A STUDENT'S READING GUIDE

Co-funded by the
Erasmus+ Programme
of the European Union

ICT-4MRPQ

INTRODUCTION TO PROGRAMMING FOR DATA ANALYTICS FOR: POSTGRADUATE DIPLOMA IN BUSINESS INTELLIGENCE AND DATA ANALYTICS (PBDA) STUDENTS

Abdal Kasule, PhD
Prof. Samali Violet Mlay, PhD

Title: Introduction To Programming For Data Analytics, A Student's Reading Guide
For: Postgraduate Diploma In Business Intelligence And Data Analytics (PBDA) Students

Copyright: © 2025 All rights reserved - Strengthening Capacity for ICT Usage and Policy Reforms for Relevant and Quality Masters Research Process Management in Uganda's HEIs (ICT-4MRPQ)

Funding and support: This book was produced with funding from the European Union, through the Erasmus+ programme, and it's a deliverable of the ICT-4MRPQ Project. Its content is the sole responsibility of the authors and do not reflect the views of the European Union.

Authors: Abdal Kasule, PhD, Prof. Samali Violet Mlay, PhD

Editing: RSET – Associação Técnico-científica para o Desenvolvimento, Lisbon Portugal

Design: Teresa Vaz

Distribution: This material is intended for free distribution only and not be sold or used for commercial purposes.

Further information: Find out more about the project at https://ict-4mrpq.uccsri.com/

ISBN:

Title: Introduction To Programming For Data Analytics, A Student's Reading Guide
For: Postgraduate Diploma In Business Intelligence And Data Analytics (PBDA) Students

Copyright: © 2025 All rights reserved - Strengthening Capacity for ICT Usage and Policy Reforms for Relevant and Quality Masters Research Process Management in Uganda's HEIs (ICT-4MRPQ)

Funding and support: This book was produced with funding from the European Union, through the Erasmus+ programme, and it's a deliverable of the ICT-4MRPQ Project. Its content is the sole responsibility of the authors and do not reflect the views of the European Union.

Authors: Abdal Kasule, PhD, Prof. Samali Violet Mlay, PhD

Editing: RSET – Associação Técnico-científica para o Desenvolvimento, Lisbon Portugal

Design: Teresa Vaz

Distribution: This material is intended for free distribution only and not be sold or used for commercial purposes.

Further information: Find out more about the project at https://ict-4mrpq.uccsri.com/

ISBN:

MAKERERE UNIVERSITY BUSINESS SCHOOL

FACULTY OF COMPUTING AND INFORMATICS

DEPARTMENT OF APPLIED COMPUTING

AND INFORMATION TECHNOLOGY

INTRODUCTION TO PROGRAMMING FOR DATA ANALYTICS

A STUDENT'S READING GUIDE

For:

POSTGRADUATE DIPLOMA IN BUSINESS INTELLIGENCE AND DATA ANALYTICS (PBDA)

PROGRAMME

November, 2025

This student guide is a product of the ICT-4MRPQ Project,

ICT-4MRPQ

Co-funded by the European Union

A project implemented by

Co-funded by the Erasmus+ Programme of the European Union

Table of Contents

PROGRAMMING FOR DATA ANALYTICS....1
TOPIC ONE: INTRODUCTION TO PROGRAMMING FOR DATA ANALYTICS....1
1.1. What is Programming?....1
1.1.1. Definition:....1
1.1.2. Purpose of Programming:....1
1.1.3. Example:....1
1.2 Overview of Data Analytics Programming Languages....1
1.2.1 Popular Languages:....1
1.2.2. Strengths....2
1.3 Characteristic Features of Programming Languages for Data Analytics....2
1.4 Differences Between Data Analytics and Conventional Programming Languages....2
1.5 Data Analytics Programming IDEs....3
1.5.1 Popular IDEs:....3
1.5.2 Key Features of IDEs for Data Analytics:....3
1.6. End of Topic One Exercises: Introduction to Programming for Data Analytics....3
1.6.1. Conceptual Questions....3
1.6.2. Analytical Questions....4
1.6.3. Practical / Research-Based Questions....4
1.6.4. Reflection....4
1.7. Topic One Recommended Reading List....4
TOPIC TWO: INTRODUCTION TO PYTHON FOR DATA ANALYTICS....5
2.1 Definition:....5
2.2 Data Structures in Python....5
2.2.1 Lists....5
2.2.2 Tuples....5
2.2.3 Dictionaries....6
2.2.4 Sets....6
2.2.5 Arrays....6
2.2.6 Vectors....6
2.3. Identifiers, Objects, Assignment Statements, and Operators....6
2.3.1 Identifiers....6
2.3.2 Objects....7
2.3.3 Assignment Statements....7
2.3.4 Operators....7
2.4. Strings....7
2.5. Python Libraries for Data Analytics....7
2.5.1 Pandas Series....7
2.5.2 Pandas DataFrame....7

