

A Text Book of

BASICS OF PYTHON PROGRAMMING FOR PHARMACEUTICAL SCIENCE

As per the New PCI Syllabus **NEP 2020**

BP101T



1st SEM.

Bachelor of Pharmacy



⊙ Dr. Shanu Kuttan Rakesh

⊙ Dr. Pratibha Singh

⊙ Dr. Rajesh Kumar Verma

⊙ Dr. Peeyush Jaiswal

BP101T



A Text Book of **BASICS OF PYTHON PROGRAMMING FOR PHARMACEUTICAL SCIENCE**

Bachelor of Pharmacy
First Semester

As per the New PCI Syllabus **NEP 2020**



Dr. Shanu Kuttan Rakesh

M.Tech, Ph.D.
Head and Associate Professor
Chouksey Engineering College,
Bilaspur, Chhattisgarh

Dr. Rajesh Kumar Verma

M.Tech, Ph.D.
Professor
CMR Engineering College,
Hyderabad, Telangana

Dr. Pratibha Singh

M.Tech, Ph.D.
Assistant Professor
Dr. C.V. Raman University,
Bilaspur, Chhattisgarh

Dr. Peeyush Jaiswal

M.Pharm, PhD.
Professor and Director
GPAT Discussion Centre Pvt. Ltd.
Bilaspur, Chhattisgarh



GDC PUBLICATION

3rd Floor, Shiv Hari Tower, Sarkanda, Bilaspur (C.G.), 495001

A Text Book of
BASICS OF PYTHON PROGRAMMING FOR PHARMACEUTICAL SCIENCES

As per the New PCI Syllabus **NEP 2020**

First Semester (Bachelor of Pharmacy)

ISBN : 978-93-49537-03-3

First Edition : 2026

Copyright ©2026, By the GDC Publication.



Head Office

3rd Floor, Shiv Hari Tower, Ashok Nagar,
Sarkanda, Bilaspur (C.G.), 495001

Phone : 6263936320, 6265846984

Email : gdcpublication037@gmail.com

Website : www.gdcclases.com

Corporate Office

Sahara Study center, Opposite CMP Degree College
George Town, Prayagraj (U.P.), 211002

Phone : 9343674693

Email : gdcgpat037@gmail.com

All rights are reserved. No part of this publication may be reproduced, stored in or introduced into a retrieval system, or transmitted in any form or by any means (electronic, mechanical, photo-copying, recording or otherwise), without the prior written permission of the above mentioned publisher of this book.

GDC PUBLICATION has taken due care in collecting the data and providing the solutions, before publishing this book. In spite of this, if any inaccuracy or printing error occurs then GDC PUBLICATION owes no responsibility. GDC PUBLICATION will be grateful if you could point out any such error. Your suggestions will be appreciated.

Any government or Private organization logo, posters, brand product name and image used in this book is for educational purposes only.

Typesetting by :

Team GDC

Composition by :

Vidyadhar Sahu and Chandranayan Sahu

Printed by :

GPAT Discussion Center Pvt. Ltd.
Bilaspur (C.G.), 495001

Price : ₹240

Jurisdiction : All disputes with respect to this publication shall be subject to the jurisdiction of the Courts, Tribunals and Forums of Bilaspur Chhattisgarh, India only.

© All rights reserved by GDC PUBLICATION. No part of this book may be reproduced or utilized in any form without the written permission from the publisher.

PREFACE

Computers and programming have become essential tools in modern pharmacy practice, education, and research. From calculating doses and recording adverse drug reactions to analyzing pharmacokinetic data and presenting results through graphs, Python offers a simple yet powerful way to support pharmacy workflows. This textbook, *Python Programming for Pharmaceutical Sciences*, is designed to introduce students and beginners to Python programming in a manner that is practical, understandable, and directly connected to pharmaceutical applications.

The primary aim of this book is to help readers learn Python step by step while developing confidence in solving pharmacy-related problems. Each chapter combines fundamental programming concepts with examples drawn from pharmacy, such as patient data handling, drug-dose calculations, file operations, data analysis, and visualization. The language has been kept simple, and the examples have been selected to support both classroom teaching and self-study.

Unlike general programming books, this text emphasizes pharmacy-oriented learning. The examples, exercises, and case studies are intended to show how Python can be used in dispensing systems, clinical documentation, pharmacovigilance, laboratory data management, and research-oriented analysis. By linking programming with real pharmaceutical contexts, the book encourages students to see Python not as an abstract subject, but as a useful professional skill.

The book is organized to gradually build knowledge from basic syntax to data handling and graphical presentation. Chapter 1 Introduces the fundamentals of Python Programming. Chapter 2 Explains decision-making and Looping structures. Chapter 3 Covers lists, dictionaries, and file handling. Chapter 4 Presents data analysis using Pandas. Chapter 5 Introduces data visualization using Matplotlib. In addition, the appendix provides case studies and practical applications to reinforce learning through hands-on examples.

Designed to meet PCI curriculum requirements, this book is for pharmacy students, teachers, researchers, and others interested in applying Python to pharmaceutical work. No prior programming experience is needed. Readers are encouraged to practice the examples and explore the exercises for a better understanding.

I hope this book serves as a useful companion in learning Python and inspires readers to apply programming creatively in pharmacy practice, education, and research.

Authors

UNIT I Introduction to Python Programming for Pharmacy **1-44**

1.1 INTRODUCTION TO PYTHON PROGRAMMING

- 1.1.1 Importance of Python in Pharmacy and Healthcare
- 1.1.2 Features of Python
- 1.1.3 Advantages of Python
- 1.1.4 Python Libraries for Scientific and Healthcare Applications
- 1.1.5 Why Pharmacy Students Should Learn Python
- 1.1.6 Example: A Simple "Hello, World!" Program

1.2 INSTALLING PYTHON AND AN INTEGRATED DEVELOPMENT ENVIRONMENTS (IDES)

- 1.2.1 Understanding Python and IDEs
- 1.2.2 Installing Python
- 1.2.3 Integrated Development Environments (IDEs)
- 1.2.4 Advantages of IDEs over Text Editors
- 1.2.5 Installing and Managing Libraries

1.3 PYTHON VARIABLES AND DATA TYPES

- 1.3.1 Variables
- 1.3.2 Basic Data Types (integers, floats, strings, booleans)

1.4 TYPE CASTING

1.5 BASIC OPERATORS

1.6 INPUT AND OUTPUT OPERATIONS

- 1.6.1 Taking Input from the User - input()
- 1.6.2 Displaying Output - print()

1.7 BASIC STRING OPERATIONS AND MANIPULATION TECHNIQUES

- 1.7.1 String Concatenation
- 1.7.2 Common String Methods
- 1.7.3 String Slicing

1.8 STANDARD LIBRARIES AND THIRD-PARTY LIBRARIES

- 1.8.1 Introduction to Standard Libraries
- 1.8.2 Installing and Uninstalling Third-Party Libraries

1.9 COMMENTS, CODE STRUCTURE, AND BASIC ERRORS

- 1.9.1 Comments in Python

- 1.9.2 Basic Code Structure and Syntax
- 1.9.3 Script Mode vs Interactive Mode
- 1.9.4 Variable Names and Naming Conventions
- 1.9.5 Simple Errors and Beginner-Level Debugging
- 1.9.6 Python File Extensions and Script Management

1.10 SUMMARY

UNIT II Control Structures and Functions in Pharmacy **45-76**

2.1 INTRODUCTION TO CONTROL STRUCTURES

2.2 CONDITIONAL STATEMENTS: if, if-else, if-elif-else, nested if conditions

- 2.2.1 Basic if Statement
- 2.2.2 if-else Statement
- 2.2.3 if-elif-else Chain
- 2.2.4 Nested if Statements

