Functions. [7 M]

- *Appeared in:* Su'24 – Q.4(c)

Solution

List Methods:

Method	Description	Example
`append(x)`	Add x at end	`[1,2].append(3)` → `[1,2,3]`
`insert(i,x)`	Insert x at index i	`[1,3].insert(1,2)` → `[1,2,3]`
`remove(x)`	Remove first x	`[1,2,2].remove(2)` → `[1,2]`
`pop(i)`	Remove at index i	`[1,2,3].pop()` → 3
`sort()`	Sort ascending	`[3,1,2].sort()` → `[1,2,3]`
`reverse()`	Reverse in place	`[1,2,3].reverse()` → `[3,2,1]`
`count(x)`	Count x occurrences	`[1,2,2].count(2)` = 2
`index(x)`	Index of first x	`[1,2,3].index(2)` = 1
`extend(l)`	Extend with iterable	`[1].extend([2,3])` → `[1,2,3]`
`clear()`	Remove all elements	`lst.clear()` → `[]`

Built-in Functions: `len()`, `min()`, `max()`, `sum()`, `sorted()`, `list()`, `enumerate()`.

27. Explain List operations with examples. **[7 M]**

- *Appeared in:* Su'25 – Q.4(c)

Solution

`lst = [5, 2, 8, 1, 9, 3]`

1. **Indexing:** `lst[0]`=5; `lst[-1]`=3
2. **Slicing:** `lst[2:5]`=`[8,1,9]`; `lst[::-1]`=`[3,9,1,8,2,5]`
3. **Concatenation:** `lst + [10,11]`=`[5,2,8,1,9,3,10,11]`
4. **Repetition:** `[0]*5`=`[0,0,0,0,0]`
5. **Membership:** `8 in lst`=True; `7 not in lst`=True
6. **Length:** `len(lst)`=6
7. **Sort:** `lst.sort()` → `[1,2,3,5,8,9]`
8. **Deletion:** `del lst[0]`; `lst.pop()`; `lst.remove(3)`
9. **Iteration:** `for x in lst: print(x)`

28. Develop Python code to create a list of prime numbers and a list of non-prime numbers in the range 1 to 50. **[7 M]**

- *Appeared in:* W'24 – Q.5(c) [OR]

Solution

```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

primes     = [n for n in range(1, 51) if is_prime(n)]
non_primes = [n for n in range(1, 51) if not is_prime(n)]

print("Prime numbers (1-50):")
print(primes)
print("\nNon-prime numbers (1-50):")
print(non_primes)
```

Prime output: [2, 3, 5, 7, 11, 13, 17, 19, 23, 29, 31, 37, 41, 43, 47]

29. Explain Indexing and Slicing in List with example. **[7 M]**

- *Appeared in:* W'25 – Q.5(c) [OR]

Solution

`lst = [10, 20, 30, 40, 50, 60, 70]`

Indexing (accessing single element):

Element:	10	20	30	40	50	60	70
+Index:	0	1	2	3	4	5	6
−Index:	−7	−6	−5	−4	−3	−2	−1

`lst[0]`=10; `lst[3]`=40; `lst[-1]`=70; `lst[-3]`=50

Slicing `lst[start:stop:step]`:

- `lst[2:5]` = `[30,40,50]` (index 2,3,4)
- `lst[:4]` = `[10,20,30,40]` (first 4)
- `lst[4:]` = `[50,60,70]` (from index 4)
- `lst[::2]` = `[10,30,50,70]` (every 2nd)
- `lst[::-1]` = `[70,60,50,40,30,20,10]` (reversed)
- `lst[1:6:2]` = `[20,40,60]` (step 2, index 1,3,5)

5.6 String — 7-mark questions

30. List various string operations and explain any three with example. **[7 M]**

- *Appeared in:* Su'23 – Q.4(c)

Solution

String Operations: Concatenation, Repetition, Indexing, Slicing, Membership, Comparison, Iteration.

