A TEXT BOOK

PHARMACY INDIA PUBLICATION

BASICS OF PYTHON PROGRAMMING FOR PHARMACEUTICAL SCIENCES

BP101T

As per NEP 2020

Bachelor of Pharmacy

1st Semester

Author

DR. LATIKA SHARMA, DR. ANUJ PATHAK
DR. PEEYUSH JAISWAL, MR. SHUBHAM KUMAR

FULLY COLOURED BOOK

A TEXT BOOK

BASICS OF PYTHON PROGRAMMING FOR PHARMACEUTICAL SCIENCES

BP101T

As per NEP 2020

Bachelor of Pharmacy 1st Semester

PHARMACY INDIA PUBLICATION

PUBLISHED BY
PHARMACY INDIA PUBLICATION

PHARMACY INDIA PUBLICATION

ISBN: 978-81-689235-2-2

1st Edition- 2026

PUBLISHED BY
PHARMACY INDIA PUBLICATION
STREET-4, DAYALPURAM, KHATAULI
MUZAFFARNAGAR, (U.P) 251201

Price:- **Rs. 325/-**

PHARMACY INDIA
STREET-4, DAYALPURAM, KHATAULI MUZAFFARNAGAR, 251201
8171313561, 8006781759 **pharmacyindia24@gmail.com**
pharmacyindia.co.in

PREFACE

The rapid advancement of digital technology, data science and computational methods has transformed pharmaceutical education, research and professional practice. Python, because of its simple syntax, flexibility and extensive scientific libraries, has become an important programming language for students and professionals in pharmaceutical and healthcare sciences. The book Basics of Python Programming for Pharmaceutical Sciences has been developed for first-semester B.Pharm students in accordance with the Pharmacy Council of India curriculum under the National Education Policy 2020.

The book introduces the fundamentals of Python programming in a simple, systematic and application-oriented manner. It covers the installation of Python and commonly used Integrated Development Environments such as Jupyter Notebook, PyCharm and Visual Studio Code, along with variables, data types, type casting, operators, input-output operations, string manipulation, libraries, conditional statements, loops, functions and modular programming. Pharmaceutical science examples such as dose calculation, BMI calculation and healthcare-related computational problems are included to connect programming concepts with practical applications.

The book also explains lists, tuples, dictionaries, NumPy arrays, CSV files, structured healthcare datasets and data handling using Pandas. Students are guided through importing and analysing pharmacokinetic datasets, adverse drug reaction reports and dissolution data. Data cleaning, handling missing values, filtering, grouping and aggregation are presented in a clear and practical manner. Matplotlib is introduced for creating and interpreting line plots, histograms, scatter plots and box plots related to concentration-time curves, ADR reporting rates and dissolution profiles.

A distinctive feature of this book is the inclusion of real coding interfaces, pharmaceutical science examples, explanatory tables, graphs, flowcharts, illustrations and coding screenshots. These visual and practical elements make programming concepts easier to understand and encourage computational thinking, data-handling ability and scientific interpretation. It is hoped that this book will provide students with a strong foundation for advanced learning in pharmaceutical data analysis, pharmacokinetics, bioinformatics, artificial intelligence, machine learning and healthcare informatics.

Authors

BASICS OF PYTHON PROGRAMMING FOR PHARMACEUTICAL SCIENCES SYLLABUS

Introduction to Python programming — HOURS: 06

- 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.

Control Structures & Functions — HOURS: 06

- 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.

Data Structures & File Handling — HOURS: 06

- 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 Handling with Pandas — HOURS: 06

- 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 Visualization with Matplotlib — HOURS: 06

- 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.

INDEX

INDEX

1 INTRODUCTION TO PYTHON PROGRAMMING

- 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.

1.1 Installing Python and an Integrated Development Environment

1.1.1 Introduction

Python is one of the most popular and widely used programming languages in the world. It is an interpreted, high-level, object-oriented, and general-purpose programming language that emphasizes readability and simplicity. Python was developed by Guido van Rossum and first released in 1991. Since then, it has become the preferred language for beginners as well as professionals working in fields such as artificial intelligence, data science, scientific computing, cybersecurity, web development, automation, robotics, cloud computing, and software engineering.

Python is one of the most widely used high-level programming languages and is available for all major operating systems, including Windows, macOS, and Linux. It is an interpreted language, meaning that Python code is executed directly by the Python interpreter as soon as the user runs the program. Unlike compiled languages such as C or C++, which require the source code to be translated into machine language before execution, Python allows immediate execution of code. This feature enables rapid development, quick testing, and easier debugging, making Python particularly suitable for scientific computing, data analysis, automation, and educational purposes. Its simple syntax and readability also make it an ideal programming language for beginners while remaining powerful enough for professional software development.

Python is free and open-source software maintained by the Python Software Foundation (PSF), a non-profit organization dedicated to its continuous development. Because Python is freely available, users can download, install, modify, and distribute it without licensing costs. Its open-source nature allows developers worldwide to inspect and improve the source code, ensuring transparency, reliability, and continuous innovation. Another significant advantage of Python is its extensive ecosystem of libraries and frameworks, which provide ready-to-use tools for numerical computation, data analysis, visualization, artificial intelligence, machine learning, web development, robotics, cybersecurity, and many other applications. These libraries save time and simplify complex programming tasks.

1.1.2 Major Reasons for Learning Python

- **Easy to Learn:** Python has a simple, readable, and user-friendly syntax that makes programming easy to understand and write. Its straightforward structure reduces complexity, making it an ideal programming language for beginners as well as experienced developers.
- **Free and Open Source:** Python is free to download, use, modify, and distribute without any licensing fees. Being open-source software, its source code is publicly available, allowing developers worldwide to contribute to its continuous improvement and development.
- **Cross-Platform Compatibility:** Python is a platform-independent programming language that runs efficiently on major operating systems such as Windows, Linux, and macOS. Programs written on one operating system can generally be executed on another with little or no modification.
- **Large Community Support:** Python has one of the largest programming communities in the world. Millions of developers, educators, researchers, and enthusiasts contribute tutorials, documentation, discussion forums, open-source projects, and technical support, making it easy to find solutions to programming problems.
- **Extensive Libraries:** Python provides a vast collection of built-in and third-party libraries that simplify programming tasks. These libraries support a wide range of applications, including scientific computing, data analysis, artificial intelligence, machine learning, web development, automation, image processing, and cybersecurity.
- **High Productivity:** Python enables programmers to accomplish complex tasks using fewer lines of code compared to many other programming languages. Its concise syntax, extensive libraries, and rapid development capabilities improve coding effi-

Figure 1.5: Python for Windows Installation

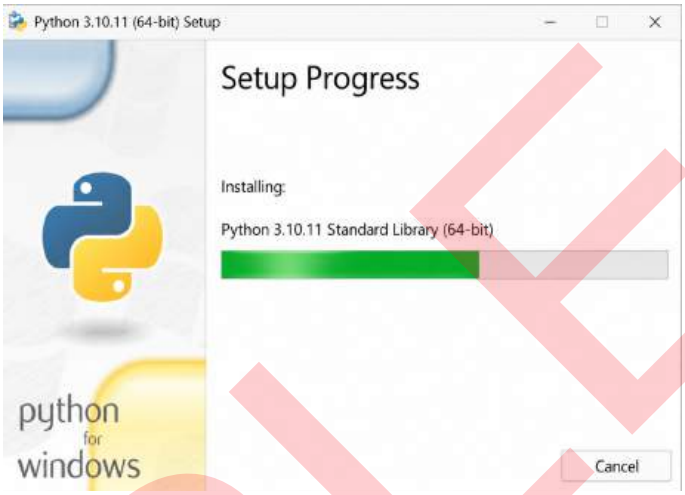


Figure 1.6: Python Setup

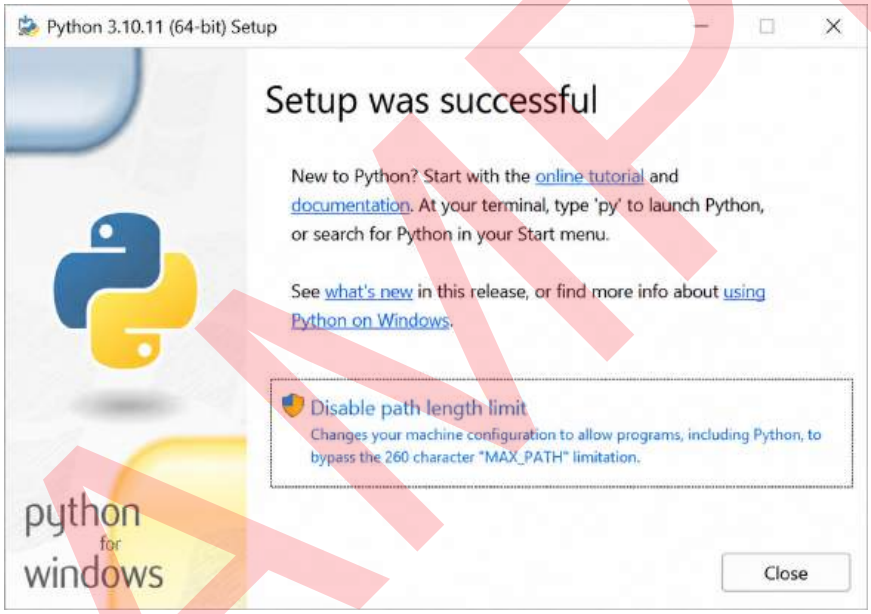


Figure 1.7: Python Successfully installed

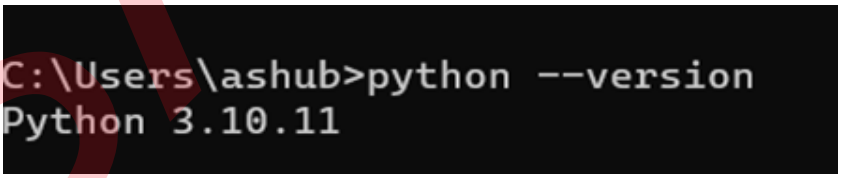


Figure 1.8: Python version

3. If Python is installed correctly, you will see the installed Python version displayed on the screen.
4. IMPORTANT: Before clicking Install, check the box that says "Add Python to PATH"