2.6. Topic two Summary....8
2.7 End of Topic two Exercises: Introduction to Python for Data Analytics....8
2.7.1. Conceptual Questions....8
2.7.2. Analytical Questions....8
2.7.3. Practical / Hands-On Exercises....8
2.7.4. Reflection / Mini-Project....9
2.8. Topic Two Recommended Reading:....9
TOPIC THREE: CONTROL FLOW AND FUNCTIONS IN PYTHON....10
3.1. Introduction to Control Flow....10
3.2. Conditionals....10
3.2.1 If Statement....10
3.2.2 If-Else Statement....10
3.2.3 If-Elif-Else Statement....11
3.3. Loops....11
3.3.1 For Loop....11
3.3.2 While Loop....11
3.3.3 Do-While Loop Concept....11
3.3.4 Continue and Break....12
3.4. Functions....12
3.5. Iterators and Generators....12
3.6. Taking Input....13
3.7. File Handling....13
3.7.1 Reading Files....13
3.7.2 Writing Files....13
3.8. Topic Three Summary....13
3.9. End of Topic Three Exercises: Control Flow and Functions in Python....13
3.9.1. Conceptual Questions....13
3.9.2. Analytical Questions....14
3.9.3. Practical / Coding Exercises....14
3.9.4. Mini-Project: Combined Control Flow and Functions....14
3.10 Topic Three References:....14
TOPIC FOUR: PROGRAMMING FRAMEWORKS AND LIBRARIES....15
4.1. Introduction to Frameworks and Libraries....15
4.1.1. Definition:....15
4.1.2. Importance in Data Analytics:....15
4.1.3. Examples of Common Libraries/Frameworks in Python for Data Analytics:....15
4.1.4. Benefits:....15
4.2. Core Libraries for Data Analytics....15
4.2.1 NumPy....15
4.2.2 Pandas....16
4.2.3 SciPy....16

4.2.4 Scikit-learn....16
4.2.5 Matplotlib....17
4.2.6 Seaborn....17
4.3. Deep Learning Frameworks....18
4.3.1 PyTorch....18
4.3.2 TensorFlow....18
4.3.3 Keras....18
4.4. Topic Four Summary....19
4.5 End of Topic Four Exercises: Programming Frameworks and Libraries....19
4.5.1. Conceptual Questions....19
4.5.2. Analytical Questions....19
4.5.3. Practical / Hands-On Exercises....19
4.5.4 Mini-Project: Combining Libraries....20
4.6 Topic Four References:....20

PROGRAMMING FOR DATA ANALYTICS

TOPIC ONE:

INTRODUCTION TO PROGRAMMING FOR DATA ANALYTICS

1.1. What is Programming?

1.1.1. Definition:

Programming is the process of designing, writing, testing, and maintaining instructions (called *code*) that a computer can execute to perform specific tasks. These instructions are written in a programming language that the computer interprets or compiles.

1.1.2. Purpose of Programming:

Programming enables the creation of software and systems that automate tasks, manipulate data, and interact with hardware. Some applications include:

- **Software development:** Creating desktop, web, and mobile applications.
- **Machine control:** Operating hardware like robots, IoT devices, and embedded systems.
- **Data analysis:** Automating data processing, statistical analysis, and reporting.
- **Problem solving:** Designing algorithms to solve complex problems efficiently.

1.1.3. Example:

A Python script that reads a dataset, calculates summary statistics, and visualizes the results automatically.

1.2 Overview of Data Analytics Programming Languages

Data analytics programming languages are specialized languages designed to process, analyze, and visualize data efficiently. These languages provide built-in functions, libraries, and frameworks that simplify common data analysis tasks.

1.2.1 Popular Languages:

1. **Python:**
    - Widely used due to simplicity and readability.
    - Rich libraries: Pandas (data manipulation), NumPy (numerical computing), Matplotlib/Seaborn (visualization), Scikit-learn (machine learning).
2. **R:**
    - Strong in statistics and data visualization.
    - Ideal for academic research and statistical modeling.

