lkhibra academy

100 Python Exercises with Solutions

100 تمرين با يثون

ملي تساليهم، الناس غادي يعيطولك هاكر

Part 1

Contents

Introduction

At Lkhibra Academy, we empower learners with practical programming skills essential for today's evolving tech industry. Whether you're new to coding or looking to advance your expertise, we make programming education accessible, engaging, and effective.

This book provides curated Python exercises bridging theory and real-world applications, enhancing your programming abilities and problem-solving skills.

We believe learning should be enjoyable and insightful—so get ready to learn, laugh, and succeed!

3lach lktab in English ?

We have chosen to present this book in English because it is the global standard in programming, the primary language for most technical resources, and a key requirement for accessing international career opportunities.

3lach lktab Fabor ?

This book is part of our paid Python Programming Training at Lkhibra Academy, but we've made it freely accessible because we believe quality education should be available to everyone, regardless of their background or circumstances, empowering all learners with essential Python programming skills.

If you want to learn Python and build real-world projects, explore our comprehensive training program: **Python Programming Training**

Lkhibra is MORE !

To support your journey beyond this book, we provide additional learning resources through:

- **Lkhibra Library** – Access a collection of books, guides, and materials on programming and technology: **Visit Lkhibra Library**.
- **Lkhibra Network** – Join our community where we host free learning sessions every month, helping learners connect, collaborate, and grow together: **Join our Network**.

For more information and resources, visit our website: **www.lkhibra.ma/ar/**.

Chapter 1

Fondamentals

1.1 Basic Syntax & Variables

1.1.1 Exercise 1: Declare and Print Variables

Objective: Understand how to declare and print variables in Python.

Task:

- Declare variables for an instructor's name, the number of students in a Python class, and the course name.
- Print them using a formatted string.

Expected Output Example:

```
The instructor is Alex, there are 30 students in the Python class!
```

1.1.2 Exercise 2: Swap Two Variables Without a Third Variable

Objective: Swap two values without using a temporary variable.

Task:

- Swap the values of `students_morning = 15` and `students_evening = 25` without using an extra variable.

Expected Output:

```
Before Swap: Morning Batch = 15, Evening Batch = 25
After Swap: Morning Batch = 25, Evening Batch = 15
```

1.1.3 Exercise 3: Assign Multiple Variables in One Line

Objective: Learn how to assign multiple variables in one line.

Task:

- Assign values to three variables representing the number of Python, Java, and AI students in a single line.

Expected Output:

```
Python = 25, Java = 18, AI = 12
```

1.1.4 Exercise 4: Check the Type of a Variable

Objective: Learn how to check variable types using `type()`.

Task:

- Declare different types of variables: an integer (age), a float (course rating), and a string (course name).
- Use `type()` to check their data types.

Expected Output:

```
21 is of type <class 'int'>
4.9 is of type <class 'float'>
Python Programming is of type <class 'str'>
```

1.1.5 Exercise 5: Concatenating Strings

Objective: Learn how to concatenate (combine) multiple strings in Python.

Task:

- Declare three separate string variables: instructor, academy name, and slogan.
- Concatenate them to form a full sentence.
- Print the combined string.

Expected Output:

```
The instructor at Lkhibra Academy says: "Learning Python is fun!"
```

1.2 Data Types & Conversions

1.2.1 Exercise 6: Convert String to Integer and Vice Versa

Objective: Learn how to convert between string and integer data types.

Task:

- Convert the string `"100"` into an integer.
- Convert the integer `42` into a string.
- Print both values along with their data types.

Expected Output:

```
Integer value: 100, Type: <class 'int'>
String value: 42, Type: <class 'str'>
```

1.2.2 Exercise 7: Convert Float to Integer and Vice Versa

Objective: Learn how to convert between float and integer values.

Task:

- Convert a float `9.75` into an integer.
- Convert an integer `50` into a float.
- Print both values along with their data types.

Expected Output:

```
Float to Int: 9, Type: <class 'int'>
Int to Float: 50.0, Type: <class 'float'>
```

1.2.3 Exercise 8: Convert a Boolean to an Integer

Objective: Learn how Boolean values convert to integers.

Task:

- Convert the Boolean values `True` and `False` into integers.
- Print their numerical values.