1.3 Integrated Development Environment

After installing Python, the next step is selecting an appropriate environment for writing and executing Python programs. Although Python programs can be written using any plain text editor such as Notepad or TextEdit, professional programmers and researchers generally use an Integrated Development Environment (IDE). An IDE is a software application that combines several development tools into a single interface, making programming more efficient, organized, and user-friendly.

5. Greater Than or Equal To Operator (>=)

Checks if the left operand is greater than or equal to the right.

NOTE: >= symbol signifies greater than or equal to. It evaluates to be True in math and programming if the value on its left is greater than the value on its right, OR if the two values are exactly the same. Otherwise, it evaluates to False

Example— Using >= to check greater than or equal conditions.

```
# Comparing two numbers using the >= (greater than or equal to)
operator

a = 9
b = 5
c = 9

print(a >= b)
print(a >= c)
print(b >= a)

Output:
True
True
False
```

Explanation

- The program will first checks condition (a >= b) whether the value of a is greater than or equal to the value of b. Since 9 is greater than 5, the result is True.
- Then second condition (a >= c) will compare a and c using the >= operator. Since both values are equal to 9, the condition is true and True is printed.
- Lastly third condition (b >= a) checks whether b is greater than or equal to a. Since 5 is smaller than 9, the condition is false and False is displayed.

6. Less Than or Equal To Operator (<=)

Checks if the left operand is less than or equal to the right.

NOTE: The symbol \(\le \) (less than or equal to) is an inequality operator that checks if a value is either strictly smaller than or exactly equal to another value. It evaluates to True if the left side is smaller than the right side, or if both sides are identical

Example: Using <= to check less than or equal conditions.

```
# Comparing numbers using the <= (less than or equal to) operator

a = 9
b = 5
c = 9

print(a <= b)
print(a <= c)
print(b <= a)

Output:
False
True
True
```

Table 1.12: Methods for Changing Letter Case

Method	Description
upper()	Converts all letters to uppercase
lower()	Converts all letters to lowercase
title()	Capitalizes the first letter of every word
capitalize()	Capitalizes only the first letter

Example

```
drug = "aspirin"
print(drug.upper())

Output:
ASPIRIN
```

Application: Standardizing medicine names in pharmaceutical records.

2. Removing Extra Spaces

Table 1.13: Methods for Removing Extra Spaces

Method	Description
strip()	Removes spaces from both ends
lstrip()	Removes leading spaces
rstrip()	Removes trailing spaces

Example

```
drug = "  Aspirin  "
print(drug.strip())

Output:
Aspirin
```

Application: Cleaning medicine and patient records before storing them in databases.

3. Replacing Text

The replace() method substitutes one substring with another.

Example

```
text = "Tablet"
print(text.replace("Tablet", "Capsule"))

Output:
Capsule
```

Application: Updating dosage forms in pharmacy databases.

4. Searching Text

The find() method returns the position of a substring.

Example

```
drug = "Paracetamol"
print(drug.find("ceta"))

Output:
4
```

Short Answer Type Questions

Q1. What is an IDE (Integrated Development Environment)? Name any three IDEs suitable for Python programming and mention one advantage of each.
Q2. What are the main differences between text editors and IDEs for programming? Why is an IDE preferred for pharmaceutical data analysis projects?
Q3. Explain the concept of a variable in Python with a pharmaceutical example. What naming conventions should be followed for variables in pharmacy applications?
Q4. Write code to create five variables of different data types relevant to pharmacy. Identify each variable's data type using the type() function.
Q5. What is type casting? Explain with two pharmaceutical examples where type casting is necessary. Write code for each example.
Q6. Explain the difference between arithmetic, comparison, and logical operators. Provide one pharmaceutical example for each.
Q7. What is the purpose of input() and print() functions in Python? Write a short program demonstrating both in a pharmacy context.
Q8. Explain five common string operations in Python with pharmaceutical examples.
Q9. What is the difference between standard libraries and third-party libraries in Python? Give three examples of each relevant to pharmacy.
Q10. Write the commands to install, verify, and uninstall a Python library using pip. Use Matplotlib as an example.

Long Answer Type Questions

Q1. Compare Jupyter Notebook, PyCharm, and VS Code in detail. For each IDE, discuss advantages, disadvantages, and the ideal pharmacy use case. Which IDE would you recommend and why?
Q2. Thoroughly explain Python data types (integers, floats, strings, booleans) with comprehensive pharmaceutical examples for each. Demonstrate creation, identification using type(), and common operations. Discuss why understanding data types is critical in pharmaceutical applications.
Q3. Write a comprehensive Python program demonstrating all three operator types (arithmetic, comparison, logical) in a realistic pharmaceutical dosing and safety validation system. The program should include:
(a) Accepting patient information via input(),
(b) Performing arithmetic operations for dose calculations,
(c) Using comparison operators to validate doses against safety limits,
(d) Applying logical operators to implement complex safety rules
(e) Displaying formatted output. Include comments explaining each step and pharmaceutical context.
Q4. Provide a comprehensive explanation of Python libraries, including the distinction between standard and third-party libraries. Explain why libraries are essential in pharmaceutical programming. Demonstrate step-by-step installation procedures using both pip and conda. Create a detailed practical program integrating NumPy, Pandas, and Matplotlib for a complete pharmacokinetic data analysis workflow, including data import, manipulation, numerical analysis, and visualization. Interpret results from a pharmaceutical perspective.

Multiple Choice Questions

Q1. Which of the following is an advantage of using Jupyter Notebook for pharmaceutical data analysis?
(a) It replaces Python programming entirely
(b) It provides interactive coding with inline graphs and outputs
(c) It only works without internet access
(d) It does not support scientific libraries
Q2. Which of the following is NOT a built-in Python data type?
(a) List (b) Tuple (c) Dictionary (d) Matrix
Q3. A drug concentration is entered as "75.6" in Python. Which function should be used before mathematical calculations?
(a) str() (b) float() (c) bool() (d) list()
Q4. Which Python operator is used to check whether one value is greater than another?
(a) + (b) > (c) = (d) %

2 CONTROL STRUCTURES & FUNCTIONS

- 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.

2.1 Conditional Statements (Decision Making)

In real life, we make decisions every day — for example: 'If the patient's temperature is above 102°F, give medicine; else, ask them to rest.' Python uses conditional statements to make such decisions in programs. These statements allow the program to execute different blocks(parts) of code depending on whether a condition is True or False.

A conditional statement is a logical "if-then" statement. So, the first part will occur in the if statement and then the second part follows after the first part. It is composed of two main parts: the Hypothesis (the "if") and the Conclusion (the "then").

Breaking it Down

- **Hypothesis:** The condition or situation.
- **Conclusion:** The result obtained is due to the condition.
- **Interpretation:** "If [Hypothesis], then [Conclusion]".

A Real-Life Example

"Forecast of rain today, then I will carry an umbrella." Here is how it breaks down:

- **Hypothesis:** "Forecasting of rain today" (The event that is supposed to happen).
- **Conclusion:** "Then I will carry an umbrella" (The result occurred as it is raining).

Other Real-Life Examples

- "If you will jump, you will fall down."
- "If you will take medicine on time, you will get better soon."

Indentation in Python

Indentation is used to group statements into blocks of code. Unlike many other programming languages, Python uses indentation instead of braces {} to define code structure. Rules of Indentation are:

1. Statements with the same indentation belong to the same block of code.
2. Indentation is created using spaces or tabs, but four spaces are commonly preferred.
3. Erratic indentation causes an IndentationError.

Indentation in Conditional Statements

All statements in a conditional block should have same alignment. Indentation in Python

The code below demonstrate how we use indentation to define seperate scopes of if-else statements:

```
a =  20
if a >= 18:
    print('Value of a is correct')
else:
    print('rewrite the value.')
print('All set !')
Output:
        Value of a is correct
        All set !
```

Explanation

- Statements inside if and else are indented, means it is in one block.
- Since a is greater than or equal to 18, the if block executes.
- The last print() statement is outside the if-else block, so it runs every time.