2.3 LOGICAL OPERATORS IN CONDITIONS

- 2.3.1 Using and
- 2.3.2 Using or
- 2.3.3 Using not

2.4 LOOPS (for loop and while loop)

- 2.4.1 for Loop – Definite Repetition
- 2.4.2 while Loop – Indefinite Repetition

2.5 break AND continue STATEMENTS

- 2.5.1 break – Exit the loop early
- 2.5.2 continue – Skip current iteration

2.6 FUNCTIONS IN PYTHON

- 2.6.1 Defining a Function and Calling Functions
- 2.6.2 Parameters and Arguments
- 2.6.3 Returning Values
- 2.6.4 Default and Keyword Arguments
- 2.6.5 Simple Pharmacy-Oriented Example

2.7 MODULAR PROGRAMS FOR PHARMACEUTICAL APPLICATIONS

- 2.7.1 Modular Dose-Calculator Program
- 2.7.2 BMI-Based Risk Categorization Program

2.7.3 Simple Clinical Decision-Support Script

2.8 SUMMARY

UNIT III Data Structures and File Handling for Pharmacy

77-106

3.1 LISTS, TUPLES, AND DICTIONARIES

- 3.1.1 Lists
- 3.1.2 Tuples
- 3.1.3 Dictionaries

3.2 INDEXING, SLICING, AND BASIC OPERATIONS

- 3.2.1 List Operations
- 3.2.2 Dictionary Operations
- 3.2.3 String Manipulation with Data Structures

3.3 INTRODUCTION TO NUMPY ARRAYS

- 3.3.1 Basic Operations Using NumPy: Array Creation & Arithmetic Operations

3.4 READING AND WRITING CSV FILES

- 3.4.1 Writing a Simple CSV
- 3.4.2 Reading a CSV File
- 3.4.3 Reading CSV Data as Dictionaries

3.5 UNDERSTANDING STRUCTURED HEALTHCARE DATASETS

- 3.5.1 What "Structured Data" Means
- 3.5.2 Common Pharmacy-related Dataset Schemas
- 3.5.3 Core Analytic Operations (Conceptual View)

3.6 IMPORTING SMALL PHARMACEUTICAL DATASETS AND PERFORMING BASIC DATA ACCESS AND MANIPULATION

3.7 SUMMARY

UNIT IV Data Handling With Pandas in Pharmacy

107-132

4.1 INTRODUCTION TO THE PANDAS LIBRARY

- 4.1.1 Core Data Structures in Pandas
- 4.1.2 Importing Pandas

4.2 PANDAS SERIES AND DATAFRAME STRUCTURES

- 4.2.1 Series: 1-D Labeled Data
- 4.2.2 DataFrame: 2-D Tabular Data

4.3 READING CSV AND EXCEL FILES - PK STUDY DATASETS AND ADR REPORTS

- 4.3.1 Reading a CSV File (PK Study Example)

4.3.2 Reading an Excel File (ADR Report Example)

4.4 INSPECTING DATASETS IN PANDAS SUCH AS shape, head(), tail(), info() and describe()

4.5 DATA CLEANING TECHNIQUES AND HANDLING MISSING VALUES

- 4.5.1 Detecting Missing Values
- 4.5.2 Dropping Rows with Missing Values
- 4.5.3 Filling Missing Values

4.6 FILTERING AND SELECTING DATA BASED ON CONDITIONS

- 4.6.1 Selecting Columns
- 4.6.2 Filtering Rows by Condition

4.7 GROUPING DATA AND PERFORMING AGGREGATION FUNCTIONS

- 4.7.1 Basic Grouping and Mean Calculation
- 4.7.2 Multiple Aggregations per Group
- 4.7.3 Grouping for Drug-Specific ADR Counts

4.8 SUMMARY

UNIT V Data Visualization With Matplotlib in Pharmacy

133-150

5.1 INTRODUCTION TO MATPLOTLIB

5.2 CREATING LINE PLOTS, HISTOGRAMS, SCATTER PLOTS, BAR CHARTS AND BOX PLOTS

- 5.2.1 Line plots
- 5.2.2 Histograms
- 5.2.3 Scatter plots
- 5.2.4 Bar Charts
- 5.2.5 Box plots

5.3 LABELING AXES, TITLES, AND LEGENDS

5.4 CREATE PLOTS AND VISUALIZE PHARMACEUTICAL DATASETS

- 5.4.1 Concentration-Time Curves (IV and Oral Administration)
- 5.4.2 ADR Reporting Rates Across Drugs
- 5.4.3 Dissolution Profiles (Time vs % Dissolved)

5.5 SCIENTIFIC INTERPRETATION OF PLOTS

5.6 SUMMARY

APPENDIX

151-160

Unit

I

Introduction to Python Programming for Pharmacy



Syllabus

- **Installing Python and an Integrated Development Environment (IDE) [Jupyter Notebook, PyCharm, VS Code etc.], Advantages of IDEs over text editors.**
- **Python variables and data types (integers, floats, strings, booleans), Type casting and basic operators (arithmetic, comparison, logical), Input and output operations.**
- **Basic string operations and manipulation techniques. Introduction to standard libraries and third-party libraries, installing and uninstalling libraries.**



INTRODUCTION TO PYTHON PROGRAMMING FOR PHARMACY



LEARNING OBJECTIVES

After studying this chapter, the learner will be able to:

- Understand the basics of Python programming.
- Install Python and use IDEs such as Jupyter Notebook, PyCharm, and VS Code.
- Use variables, data types, type casting, and basic operators in Python.
- Perform input and output operations.
- Apply basic string operations and manipulation techniques.
- Understand standard and third-party libraries and manage their installation.
- Write simple Python programs using proper syntax, comments, and indentation.

1.1 INTRODUCTION TO PYTHON PROGRAMMING

Python is a **high-level, interpreted** programming language known for its simplicity, readability, and versatility. It was created by Guido van Rossum and first released in 1991. Python was designed with a strong emphasis on code readability and simplicity, which is why it is considered one of the most beginner-friendly programming languages. It is widely used in scientific computing, data analysis, artificial intelligence, automation, web development, and healthcare analytics. In pharmaceutical sciences, Python helps in performing calculations, managing data, automating tasks, and visualizing results.

Python is called a **high-level language** because it is easy for humans to read and write. Its syntax is close to natural language, making it easier to understand than machine-level instructions. Python is also an **interpreted language**, which means the code is executed line by line. This allows beginners to test programs quickly and identify errors more easily. Python is suitable for pharmacy students because it can be used for simple calculations as well as advanced research applications.

One of the major reasons for Python's popularity is its **simple and clear syntax**. Programs can often be written in fewer lines compared to many other programming languages. Python also supports essential programming concepts such as variables, data types, operators, input/output, conditions, loops, and functions. These concepts form the foundation of programming and are highly useful in pharmacy-related problem solving.

1.1.1 Importance of Python in Pharmacy and Healthcare

The pharmaceutical and healthcare sectors generate large amounts of data related to patients, medications, laboratory tests, clinical trials, and drug development. Python provides efficient tools to process, analyze, and visualize such data.

Interactive Mode vs. Script Mode

Once Python is installed, programs can be executed in two different **modes**:

1. **Interactive Mode (Immediate Mode):** Commands are entered and executed one at a time, providing immediate results.
2. **Script Mode:** Python code is written in a file and executed as a complete program.

Summary: Python installation is the first step in programming. Choosing the right distribution and development environment helps students learn more effectively and work more efficiently in practical applications.

1.2.2 Installing Python

Follow these steps to install Python on your computer:

Step 1 - Download Python

1. Open a browser and go to the official Python website: <https://www.python.org>
2. Under the “**Downloads**” section, click the latest **stable Python 3.x** release for your operating system (Windows, macOS, or Linux).