Expected Output:

```
True as an integer: 1
False as an integer: 0
```

1.2.4 Exercise 9: Convert List to a String and Back

Objective: Learn how to convert between lists and strings.

Task:

- Convert a list of words into a single string.
- Convert the string back into a list.

Expected Output:

```
List to String: Python, is, amazing
String to List: ['Python', 'is', 'amazing']
```

1.2.5 Exercise 10: Convert Dictionary Keys and Values to Lists

Objective: Learn how to extract dictionary keys and values as lists.

Task:

- Convert dictionary keys into a list.
- Convert dictionary values into a list.

Expected Output:

```
Keys: ['name', 'age', 'language']
Values: ['Lkhibra Academy', 5, 'Python']
```

1.3 Operators & Expressions

1.3.1 Exercise 11: Perform Arithmetic Operations

Objective: Learn how to use arithmetic operators in Python.

Task:

- Perform addition, subtraction, multiplication, division, and modulus operations.
- Print the results.

Expected Output:

```
Addition: 15
Subtraction: 5
Multiplication: 50
Division: 2.5
Modulus: 0
```

1.3.2 Exercise 12: Use Comparison Operators

Objective: Understand comparison operators in Python.

Task:

- Compare two numbers using comparison operators.
- Print the results.

Expected Output:

```
10 > 5: True
10 < 5: False
10 == 10: True
10 != 5: True
10 >= 5: True
10 <= 5: False
```

1.3.3 Exercise 13: Use Logical Operators

Objective: Learn how logical operators work in Python.

Task:

- Use `and`, `or`, and `not` to evaluate logical expressions.

Expected Output:

```
True and False: False
True or False: True
Not True: False
```

1.3.4 Exercise 14: Use Assignment Operators

Objective: Learn how assignment operators modify variables.

Task:

- Use different assignment operators to modify a variable.

Expected Output:

```
Initial Value: 10
After += : 15
After -= : 12
After *= : 24
After /= : 8.0
After %= : 0.0
```

1.3.5 Exercise 15: Use Bitwise Operators

Objective: Learn how to manipulate binary numbers using bitwise operators.

Task:

- Perform bitwise operations on integers.

Expected Output:

```
5 & 3 = 1
5 | 3 = 7
5 ^ 3 = 6
5 << 1 = 10
5 >> 1 = 2
```

1.4 Control Flow (Conditionals & Loops)

1.4.1 Exercise 16: Check if a Number is Even or Odd

Objective: Learn how to use the `if-else` statement in Python.

Task:

- Write a program that asks the user for a number.
- Check if the number is even or odd.
- Print the result.

Expected Output:

```
Enter a number: 7
7 is an odd number.
```

1.4.2 Exercise 17: Find the Largest Number

Objective: Learn how to compare values using `if-elif-else`.

Task:

- Take three numbers as input.
- Find and print the largest number.

Expected Output:

```
Enter three numbers: 5 12 9
The largest number is 12.
```

1.4.3 Exercise 18: Check if a Year is a Leap Year

Objective: Use conditional logic to determine leap years.

Task:

- Ask the user for a year.
- Check if it is a leap year.

Expected Output:

```
Enter a year: 2024
2024 is a leap year.
```

1.4.4 Exercise 19: Print Numbers from 1 to N

Objective: Learn how to use a `for` loop.

Task:

- Take an integer input `N`.
- Print numbers from 1 to `N`.

Expected Output:

```
Enter a number: 5
1 2 3 4 5
```

1.4.5 Exercise 20: Print Even Numbers up to N

Objective: Use loops with conditions.

Task:

- Take an integer input `N`.
- Print all even numbers from 1 to `N`.

Expected Output:

```
Enter a number: 10
2 4 6 8 10
```

1.4.6 Exercise 21: Sum of First N Natural Numbers

Objective: Learn how to use a loop to compute a sum.

Task:

- Take an integer input `N`.
- Compute the sum of numbers from 1 to `N`.

Expected Output:

```
Enter a number: 5
Sum of first 5 numbers: 15
```

1.4.7 Exercise 22: Factorial of a Number

Objective: Learn how to calculate factorial using loops.

Task:

- Take an integer input `N`.
- Compute `N!` (Factorial of `N`).