3. **SQL (Structured Query Language):**
    - Used for querying and managing relational databases.
    - Supports aggregation, filtering, joining, and data extraction.
4. **SAS:**
    - Common in finance, healthcare, and corporate analytics.
    - Known for robustness and enterprise-level data management.
5. **Julia:**
    - Emerging language for high-performance numerical and scientific computing.
6. **Scala (with Apache Spark):**
    - Suited for big data analytics and distributed computing.

1.2.2. Strengths

Each language has unique strengths depending on:

- Data size (small vs. big data)
- Purpose (statistical analysis, machine learning, visualization)
- Environment (academic, enterprise, cloud)

1.3 Characteristic Features of Programming Languages for Data Analytics

Data analytics languages have features tailored for data-focused tasks:

1. **Rich Libraries and Packages:** Pre-built functions for cleaning, transforming, analyzing, and visualizing data.
2. **Vectorized Operations:** Perform calculations efficiently on arrays or data frames without explicit loops.
3. **Interactive Environments:** Tools like Jupyter Notebook (Python) and RStudio enable exploratory data analysis.
4. **Data Integration:** Connect to databases, cloud storage, and APIs.
5. **Built-in Statistical and ML Tools:** Simplify regression, classification, clustering, and predictive modeling.
6. **Scalability:** Support for parallel and distributed computing to handle large datasets.

1.4 Differences Between Data Analytics and Conventional Programming Languages

Feature	Conventional Programming	Data Analytics Programming
Purpose	General-purpose apps, systems, games	Data processing, analysis, visualization
Libraries	Limited; must implement algorithms manually	Rich pre-built statistical and ML libraries
Environments	Compilers, IDEs (VS, Eclipse)	Interactive environments (Jupyter, RStudio)

Focus	Performance, control structures, application architecture	Data manipulation, insight generation, exploratory analysis
Typical Use	Software engineering, systems development	Business intelligence, data science, research

1.5 Data Analytics Programming IDEs

Integrated Development Environments (IDEs) enhance programming efficiency by providing tools for writing, testing, debugging, and visualizing code.

1.5.1 Popular IDEs:

- **Python:**
  - *Jupyter Notebook / JupyterLab*: Interactive, ideal for exploratory analysis and visualizations.
  - *Spyder*: Combines code editor, console, and data inspection tools.
- **R:**
  - *RStudio*: Comprehensive IDE for statistical analysis, plotting, and package management.
- **SQL:**
  - *DBeaver*: Multi-database query and management tool.
  - *MySQL Workbench*: GUI for MySQL database design and queries.

1.5.2 Key Features of IDEs for Data Analytics:

- Syntax highlighting
- Code completion
- Debugging tools
- Integrated visualization
- Variable inspection and memory management

1.6. End of Topic One Exercises: Introduction to Programming for Data Analytics

1.6.1. Conceptual Questions

1. Define programming and explain its main purposes in the context of data analytics. (5 marks)
2. List three programming languages commonly used for data analytics and explain one strength of each. (6 marks)
3. Differentiate between conventional programming languages and data analytics programming languages with examples. (5 marks)
4. Explain the importance of vectorized operations in data analytics programming. Give an example in Python or R. (4 marks)

1.6.2. Analytical Questions

5. Suppose you are given a dataset containing sales records for a retail store. Discuss which programming language you would choose for:
   a) Performing complex statistical analysis.
   b) Querying large amounts of transactional data in a relational database.
   c) Visualizing sales trends interactively for management.
   Justify your choices. (6 marks)

6. Compare the benefits of using Jupyter Notebook versus RStudio for a data analytics project. (4 marks)

1.6.3. Practical / Research-Based Questions

7. Research and list two emerging programming languages or tools for data analytics. Explain what unique feature each offers compared to traditional languages like Python or R. (4 marks)

8. Identify one scenario in a real-world organization where automation of data analysis using programming can significantly improve decision-making. Describe the expected outcome. (5 marks)

1.6.4. Reflection

9. Reflect on how learning data analytics programming languages can contribute to career growth in business intelligence, research, or technology. Provide at least three points. (5 marks)

1.7. Topic One Recommended Reading List

1. McKinney, W. (2018). *Python for Data Analysis*. O'Reilly Media.
2. Wickham, H., & Grolemund, G. (2017). *R for Data Science*. O'Reilly Media.
3. VanderPlas, J. (2016). *Python Data Science Handbook*. O'Reilly Media.
4. James, G., Witten, D., Hastie, T., & Tibshirani, R. (2013). *An Introduction to Statistical Learning*. Springer.
5. Sebesta, R. W. (2016). *Programming Language Concepts*. Pearson.
6. Dehghani, M. (2020). *Mastering SQL for Data Analytics*. Packt Publishing.
7. Grus, J. (2019). *Data Science from Scratch*. O'Reilly Media.