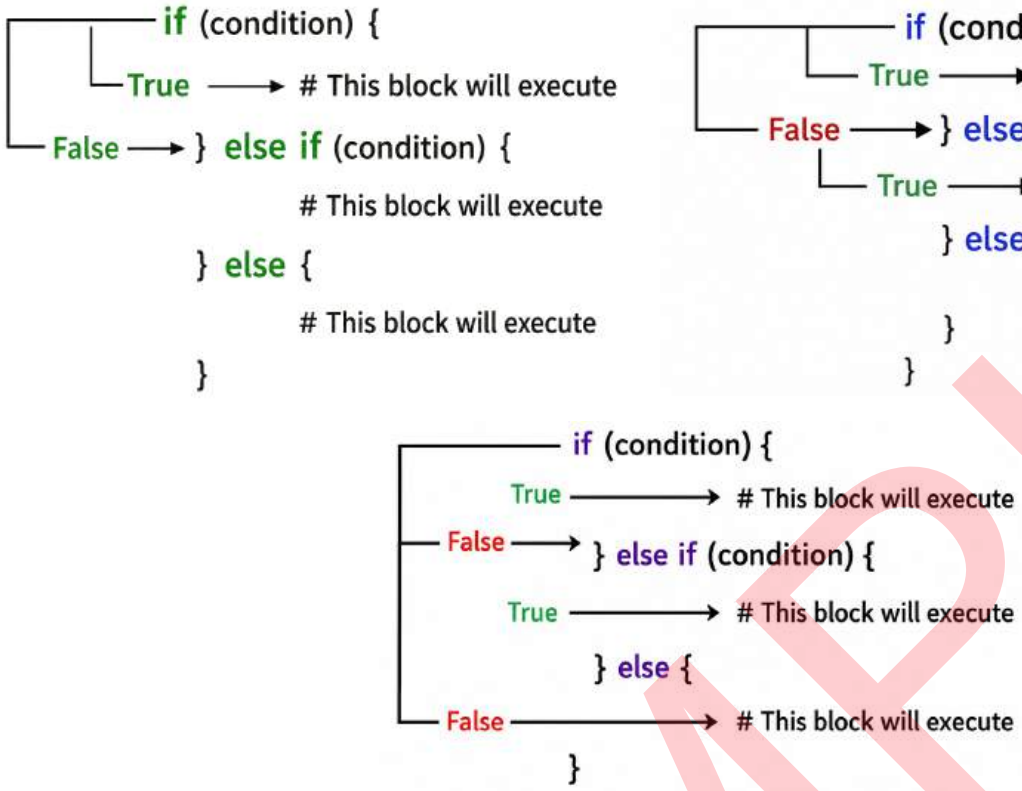


Figure 2.3: If-elif-else Statement

Syntax:

```
if condition1:
   # runs if condition1 is True
elif condition2:
   # runs if condition1 False AND condition2 True
elif condition3:
   # runs if condition1, 2 False AND condition3 True
else:
   # runs if ALL conditions above are False
```

Example 1: BMI Category Classification

```
bmi = 27.5

if bmi < 18.5:
   print('Category: Underweight')
   print('Advice: Increase caloric intake. Consult nutritionist.')
elif bmi < 25.0:
   print('Category: Normal weight')
   print('Advice: Maintain current diet and activity.')
elif bmi < 30.0:
   print('Category: Overweight')
   print('Advice: Reduce caloric intake. Increase physical activity.')
else:
   print('Category: Obese')
   print('Advice: Seek medical guidance immediately.')
```

Looping Over a List of Drug Names

```
drug_list = ["Paracetamol", "Ibuprofen", "Amoxicillin", "Metformin"]

for drug in drug_list:
    print("Checking inventory for:", drug)

Output:
Checking inventory for: Paracetamol
Checking inventory for: Ibuprofen
Checking inventory for: Amoxicillin
Checking inventory for: Metformin
```

The range() Function

When you need to repeat an action a specific number of times, or step through numeric values, the built-in range() function generates a sequence of numbers on demand.

Table 2.2: The range() Function

Call	Produces	Typical Use
range(5)	0, 1, 2, 3, 4	Repeat something 5 times
range(2, 8)	2, 3, 4, 5, 6, 7	Loop from a start to an end value
range(0, 20, 5)	0, 5, 10, 15	Step through values (e.g., mg increments)

Example: Generating a dose-escalation schedule in 50 mg increments up to 250 mg.

```
for dose in range(50, 300, 50):
    print(f"Administer {dose} mg")
Administer 50 mg
Administer 100 mg
Administer 150 mg
Administer 200 mg
Administer 250 mg
```

Examples of the for Loop

Example 1: Print Medicine Schedule for 7 days

```
for day in range(1, 8):  # range(1,8) gives numbers 1,2,3,4,5,6,7
    print(f'Day {day}: Take 1 tablet of Amoxicillin after meals')

# Output:
# Day 1: Take 1 tablet of Amoxicillin after meals
# Day 2: Take 1 tablet of Amoxicillin after meals
# ... (continues till Day 7)
```

Explanation

The program uses a for loop to display a medicine reminder for seven consecutive days. The range(1, 8) function generates numbers from 1 to 7 because index starts from "0"(zero), representing daily treatment schedule. For every iteration, the program prints a message instructing the patient to take one tablet of Amoxicillin after meals.

Example 2: Process a List of Patients

```
patients = ['Ramesh', 'Sita', 'Arun', 'Priya']

for patient in patients:
    print(f'Preparing prescription for: {patient}')
```

Real-World Analogy
Think of a function like a medicine formula in a pharmacopoeia. The formula is defined once and can be used (called) multiple times for different patients with different ingredient amounts (arguments).

2.7.1 Defining and Calling a Function

A function is created with the def keyword, followed by the function name, a pair of parentheses that may contain parameters, and a colon. The function's code block is indented beneath this line.

A function is called (or invoked) by writing its name followed by parentheses containing the values to pass in. These values are supplied in the same order as the parameters were defined, unless keyword arguments are used.

Syntax:

```
# DEFINING a function
def function_name():
    # body of the function
    # (code that runs when function is called)

# CALLING a function
function_name()
```

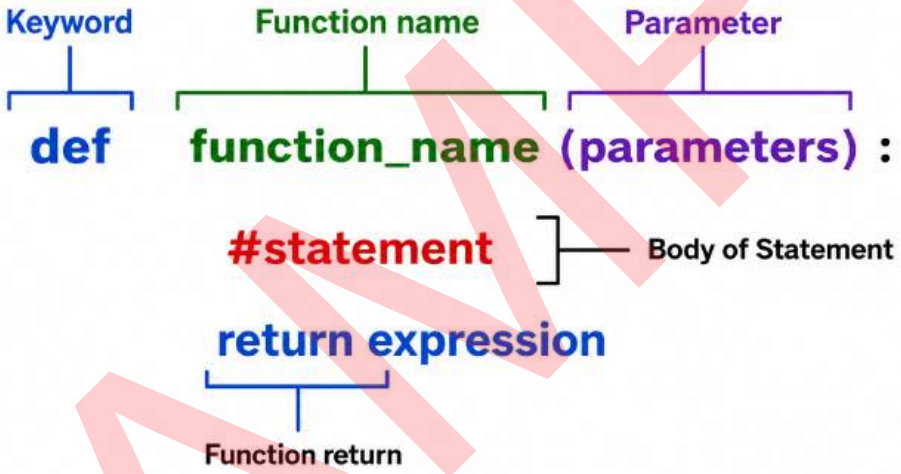


Figure 2.6: Explanation of Syntax

A few points about this definition are worth highlighting:

- The function name, dose_per_kg, is descriptive. Following Python convention, function names use lowercase words separated by underscores (snake_case).
- dose_mg and weight_kg are parameters — placeholders for the values the function will receive when it is called.
- The triple-quoted string immediately below the def line is a docstring. It documents what the function does and is good practice in any script intended for research or regulated work, since it can be read later with help(dose_per_kg).
- The return statement sends a value back to whatever code called the function.

Defining vs. running
Writing a def block only defines the function; it does not execute the code inside it. Nothing is calculated until the function is called.

Example: Greeting function for Patients

```
# DEFINE the function
def greet_patient():
    print('Welcome to City Pharmacy!')
    print('Please have a seat. A pharmacist will assist you shortly.')
```

3 DATA STRUCTURES & FILE HANDLING

- 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.

3.1 Lists

A list is one of the most commonly used data structures in Python. It is a built-in data type used to store a collection of multiple items in a single, just like how arrays work. It is an ordered, mutable (changeable) collection that can store multiple items of different types in a single variable.

Example 1: Marks of Students

Think of a list like a marks register that stores the marks of several students. [85, 90, 78, 92, 88] contains marks of five students. You can add a new mark, update a student's mark, or remove an incorrect entry whenever needed.

Example 2:

Think of a list like a medicine tray that holds several items in slots, you can add, remove, or change any item at any time.

Example 3: Patient Heart Rates

A collection of heart rate readings taken from a patient over time. For example, [72, 75, 70, 78, 74] stores multiple heart rate measurements. Doctors can review, update, or analyze these readings as required.

NOTE:

Also, we can have lists inside lists, e.g.,

a = [3.2, 'hello', [4, 8, 17.1]]

Key Properties of a List

They are dynamic, resizable and capable of storing multiple data types.