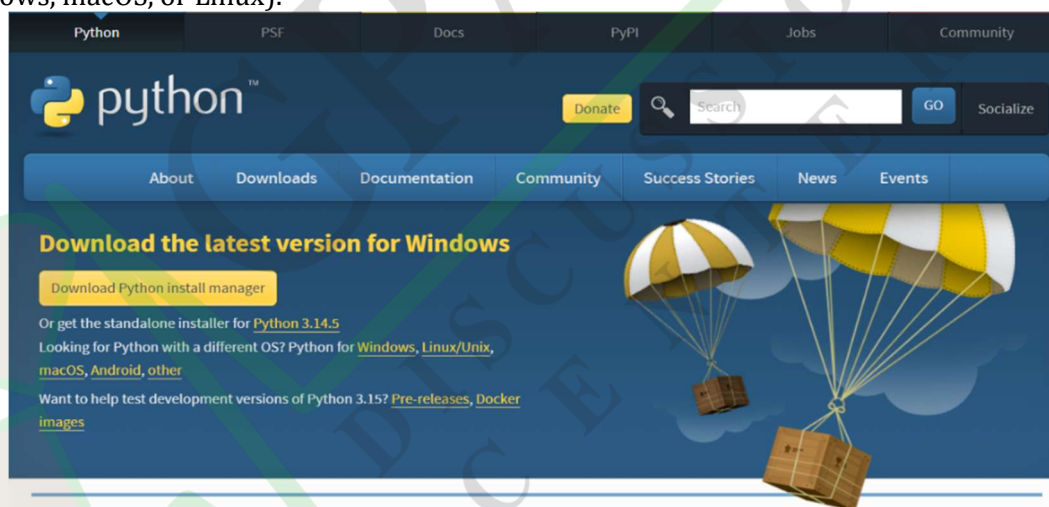


Fig 1.3: Python download page

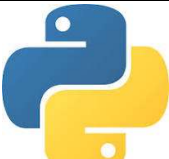
Looking for a specific release?
Python releases by version number:

Release version	Release date	Click for more	
Python 3.14.5	May 10, 2026	Download	Release notes
Python 3.14.4	April 7, 2026	Download	Release notes
Python 3.14.3	Feb. 3, 2026	Download	Release notes
Python 3.14.2	Dec. 5, 2025	Download	Release notes
Python 3.14.1	Dec. 2, 2025	Download	Release notes
Python 3.14.0	Oct. 7, 2025	Download	Release notes
Python 3.13.13	April 7, 2026	Download	Release notes
Python 3.13.12	Feb. 3, 2026	Download	Release notes

[View older releases](#)

Fig 1.4: Python release page

Table 1.2: Popular Python IDEs

IDE / Tool	Logo	Best For
Python IDLE		Simple, built-in Python IDE; ideal for beginners.
Jupyter Notebook		Interactive coding, data analysis and visualization
Google Colab		Cloud-based Python programming without installation.
VS Code		Lightweight and extensible editor for Python development; popular among students and professionals.
PyCharm		Professional IDE for larger projects and applications.
Spyder		Scientific computing, data analysis, and research.

For beginners, **IDLE** is a good starting point. For data analysis and research applications, **Jupyter Notebook**, **Google Colab**, and **Spyder** are widely used. **PyCharm** and **VS Code** are suitable for developing larger and more complex Python projects.

1.2.4 Advantages of IDEs over Text Editors

A **text editor**, such as Notepad, can be used to write Python code. However, **IDEs** and **advanced editors** such as IDLE, Spyder, PyCharm, VS Code, and Jupyter Notebook provide several additional features that make programming easier and more efficient. Table 1.3 shows Advantages of IDEs over Text Editors.

Table 1.3: Advantages of IDEs over Text Editors

Feature	Text Editor (e.g., Notepad)	Python IDE / Advanced Editor (e.g., PyCharm, VS Code, Jupyter)
Syntax highlighting	No or minimal	Yes (colors keywords, strings, comments, etc.)
Code completion	No	Yes (auto-suggests function names, parameters, and library items)

Table 1.4: Common Python Data Types

Data Type	Description	Example
int	Whole Numbers	500
float	Decimal Numbers	2.5
str	Text Data	"Amoxicillin"
bool	Logical Values (True/False)	True

These basic data types form the foundation of Python programming and are essential for storing and processing healthcare, pharmaceutical, and scientific data.

Checking Data Types with `type()`:

Python provides the built-in `type()` function to determine the data type of a variable.

Example:

Python

```
dose = 500
weight_kg = 70.2
drug_name = "Paracetamol"
is_adult = True
```

```
print(type(dose))    # <class 'int'>
print(type(weight_kg)) # <class 'float'>
print(type(drug_name)) # <class 'str'>
print(type(is_adult)) # <class 'bool'>
```

Output

```
<class 'int'>
<class 'float'>
<class 'str'>
<class 'bool'>
```

Explanation of `type()`

- The `type()` function is used to identify the data type of a variable.
- It helps verify whether a value is an `int`, `float`, `str`, or `bool`.
- This is useful for debugging and ensuring that data is stored in the correct format.

Example: Simple Pharmacy-Related Variables

Python

```
dose = 500
weight_kg = 70.2
drug_name = "Paracetamol"
is_adult = True
```

```
print("Dose:", dose)
print("Weight:", weight_kg)
print("Drug:", drug_name)
print("Adult patient:", is_adult)
```

Output

Enter patient weight (kg): 70
Total dose: 350.0 mg

Explanation:

- The `input()` function stores user input as a string.
- `float()` converts the entered weight into a numeric value.
- The total dose is calculated using the formula:
Total Dose = Weight × Dose per kg
- For a weight of **70 kg** and a dose of **5 mg/kg**:
Total Dose = 70 × 5 = 350 mg

Concepts Used: Input, type casting (`float()`), arithmetic operations, and output.

Importance of Type Casting in Pharmaceutical Applications

Type casting is essential in pharmacy and healthcare programming because:

- User input is usually received as text.
- Mathematical calculations require numeric values.
- Dose calculations often involve both integers and floating-point values.
- BMI, creatinine clearance, and PK calculations require accurate numeric conversions.

1.5 BASIC OPERATORS

Operators are symbols used to perform operations on variables and values. Python supports several types of operators that are extensively used in pharmacy-related programs and calculations.

Arithmetic Operators

Arithmetic operators are used to perform mathematical calculations.

Table 1.6: Arithmetic Operators

Operator	Name	Example
+	Addition	2 + 3 → 5
-	Subtraction	7 - 2 → 5
*	Multiplication	4 * 3 → 12
/	Division	10 / 3 → 3.333...
//	Floor division	10 // 3 → 3
%	Modulo	10 % 3 → 1

Example 1: Tablet Distribution Calculation**Python**

```
tablets = 22
days = 5
```

```
tablets_per_day = tablets // days
remaining = tablets % days
```


The letter **f** before a string indicates that variables can be inserted directly inside the string using curly braces **{}**.

Example: Using f-Strings

Python

```
drug_name = "Paracetamol"
dose_mg = 500.0
print(f'Drug: {drug_name}, Dose: {dose_mg} mg')
```

Output

```
Drug: Paracetamol, Dose: 500.0 mg
```

Explanation of f-Strings:

- The **f** in **f"..."** stands for “**formatted string**”.
- Variables placed inside **{}** are automatically replaced with their values.
- f-strings make output more readable and easier to format.
- **Without f**, the variable names would appear as **plain text**.

Without f

```
print("Drug: {drug_name}")
```

Output

```
Drug: {drug_name}
```

With f

```
print(f'Drug: {drug_name}')
```

Output

```
Drug: Paracetamol
```

Note: Python treats lowercase **f** and uppercase **F** identically in f-strings.

```
print(f'Hello')
print(F'Hello')
```

Formatting Numbers in Output:

f-Strings can also control the number of decimal places displayed.