Expected Output:

```
Enter a number: 5
Factorial of 5 is 120.
```

1.4.8 Exercise 23: Reverse a Number

Objective: Use loops to reverse a number.

Task:

- Take an integer input.
- Reverse its digits and print the result.

Expected Output:

```
Enter a number: 1234
Reversed number: 4321
```

1.4.9 Exercise 24: Multiplication Table

Objective: Use loops to generate a multiplication table.

Task:

- Take an integer input `N`.
- Print its multiplication table up to 10.

Expected Output:

```
Enter a number: 7
7 x 1 = 7
7 x 2 = 14
...
7 x 10 = 70
```

1.4.10 Exercise 25: Count Digits in a Number

Objective: Use loops to count the digits in a number.

Task:

- Take an integer input.
- Count and print the number of digits.

Expected Output:

```
Enter a number: 56473
The number has 5 digits.
```

1.5 Functions & Scope

1.5.1 Exercise 26: Define and Call a Function

Objective: Learn how to define and call a function in Python.

Task:

- Define a function called `greet()` that prints `"Hello, Lkhibra Academy!"`.
- Call the function.

Expected Output:

```
Hello, Lkhibra Academy!
```

1.5.2 Exercise 27: Function with Parameters

Objective: Learn how to pass parameters to a function.

Task:

- Define a function `welcome(name)` that prints a personalized greeting.
- Call the function with different names.

Expected Output:

```
Hello, Sara! Welcome to Lkhibra Academy!
Hello, Youssef! Welcome to Lkhibra Academy!
```

1.5.3 Exercise 28: Function Returning a Value

Objective: Learn how a function can return a value.

Task:

- Define a function `square(num)` that returns the square of a number.
- Print the result.

Expected Output:

```
The square of 5 is 25.
```

1.5.4 Exercise 29: Function with Default Parameter

Objective: Learn how to use default parameters in functions.

Task:

- Define a function `greet(name="Student")` that greets a user.
- Call it with and without a parameter.

Expected Output:

```
Hello, Student!
Hello, Amine!
```

1.5.5 Exercise 30: Function Using Global Variable

Objective: Learn how functions interact with global variables.

Task:

- Declare a global variable `academy = "Lkhibra"`.
- Define a function that prints the value of the global variable.

Expected Output:

```
The best academy is Lkhibra!
```

1.5.6 Exercise 31: Function Using Local Variable

Objective: Learn about local variables inside a function.

Task:

- Declare a local variable inside a function and try accessing it outside.

Expected Output:

```
Inside function: Lkhibra Rocks!
Error: name 'message' is not defined
```

1.5.7 Exercise 32: Recursive Function

Objective: Learn how to use recursion in functions.

Task:

- Define a recursive function to compute factorial.

Expected Output:

```
Factorial of 5 is 120.
```

1.5.8 Exercise 33: Function with Multiple Return Values

Objective: Learn how a function can return multiple values.

Task:

- Define a function `calculate(x, y)` that returns both sum and product.

Expected Output:

```
Sum: 8, Product: 15
```

1.6 Basic Data Structures

1.6.1 Exercise 34: Manage a Shopping List

Objective: Use a list to manage a shopping list dynamically.

Task:

- Create a list called `shopping_list` containing `"Milk"`, `"Eggs"`, `"Bread"`.
- Add ”Tomatoes” to the list.
- Print the updated list.

Expected Output:

```
['Milk', 'Eggs', 'Bread', 'Tomatoes']
```

1.6.2 Exercise 35: Sort a List of Student Scores

Objective: Learn how to sort a list.

Task:

- Given the list of scores `[88, 92, 75, 100, 89]`, sort them in ascending order.

Expected Output:

```
[75, 88, 89, 92, 100]
```

1.6.3 Exercise 36: Find the Most Expensive Product

Objective: Identify the most expensive product in a list.

Task:

- Given the list of prices `[299, 159, 499, 120, 349]`, find and print the highest price.

Expected Output:

```
Most expensive product costs: 499
```

1.6.4 Exercise 37: Count Word Frequency in a Sentence

Objective: Count how often words appear in a sentence using a dictionary.

Task:

- Given the sentence: ”Python is great and Python is fun”, count how many times each word appears.