- Mutable: list elements can be changed, updated, added, or removed after the list is created.
- Ordered: elements maintain the order in which they are inserted.
- Index-based: elements are accessed using their position, starting from index 0.

Can hold any data type, numbers, strings, floats, even other lists.

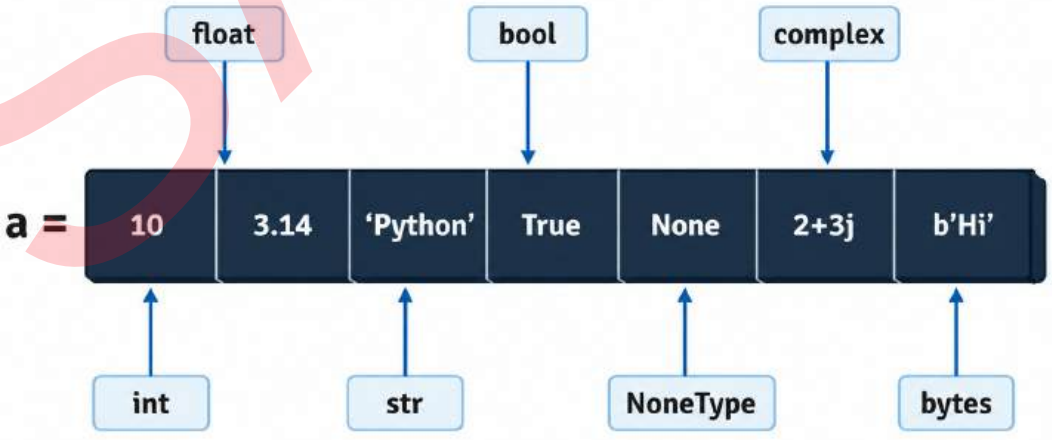


Figure 3.1: A List containing items of different data types

```
print(sum([100, 250, 500]))          //Calculates and displays the sum of all numbers.
Output:
850
```

Table 3.1: Common Python List Methods with Pharmaceutical Examples

Method	Action	Pharma Example
append(x)	Add x to end	meds.append('Aspirin')
insert(i,x)	Insert x at index i	meds.insert(0,'Vitamin C')
remove(x)	Remove first x	meds.remove('Expired Drug')
pop(i)	Remove & return item at i	expired = meds.pop(-1)
sort()	Sort in place	dosages.sort()
reverse()	Reverse in place	meds.reverse()
index(x)	Find index of x	meds.index('Metformin')
count(x)	Count occurrences of x	meds.count('Paracetamol')

3.2 Tuples

A tuple is similar to a list but with one crucial difference it is immutable, meaning once created, its contents cannot be changed. Tuples are used for data that should remain fixed throughout the program. It can hold elements of different data types. These are ordered, heterogeneous and immutable. In pharmacy, a drug's approved chemical formula, regulatory code, or a standard reference range should never change these are ideal tuple candidates.

List vs Tuple Core Difference

- List: uses square brackets [ ], mutable (changeable).
- Tuple: uses round brackets ( ), immutable (cannot be changed).
- Both are ordered and support indexing and slicing.

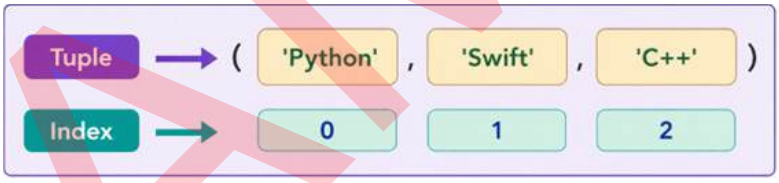


Figure 3.2: Python Tuple and Indexing

Explanation

- A tuple is an ordered collection of items enclosed in parentheses ( ).
- Each item in the tuple has an index number, starting from 0.
- We can use the index to access items in the tuple.

3.2.1 Creating and Accessing Tuples

Example 1: Program for creating all types of List in Python

```
Create a Python Tuple
We create a tuple by placing items inside parentheses (). For example,
numbers = (1, 2, -5)

print(numbers)

Output: (1, 2, -5)
```

Example 2: Program for creating all types of List in Python

3.3 Dictionaries

A dictionary stores data as key-value pairs. Instead of accessing items by position (index), you access them by a unique key just like looking up a word in a real dictionary to find its meaning. Each key acts as a unique identifier, while the corresponding value represents the data associated with that key. Dictionaries are enclosed within curly braces {} and multiple key–value pairs are separated by commas. In pharmacy, a patient record with fields like name, age, and diagnosis is perfectly suited to a dictionary structure.

Key Properties of a Dictionary

Unordered in older Python (ordered in Python 3.7+). Keys must be unique and immutable (strings, numbers). Values can be any data type. Access by key, not by position. Mutable — can add, modify, or delete key-value pairs.

3.3.1 Creating a Dictionary

Example 1: Creating a dictionary of Country Capitals

```
# Creating a dictionary of country capitals
country_capitals = {
   “Germany”: “Berlin”,
   “Canada”: “Ottawa”,
   “England”: “London”
}
# Displaying the dictionary print(country_capitals)
print(country_capitals)

Output:
{‘Germany’: ‘Berlin’, ‘Canada’: ‘Ottawa’, ‘England’: ‘London’}
```

Explanation

country_capitals is a dictionary containing three key–value pairs. The keys (“Germany”, “Canada”, and “England”) represent country names, while the values (“Berlin”, “Ottawa”, and “London”) represent their respective capital cities. Python uses the key to efficiently retrieve the associated value.

Example 2: Dictionary Creation

```
# Patient dictionary
patient = {
   ‘name’     : ‘Ramesh Kumar’,
   ‘age’      : 45,
   ‘weight’   : 72.5,
   ‘diagnosis’ : ‘Type 2 Diabetes’,
   ‘medicines’ : [‘Metformin’, ‘Glipizide’]  # Value can be a list!
}

# Drug dictionary
drug = {
   ‘name’   : ‘Amoxicillin’,
   ‘class’  : ‘Antibiotic’,
   ‘dose_mg’ : 500,
   ‘route’  : ‘Oral’,
   ‘frequency’: ‘Three times daily’
}
```

Explanation
The string "Hello " is multiplied by 3. Python repeats the word Hello three times and combines them into a single string.

Example 2: Creating a Decorative Line

```
line = "-" * 20
print(line)
Output:
--------------------
```

Explanation
The dash character (-) is repeated 20 times. This technique is often used to create separators in reports and menus.

Example 3: Displaying a Warning Message

```
warning = "Take Medicine On Time! " * 3
print(warning)
Output:
Take Medicine On Time! Take Medicine On Time! Take Medicine On Time!
```

Explanation
The message is repeated three times. In healthcare applications, such repeated messages can be used to emphasize important instructions for patients.

Real-Life Analogy
Imagine a pharmacist needs five copies of the same medicine label. Instead of writing the label five times manually, a machine can print the same label repeatedly. The multiplication operator performs a similar task in Python by automatically repeating the same text multiple times.

Difference Between Numeric and String Multiplication

a) Numeric Multiplication

```
print(5 * 3)
Output:
15
```

The numbers are mathematically multiplied.

b) String Multiplication

```
print("Hi " * 3)
Output: Hi Hi Hi
```

The string is repeated three times instead of being mathematically multiplied.

3. Append Operator (+=)

The append operator adds new content to the end of an existing string.

In Python, the Append Operator (+=) is used to add new content to the end of an existing string. Instead of creating a completely new string variable, this operator allows us to extend the current string by attaching additional text to it. The append operator is a combination of two operators:

- + (concatenation operator) → joins strings together
- = (assignment operator) → stores the result back into the same variable

Therefore, the statement:

```
message += "Python"
```

is equivalent to:

```
message = message + "Python"
```

Both statements produce the same result, but the += operator is shorter, easier to read, and more convenient when text needs to be added repeatedly.

2. Median

The median is the middle value after arranging data in ascending order.

Example:

```
70, 75, 80, 85, 90
Middle value:
80
NumPy Function
print(np.median(marks))
Output
80.0
```

3. Mode (Concept)

The mode is the value that occurs most frequently.

Example:

70, 80, 80, 90, 80
Mode:
80
Although NumPy does not directly provide a mode function, understanding mode is important in statistics.

4. Minimum Value

The minimum value is the smallest number in the dataset.

Example:

```
print(np.min(marks))
Output:  70
```

Explanation
The lowest mark obtained by any student is 70.

5. Maximum Value

The maximum value is the largest number in the dataset.

Example:

```
print(np.max(marks))
Output: 90
```

Explanation
The highest mark obtained by any student is 90.

6. Sum

The sum function adds all values together.

Example:

```
print(np.sum(marks))
Output: 400
```

Explanation
The total marks of all students are 400.

7. Standard Deviation

Standard deviation measures how spread out the data values are from the average.