Example: Formatting Decimal Values

Python

```
weight = 72.5
print(f'Body weight: {weight:.2f} kg') # 72.50 kg
```

Output

```
Body weight: 72.50 kg
```

Explanation:

- Changing letter case
- Removing extra spaces
- Formatting text
- Replacing characters

Since strings are immutable, these methods return new strings instead of modifying the original string.

Case-Conversion Methods:

- `.lower()` → converts all characters to lowercase
- `.upper()` → converts all characters to uppercase
- `.title()` → converts to title-case (first letter of each word capitalized)

Example: Drug Name Normalization

Python

```
drug = " Paracetamol "

drug_lower = drug.lower()      # " paracetamol "
drug_upper = drug.upper()      # " PARACETAMOL "
drug_title = drug.strip().title() # "Paracetamol"

print("Original:", repr(drug))
print("Lower:", drug_lower)
print("Upper:", drug_upper)
print("Title:", drug_title)
```

Output

```
Original: ' Paracetamol '
Lower: paracetamol
Upper: PARACETAMOL
Title: Paracetamol
```

Explanation:

- `.lower()` converts all letters to lowercase.
- `.upper()` converts all letters to uppercase.
- `.title()` capitalizes the first letter of each word.
- `.strip()` removes unwanted spaces before applying formatting.

These methods help standardize user-entered drug names such as "paracetamol," "PARACETAMOL," and "Paracetamol," before comparison or database storage.

Key Point: String methods do not modify the original string; they return a new string.

Whitespace-Trimming Method:

The `.strip()` method removes leading and trailing whitespace (spaces, tabs, newlines).

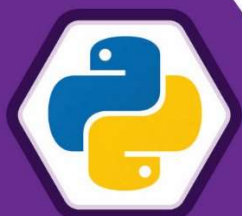
Unit II

Control Structures and Functions in Pharmacy



Syllabus

- Conditional statements (if, if-else, if-elif-else), nested conditions
- Loops (for loop, while loop).
- Break and continue statements.
- Defining and calling functions, passing arguments and returning values.
- Writing modular programs for simple pharmaceutical applications- dosage calculation and BMI calculation.



CONTROL STRUCTURES AND FUNCTIONS IN PHARMACY



LEARNING OBJECTIVES

After studying this chapter, the learner will be able to:

- Use conditional statements (if, if-else, and if-elif-else) for dose- and BMI-based decision making.
- Implement nested conditions for multi-level clinical logic.
- Use for and while loops for repetitive pharmacy-related tasks.
- Apply break and continue statements to control loop execution.
- Define and call functions, pass arguments, and return values.
- Develop modular programs for dosage calculation and BMI-based categorization.

2.1 INTRODUCTION TO CONTROL STRUCTURES

In Chapter 1, you learned how to store data, perform calculations, accept input, and display output. In this chapter, you will learn how to make programs take decisions and repeat tasks automatically. This is achieved using control structures.

A **control structure** is a programming construct that determines the order of execution of statements in a program. Control structures allow programs to make decisions, repeat actions, and execute instructions in a logical sequence.

In pharmacy-oriented programming, control structures are important because many clinical and pharmaceutical tasks depend on conditions and repeated operations.

For example:

- A warning may appear only when a calculated dose exceeds a safe limit.
- A BMI category may change depending on the patient's value.
- A report may process multiple patient records repeatedly.

Without control structures, a program would execute the same instructions every time and would not be able to respond intelligently to patient-specific data.

The two major **types of control structures** covered in this chapter are shown in Table 2.1:

Table 2.1: Types of control structure

Control Structure	Purpose
Conditional structures (if, if-else, if-elif- else)	Used for selection and decision-making
Loop structures (for, while)	Used for repetition and iteration

These structures are essential for building dose-adjustment rules, BMI classification tools, ADR risk categorization systems, and iterative pharmacokinetic (PK) calculations such as time-series and batch processing.

The `range()` function is useful for generating repeated schedules, fixed iterations, and sequential calculations.

Example 1: Print Numbers from 1 to 5

```
for number in range(1, 6):  
    print(number)
```

Output

```
1  
2  
3  
4  
5
```

Explanation:

- `range(1, 6)` generates the numbers 1, 2, 3, 4, and 5.
- The loop variable `number` takes each value one by one.
- The `print()` statement displays the current value.

Example 2: Print a Message 3 Times

```
for i in range(3):  
    print("Welcome to Python Programming")
```

Output

```
Welcome to Python Programming  
Welcome to Python Programming  
Welcome to Python Programming
```

Explanation:

- `range(3)` generates 0, 1, and 2.
- The loop executes three times.
- The same message is displayed during each iteration.

Python Oriented Example:**Example 3: Print a Dose Reminder for 5 Days****Python**

```
for day in range(1,6):          # 1 to 5  
    print(f"Day {day}: Take Paracetamol 500 mg")
```

Output

```
Day 1: Take Paracetamol 500 mg  
Day 2: Take Paracetamol 500 mg  
Day 3: Take Paracetamol 500 mg  
Day 4: Take Paracetamol 500 mg  
Day 5: Take Paracetamol 500 mg
```


Explanation:

- The loop repeats for 5 days.
- `day` stores the current day number.
- A medication reminder is printed for each day.
- The `f` before the string allows the value of `day` to be inserted into the message.

Key Point: A **for loop** is used to repeat a task a specific number of times. Here, it prints the daily dose reminder for **5 days**.

Applications in Pharmacy:

for Loops with `range()`:

- Generating dose schedules
- Processing fixed numbers of patients
- Repeating ADR screening tasks
- Simulating repeated measurements

for-loop with Lists

The **for loop** can also iterate through **lists** containing pharmacy-related data such as patient names, drug doses, or laboratory values.

Example 4: Display Drug Names**Python**

```
drugs = ["Paracetamol", "Ibuprofen", "Amoxicillin"]  
  
for drug in drugs:  
    print(drug)
```

Output

```
Paracetamol  
Ibuprofen  
Amoxicillin
```

Explanation:

- The list contains three drug names.
- The loop accesses each item one by one.
- Each drug name is displayed on the screen.

Example 5: Processing Patient Dose Lists**Python**

```
patient_names = ["Ramesh", "Sunita", "Vikram"]  
doses = [500, 750, 400]  
  
for i in range(len(patient_names)):  
    print(f"Patient: {patient_names[i]}, Dose: {doses[i]} mg")
```

Output

```
Patient: Ramesh, Dose: 500 mg
```

Here:

- `def` starts the function definition.
- `function_name` is the name of the function you choose
 - e.g. `calculate_total_daily_dose`
- `Parameters` are variables that receive input values from outside.
- `return` sends the computed value back to the caller.

Example 1: Simple Greeting Function

Python

```
def greet():  
    print("Hello, Pharmacy Student!")
```

```
#Calling a Function  
greet()
```

Output

```
Hello, Pharmacy Student!
```

Explanation:

- `greet()` is a function.
- The function prints a greeting message.
- The function runs when it is called using `greet()`.
- Functions help avoid repeating the same code.

Key Point:

A **function** is a reusable block of code that performs a specific task and can be called whenever needed.

2.6.2 Parameters and Arguments

Functions often need data to perform calculations. This data is passed using parameters and **arguments**.

- **Parameters** are variables defined inside the function.
- **Arguments** are the actual values (inputs) supplied during the function call.