Expected Output:

```
{'Python': 2, 'is': 2, 'great': 1, 'and': 1, 'fun': 1}
```

1.6.5 Exercise 38: Store Student Grades in a Dictionary

Objective: Store and retrieve student grades using a dictionary.

Task:

- Create a dictionary where student names are keys and their grades are values.
- Retrieve the grade of ”Sarah”.

Expected Output:

```
Sarah's grade: A
```

1.6.6 Exercise 39: Find the Common Elements in Two Sets

Objective: Use sets to find common elements between two groups.

Task:

- Given two sets of students enrolled in Python and Java, find students taking both courses.

Expected Output:

```
{'Alice', 'David'}
```

1.6.7 Exercise 40: Identify Unique Words in a Text

Objective: Use a set to find unique words in a text.

Task:

- Extract all unique words from the text: ”Python is amazing and Python is powerful”.

Expected Output:

```
{'Python', 'is', 'amazing', 'and', 'powerful'}
```

1.6.8 Exercise 41: Create a Menu Using a Dictionary

Objective: Use a dictionary to represent a restaurant menu.

Task:

- Create a menu where dish names are keys and prices are values.
- Retrieve the price of ”Pasta”.

Expected Output:

```
Pasta costs: 12.99
```

1.6.9 Exercise 42: Track Inventory in a Warehouse

Objective: Use a dictionary to manage stock inventory.

Task:

- Create a dictionary where items are keys and stock quantities are values.
- Retrieve the stock quantity of ”Laptops”.

Expected Output:

```
Laptops in stock: 15
```

1.6.10 Exercise 43: Find the Most Frequent Word in a List

Objective: Use dictionaries to count occurrences in a dataset.

Task:

- Given a list of customer reviews, find the most frequently mentioned word.

Expected Output:

```
Most frequent word: excellent
```

1.6.11 Exercise 44: Find the Unique Visitors on a Website

Objective: Use sets to track unique visitors.

Task:

- Given a list of visitor IPs, find the number of unique visitors.

Expected Output:

```
Unique visitors: 3
```

1.6.12 Exercise 45: Filter Customers Who Purchased a Specific Product

Objective: Use sets to find customers who bought a product.

Task:

- Given two sets of customers who bought ”Laptops” and ”Monitors,” find those who bought both.

Expected Output:

```
Customers who bought both: {'Sarah', 'Alex'}
```

1.6.13 Exercise 46: Find the Missing Student in a Class List

Objective: Use sets to compare two lists and find missing data.

Task:

- Given a list of registered students and a list of those who attended, find the missing students.

Expected Output:

```
Absent students: {'Lina', 'Omar'}
```

1.6.14 Exercise 47: Merge Two Dictionaries

Objective: Learn how to merge dictionaries in Python.

Task:

- Given two dictionaries with product stock, merge them into one.

Expected Output:

```
{'Laptops': 10, 'Monitors': 5, 'Keyboards': 8, 'Mice': 12}
```

1.6.15 Exercise 48: Remove Duplicates from a List

Objective: Use sets to remove duplicate values from a list.

Task:

- Given a list of product IDs, remove duplicates.

Expected Output:

```
Unique product IDs: [101, 102, 103, 104]
```

1.7 String Operations & Formatting

1.7.1 Exercise 49: Extract the Domain from an Email

Objective: Extract the domain name from an email address.

Task:

- Given an email address, extract and print the domain name.

Expected Output:

```
Domain: example.com
```

1.7.2 Exercise 50: Count the Occurrences of a Word in a Review

Objective: Count how many times a specific word appears in customer reviews.

Task:

- Given a review, count how often the word ”quality” appears.

Expected Output:

```
The word 'quality' appears 3 times.
```

1.7.3 Exercise 51: Format an Invoice

Objective: Properly align items and prices in an invoice using string formatting.

Task:

- Format the invoice for items purchased and their prices.

Expected Output:

```
Item         Price
--------------------
Laptop      $1200.99
Mouse       $25.50
```

1.7.4 Exercise 52: Reverse Words in a Sentence

Objective: Reverse the order of words in a sentence while keeping their individual order intact.

Task:

- Reverse the words in the sentence: ”Lkhibra Academy is great”.

Expected Output:

```
great is Academy Lkhibra
```

1.7.5 Exercise 53: Extract Hashtags from a Social Media Post

Objective: Identify and extract all hashtags from a post.