1. Concatenation and Repetition:

`"Hello" + " " + "World" = "Hello World"`

`"ab" * 3 = "ababab"`

2. Slicing:

`s = "Programming"`

`s[0:7] = "Program"`; `s[::-1] = "gnimmargorP"`

3. String Methods:

`"hello world".title() = "Hello World"`

`"Python".replace("P","J") = "Jython"`

`"a,b,c".split(",") = ["a","b","c"]`

`" trim ".strip() = "trim"`

31. Explain string operations with examples. **[7 M]**

- *Appeared in:* W'23 – Q.5(c)

Solution

`s = "Hello, Python!"`

1. Concatenation: `s + " Rocks" = "Hello, Python!  Rocks"`

2. Repetition: `"Hi" * 3 = "HiHiHi"`

3. Indexing: `s[0]='H'`; `s[-1]='!'`

4. Slicing: `s[7:13]="Python"`; `s[:5]="Hello"`

5. Membership: `"Python" in s=True`

6. upper/lower: `s.upper() = "HELLO, PYTHON!"`

7. find/count: `s.find("Python")`=7; `s.count("l")`=2

8. replace: `s.replace("Python","Java") = "Hello, Java!"`

9. split: `s.split(",") = ["Hello", " Python!"]`

10. Comparison: `"apple" < "banana" = True` (lexicographic)

32. Write Python program to count and display vowels, consonants, uppercase letters and lowercase letters in a string. **[7 M]**

- *Appeared in:* Su'24 – Q.4(c) [OR]

Solution

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

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

print(f"String : {s}")
print(f"Vowels     : {vowels}")
print(f"Consonants : {consonants}")
print(f"Uppercase  : {upper}")
print(f"Lowercase  : {lower}")
print(f"Others     : {others}")
```

Output: Input "Hello World" → Vowels:3 Consonants:7 Upper:2 Lower:8 Others:1

5.7 Tuple — 7-mark questions

33. Write Python code to demonstrate tuple functions and operations. [7 M]

- *Appeared in:* Su'23 – Q.5(c)

Solution

```
t1 = (3, 1, 4, 1, 5, 9, 2, 6)
t2 = (7, 8)

print("Tuple t1:", t1)
print("len()    :", len(t1))          # 8
print("min()    :", min(t1))          # 1
print("max()    :", max(t1))          # 9
print("sum()    :", sum(t1))          # 31
print("count(1):", t1.count(1))       # 2
print("index(5):", t1.index(5))       # 4
print("sorted():", sorted(t1))        # ascending list
print("t1+t2    :", t1 + t2)         # concatenation
print("t2*3     :", t2 * 3)          # repetition
print("4 in t1 :", 4 in t1)          # True
print("t1[2:5] :", t1[2:5])          # slicing
```

34. Explain tuple in brief with necessary example. [7 M]

- *Appeared in:* W'23 – Q.5(c) [OR]

Solution

Tuple: An ordered, **immutable** collection of items enclosed in parentheses `( )`.

Creating Tuples:

`t = (1, 2, 3)  single = (5,)  empty = ()`

Key Properties:

- Immutable — elements cannot be changed after creation
- Ordered — maintains insertion order
- Can contain duplicate elements
- Can hold mixed data types: `t = ("Raj", 20, 3.14, True)`

Operations:
`t = (10, 20, 30, 40, 50)`
`t[0]` = 10 (indexing) `t[1:4]` = `(20,30,40)` (slicing)
`len(t)` = 5 `20 in t` = `True` (membership)
`(1,2)+(3,4)` = `(1,2,3,4)` (concatenation)
When to use: Use tuple for data that should not change (coordinates, RGB values, database records).

35. Explain tuple Operations, Functions and Methods. [7 M]

- *Appeared in:* Su'24 – Q.5(c)

Solution

`t = (5, 10, 15, 10, 20, 10)`
Operations:

- Indexing: `t[0]`=5; `t[-1]`=10
- Slicing: `t[1:4]`=`(10,15,10)`; `t[::-1]`=`(10,20,10,15,10,5)`
- Concatenation: `t+(25,30)`=`(5,10,15,10,20,10,25,30)`
- Repetition: `(1,2)*3`=`(1,2,1,2,1,2)`
- Membership: `15 in t`=`True`
- Comparison: `(1,2)<(1,3)`=`True`
- Deletion: `del t` (delete entire tuple)

Built-in Functions: `len(t)`=6; `min(t)`=5; `max(t)`=20; `sum(t)`=70; `sorted(t)` returns sorted list.
Tuple Methods (only 2): `t.count(10)`=3; `t.index(15)`=2.

36. Explain tuple operations with examples. [7 M]

- *Appeared in:* Su'25 – Q.4(c) [OR]; W'25 – Q.4(c) [OR]

Solution

`a = (1, 2, 3); b = (4, 5)`

Operation	Code	Result
Concatenation	`a + b`	`(1,2,3,4,5)`
Repetition	`a * 2`	`(1,2,3,1,2,3)`
Indexing	`a[1]`	`2`
Neg indexing	`a[-1]`	`3`
Slicing	`a[0:2]`	`(1,2)`
Membership	`3 in a`	`True`
Length	`len(a)`	`3`
Min/Max	`min(a)`	`1`
Iteration	`for x in a`	1, 2, 3
Comparison	`a < (1,3,3)`	`True`

Tuple Packing/Unpacking:
`t = 10, 20, 30` (packing) `x, y, z = t` (unpacking)

5.8 Set — 7-mark questions

37. List set functions and operations; develop a Python program to demonstrate set functions and operations. [7 M]

- *Appeared in:* Su'23 – Q.5(c) [OR]; Su'25 – Q.5(c); W'25 – Q.4(c)

Solution

Set: Unordered, mutable collection of unique elements enclosed in `{ }`.
Operations: union (`|`), intersection (`&`), difference (`-`), symmetric difference (`^`).