Functions with One Parameters:

Example: Dose-calculator with one parameter

Python

```
def print_dose_report(dose_mg):  
    print(f"Dose: {dose_mg} mg")
```

```
# Call with argument 500  
print_dose_report(500)
```

Output

```
Dose: 500 mg
```

$$BMI = \frac{weight\ (kg)}{height^2\ (m^2)}$$

- The function `get_bmi_advice()` categorizes the BMI and provides appropriate pharmacy-oriented advice.
- The calculated BMI and the corresponding advisory message are then displayed to the user.

Key Points:

- Functions help organize calculations and decision-making into reusable blocks.
- BMI is commonly used to assess body weight status.
- The `if-elif-else` structure is used to classify BMI into different categories.
- Formatted output (`{bmi:.2f}`) displays the BMI value rounded to two decimal places.
- Different BMI categories may require different clinical considerations and dosing strategies.

Pharmacy Relevance:

BMI is an important parameter in healthcare and pharmacy practice. It can influence medication dosing, nutritional assessment, risk evaluation, and patient monitoring. Programs such as this can serve as the foundation for more advanced clinical decision-support systems.

2.7.3 Simple Clinical Decision-Support Script

A more advanced example that combines **loops**, **functions**, and **decision-logic** to simulate a small clinical-support script.

Python

```
def recommend_monitoring(bmi):
    if bmi >= 30.0:
        return "Monitor for dose-related toxicity and organ function."
    else:
        return "Routine monitoring sufficient."

def print_dose_plan(dose, days):
    for day in range(1, days + 1):
        print(f"Day {day}: Take {dose} mg")

def calculate_bmi(weight_kg, height_m):
    return weight_kg / (height_m ** 2)

def get_dose_details(drug_name):
    dose_db = {
        "paracetamol": 10.0,
        "ibuprofen": 5.0,
        "amoxicillin": 15.0,
    }
    return dose_db.get(drug_name.lower(), None)

print("=== Simple Clinical Decision Script ===")
drug = input("Enter drug name: ")
wt = float(input("Enter weight (kg): "))
ht = float(input("Enter height (m): "))
```

- (b) It is used only in loops
 (c) It defines code blocks
 (d) It is ignored by Python

24. What is the output?

```
x = 4
```

```
if x % 2 == 0:
```

```
    print("Even")
```

- (a) Odd (b) Even
 (c) 4 (d) Error

25. Which of the following creates an infinite loop?

- (a) while False: (b) while True:
 (c) for i in range(0): (d) if True:

26. What is the purpose of range(5)?

- (a) Generates numbers 1 to 5
 (b) Generates numbers 0 to 5
 (c) Generates numbers 0 to 4
 (d) Generates numbers 1 to 4

27. Which keyword is used to exit a function and optionally return a value?

- (a) stop (b) break
 (c) exit (d) return

28. What will be the output?

```
for i in range(1, 3):
```

```
    print(i)
```

- (a) 0 1 2 (b) 1 2
 (c) 1 2 3 (d) 0 1

29. Which of the following is a valid function definition?

- (a) def add(a, b): (b) function add(a, b):
 (c) add(a, b) def: (d) define add(a, b):

30. What is the output?

```
def square(x):
```

```
    return x * x
```

```
print(square(4))
```

- (a) 4 (b) 8
 (c) 16 (d) Error

ANSWER KEY

1-d	2-c	3-c	4-b	5-b	6-a	7-b	8-c	9-b	10-a
11-c	12-b	13-c	14-b	15-c	16-b	17-c	18-c	19-b	20-a
21-c	22-c	23-c	24-b	25-b	26-c	27-d	28-b	29-a	30-c

Short Answer Questions

- What are control structures in Python?
- Differentiate between if and if-else statements.
- Explain the purpose of nested if statements.
- What are logical operators? Give examples.
- Differentiate between for loop and while loop.
- Explain the use of break statement.
- Explain the use of continue statement.
- What is a function in Python?
- Differentiate between parameters and arguments.
- What is modular programming?

Long Answer Questions

- Explain conditional statements in Python with pharmacy-oriented examples.
- Discuss for and while loops with suitable examples.
- Explain break and continue statements with illustrations.
- Describe functions, parameters, arguments, and return values in Python.
- Explain the importance of modular programming in pharmaceutical applications.

Unit III

Data Structures and File Handling for Pharmacy



Syllabus

- Lists, tuples, and dictionaries.
- Indexing and slicing lists, basic operations on lists and dictionaries, string manipulation techniques.
- Introduction to NumPy arrays, basic operations using NumPy (array creation, arithmetic operations).
- Reading and writing CSV files.
- Understanding structured healthcare datasets.
- Importing small pharmaceutical datasets and performing basic data access and manipulation tasks.



DATA STRUCTURES AND FILE HANDLING FOR PHARMACY



LEARNING OBJECTIVES

After studying this chapter, the learner will be able to:

- Use lists, tuples, and dictionaries to store and organize pharmaceutical and healthcare data.
- Perform indexing, slicing, and basic operations on lists and dictionaries.
- Apply basic string manipulation techniques in Python programs.
- Create and manipulate NumPy arrays for numerical healthcare and pharmaceutical data.
- Perform basic NumPy operations such as array creation and arithmetic operations.
- Read from and write data to CSV files.
- Understand structured healthcare datasets and their organization.
- Import small pharmaceutical datasets and perform basic data access and manipulation tasks.

3.1 LISTS, TUPLES, AND DICTIONARIES

In Python, **data structures** are used to store and organize collections of data efficiently. In pharmaceutical and healthcare applications, they help manage patient details, drug information, dose schedules, laboratory values, and pharmacokinetic (PK) datasets.

Choosing the correct data structure is important because different structures are designed for different purposes. Some allow modification of data, while others are intended for fixed or labeled information.

The most commonly used built-in Python **data structures** are:

- **Lists** → ordered and mutable collections
- **Tuples** → ordered and immutable collections
- **Dictionaries** → key-value based collections

These structures form the foundation for handling healthcare datasets before moving to advanced tools such as NumPy arrays and pandas DataFrames.

3.1.1 Lists

A **list** is an **ordered** and **mutable** collection of items, enclosed in square brackets `[]`.

- **Ordered** means elements maintain their position.
- **Mutable** means elements can be added, removed, or modified after creation.

Lists are widely used in pharmacy-oriented programming because patient data, drug records, and laboratory values often change during processing.

3.1.3 Dictionaries

A **dictionary** stores data as **key-value pairs** inside curly braces {}.

- Each **key** is unique.
- Each **key** is associated with a corresponding **value**.
- **Values** can be accessed using their keys rather than numeric positions.

Dictionaries are extremely useful in healthcare and pharmacy programming because clinical information is naturally organized using labels such as patient ID, age, drug name, dose, and laboratory values.

Example: Creating a Dictionary

Python

```
drug_dict = {  
    "name": "Amoxicillin",  
    "dose_per_kg": 15.0,  
    "route": "oral",  
    "max_daily": 1000.0  
}  
  
print("Drug name:", drug_dict["name"])  
print("Dose per kg:", drug_dict["dose_per_kg"])  
print("Max daily dose:", drug_dict["max_daily"])
```

Output

```
Drug name: Amoxicillin  
Dose per kg: 15.0  
Max daily dose: 1000.0
```

Explanation:

The dictionary `drug_dict` stores drug information using meaningful labels (keys). The values are retrieved by specifying the corresponding key inside square brackets.

Key Point: Dictionaries provide a convenient way to store and access labeled information.

Adding and Updating Dictionary Values:

Python

```
drug_dict["frequency"] = "twice daily" # add  
drug_dict["dose_per_kg"] = 20.0        # update  
  
print("Updated drug dict:", drug_dict)
```

Output

```
Updated drug dict: {  
'name': 'Amoxicillin',  
'dose_per_kg': 20.0,  
'route': 'oral',  
'max_daily': 1000.0,  
'frequency': 'twice daily'}
```

Chapter Insight: Lists, tuples, and dictionaries are the most commonly used Python data structures for storing and managing healthcare and pharmacy data.