Task:

- Extract all hashtags from: ”Loving #Python and #Coding at #LkhibraAcademy ”

Expected Output:

```
Hashtags: ['#Python', '#Coding', '#LkhibraAcademy']
```

1.7.6 Exercise 54: Validate a Password Strength

Objective: Check if a password meets security criteria.

Task:

- Ensure the password has at least 8 characters, including a number and special character.

1.7.7 Exercise 55: Remove Extra Spaces from a String

Objective: Clean up a messy text by removing unnecessary spaces.

Task:

- Remove excess spaces from: ” Hello World ! ”

Expected Output:

```
Hello World !
```

1.7.8 Exercise 56: Convert a String to Title Case

Objective: Convert a string so that each word starts with a capital letter.

Task:

- Convert ”lkhibra academy python training” to title case.

Expected Output:

```
Lkhibra Academy Python Training
```

1.7.9 Exercise 57: Replace Words in a Text

Objective: Replace certain words in a text dynamically.

Task:

- Replace ”Python” with ”Java” in ”I love Python programming”.

Expected Output:

```
I love Java programming
```

1.8 I/O & Error Handling

1.8.1 Exercise 58: Read a File and Count Lines

Objective: Read a text file and count the number of lines.

Task:

- Prompt the user for the file name, e.g., ”transactions.txt”.
- Open and read each line in the file.
- Print the total number of lines.

Expected Output:

```
Enter file name: transactions.txt
Total lines: 12
```

1.8.2 Exercise 59: Write a Sales Record to a File

Objective: Store sales records in a text file.

Task:

- Prompt the user for customer name, item, and price.
- Write a formatted record (e.g., "Customer: John, Item: Laptop, Price: $1200") to `sales.txt`.

Expected Output:

```
Enter customer name: John
Enter item: Laptop
Enter price: 1200
Record written to sales.txt
```

1.8.3 Exercise 60: Read a File and Handle Missing File Errors

Objective: Handle file-related exceptions in Python.

Task:

- Prompt the user for a file name, e.g., "customers.txt".
- Attempt to read and print its contents.
- If the file is missing, print an error message without crashing.

Expected Output:

```
Enter file name: customers.txt
[File contents printed here...]
--- or ---
Error: customers.txt not found.
```

1.8.4 Exercise 61: Handle Division by Zero in a Calculator

Objective: Use `try-except` to handle division errors in a simple calculator.

Task:

- Ask the user to enter two numbers.
- Attempt to divide the first by the second.
- Handle cases where the user tries to divide by zero.

Expected Output:

```
Enter first number: 10
Enter second number: 0
Error: Division by zero is not allowed.
```

1.8.5 Exercise 62: Validate User Input as an Integer

Objective: Ensure user input is an integer before proceeding.

Task:

- Ask the user for a number.
- If the input is not an integer, prompt again until a valid integer is entered.

Expected Output:

```
Enter a number: abc
Invalid input. Please enter an integer.
Enter a number: 42
You entered: 42
```

1.8.6 Exercise 63: Log Errors to a File

Objective: Store error messages in a log file.

Task:

- Handle a division error (e.g., dividing by zero).
- Log the error message to `error_log.txt`.
- Continue program execution.

Expected Output:

```
Enter numerator: 10
Enter denominator: 0
Error logged to file.
```

1.8.7 Exercise 64: Read and Process a CSV File

Objective: Read structured data from a CSV file and process it.

Task:

- Read `sales_data.csv` containing rows like `Product,Amount`.
- Print each row in a readable format.

Expected Output:

```
Reading sales_data.csv...
Product: Laptop, Amount: 1200
Product: Mouse, Amount: 25
...
```

1.8.8 Exercise 65: Automatically Close a File Using a Context Manager

Objective: Use the `with` statement to properly close a file after reading.

Task:

- Read `notes.txt` and print its contents.
- Ensure the file is automatically closed via a context manager.

Expected Output:

```
File content from notes.txt:
[File contents here...]
```

Chapter 2

Mini Projects

2.0.1 Project 66: Student Grade Management System

Objective: Build a system to store, update, and retrieve student grades.

Task:

- Create a dictionary where student names are keys and their grades are values.
- Allow users to: Add a new student. Update an existing grade. Retrieve a student's grade. Display all students.