```
A = {1, 2, 3, 4, 5}
```

```
B = {3, 4, 5, 6, 7}

print("A:", A); print("B:", B)
print("Union (A|B)         :", A | B)
print("Intersection (A&B) :", A & B)
print("Difference (A-B)    :", A - B)
print("Sym Diff (A^B)      :", A ^ B)

# Set methods
A.add(10)
print("After add(10):", A)
A.remove(1)
print("After remove(1):", A)
A.discard(99)           # no error if not found
print("len(A):", len(A))
print("3 in A:", 3 in A)
print("issubset:", {2,3}.issubset(A))
```

38. Develop code for two sets and perform union, intersection, difference, and symmetric difference operations. **[7 M]**

- *Appeared in:* W'24 – Q.5(c)

Solution

```
# Create two sets
set1 = {10, 20, 30, 40, 50}
set2 = {30, 40, 50, 60, 70}

print("Set 1:", set1)
print("Set 2:", set2)

print("\n--- Set Operations ---")
print("Union           :", set1 | set2)
# {10,20,30,40,50,60,70}

print("Intersection    :", set1 & set2)
# {30,40,50}

print("Difference(1-2):", set1 - set2)
# {10,20}

print("Difference(2-1):", set2 - set1)
# {60,70}

print("Symmetric Diff :", set1 ^ set2)
# {10,20,60,70}

# Using methods
print("\nUsing methods:")
print("union()     :", set1.union(set2))
print("intersect():", set1.intersection(set2))
print("difference():", set1.difference(set2))
print("sym_diff() :", set1.symmetric_difference(set2))
```

5.9 Dictionary — 7-mark questions

39. Write program to demonstrate dictionary functions and operations. **[7 M]**

- *Appeared in:* Su'24 – Q.5(c) [OR]; Su'25 – Q.5(c) [OR]

Solution

```python
# Create dictionary
student = {"name": "Raj", "age": 20,
           "marks": 85, "city": "Surat"}

print("Dictionary:", student)
print("Keys   :", list(student.keys()))
print("Values:", list(student.values()))
print("Items  :", list(student.items()))

# Access
print("\nname   :", student["name"])
print("marks  :", student.get("marks", 0))

# Update and add
student["marks"] = 90
student["course"] = "Engineering"
print("\nAfter update:", student)

# Delete
student.pop("age")
print("After pop('age'):", student)

# Iterate
print("\nAll key-value pairs:")
for key, val in student.items():
    print(f"  {key}: {val}")

print("Length:", len(student))
```

40. Explain Dictionary Built-in Functions with examples. [7 M]

- *Appeared in:* W'25 – Q.5(c)

Solution

```python
d = {"a": 1, "b": 2, "c": 3, "d": 4}
print("dict:", d)

print("keys()  :", list(d.keys()))     # ['a','b','c','d']
print("values():", list(d.values())) # [1,2,3,4]
print("items() :", list(d.items()))  # [('a',1),...]

print("get('b')     :", d.get("b"))        # 2
print("get('z',0)   :", d.get("z", 0))    # 0 (default)

# update() -- merge another dict
d.update({"e": 5, "a": 10})    # adds 'e', updates 'a'
print("after update:", d)

# pop() -- remove and return
val = d.pop("b")
print("pop('b'):", val, "| dict:", d)

# popitem() -- remove last item (Python 3.7+)
k, v = d.popitem()
print("popitem():", k, v)

# setdefault() -- get or set default
d.setdefault("f", 6)
print("setdefault('f',6):", d)
```

```python
# copy() and clear()
d2 = d.copy()
d2.clear()
print("cleared copy:", d2)           # {}
print("original:", d)               # unchanged
```