3.2 INDEXING, SLICING, AND BASIC OPERATIONS

After creating data structures such as lists and dictionaries, the next step is learning how to access, modify, and manipulate stored data. Python provides **indexing**, **slicing**, and built-in operations that make data handling simple and efficient.

These operations are especially useful in pharmacy-oriented programming for processing patient records, laboratory values, dose schedules, and drug-related datasets.

3.2.1 List Operations

Lists are dynamic data structures, meaning elements can be added, removed, updated, or reordered after creation. Python provides several built-in methods for manipulating lists efficiently.

Basic List Operations:

Python provides several built-in methods for working with lists. Common operations include:

- **adding elements** (append(), extend())
- **removing elements** (pop(), remove())
- **modifying elements** using indexing
- **sorting elements** (sort())
- **finding the number of elements** (len())

These operations allow lists to be updated and managed efficiently during program execution.

Example: Basic List Operations

Python

```
# Create a list of lab values
lab_values = [1.2, 1.5, 1.3, 1.4]

print("Original lab values:", lab_values)

# Append a new value
lab_values.append(1.6)
print("After append:", lab_values)

# Extend with multiple values
lab_values.extend([1.7, 1.8])
print("After extend:", lab_values)

# Remove last element
lab_values.pop()
print("After pop:", lab_values)

# Sort in ascending order
lab_values.sort()
print("Sorted lab values:", lab_values)
```

After cleaning, all entries can be converted to a consistent format such as:

- "Paracetamol"

Key Point: Data cleaning is an important preprocessing step before performing analysis, visualization, or clinical decision support tasks.

3.3 INTRODUCTION TO NUMPY ARRAYS

NumPy (Numerical Python) is a powerful Python library used for **numerical computing and array-based operations**. It provides efficient data structures called **arrays**, which is optimized for mathematical and scientific calculations. Compared to Python lists, **NumPy arrays** are faster, more memory-efficient and better suited for large-scale numerical processing.

A **NumPy array** is a collection of elements if the **same data type** stored in a single variable. It is designed to support mathematical operations on entire datasets without the need of explicit loops.

In pharmaceutical and healthcare applications, NumPy is widely used for:

- Dose and concentration calculations
- Pharmacokinetic (PK) time-concentration analysis
- Processing laboratory and clinical datasets
- Performing vectorized calculations on multiple patients or samples simultaneously

A key feature of NumPy is **vectorization**, where mathematical operations are applied directly to entire arrays at once. This improves performance and makes code concise and readable.

Basic NumPy Usage:

Python

```
import numpy as np
```

```
doses = np.array([500, 650, 400, 550])  
print("Doses:", doses)
```

Output

```
Doses: [500 650 400 550]
```

Explanation:

- `import numpy as np` imports the NumPy library and assigns it the alias `np`.
- `np.array()` creates a NumPy array.
- The array stores multiple numerical values in a compact and efficient format.
- The `print()` function displays the contents of the array.

Key Point: NumPy arrays provide faster and more efficient numerical processing than standard Python lists.

Vectorized Dose Calculation:

Python

```
import numpy as np
```

Key Point:

Even when working with simple CSV files, lists, dictionaries, and Python loops, it is useful to think in terms of **rows, columns, filtering, grouping, and aggregation**. These concepts form the foundation for advanced data analysis using tools such as **pandas** and other healthcare analytics libraries.

Pharmacy Significance:

These operations are commonly used in pharmacy and healthcare to analyze patient records, summarize laboratory results, monitor adverse drug reactions, evaluate treatment outcomes, and support clinical decision-making.

3.6 IMPORTING SMALL PHARMACEUTICAL DATASETS AND PERFORMING BASIC DATA ACCESS AND MANIPULATION

Small datasets are often the starting point for healthcare data analysis. Before performing calculations, visualization, or advanced analytics, healthcare professionals must be able to import, access, and manipulate data efficiently. Python provides simple tools that make these tasks straightforward and practical.

In pharmacy practice, clinical research, and healthcare analytics, data is often stored in small tabular formats such as **CSV (Comma-Separated Values) files**. These datasets may contain patient details, laboratory values, drug administration records, adverse drug reaction (ADR) reports, or dosage information.

Importing such datasets into Python is an important first step in organizing, examining, and analyzing healthcare information in a structured manner.

Small pharmaceutical datasets can be imported using:

- Python file handling
- The built-in csv module
- Lists and dictionaries for in-memory data storage

Once imported, the data can be accessed, filtered, updated, and summarized using basic Python operations.

Basic Data Access:

Data access refers to retrieving information from a dataset without modifying its contents. For example, a pharmacist may:

- View the names of drugs in a record
- Identify patients with severe ADRs
- Retrieve laboratory test results
- Examine dosage information

Basic Data Manipulation:

Data manipulation involves modifying data or creating useful summaries. Common manipulation tasks include:

- Updating existing values
- Filtering records based on conditions

7. Which function or method is used to write rows into a CSV file in Python?
 (a) csv.reader() (b) csv.writerows()
 (c) print_csv() (d) open_csv()
8. In a CSV file used for lab results, which of the following is most likely to be a column (field) name?
 (a) [1, 2, 3, 4] (b) "patient_id"
 (c) "Paracetamol" (d) 500
9. What is the primary advantage of using structured datasets (CSV, Excel) in pharmacy?
 (a) They are always encrypted
 (b) They allow easy sharing and automation of analysis
 (c) They cannot be edited
 (d) They are limited to text only
10. Which Python code correctly opens a CSV file for reading using the csv module?
 (a) open("file.csv", "read")
 (b) open("file.csv", "r")
 (c) open("file.csv", "rb")
 (d) open(file.csv, "r")
11. Suppose you have a list of lab values: lab_results = [1.2, 1.5, 1.3]. Which method will add a new value 1.4 to the end of this list?
 (a) lab_results.add(1.4)
 (b) lab_results.append(1.4)
 (c) lab_results.extend(1.4)
 (d) lab_results.insert(1.4)
12. Which of the following is a valid dictionary key in Python?
 (a) a list such as ["Paracetamol", "Ibuprofen"]
 (b) a string such as "drug_name"
 (c) a dictionary such as {"name": "Paracetamol"}
 (d) a function call such as print("drug")
13. What does lab_results[1:3] return for lab_results = [1.2, 1.5, 1.3, 1.8]?
 (a) [1.2, 1.5] (b) [1.5, 1.3]
 (c) [1.3, 1.8] (d) [1.2, 1.5, 1.3]
14. Which import statement is needed to work with NumPy arrays in Python?
 (a) import csv as np
 (b) import numpy
 (c) import math as np
 (d) import array
15. Which data structure stores key-value pairs in Python?
 (a) List (b) Tuple
 (c) Dictionary (d) String
16. What is the output of the following code?
 numbers = [10, 20, 30]
 print(len(numbers))
 (a) 2 (b) 3
 (c) 10 (d) Error
17. Which method removes an item from a list by its value?
 (a) delete() (b) remove()
 (c) pop() (d) clear()
18. What is the index of the first element in a Python list?
 (a) 1
 (b) -1
 (c) 0
 (d) Depends on the list
19. Which of the following creates a tuple?
 (a) [1, 2, 3] (b) {1, 2, 3}
 (c) (1, 2, 3) (d) <1, 2, 3>
20. Which method returns all keys of a dictionary?
 (a) keys() (b) values()
 (c) items() (d) getkeys()
21. What is the output of the following code?
 drug = {"name": "Paracetamol"}
 print(drug["name"])
 (a) drug (b) name
 (c) Paracetamol (d) Error
22. Which NumPy function is used to create an array?
 (a) np.list() (b) np.array()
 (c) np.makearray() (d) np.create()
23. What is the output of the following code?
 import numpy as np
 arr = np.array([1, 2, 3])
 print(arr + 1)
 (a) [1, 2, 3, 1] (b) [2, 3, 4]
 (c) [1, 2, 3] (d) Error
24. Which mode is commonly used to write data to a CSV file?
 (a) "r" (b) "rb"
 (c) "w" (d) "read"
25. Which module is required for CSV file handling in Python?
 (a) numpy (b) csv
 (c) math (d) os
26. What is the result of the following code?
 data = [100, 200, 300]
 print(data[-1])
 (a) 100 (b) 200
 (c) 300 (d) Error
27. Which method is used to add a new key-value pair to a dictionary?
 (a) append()
 (b) add()