Expected Output:

```
1. Add Student
2. Update Grade
3. Get Grade
4. Display All
5. Exit
Enter choice: 1
Enter student name: Alice
Enter grade: A
Student added successfully.
```

2.0.2 Project 67: Expense Tracker

Objective: Track and categorize personal expenses.

Task:

- Allow users to: Add expenses (amount & category). View total expenses by category. Display all transactions. Save transactions to a file.

Expected Output:

```
1. Add Expense
2. View Expenses by Category
3. Save & Exit
Enter choice: 1
Enter category (Food, Transport, etc.): Food
Enter amount: 12.5
Expense added.
```

2.0.3 Project 68: Contact Book

Objective: Store and retrieve contact details.

Task:

- Allow users to: Add new contacts. Search for a contact. Display all contacts. Save contacts to a file.

Expected Output:

```
1. Add Contact
2. Search Contact
3. Display Contacts
4. Save & Exit
Enter choice: 1
Enter name: Alice
Enter phone number: 123-456-7890
Contact saved.
```

2.0.4 Project 69: Temperature Converter

Objective: Convert Celsius to Fahrenheit and vice versa.

Task:

- Allow users to: Convert from Celsius to Fahrenheit. Convert from Fahrenheit to Celsius.

Expected Output:

```
Convert to (C/F): F
Enter temperature: 0
Temperature in Fahrenheit: 32.00°F
```

2.0.5 Project 70: Number Guessing Game

Objective: Implement a simple number guessing game using loops.

Task:

- The program generates a random number.
- The user must guess the number with hints provided.

Expected Output:

```
Guess a number (1-100): 50
Too low! Try again.
Guess a number (1-100): 75
Too high! Try again.
Guess a number (1-100): 65
Congratulations! You guessed it in 3 tries.
```

2.0.6 Project 71: To-Do List App

Objective: Manage daily tasks using a list-based system.

Task:

- Allow users to: Add a task. Remove a task. View all tasks.

Expected Output:

```
1. Add Task
2. Remove Task
3. View Tasks
4. Exit
Enter choice: 1
Enter task: Buy groceries
Task added.
```

2.0.7 Project 72: Banking System

Objective: Simulate a simple bank account system.

Task:

- Allow users to: Deposit money. Withdraw money. Check balance.

Expected Output:

```
1. Deposit
2. Withdraw
3. Check Balance
4. Exit
Enter choice: 1
Enter deposit amount: 200
Deposited $200.00. New balance: $200.00
```

2.0.8 Project 73: BMI Calculator

Objective: Calculate the Body Mass Index (BMI) using functions.

Task:

- Prompt the user for weight (in kilograms) and height (in meters).
- Compute BMI = $\frac{\text{weight}}{\text{height}^2}$.
- Print the user's BMI with a simple interpretation.

Expected Output:

```
Enter weight in kg: 70
Enter height in m: 1.75
Your BMI is: 22.86
You are in the Normal weight range.
```

2.0.9 Project 74: Unit Converter

Objective: Convert different measurement units using modular functions.

Task:

- Allow the user to choose between different conversions (e.g., km to miles, kg to lbs).
- Perform the calculation and display the result.

Expected Output:

```
1. KM to Miles
2. KG to Lbs
Enter choice: 1
Enter value in KM: 5
Miles: 3.11
```

2.0.10 Project 75: PDF Merger

Objective: Merge multiple PDF files into a single PDF document.

Task:

- Ask the user for the names of several PDF files.
- Combine them in the specified order into a single PDF.
- Save the merged PDF as `merged.pdf`.

Expected Output:

```
Enter PDFs to merge (comma-separated): file1.pdf, file2.pdf
Merging PDF files...
PDF files merged and saved as merged.pdf
```

2.0.11 Project 76: IP Location Finder

Objective: Query an external API to find geographical information about a given IP address.

Task:

- Prompt the user for an IP address.
- Send a request to a geolocation API.
- Display city, region, and country.

Expected Output:

```
Enter IP address: 8.8.8.8
City: Mountain View
Region: California
Country: US
```

2.0.12 Project 77: Basic Maze Generator

Objective: Generate and display a simple ASCII maze using randomization.