INTRODUCTION TO PYTHON FOR DATA ANALYTICS

2.1 Definition:

Python is a high-level, versatile, and interpreted programming language widely used in data analytics. Its simple syntax, readability, and extensive ecosystem of libraries make it ideal for beginners and professionals.

Key Features for Data Analytics:

- **Ease of Learning:** Simple syntax and readable code.
- **Rich Libraries:** Pandas (data manipulation), NumPy (numerical computation), Matplotlib & Seaborn (visualization), Scikit-learn (machine learning).
- **Integration:** Connects easily with databases, big data platforms (e.g., Hadoop, Spark), and APIs.
- **Community Support:** Strong open-source community provides tutorials, packages, and problem-solving support.
- **Applications:** Data cleaning, exploration, modeling, visualization, and predictive analytics.

2.2 Data Structures in Python

Python provides built-in data structures to efficiently organize and manage data. Common data structures include:

2.2.1 Lists

- Ordered, mutable collections.
- Can hold mixed data types.
- Example:

```
my_list = [1, 'apple', 3.14, True]
my_list.append('new item')
print(my_list[1])  # Output: apple
```

2.2.2 Tuples

- Ordered, immutable sequences.
- Often used for grouping related data.
- Example:

```
my_tuple = (10, 20, 30)
print(my_tuple[1])  # Output: 20
# my_tuple[1] = 40  # Error: tuples are immutable
```

2.2.3 Dictionaries

- Key-value pairs allowing fast lookups.
- Example:

```
my_dict = {'name': 'Alice', 'age': 25}
print(my_dict['name'])  # Output: Alice
my_dict['city'] = 'New York'
```

2.2.4 Sets

- Unordered collections of unique elements.
- Useful for membership tests and removing duplicates.
- Example:

```
my_set = {1, 2, 2, 3, 4}
print(my_set)  # Output: {1, 2, 3, 4}
my_set.add(5)
```

2.2.5 Arrays

- Fixed-type collections stored in contiguous memory locations.
- Python provides the array module, but lists are more commonly used.
- Example:

```
import array
arr = array.array('i', [1, 2, 3, 4])
print(arr[2])  # Output: 3
```

2.2.6 Vectors

- Represented using lists or NumPy arrays for numerical computations.
- Example:

```
import numpy as np
vector = np.array([1, 2, 3])
print(vector + 5)  # Output: [6 7 8]
```

2.3. Identifiers, Objects, Assignment Statements, and Operators

2.3.1 Identifiers

- Names for variables, functions, classes, or objects.
- Rules: Start with a letter or underscore, contain letters, digits, or underscores.

2.3.2 Objects

- Any data stored in memory (numbers, strings, lists, functions).
- Everything in Python is an object.

2.3.3 Assignment Statements

- Assign a value to an identifier using =.

```
x = 5  # Assigns integer object 5 to variable x
```

2.3.4 Operators

- **Arithmetic:** +, -, *, /
- **Comparison:** ==, <, >
- **Logical:** and, or, not

```
3 + 4 * 2  # Output: 11 (multiplication has higher precedence)
(3 + 4) * 2  # Output: 14
```

2.4. Strings

- Immutable sequences of characters.
- Rich methods for manipulation.

```
s = "Hello, World!"
print(s.lower())  # Output: hello, world!
print(s[7:12])    # Output: World
```

2.5. Python Libraries for Data Analytics

2.5.1 Pandas Series

- One-dimensional labeled array, like a spreadsheet column.

```
import pandas as pd
data = pd.Series([10, 20, 30, 40], index=['a', 'b', 'c', 'd'])
print(data['b'])  # Output: 20
print(data * 2)
```

2.5.2 Pandas DataFrame

- Two-dimensional, size-mutable, heterogeneous tabular data structure.

```
data = {'Name': ['Alice', 'Bob', 'Charlie'], 'Age': [25, 30, 35]}
df = pd.DataFrame(data)
print(df)
print(df['Age'] > 28)
```

2.6. Topic two Summary

Python is a versatile tool for data analytics due to:

- **Ease of learning and readability**
- **Rich ecosystem of libraries**
- **Support for multiple data structures** (lists, tuples, dictionaries, sets, arrays, vectors)
- **Integration with data analytics workflows** using Pandas, NumPy, and visualization libraries

2.7 End of Topic two Exercises: Introduction to Python for Data Analytics

2.7.1. Conceptual Questions

1. Define Python and explain why it is widely used in data analytics. (4 marks)
2. List four key features of Python that make it suitable for data analytics and give one example for each. (8 marks)
3. Differentiate between lists, tuples, dictionaries, and sets in Python, giving one example of when each might be used. (8 marks)
4. Explain the difference between mutable and immutable data structures in Python. Provide two examples of each. (4 marks)

2.7.2. Analytical Questions

5. Explain the following concepts with examples in Python:
   a) Identifier
   b) Object
   c) Assignment statement
   d) Operators (arithmetic, comparison, logical)
   (8 marks)
6. Describe how Python integrates with data analytics workflows, mentioning at least two libraries and their roles. (4 marks)
7. Explain the difference between a Pandas Series and a Pandas DataFrame. Give one example of when you would use each. (4 marks)

2.7.3. Practical / Hands-On Exercises

8. **Python Data Structures**
   a) Create a list of five different fruits and add one more fruit to the list. Print the updated list.
   b) Create a tuple of five integers and attempt to change one of the values. Explain the result.
   c) Create a dictionary with three students' names and their ages. Add a fourth student and print the dictionary.
   d) Create a set with the numbers {1, 2, 2, 3, 4} and add a new number. Print the set. (10 marks)
9. **Strings**
   a) Create a string with your full name. Convert it to lowercase and uppercase.
   b) Extract and print only the first name from the string using slicing.
   c) Count the number of characters in your name. (6 marks)

10. Pandas Series and DataFrame

a) Create a Pandas Series representing the marks of five students in a subject. Multiply all marks by 2 and display the result.

b) Create a Pandas DataFrame containing the following data:

Name	Age	Grade
Alice	25	A
Bob	30	B
Charlie	28	A

i) Print the names of students older than 26.

ii) Add a new column called 'City' and assign a city to each student.

iii) Display the final DataFrame. (10 marks)

2.7.4. Reflection / Mini-Project

11. Mini-Project: Using Python, write a program that:

a) Creates a list of 10 random integers between 1 and 100.

b) Calculates the maximum, minimum, and average of the numbers.

c) Creates a Pandas DataFrame with the numbers and a column indicating whether each number is "Even" or "Odd".

d) Print the DataFrame and summarize the results in 3–5 sentences. (10 marks)

2.8. Topic Two Recommended Reading:

1. McKinney, W. (2018). *Python for Data Analysis*. O'Reilly Media.
2. VanderPlas, J. (2016). *Python Data Science Handbook*. O'Reilly Media.
3. Lutz, M. (2013). *Learning Python*. O'Reilly Media.
4. Downey, A. (2015). *Think Python*. Green Tea Press.

TOPIC THREE:

CONTROL FLOW AND FUNCTIONS IN PYTHON

3.1. Introduction to Control Flow

Definition:
Control flow in Python refers to the order in which individual statements, instructions, or function calls are executed in a program. It determines how a program makes decisions, repeats tasks, and organizes modular code execution.

Key Constructs:

- **Conditional statements:** if, elif, else
- **Loops:** for, while
- **Loop control statements:** break, continue, pass
- **Function calls:** Modular execution of reusable code

Purpose:
Control flow enables decision-making, iteration, and modularity, allowing programs to respond dynamically to inputs and conditions.

3.2. Conditionals

Definition:
Conditionals allow a program to execute different blocks of code depending on whether a condition evaluates to True or False.

Types:

- if
- if-else
- if-elif-else

3.2.1 If Statement

Executes a code block only when a specified condition is True.

```
x = 10
if x > 5:
    print("x is greater than 5")  # Output: x is greater than 5
```

3.2.2 If-Else Statement

Provides two execution paths based on a Boolean condition.

```
x = 3
if x > 5:
```

```
    print("x is greater than 5")
else:
    print("x is not greater than 5")  # Output: x is not greater than 5
```

3.2.3 If-Elif-Else Statement

Handles multiple sequential conditions.

```
x = 15
if x < 10:
    print("x is less than 10")
elif x < 20:
    print("x is between 10 and 19")  # Output: x is between 10 and 19
else:
    print("x is 20 or more")
```

3.3. Loops

Loops enable repeated execution of code blocks, automating repetitive tasks.