Unit IV

Data Handling with Pandas in Pharmacy



Syllabus

- Introduction to Pandas library.
- Pandas Series and DataFrame structures.
- Reading CSV and Excel files-PK study datasets and ADR reports
- Inspecting datasets using functions such as head(), tail(), info(), and describe().
- Data cleaning techniques and handling missing values.
- Filtering and selecting data based on conditions.
- Grouping data and performing aggregation functions.



DATA HANDLING WITH PANDAS IN PHARMACY



LEARNING OBJECTIVES

After studying this chapter, the learner will be able to:

- Understand the role of the Pandas library in healthcare and pharmaceutical data analysis.
- Create and use Pandas Series and DataFrame structures.
- Read CSV and Excel files containing PK study datasets and ADR reports.
- Inspect datasets using functions such as `head()`, `tail()`, `info()`, and `describe()`.
- Apply basic data cleaning techniques and handle missing values.
- Filter and select data based on specific conditions.
- Group data and perform aggregation functions such as mean, count, minimum, and maximum.

4.1 INTRODUCTION TO THE PANDAS LIBRARY

Pandas is a powerful Python library used for **data manipulation and analysis**, especially for **tabular data** organized in **rows and columns**. It provides fast, flexible, and easy-to-use data structures that are ideal for handling real-world datasets.

In pharmaceutical sciences, Pandas is widely used because most healthcare data is naturally stored in structured tabular formats such as **Excel sheets, CSV files, laboratory reports, clinical trial datasets, and electronic health records (EHR)**. It enables efficient loading, cleaning, filtering, and analysis of such data, making it a core tool for pharmaceutical data processing and healthcare analytics.

Pharmaceutical Applications of Pandas:

Pandas is commonly used for:

- Loading and analyzing Pharmacokinetic (PK) datasets
- Processing Adverse Drug Reaction (ADR) reports
- Managing clinical trial datasets
- Handling laboratory reports and patient records
- Analyzing drug inventory and prescription data
- Preparing healthcare datasets for statistical analysis and visualization

4.1.1 Core Data Structures in Pandas

Pandas provides two fundamental data structures:

1. Series
2. DataFrame

Explanation:

- The dictionary `data` stores patient information in separate lists.
- `pd.DataFrame(data)` converts the dictionary into a **DataFrame**.
- Each dictionary **key** becomes a **column** name.
- Each **row** represents a patient **record**.

Pharmaceutical Interpretation:

DataFrames are widely used for:

- Prescription records
- ADR monitoring tables
- Clinical trial datasets
- Pharmacokinetic data tables
- Hospital and patient management systems

Key Point:

Pandas provides powerful data structures—**Series and DataFrame**—that allow efficient handling of structured pharmaceutical data, enabling easy analysis, filtering, and interpretation of clinical and research datasets.

4.3 READING CSV AND EXCEL FILES – PK STUDY DATASETS AND ADR REPORTS

In pharmaceutical research and healthcare settings, large amounts of data are stored electronically for analysis and reporting. Common examples include pharmacokinetic (PK) study data, adverse drug reaction (ADR) reports, laboratory results, patient records, and clinical trial datasets.

One of the most widely used file formats for storing such data is the **CSV (Comma-Separated Values)** file. CSV files are simple text files in which values are separated by commas, making them easy to create, share, and analyze.

The Pandas library provides powerful functions for reading and processing CSV and Excel files. Once loaded into a DataFrame, the data can be inspected, cleaned, analyzed, and visualized efficiently.

4.3.1 Reading a CSV File (PK Study Example)

Pharmacokinetic (PK) and clinical datasets are commonly stored in CSV (Comma-Separated Values) files. Pandas provides a simple function `read_csv()` to load such data into a DataFrame for analysis.

Example PK dataset: `pk_data.csv`

csv file

```
patient_id,time_h,concentration_mg_L
P001,0,0.0
P001,1,5.2
P001,2,4.1
P001,3,2.8
P002,0,0.0
P002,1,6.0
P002,2,4.5
```

Output

Aggregation per patient:

	count	mean	min	max
patient_id				
P001	4	3.025	0.0	5.2
P002	4	3.425	0.0	6.0

Explanation:

- `groupby("patient_id")` forms separate groups for each patient.
- `agg()` applies multiple aggregation functions simultaneously.
- `count` → Number of observations
- `mean` → Average value
- `min` → Lowest recorded value
- `max` → Highest recorded value

Pharmacy Relevance:

- Evaluating variability in drug response
- Identify high or low drug exposure
- Summarize concentration statistics efficiently

4.7.3 Grouping for Drug-Specific ADR Counts

Grouping is also useful for counting ADR events associated with different drugs, in an ADR (Adverse Drug Reaction) dataset.

Python

Example ADR dataset

```
data_adr = {
    "drug": ["Amoxicillin", "Lisinopril", "Amoxicillin", "Metformin"],
    "reaction": ["Rash", "Cough", "Diarrhea", "Hypoglycemia"],
    "severity": ["mild", "moderate", "moderate", "severe"]
}
df_adr = pd.DataFrame(data_adr)
```

```
print("\n ADR count per drug:")
adr_count = df_adr.groupby("drug")["reaction"].count()
print(adr_count)
```

Output

ADR count per drug:

drug	
Amoxicillin	2
Lisinopril	1
Metformin	1

Name: reaction, dtype: int64

Explanation:

- `groupby("drug")` groups ADR records (rows) by drug name.
- `count()` counts the number of ADR records for each drug.
- Each value represents how frequently a drug is associated with reported reactions.

Unit V

Data Visualization with Matplotlib in Pharmacy



Syllabus

- Introduction to Matplotlib.
- Creating line plots, histograms, scatter plots, and box plots.
- Labeling axes, titles, and legends.
- Create plots and visualize pharmaceutical datasets - concentration-time curves for oral and IV administration, ADR reporting rates across drugs, dissolution profiles.
- Scientific interpretation of plots.



DATA VISUALIZATION WITH MATPLOTLIB IN PHARMACY



LEARNING OBJECTIVES

After studying this chapter, the learner will be able to:

- Understand the role of Matplotlib in pharmaceutical data visualization.
- Create line plots, histograms, scatter plots, bar plots and box plots using simple generic examples.
- Use titles, labels, and legends to improve plot readability.
- Visualize pharmaceutical datasets such as concentration–time curves, ADR reporting rates, and dissolution profiles.
- Interpret graphical outputs scientifically for healthcare and pharmaceutical data analysis.

5.1 INTRODUCTION TO MATPLOTLIB

Matplotlib is a powerful Python library used for creating graphs and visual representations of data. It is the most widely used 2D plotting library in Python and is extensively applied in scientific research, healthcare analytics, and pharmaceutical data analysis.

In pharmaceutical sciences, visualization plays an important role in understanding patterns, trends, and relationships in datasets. Matplotlib helps transform numerical healthcare data into meaningful graphical representations that are easier to interpret and communicate.