Example:

Q3. A tuple containing (drug_name, strength, form) is immutable because:
(a) It's stored in memory (b) Once created, its elements cannot be changed
(c) It has three elements (d) It contains strings

Q4. Dictionary keys must be:
(a) Strings only (b) Numbers only
(c) Unique and immutable (d) Lists

Q5. What is the output of "Amoxicillin".find("mox")?
(a) True (b) 1 (c) "mox" (d) -1

Q6. String slicing s[2:5] returns:
(a) Characters at positions 2, 3, 4 (b) Characters at positions 2, 3, 4, 5
(c) First 5 characters (d) Error

Q7. NumPy arrays are preferred for pharmaceutical calculations because:
(a) They look nicer (b) They're built-in
(c) They enable fast vectorized operations (d) They can store any data type

Q8. Creating a NumPy array from list [1, 2, 3]:
(a) np.list([1, 2, 3]) (b) np.array([1, 2, 3])
(c) array([1, 2, 3]) (d) np.create([1, 2, 3])

Q9. What does np.trapz(concentrations, time) calculate?
(a) Maximum value (b) Mean value
(c) Area Under Curve (AUC) (d) Standard deviation

Q10. CSV files are suitable for pharmaceutical data because:
(a) They're encrypted
(b) They're universal, human-readable, and compatible with all software
(c) They automatically validate data
(d) They're faster than databases

Q11. To read a CSV file using pandas:
(a) pd.open_csv("file.csv") (b) pd.read_csv("file.csv")
(c) pandas.import_csv("file.csv") (d) csv.read("file.csv")

Q12. Filtering pandas DataFrame for concentrations > 50:
(a) df[df["Concentration"] > 50] (b) df.filter(Concentration > 50)
(c) df.where(Concentration > 50) (d) df("Concentration" > 50)

Q13. What does df.shape return?
(a) Column names (b) First few rows
(c) Tuple of (rows, columns) (d) Data types

Q14. String method .upper() on "amoxicillin" returns:
(a) "A" (b) "AMOXICILLIN" (c) "amoxicillin" (d) Error

Q15. Accessing dictionary value with key "patient_id":
(a) dict["patient_id"] (b) dict.patient_id (c) dict[0] (d) dict.get(0)

Answer Keys

1-(c)	2-(b)	3-(b)	4-(c)	5-(b)	6-(a)	7-(c)	8-(b)	9-(c)	10-(b)
11-(b)	12-(a)	13-(c)	14-(b)	15-(a)					

4 DATA HANDLING WITH PANDAS

Definition of Acids, Bases, Buffers, pH Scale And its Significance, Buffer Equation, Calculation of pH for Buffer Solution, Isotonicity and its Application in IV Fluids and Ophthalmic Solutions.

4.1 Introduction to the Pandas Library

4.1.1 Pandas

Pandas is an open-source, high-performance data analysis and data manipulation library for the Python programming language. The name 'Pandas' is derived from the term Panel Data — a type of multidimensional data structure used in statistics and econometrics. It was originally developed by Wes McKinney in 2008 while he was working at AQR Capital Management, and it has since become the standard tool for data wrangling in Python across science, engineering, and business domains.

Pandas introduces two important data structures:

1. **Series:** A Series is a one-dimensional labelled array that can store data such as numbers, text, or other Python objects. It is similar to a single column in a spreadsheet.
2. **DataFrame:** A DataFrame is a two-dimensional tabular data structure consisting of rows and columns. It is similar to an Excel worksheet or a database table and is the most commonly used Pandas object.

Pandas is widely used in data science, healthcare analytics, pharmacy research, business intelligence, finance, and machine learning because it integrates seamlessly with several other Python libraries:

- NumPy for numerical and mathematical computations.
- Matplotlib and Seaborn for creating graphs and visualizing data.
- SciPy for statistical analysis and scientific computing.
- Scikit-learn for machine learning and predictive modeling.

Pandas provides a comprehensive set of tools for handling real-world datasets and is commonly used for:

1. **Data Cleaning:** Removing missing values, correcting errors, handling duplicates, and preparing raw data for analysis.
2. **Data Transformation**: Changing the structure or format of data, filtering records, sorting values, and creating new calculated fields.
3. **Data Analysis:** Performing statistical calculations, summarizing datasets, identifying trends, and generating meaningful insights.
4. **Machine Learning Preparation:** Organizing and preprocessing data before it is used for training machine learning models.
5. **Data Visualization Support:** Preparing and formatting data for graphical representation using visualization libraries.

4.1.1.1 Importance of Pandas in Pharmacy and Healthcare

In pharmacy and healthcare, large amounts of patient and medical data are generated every day. Pandas helps healthcare professionals and researchers efficiently manage and analyze information such as:

- Patient records and demographics
- Blood glucose and blood pressure readings
- Medicine inventory and stock management
- Clinical trial datasets
- Laboratory test results
- Prescription and treatment records

For example, a pharmacy can use Pandas to analyze medicine stock levels, identify low-stock drugs, calculate average patient readings, and generate reports for decision-making.

In the context of pharmaceutical sciences, Pandas is especially relevant because:

- Clinical trial data, patient records, drug concentration measurements, and ADR reports are almost always stored in tabular formats (spreadsheets or CSV files).
- Pharmacokinetic (PK) studies generate time-series data that requires careful handling, cleaning, and statistical summarization.
- Regulatory agencies such as the CDSCO, FDA, and EMA require structured, auditable data, Pandas helps create reproducible data pipelines

P001	1.0	500	52.10	68.2
P001	2.0	500	68.40	68.2
P002	0.0	500	0.00	72.5
P002	0.5	500	18.90	72.5

Example: Reading an ADR Report from an Excel File

```
# Import the Pandas library
import pandas as pd

# Read the CSV file into a DataFrame
pk_df = pd.read_csv("pk_study_data.csv")

# Display the DataFrame
print(pk_df)

Output:
  Subject_ID  Time_hr  Dose_mg  Conc_ngmL  Body_Weight_kg
0       P001      0.0      500        0.0            68.2
1       P001      0.5      500       15.3            68.2
2       P001      1.0      500       52.1            68.2
3       P001      2.0      500       68.4            68.2
4       P002      0.0      500        0.0            72.5
5       P002      0.5      500       18.9            72.5
```

Explanation

- The first row of the CSV file is automatically treated as the column names.
- Every remaining row becomes a record in the DataFrame.
- Pandas automatically assigns row numbers (0, 1, 2, ...) known as the DataFrame index.
- The data is now ready for filtering, statistical analysis, plotting, or machine learning.

Example: Read and display csv file

```
# Import the Pandas library
import pandas as pd

# ------------------------------------------------
# Step 1: Create sample Pharmacokinetic (PK) data
# ------------------------------------------------

# Create a dictionary containing sample PK study data
data = {
    'Subject_ID': [101, 102, 103, 104],        # Patient IDs
    'Time_hr': [0, 1, 2, 4],                   # Time after drug administration (hours)
    'Conc_ngmL': [0.0, 25.6, 48.2, 35.4]       # Drug concentration (ng/mL)
}

# Convert the dictionary into a DataFrame
df = pd.DataFrame(data)

# Save the DataFrame as a CSV file
df.to_csv('pk_study_data.csv', index=False)
```

Hospitals and pharmaceutical companies collect large amounts of patient-related information every day. Before this information can be analyzed, it must be carefully verified and cleaned. Typical pharmacy datasets include:

- Electronic Health Records (EHR)
- Patient demographic information
- Drug inventory databases
- Clinical trial datasets
- Pharmacokinetic (PK) studies
- Pharmacodynamic (PD) studies
- Laboratory reports
- Adverse Drug Reaction (ADR) reports
- Prescription records

For example, if the body weight of several patients is missing in a clinical trial dataset, calculating the average Body Mass Index (BMI) without handling these missing values may produce misleading results. Therefore, missing information must be properly treated before performing any analysis.

4.5.2 Missing Values

A missing value represents information that is unavailable, incomplete, or not recorded in a dataset. During data collection, certain observations may be accidentally omitted, lost, or intentionally left blank. Such incomplete entries are referred to as missing values. In Pandas, missing values are commonly represented by:

- NaN (Not a Number): Used mainly for missing numerical values.
- None: Represents missing values in Python objects.
- Empty (blank) cells in a dataset.
- Special placeholders such as "Unknown", "NA", "N/A", or "Missing".

For example, a patient database may contain the following incomplete information:

Patient ID	Weight (kg)	Blood Glucose (mg/dL)
101	72.5	110
102	NaN	95
103	88.0	NaN
104	78.5	125

Here, the missing values are represented by NaN, indicating that the corresponding information was not available during data collection.

Table 4.7: Student Academic Records

School ID	Name	Address	City	Subject	Marks	Rank	Grade
101.0	A	123 Main St	Mumbai	Math	85.0	2	B
102.0	B	456 Oak Ave	Delhi	English	92.0	1	A
103.0	C	789 Pine Ln	Bengaluru	Science	78.0	4	C
NaN	D	101 Elm St	Chennai	Math	89.0	3	B
105.0	E	NaN	Kolkata	History	NaN	8	D
106.0	F	222 Maple Rd.	NaN	Math	95.0	1	A
107.0	G	444 Cedar Blvd	Pune	Science	80.0	5	C
108.0	H	555 Birch Dr	Jaipur	English	88.0	3	B

4.5.2.1 Reasons for Handling Missing Values

Ignoring missing values can significantly reduce the quality and reliability of data analysis. Many statistical techniques and machine learning algorithms require complete datasets. If missing values are not handled properly, they may lead to incorrect calculations, biased conclusions, or even software errors. Proper handling of missing values helps to:

- Improve the accuracy of statistical calculations.
- Prevent biased or misleading results.
- Preserve valuable patient records whenever possible.