3.3.1 For Loop

Iterates over sequences like lists, tuples, or ranges.

```
for i in range(5):
    print(i)  # Output: 0 1 2 3 4
```

3.3.2 While Loop

Executes as long as a condition is True.

```
count = 0
while count < 5:
    print(count)
    count += 1
```

3.3.3 Do-While Loop Concept

Python does not have a built-in do-while loop, but it can be simulated using while True with a break statement.

```
count = 0
while True:
    print(count)
    count += 1
    if count >= 5:
        break
```

3.3.4 Continue and Break

- **continue:** Skips the current iteration and moves to the next.
- **break:** Exits the loop immediately.

```
for i in range(5):
    if i == 2:
        continue  # Skip 2
    if i == 4:
        break     # Stop loop
    print(i)
# Output: 0 1 3
```

3.4. Functions

Definition:
A function is a reusable block of code defined with def that may take parameters and optionally return a value.

```
def greet(name):
    return f"Hello, {name}!"
print(greet("Alice"))  # Output: Hello, Alice!
```

Information Passing:

- **Arguments:** Values passed to a function.
- **Return Values:** Values sent back from the function.

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


result = add(3, 5)
print(result)  # Output: 8
```

3.5. Iterators and Generators

- **Iterator:** An object that allows traversal of a collection one element at a time.
- **Generator:** A function using yield to produce values lazily, saving memory.

```
def countdown(n):
    while n > 0:
        yield n
        n -= 1
for number in countdown(3):
```

```
    print(number)
# Output: 3 2 1
```

3.6. Taking Input

- Use input() to read user input as a string.
- Convert to other types as needed.

```
name = input("Enter your name: ")
print(f"Hello, {name}!")
```

3.7. File Handling

3.7.1 Reading Files

```
with open('data.txt', 'r') as file:
    content = file.read()
print(content)
```

3.7.2 Writing Files

```
with open('output.txt', 'w') as file:
    file.write("Hello, file!\n")
```

3.8. Topic Three Summary

- **Control Flow** manages program execution based on conditions and loops.
- **Conditionals** enable decision-making (if, if-else, if-elif-else).
- **Loops** automate repetitive tasks (for, while, simulated do-while).
- **Loop Control** (break, continue) provides flexible iteration.
- **Functions** modularize code, allowing input/output through arguments and return values.
- **Iterators and Generators** support efficient data traversal.
- **Input/Output** allows user interaction and file handling.

3.9. End of Topic Three Exercises: Control Flow and Functions in Python

3.9.1. Conceptual Questions

1. Define control flow in Python and explain its purpose in data analytics programs. (4 marks)
2. Differentiate between if, if-else, and if-elif-else statements with examples. (6 marks)
3. Explain the difference between for and while loops in Python. Give one scenario where each would be appropriate. (6 marks)
4. Describe the purpose of break and continue statements in loops and provide an example of each. (4 marks)
5. Define a function in Python. Explain the difference between arguments and return values with an example. (4 marks)

3.9.2. Analytical Questions

6. Explain the concepts of iterators and generators in Python. Why are generators useful for handling large datasets? (4 marks)
7. Describe the steps to take user input in Python and convert it to an integer or float. Provide a brief code snippet. (4 marks)
8. Explain how file reading and writing works in Python. Give one example use case of each in data analytics. (4 marks)

3.9.3. Practical / Coding Exercises

9. Conditionals

a) Write a Python program that takes a number from the user and prints whether it is positive, negative, or zero.
b) Modify the program to also print whether the number is even or odd. (6 marks)

10. Loops

a) Write a Python program using a for loop to print all numbers from 1 to 20 that are divisible by 3.
b) Write the same program using a while loop. (6 marks)

11. Loop Control

a) Write a for loop that prints numbers from 0 to 10, but skips 5 and stops entirely when it reaches 8. (4 marks)

12. Functions

a) Define a function calculate_area(radius) that returns the area of a circle given its radius.
b) Use the function to calculate and print the areas of circles with radii 3, 5, and 7. (6 marks)

13. Iterators and Generators

a) Create a generator function fibonacci(n) that yields the first n Fibonacci numbers.
b) Use a for loop to print the Fibonacci numbers generated for n = 10. (6 marks)

14. File Handling

a) Create a text file students.txt and write the names of five students, one per line.
b) Read the file and print each name in uppercase. (6 marks)

3.9.4. Mini-Project: Combined Control Flow and Functions

Write a Python program that:
a) Reads a list of integers from the user (input separated by commas).
b) Uses a function to calculate the sum, maximum, and minimum values.
c) Prints all numbers greater than the average.
d) Writes the results to a text file named results.txt. (10 marks)

3.10 Topic Three References:

1. Lutz, M. (2013). *Learning Python*. O'Reilly Media.
2. Downey, A. (2015). *Think Python*. Green Tea Press.
3. Sweigart, A. (2015). *Automate the Boring Stuff with Python*. No Starch Press.
4. McKinney, W. (2018). *Python for Data Analysis*. O'Reilly Media.

TOPIC FOUR:

PROGRAMMING FRAMEWORKS AND LIBRARIES

4.1. Introduction to Frameworks and Libraries

4.1.1. Definition:

- **Libraries** are collections of pre-written code that provide specific functionalities or tools, which programmers can use to perform tasks without writing code from scratch.
- **Frameworks** are more structured, providing a foundation and guidelines for building applications, often enforcing certain patterns or workflows.

4.1.2. Importance in Data Analytics:

Python offers a rich ecosystem of libraries and frameworks that simplify data manipulation, statistical analysis, machine learning, and visualization. Using these tools allows analysts to work efficiently, handling everything from data cleaning to building complex machine learning models.

4.1.3. Examples of Common Libraries/Frameworks in Python for Data Analytics:

- **Numerical computing:** NumPy, SciPy
- **Data manipulation:** Pandas
- **Visualization:** Matplotlib, Seaborn
- **Machine learning:** Scikit-learn
- **Deep learning:** TensorFlow, Keras, PyTorch

4.1.4. Benefits:

- Reduces coding effort
- Provides optimized, tested, and reusable functions
- Speeds up development and experimentation
- Facilitates complex computations and visualizations

4.2. Core Libraries for Data Analytics

4.2.1 NumPy

- Foundation for numerical computing in Python.
- Provides multi-dimensional arrays and fast mathematical operations.

Example:

```
import numpy as np

arr = np.array([1, 2, 3])

print(arr + 5)  # Output: [6 7 8]
```

Use Cases:

- Array manipulation
- Linear algebra
- Numerical simulations

4.2.2 Pandas

- Library for data manipulation and analysis.
- Key structures:
    - **Series:** One-dimensional labeled array
    - **DataFrame:** Two-dimensional labeled table

Example:

```
import pandas as pd
data = pd.DataFrame({'Name': ['Alice', 'Bob'], 'Age': [25, 30]})
print(data)
```

Use Cases:

- Data cleaning and preprocessing
- Aggregation, filtering, and joining datasets
- Time series analysis

4.2.3 SciPy

- Builds on NumPy and adds modules for:
    - Optimization
    - Integration
    - Interpolation
    - Linear algebra and statistics

Example:

```
from scipy import integrate
result, _ = integrate.quad(lambda x: x**2, 0, 1)
print(result)  # Output: 0.33333333333333337
```

Use Cases:

- Scientific and engineering computations
- Statistical analysis and signal processing

4.2.4 Scikit-learn

- Provides simple and efficient tools for machine learning.
- Supports classification, regression, clustering, and dimensionality reduction.

Example:

```
from sklearn.linear_model import LinearRegression
import numpy as np
X = np.array([[1], [2], [3]])
y = np.array([2, 4, 6])
model = LinearRegression()
model.fit(X, y)
print(model.predict([[4]]))  # Output: [8.]
```

Use Cases:

- Predictive modeling
- Feature engineering
- Model evaluation and selection

4.2.5 Matplotlib

- Comprehensive library for creating static, animated, and interactive visualizations.

Example:

```
import matplotlib.pyplot as plt
plt.plot([1, 2, 3], [4, 5, 6])
plt.show()
```

Use Cases:

- Line charts, bar plots, histograms
- Customizable visualizations for presentations and reports

4.2.6 Seaborn

- Built on top of Matplotlib, offers a higher-level interface for attractive statistical graphics.

Example:

```
import seaborn as sns
import matplotlib.pyplot as plt
tips = sns.load_dataset("tips")
sns.scatterplot(data=tips, x="total_bill", y="tip")
plt.show()
```

Use Cases:

- Statistical plots (boxplots, violin plots, heatmaps)
- Data exploration and visualization

4.3. Deep Learning Frameworks

4.3.1 PyTorch

- Open-source deep learning framework with dynamic computation graphs.

Example:

```
import torch
x = torch.tensor([1, 2, 3])
y = x + 5
print(y)  # Output: tensor([6, 7, 8])
```

Use Cases:

- Neural networks
- Research and experimentation in AI

4.3.2 TensorFlow

- Open-source platform for machine learning and deep learning.
- Supports large-scale computation and deployment.

Example:

```
import tensorflow as tf
a = tf.constant([1, 2, 3])
b = a + 5
print(b.numpy())  # Output: [6 7 8]
```

Use Cases:

- Production-ready ML models
- Deployment on cloud platforms

4.3.3 Keras

- High-level API for building and training neural networks.
- Runs on top of TensorFlow for easy experimentation.

Example:

```
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense
model = Sequential([
    Dense(10, activation='relu', input_shape=(5,)),
    Dense(1)
])
model.summary()
```

Use Cases:

- Rapid prototyping of deep learning models
- Image recognition, NLP, and time-series prediction

4.4. Topic Four Summary

Library / Framework	Purpose / Use Case
NumPy	Numerical computing, arrays, linear algebra
Pandas	Data manipulation, Series/DataFrame
SciPy	Scientific computing, integration, optimization
Scikit-learn	Machine learning, regression, classification
Matplotlib	Visualization (line, bar, histograms)
Seaborn	Statistical graphics, data visualization
PyTorch	Deep learning, dynamic computation graphs
TensorFlow	Deep learning, scalable ML models
Keras	High-level deep learning API, rapid prototyping

4.5 End of Topic Four Exercises: Programming Frameworks and Libraries

4.5.1. Conceptual Questions

1. Define a library and a framework in programming. Explain one key difference between them. (4 marks)
2. List five Python libraries commonly used in data analytics and briefly describe the primary purpose of each. (5 marks)
3. Explain the advantages of using libraries and frameworks in data analytics projects. Provide two practical examples. (4 marks)
4. Differentiate between PyTorch, TensorFlow, and Keras in terms of purpose and use case. (6 marks)

4.5.2. Analytical Questions

5. Explain how NumPy arrays differ from Python lists and why NumPy is preferred for numerical computations in data analytics. (4 marks)
6. Describe the role of Pandas Series and DataFrames in data analysis. Give an example scenario for using each. (4 marks)
7. Discuss the difference between Matplotlib and Seaborn for data visualization. When would you choose one over the other? (4 marks)
8. Explain how Scikit-learn supports machine learning workflows, from data preprocessing to model evaluation. (4 marks)

4.5.3. Practical / Hands-On Exercises

9. **NumPy**
   a) Create a NumPy array of integers from 1 to 10. Multiply all elements by 3 and print the result.
   b) Create a 2×3 NumPy array of random integers between 1 and 20 and compute the sum of each column. (6 marks)

10. Pandas

a) Create a Pandas DataFrame for five students containing columns: Name, Age, and Score.

b) Filter and display only the students with scores greater than 70.

c) Add a new column 'Grade' assigning 'Pass' if score ≥ 50 and 'Fail' otherwise. (6 marks)

11. Matplotlib & Seaborn

a) Using Matplotlib, plot a line chart of the scores of five students.

b) Using Seaborn, create a boxplot for the same scores. Include labels and title. (6 marks)

12. Scikit-learn

a) Create a simple linear regression model using Scikit-learn with the following data:
X = [[1], [2], [3], [4]] and y = [2, 4, 6, 8]

b) Predict the value of y when X = 5. (4 marks)

1.1.4 **Mini-Project: Combining Libraries**

Write a Python program that:

a) Creates a NumPy array of 20 random integers between 1 and 100.

b) Converts it to a Pandas DataFrame with a column 'Value'.

c) Calculates the mean, maximum, and minimum values and adds them as new rows in the DataFrame.

d) Visualizes the data using a Matplotlib bar chart and a Seaborn boxplot.

e) Optionally, save the DataFrame to a CSV file named data_summary.csv. (10 marks)

4.6 Topic Four References:

1. McKinney, W. (2018). *Python for Data Analysis*. O'Reilly Media.
2. VanderPlas, J. (2016). *Python Data Science Handbook*. O'Reilly Media.
3. Géron, A. (2019). *Hands-On Machine Learning with Scikit-Learn, Keras, and TensorFlow*. O'Reilly Media.
4. Chollet, F. (2018). *Deep Learning with Python*. Manning Publications.

Implemented by:

Scan me for a soft copy