Matplotlib is commonly used in pharmacy for:

- Plotting pharmacokinetic (PK) concentration–time curves.
- Visualizing dose distributions and laboratory values.
- Comparing ADR counts and medication usage.
- Presenting clinical and research findings in reports and publications.
- Analyzing trends in patient monitoring datasets.

Some of the most commonly used plots in pharmacy and healthcare include:

- Line plots
- Histograms
- Scatter plots
- Bar charts
- Box plots

Before creating plots, **Matplotlib** is usually **imported** as follows:

Python

```
import matplotlib.pyplot as plt
import numpy as np
import pandas as pd
```

- Frequency distribution
- Data spread
- Skewness
- Outliers

Example: Histogram

Python

```
import matplotlib.pyplot as plt
import numpy as np

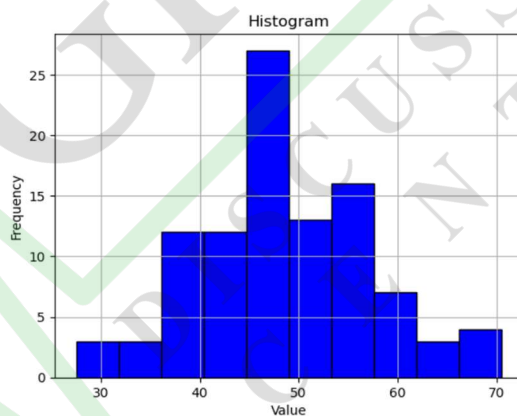
data = np.random.normal(50, 10, 100)

plt.hist(data, bins=10, color="blue", edgecolor="black")

plt.title("Histogram")
plt.xlabel("Value")
plt.ylabel("Frequency")

plt.grid(True)
plt.show()
```

Output



Explanation of Code:

- `np.random.normal(50, 10, 100)` generates 100 random values following a normal distribution with mean 50 and standard deviation 10.
- `plt.hist()` creates the **histogram** from the data.
- `bins=10` divides the data into 10 equal intervals (classes).
- `edgecolor="black"` adds borders to bars for better visibility.
- `plt.title()` sets the graph title.
- `plt.xlabel()` and `plt.ylabel()` label the axes.
- `plt.grid(True)` adds grid lines for easier interpretation.
- `plt.show()` displays the plot.

Explanation of Code:

- `plt.plot()` is used twice to create two different data series.
- `marker="o"` and `marker="s"` differentiate the data points visually.
- `label=` assigns names to each series for the legend.
- `plt.title()` adds the main heading of the plot.
- `plt.xlabel()` and `plt.ylabel()` define axis labels for interpretation.
- `plt.legend()` displays the legend box identifying each series.
- `plt.grid(True)` adds grid lines for better visual clarity.
- `plt.show()` renders the final graph.

Pharmacy Relevance:

Well-labeled plots are important for:

- Clinical reports
- PK profile interpretation
- Research presentations
- Scientific publications
- Healthcare dashboards

Key Point:

Data visualization converts pharmaceutical and healthcare datasets into graphical forms that are easier to interpret, compare, and analyze. Matplotlib provides powerful tools for creating professional scientific plots used in pharmacy education, clinical research, and healthcare analytics.

5.4 CREATE PLOTS AND VISUALIZE PHARMACEUTICAL DATASETS

In pharmaceutical sciences, data visualization helps transform numerical data into meaningful graphical representations that are easier to understand and interpret. Matplotlib can be used to visualize important pharmaceutical datasets such as PK concentration–time curves, ADR reporting rates, and dissolution profiles.

This section applies basic plot types such as line plots, bar charts, scatter plots, histograms, and box plots to real-world pharmaceutical and healthcare data. It demonstrates how pharmaceutical datasets can be created or loaded in Python and converted into clear, labeled visualizations suitable for laboratory reports, research analysis, presentations, and clinical interpretation.

In this section, the following pharmaceutical visualizations are demonstrated:

- PK concentration–time curves for IV and oral administration
- ADR reporting rates across drugs
- Dissolution profiles (time vs % dissolved)

These plots are widely used in pharmacokinetics, pharmacovigilance, quality control, and clinical research.

5.4.1 Concentration–Time Curves (IV and Oral Administration)

Line plots are widely used to represent **pharmacokinetic (PK) concentration–time curves** because they clearly show how drug concentration changes over time. These plots help visualize important pharmacokinetic processes such as absorption, distribution, metabolism, and elimination of a drug in the body.

About the Authors:

A Text Book of BASICS OF PYTHON PROGRAMMING FOR PHARMACEUTICAL SCIENCES

As per the New PCI Syllabus NEP 2020 **BP101T**

Dr. Shanu Kuttan Rakesh is an academic leader, researcher, and educator with 18+ years of experience in teaching, research, and administration. She holds a B.E., M.Tech., and Ph.D. in Computer Science & Engineering, along with certifications in Python, Machine Learning, and Deep Learning. Her research focuses on AI, Machine Learning, Deep Learning, Swarm Intelligence, and Swarm Robotics. She is currently serving as Head and Associate Professor in the Department of Computer Science and Engineering at Chouksey Engineering College, Bilaspur (C.G.).

She has published 70+ papers in IEEE, Scopus, SCI, and WoS-indexed venues, holds six patents, and authored a book and a book chapter. An IEEE reviewer and award-winning educator, she runs the YouTube channel 'Shanu Kuttan CSE Classes,' with 67,000+ subscribers and 1 million+ views.



Dr. Rajesh Kumar Verma is a distinguished academician and industry expert, currently serving as Professor in the Department of Computer Science and Engineering at CMR Engineering College, Hyderabad, Telangana. With over 30 years of experience, he has worked with leading organizations such as InterGraph India, TCS, Infosys, and Codebees, and served as Professor of Practice at VNR VJIT.

He holds a B.Tech. from NERIST, an M.Tech. in Artificial Intelligence and Robotics from the University of Hyderabad, and a Ph.D. in CSE from BPUT, Odisha. His expertise includes AI, Big Data, Cloud Computing, and Data Science, and he has delivered numerous lectures, workshops, and FDPs nationwide.



Dr. Pratibha Singh is an accomplished academician, researcher, and computer science professional, currently serving as Assistant Professor in the Department of Computer Science at Dr. C. V. Raman University, Bilaspur, Chhattisgarh. She holds a Ph.D. from Guru Ghasidas Vishwavidyalaya, an M.Tech. from RITEE, Raipur, and a B.Tech. from Government Engineering College, Raipur.

Her research expertise includes Computer Vision, Artificial Intelligence, and Deep Learning. She has published multiple papers in SCI-indexed journals and international conferences, holds two Indian patents and one international design patent, and serves as an editor and reviewer for Springer journals and IEEE conferences.



Dr. Peeyush Jaiswal, Professor and Director at GPAT Discussion Centre Pvt. Ltd., Bilaspur, Chhattisgarh, is a Pharma training professional with extensive leadership experience in various competitive exam preparation. He had qualified GATE, GPAT 4 times. He has more than 20 years of experience in shaping the Pharma aspirants for their bright future, around 25,000+ students qualified GPAT, NIPER, Drug Inspector, Pharmacist exams under his guidance.

He is recipient of Master program fellowship by the MHRD, Govt. of India in 2009. He has delivered more than 1000 Seminars in different Colleges across India, has authored 75 books, 3 book chapters and has to his credit 40 publications in different National and International Journals.



GPAT DISCUSSION CENTER

Head Office : 3rd Floor, Shiv Hari Tower,
Ashok Nagar, Sarkanda, Bilaspur (C.G.)

Phone : 6263936320, 6265846984, 9343674693

E-mail : gdcpublication037@gmail.com

Website : www.gdcclases.com



Scan the
QR Code and
get the
Book catalogue

ISBN : 978-93-49537-03-3



Price : ₹ 240.00