Step 1: Calculate Average Weight

Available weights:
72.5, 88.0, 78.5
Average weight

$$\frac{72.5+88.0+78.5}{3} = 79.67$$

So, Sita's missing weight becomes 79.67 kg.

Step 2: Calculate Average Glucose

Available glucose values:
110, 95, 125
Average glucose

$$\frac{110+95+125}{3} = 110$$

So, Arun's missing glucose value becomes 110 mg/dL.

Final Clean Dataset

All missing values have now been replaced with meaningful values.

Explanation

- **Step 1:** The program imports the Pandas library to work with tabular data.
- **Step 2:** A DataFrame named patients is created. Some values are intentionally left blank using None, which Pandas converts into NaN.
- **Step 3:** The original dataset is displayed so that the missing values can be clearly observed.
- **Step 4:** The program calculates the average glucose level using the mean() function.
- **Step 5:** The fillna() function replaces the missing glucose value with the calculated average.
- **Step 6:** Similarly, the average body weight is calculated and used to replace the missing weight value.
- **Step 7:** Finally, the cleaned dataset is displayed. All missing values have been replaced, making the dataset complete and ready for statistical analysis or machine learning.

4.5.2.7 Removing Duplicate Rows

In real-world datasets, the same record may sometimes appear more than once. These repeated records are called duplicate rows. Duplicate rows usually occur because of data entry mistakes, importing data multiple times, software errors, or combining data from different sources.

In Pandas, duplicate rows can be identified and removed using the drop_duplicates() function. Removing duplicate records is an important step in data cleaning because duplicate data can produce incorrect calculations, inaccurate statistical results, and misleading conclusions.

For example, if the same patient's laboratory report is entered twice, the average blood glucose level may become incorrect. Therefore, duplicate records should be removed before performing any data analysis.

Pharmacy Perspective

In hospitals and pharmaceutical companies, patient information, prescriptions, laboratory reports, and clinical trial records are collected from multiple departments. Sometimes the same patient's record is entered more than once due to human error or repeated data import. If duplicate records are not removed, researchers may count the same patient multiple times, leading to incorrect drug efficacy analysis or inaccurate clinical study results.

Using drop_duplicates(), pharmacists and researchers can quickly remove repeated records and maintain a clean, reliable dataset.
Syntax
DataFrame.drop_duplicates()

Example 1: Program for Filling the Missing Values of Glucose level of patients

```
# Import the Pandas library
import pandas as pd
```

Original Dataset

The dataset contains drug concentration values measured for different patients.

Subject_ID	Time_hr	Conc_ngmL
P001	0	0.0
P001	1	35.4
P001	2	58.6
P002	0	72.5
P002	1	44.8
P003	2	85.3

Step 1: Boolean Condition

The program checks every concentration value.

Condition:

Conc_ngmL > 50

Result:

Concentration	Condition
0.0	False
35.4	False
58.6	True
72.5	True
44.8	False
85.3	True

Step 2: Filter Rows

Only rows marked True are selected.

Subject_ID	Time_hr	Conc_ngmL
P001	2	58.6
P002	0	72.5
P003	2	85.3

Explanation

Step 1: The program imports the Pandas library for data manipulation.
Step 2: A sample pharmacokinetic (PK) DataFrame named pk_df is created. It stores patient IDs, sampling time, and drug concentration values.
Step 3: The original dataset is displayed.
Step 4: A Boolean condition is created using:
pk_df['Conc_ngmL'] > 50
Pandas compares every concentration value with 50 and generates a Boolean Series containing True and False values.
Step 5: This Boolean Series is stored in the variable condition and displayed.
Step 6: The Boolean Series is then used inside square brackets:
pk_df[condition]
Only the rows where the condition is True are selected.
Step 7: Finally, the program demonstrates the shorter and most commonly used one-line Boolean indexing method:
pk_df[pk_df['Conc_ngmL'] > 50]
Both methods produce the same filtered dataset.

4.6.2 Filtering with Multiple Conditions

In real-world data analysis, we often need to filter records based on more than one condition. For example, a pharmacist may want to find patients who:

```
2     P002     2     55.8
3     P003     2     68.4
5     P005     4     82.1
6     P006     5     95.6

Output of Example 2
Example 2 : Concentration > 50 AND Time >= 2 hr

  Subject_ID  Time_hr  Conc_ngmL
2     P002     2     55.8
3     P003     2     68.4
5     P005     4     82.1
6     P006     5     95.6

Output of Example 3
Example 3 : Concentration greater than variable threshold

  Subject_ID  Time_hr  Conc_ngmL
3     P003     2     68.4
5     P005     4     82.1
6     P006     5     95.6
```

4.7 Grouping Data and Performing Aggregation Functions

In real-world healthcare and pharmaceutical research, datasets often contain thousands of patient records collected from different hospitals, clinical trials, or laboratories. Instead of analyzing each individual record separately, researchers frequently need to group similar records together and calculate summary statistics such as the average, maximum, minimum, count, or total.

Pandas provides a powerful function called groupby() that allows us to divide data into groups based on one or more columns and then perform various statistical calculations on each group. This process is known as Grouping and Aggregation.

Grouping helps transform large amounts of raw data into meaningful summaries, making it easier to understand trends, compare different groups, and make informed decisions.

Grouping

- Grouping is the process of dividing a dataset into smaller groups based on the values of one or more columns.
- After creating these groups, we can perform calculations separately on each group instead of the entire dataset.
- For example, suppose a dataset contains information about patients suffering from different diseases.

Patient	Disease	Glucose
Ramesh	Diabetes	180
Sita	Hypertension	110
Arun	Diabetes	200
Priya	Hypertension	115

Instead of calculating one average glucose level for all patients, we may want to calculate the average glucose level for each disease separately. This is exactly what groupby() does.

Aggregation

After grouping the data, we usually want to perform some calculations on each group. These calculations are called aggregation functions. Aggregation combines multiple values into a single summarized value. For example,

Patient glucose readings:

120
130
140
110

healthcare data based on specific conditions.

8. Explain filtering and selecting data based on conditions in Pandas. Discuss multiple-condition filtering, the .isin() function, and the use of .loc[] and .iloc[] for accessing rows, columns, and cells.
9. What is meant by grouping and aggregation in Pandas? Explain the groupby() function along with common aggregation functions, using examples relevant to PK studies and ADR reports.
10. Explain the use of the .agg() method for applying multiple aggregation functions, and differentiate between .agg() and .transform() in a groupby operation, with suitable examples.

Multiple Choice Questions

1. What is Pandas in Python?
(a) A database management system
(b) A Python library for data manipulation and analysis
(c) A programming language
(d) A web development framework

2. Which data structure in Pandas is one-dimensional?
(a) DataFrame (b) Series (c) Array (d) Matrix

3. A Pandas DataFrame is best described as:
(a) A single variable
(b) A two-dimensional table with rows and columns
(c) A text file
(d) A mathematical formula

4. Which function is used to read a CSV file in Pandas?
(a) pd.load_csv() (b) pd.import_csv() (c) pd.read_csv() (d) pd.csv_read()

5. Which function is used to read an Excel file in Pandas?
(a) pd.read_excel() (b) pd.load_excel() (c) pd.import_excel() (d) pd.open_excel()

6. Which function displays the first five rows of a DataFrame by default?
(a) tail() (b) info() (c) head() (d) describe()

7. Which Pandas function displays the last five rows of a DataFrame?
(a) tail() (b) head() (c) describe() (d) shape()

8. Which function provides information about column names, data types, and missing values?
(a) summary() (b) describe() (c) info() (d) tail()

9. Which function generates descriptive statistics for numerical columns?
(a) info() (b) describe() (c) head() (d) shape()

10. Which function removes rows containing missing values?
(a) fillna() (b) dropna() (c) drop() (d) replace()

11. Which function is used to replace missing values with a specified value?
(a) replace() (b) dropna() (c) fillna() (d) update()

12. Which statement correctly filters patients whose glucose level is greater than 140?
(a) df[df['Glucose'] > 140] (b) df.Glucose = 140 (c) df.filter(140) (d) df.select('Glucose')

13. Which method allows SQL-like filtering of a DataFrame?
(a) filter() (b) query() (c) where() (d) loc()

14. Which Pandas function is used to group data before performing calculations?
(a) merge() (b) join() (c) groupby() (d) sort_values()

15. Which aggregation function calculates the average value of each group?
(a) count() (b) sum() (c) mean()\ (d) max()

Answer Keys

1-(b)	2-(b)	3-(b)	4-(c)	5-(a)	6-(c)	7-(a)	8-(c)	9-(b)	10-(b)
11-(c)	12-(a)	13-(b)	14-(c)	15-(c)					

5 DATA VISUALIZATION WITH MATPLOTLIB

- 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.

5.1 Introduction to Matplotlib

5.1.1 Introduction

Matplotlib is a comprehensive, open-source Python library for creating static, animated, and interactive visualizations. Developed by John D. Hunter in 2003 and now maintained by a large community, Matplotlib has become the de facto standard for scientific and technical plotting in Python. It integrates seamlessly with NumPy and Pandas, making it ideal for analysing pharmaceutical data stored in DataFrames.

Features of Matplotlib

- **Simple API:** easy to create basic plots with just a few lines of code.
- **Customizable:** nearly every aspect of a plot (colours, fonts, line styles, annotations) can be modified.
- **Publication-quality output:** figures are suitable for journal articles, theses, and presentations.
- **Multiple plot types:** line plots, scatter plots, histograms, bar charts, box plots, heatmaps, and more.
- **LaTeX integration:** ability to use mathematical symbols and equations in labels.
- **Interactive features:** zoom, pan, and save figures in multiple formats (PNG, PDF, SVG, EPS).

5.1.2 Why Matplotlib for Pharmaceutical Visualization?

In the pharmaceutical field, data visualization serves several critical functions:

1. Exploratory Data Analysis (EDA) understanding the distribution and relationships in your data before performing statistical tests.
2. Communication presenting findings to colleagues, regulators, and patients in an intuitive, visually compelling format.
3. Quality Control spotting outliers and errors in laboratory or clinical data.
4. Regulatory Compliance meeting FDA, EMA, and CDSCO requirements for data presentation in submissions.

A concentration-time plot, for example, is more informative than a table of 50 plasma concentrations. A box plot of adverse event frequencies by drug class conveys risk at a glance. Dissolution profile curves demonstrate product quality and consistency.

5.1.3 Installation and Import

Matplotlib is typically included with Anaconda/Miniconda. For standalone installation via pip:

Matplotlib is one of the most popular Python libraries used for creating graphs, charts, and other visual representations of data. Before using Matplotlib, it must be installed on your computer and then imported into your Python program. Once installed, it enables users to transform numerical data into meaningful visualizations that are easier to interpret and communicate. In pharmaceutical sciences, Matplotlib is widely used to visualize experimental data, drug dissolution profiles, pharmacokinetic curves, adverse drug reaction trends, clinical trial outcomes, and quality control results.

Matplotlib is included by default in many Python distributions such as Anaconda and Google Colab. Therefore, users working with these platforms generally do not need to install it separately. However, if Python has been installed independently from the official Python website, Matplotlib must be installed using a package manager such as pip.

```
# Install Matplotlib
pip install matplotlib

# Import Matplotlib and its pyplot module (standard practice)
import matplotlib.pyplot as plt
import matplotlib as mpl
```

```
plt.xlabel('Concentration (ng/mL)', fontsize=12)
plt.ylabel('Frequency (Number of Subjects)', fontsize=12)
plt.title('Distribution of Plasma Concentrations at 4 Hours', fontsize=14, fontweight='bold')
plt.grid(axis='y', alpha=0.3)
plt.show()
```

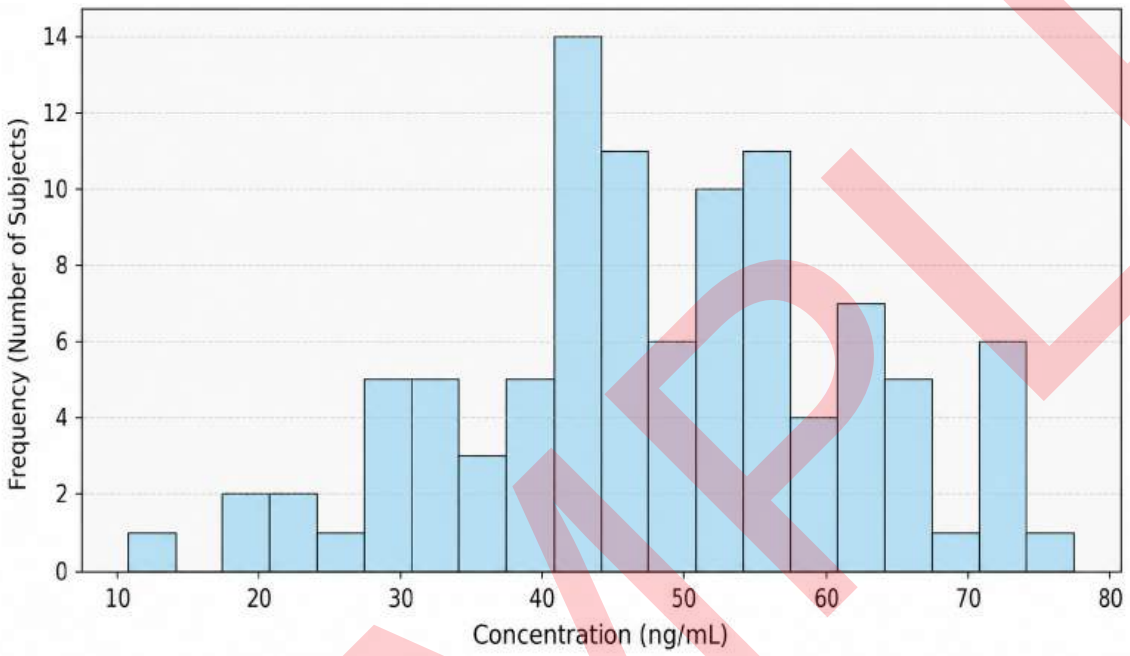


Figure 5.5: Histogram Showing the Distribution of Plasma Concentrations at 4 Hours

Histogram Parameters

The plt.hist() function provides several parameters that allow users to customize the appearance and behavior of a histogram. By modifying these parameters, users can control the number of intervals, colors, transparency, labels, and other graphical properties, making the histogram more informative and visually appealing.

Table 5.2: Common Parameters Used in Matplotlib Histograms

Parameter	Description	Example
bins	Number of bins (bars) or bin edges	bins=20 or bins=[0, 10, 20, ..., 100]
color	Colour of bars	color='skyblue'
edgecolor	Colour of bar borders	edgecolor='black'
alpha	Transparency (0–1)	alpha=0.7
density	Normalize to probability density	density=True
cumulative	Cumulative histogram	cumulative=True

5.2.4 Box Plots — plt.boxplot()

A box plot (also called a box-and-whisker plot) is a statistical graph used to summarize the distribution of a numerical dataset. It provides a quick visual representation of the minimum value, first quartile (Q1), median, third quartile (Q3), maximum value, and any outliers present in the data.

In Matplotlib, the plt.boxplot() function is used to create box plots. Box plots are especially useful for comparing multiple datasets and identifying variations or unusual observations. In pharmaceutical sciences, they are widely used in quality control, formulation development, clinical research, and bioequivalence studies.

Unlike histograms, which show the frequency distribution of data, box plots summarize the distribution using key statistical measures.

Box plots summarize distributions using quartiles and extremes. They show median (middle line), interquartile range (IQR,

```
total_reports = np.array(
  [15, 18, 22, 19, 23, 25, 28, 31, 35, 38, 42, 45,
   48, 52, 58, 65, 72, 68, 75, 82, 78, 85, 88, 92])
serious_reports = np.array(
  [2, 3, 4, 3, 5, 6, 7, 8, 10, 11, 13, 15,
   17, 19, 22, 25, 28, 26, 31, 35, 32, 38, 40, 43])

# Create figure
fig, (ax1, ax2) = plt.subplots(2, 1, figsize=(11, 8))

# ── Top plot: Total vs Serious reports ──
ax1.plot(months, total_reports, 'b-o', linewidth=2, markersize=6, label='Total Reports')
ax1.plot(months, serious_reports, 'r-s', linewidth=2, markersize=6, label='Serious Reports')
ax1.fill_between(months, total_reports, alpha=0.2, color='blue')
ax1.set_ylabel('Number of Reports', fontsize=11, fontweight='bold')
ax1.set_title('ADR Reporting Trend Over 24 Months', fontsize=13, fontweight='bold')
ax1.legend(fontsize=10, loc='upper left')
ax1.grid(True, alpha=0.3)
ax1.set_xlim(0, 25)

# ── Bottom plot: Proportion of serious reports ──
proportion_serious = serious_reports / total_reports
colors = ['orange' if p > 0.5 else 'green' for p in proportion_serious]
ax2.bar(months, proportion_serious * 100, color=colors, alpha=0.7, edgecolor='black')
ax2.axhline(y=50, color='red', linestyle='--', linewidth=2, label='50% threshold')
ax2.set_xlabel('Month', fontsize=11, fontweight='bold')
ax2.set_ylabel('Serious Reports (%)', fontsize=11, fontweight='bold')
ax2.set_title('Proportion of Serious Adverse Events', fontsize=13, fontweight='bold')
ax2.legend(fontsize=10)
ax2.set_ylim(0, 100)
ax2.grid(axis='y', alpha=0.3)

plt.tight_layout()
plt.savefig('adr_trend_analysis.png', dpi=300)
plt.show()
```

Figure 5.11: Trends in Adverse Drug Reaction Reports and the Proportion of Serious Adverse Events Over 24 Months

5.7.2.2 Plot Not Showing

One of the most common problems encountered by beginners while working with Matplotlib is that the program executes without displaying any graph. In most cases, the code runs successfully, but no plot window appears. This issue is commonly referred to as the "Plot Not Showing" problem.

The most frequent cause is forgetting to call the plt.show() function, which instructs Matplotlib to display the figure. Other possible causes include closing the figure before displaying it, using the wrong backend, or running the program in an environment that does not automatically render plots.

Understanding the reasons behind this issue and knowing how to resolve it enables users to display graphs correctly and continue their data visualization tasks without interruption.

"Plot Not Showing" Error

The "Plot Not Showing" problem occurs when a Python program executes successfully but no graph or figure window appears. Unlike syntax or runtime errors, Python usually does not display an error message. Instead, the program finishes execution without showing the plot.

```
# Problem: You create a plot but no window opens

# Solution 1: Add plt.show() at the end
plt.plot([1, 2, 3], [1, 4, 9])
plt.show()  # <-- Essential for interactive use

# Solution 2: Check your backend (Jupyter vs script)
# In Jupyter notebooks, add to first cell:
%matplotlib inline

# In Spyder or command-line script, plt.show() is required
```

5.7.2.3 Figure Quality Issues

High-quality figures are essential for effective scientific communication. In pharmaceutical sciences, graphs are used to present experimental data such as dissolution profiles, calibration curves, pharmacokinetic studies, stability testing, and clinical trial results. A poorly designed figure may appear blurred, pixelated, difficult to read, or unsuitable for publication, reducing the clarity and credibility of the presented data.

Figure quality issues in Matplotlib usually arise from using a low resolution, inappropriate figure size, small font sizes, poor color choices, or saving images in unsuitable file formats. Understanding these issues and applying the appropriate solutions helps produce clear, professional, and publication-quality figures.

Figure Quality Issues

Figure quality issues refer to problems that reduce the visual clarity, readability, or publication suitability of a graph. These issues may include blurry images, low resolution, overlapping labels, distorted proportions, unreadable text, or improper image formats.

Such problems become particularly noticeable when figures are inserted into research papers, dissertations, presentations, or textbooks.

```
# Problem: Figure looks pixelated or blurry when printed
# Solution: Save with high DPI and/or vector format

# Low DPI (default 100 — too low for print)
plt.savefig('plot.png')  # □

# High DPI (300 is publication standard)
plt.savefig('plot.png', dpi=300)  # □

# Vector format (infinitely scalable)
plt.savefig('plot.pdf', bbox_inches='tight')  # □ Best for submissions
```

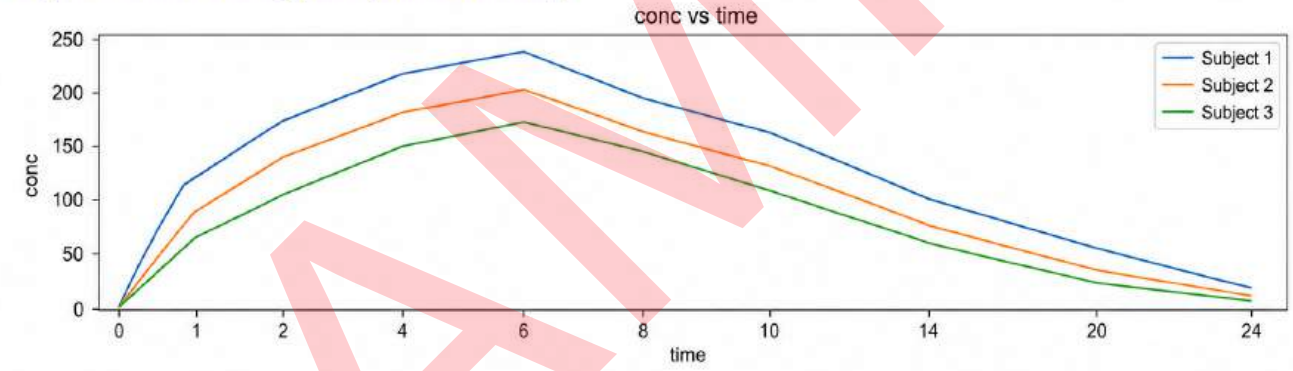
```
import pandas as pd
import matplotlib.pyplot as plt

df = pd.read_csv('pk_data.csv')  # Has columns: time, subject, conc

# ——— Pandas (quick, but limited control) ————————————————
df.plot(x='time', y='conc')
plt.show()

# ——— Matplotlib (more control) ————————————————————————
fig, ax = plt.subplots(figsize=(10, 6))
for subject in df['subject'].unique():
    subject_data = df[df['subject'] == subject]
    ax.plot(subject_data['time'], subject_data['conc'], marker='o', label=subject)
ax.set_xlabel('Time (hours)', fontsize=12, fontweight='bold')
ax.set_ylabel('Concentration (ng/mL)', fontsize=12, fontweight='bold')
ax.set_title('PK Curves by Subject', fontsize=13, fontweight='bold')
ax.legend()
ax.grid(True, alpha=0.3)
plt.tight_layout()
plt.show()
```

Output 1: Pandas Plotting (Quick, Limited Control)



Output 2: Matplotlib Plotting (More Control)

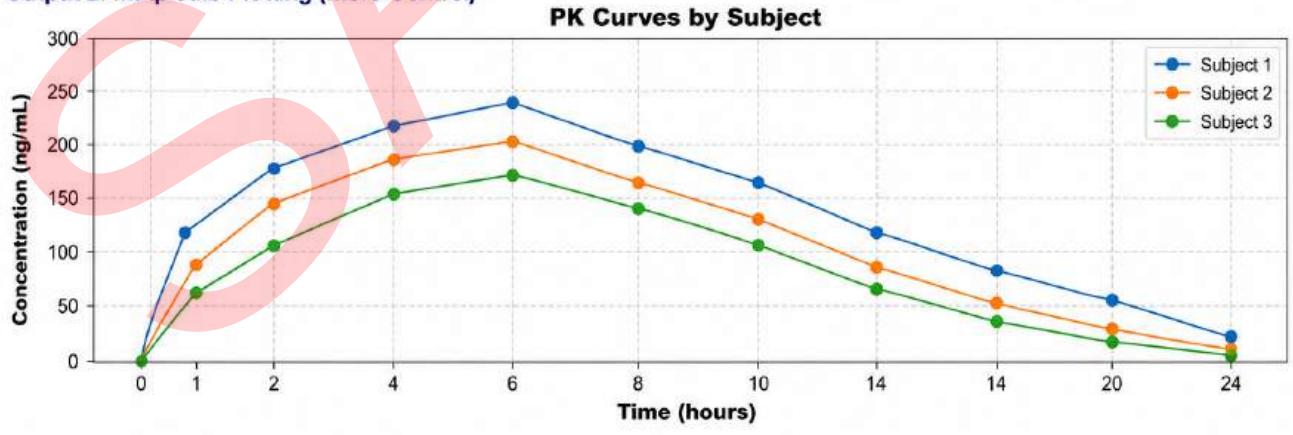


Figure 5.22: Comparison of Pharmacokinetic Curve Plotting Using Pandas and Matplotlib

Author Profile

DR. LATIKA SHARMA

Dr. Latika Sharma holds a B.Tech degree, an M.Tech from the National Institute of Technology (NIT), Kurukshetra, and a Ph.D. in Computer Science. With over a decade of academic and research experience, her work focuses on Artificial Intelligence, Machine Learning, Deep Learning, Computer Vision, Medical Image Analysis, Biomedical Signal Processing, Data Science, and Intelligent Healthcare Systems. Her research is particularly centered on the development of AI-driven frameworks for neurological disorder diagnosis, EEG signal analysis, epileptic seizure prediction, and multimodal medical data analytics. She has authored numerous research publications in reputed journals and international conferences and has contributed to several patents and innovative technology solutions in collaboration with multidisciplinary research teams.

DR. ANUJ PATHAK

Dr. Anuj Pathak, M.Pharm., Ph.D., is an Associate Professor at KIET Deemed to be University, Ghaziabad, with over 20 years of experience in pharmaceutical, cosmetic, and food product development. He has worked with leading organizations, including Sun Pharma, Glenmark, Ajanta Research Centre, and Krishnapath Industrial Research Consultancy Foundation. His expertise encompasses pharmaceutical formulations, novel drug delivery systems, cosmetic product innovation, nutraceuticals, industrial consultancy, and intellectual property. Dr. Pathak has authored more than 50 research publications and holds 35 patents, reflecting his significant contributions to research, innovation, and product development.

DR. PEEYUSH JAISWAL

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 preparations. 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.

MR. SHUBHAM KUMAR

MR. Shubham Kumar, M.Pharm. (Pharmaceutics), is a graduate with academic specialization in pharmaceutics and research interests in novel drug delivery systems and formulation development. He has contributed to research publications, review articles, and academic book chapters in the field of pharmaceutical sciences. His areas of interest include pharmaceutical formulation, drug delivery technologies, and evidence-based research, with a commitment to advancing pharmaceutical education and innovative healthcare solutions.

PHARMACY INDIA
Street No.-4, Dayalpuram, Khatauli, Muzaffarnagar, 251201
8171313561, 8006781759
pharmacyindia24@gmail.com
Pharmacyindia.co.in

NOW WE ARE AVAILABLE ON
Flipkart amazon

ISBN : 978-81-689235-2-2
9 788168 923522
PRICE:- ₹325.00/-