Task:

- Prompt user for maze dimensions (width, height).
- Build a random maze with walls and open paths.
- Print the maze in the console.

Expected Output:

```
Enter width: 10
Enter height: 5
Maze:
##########
#   ##   #
# ##   # #
#        #
##########
```

2.0.13 Project 78: File Duplicate Finder

Objective: Identify duplicate files in a directory by comparing file hashes.

Task:

- Ask the user for a directory path.
- Compute a hash (e.g., MD5) for each file.
- Print sets of files that share the same hash.

Expected Output:

```
Enter directory path: ./test_files
Duplicates found:
fileA.txt, fileB.txt
fileC.jpg, fileD.jpg
```

2.0.14 Project 79: Student Attendance System

Objective: Track attendance for students in a class using dictionary-based storage.

Task:

- Maintain a dictionary mapping student names to attendance counts.
- Allow marking attendance, viewing attendance, and resetting counts.

Expected Output:

```
1. Mark Attendance
2. View Attendance
3. Reset Attendance
4. Exit
Enter choice: 1
Enter student name: Alice
Alice's attendance marked.
```

2.0.15 Project 80: Basic Chat Application

Objective: Implement a simple console-based chat app using sockets.

Task:

- Create a server that listens for incoming messages.
- A client connects to the server and sends text messages.

Expected Output:

```
Server:
Waiting for client...
Connection established.
Client:
Connected to server.
Type your message.
Hello from client -> received on server
```

2.0.16 Project 81: Library Management System

Objective: Manage a digital library where users can borrow and return books.

Task:

- Allow users to: View available books. Borrow a book. Return a book.

Expected Output:

```
1. View Books
2. Borrow Book
3. Return Book
4. Exit
Enter choice: 1
Available Books:
- Python Basics
- Data Science
- AI Fundamentals
```

2.0.17 Project 82: Voting System

Objective: Use sets to track unique votes and prevent duplicate voting.

Task:

- Allow users to: Vote for a candidate. Prevent multiple votes from the same user. Display results.

Expected Output:

```
1. Vote
2. Show Results
3. Exit
Enter choice: 1
Enter your ID: user123
Vote for (Alice/Bob/Charlie): Alice
Vote recorded!
```

2.0.18 Project 83: Resume Formatter

Objective: Format and structure resume details into a clean format.

Task:

- Take user input for name, skills, and experience.
- Format the information into a readable resume.

Expected Output:

```
Enter your name: Alice Brown
Enter your skills (comma-separated): Python, Data Analysis
Enter years of experience: 3

----------------------------
    ALICE BROWN
----------------------------
Skills: Python, Data Analysis
Experience: 3 years
```

2.0.19 Project 84: Palindrome Checker

Objective: Check if a word or phrase is a palindrome.

Task:

- Take user input and check if it's a palindrome (ignoring case & spaces).

Expected Output:

```
Enter a word or phrase: Racecar
Palindrome
```

2.0.20 Project 85: CSV File Analyzer

Objective: Read and process CSV data from a sales report.

Task:

- Read `sales.csv` and calculate total sales.

Expected Output:

```
Total Sales: $1234.56
```

2.0.21 Project 86: Password Manager

Objective: Securely store and retrieve passwords from a file.

Task:

- Allow users to save and retrieve encrypted passwords.

Expected Output:

```
1. Save Password
2. View Passwords
3. Exit
Enter choice: 1
Enter website: example.com
Enter password: mySecret123
Password saved!
```

2.0.22 Project 87: Random Password Generator

Objective: Generate a secure password using randomization.

Task:

- Generate a password containing letters, numbers, and symbols.

Expected Output:

```
Generated Password: #X3Pa9&AL!
```

2.0.23 Project 88: Email Validator

Objective: Check if an email address follows a valid format.

Task:

- Validate an email using regex.

Expected Output:

```
Enter an email: user@example.com
Valid
```

2.0.24 Project 89: Weather Forecast App

Objective: Retrieve and display real-time weather data using an API.

Task:

- Fetch and display weather information for a given city.

Expected Output:

```
Enter city name: London
Temperature: 18°C
Weather: Clear sky
```

2.0.25 Project 90: Stock Price Tracker

Objective: Fetch real-time stock prices using an API.

Task:

- Get the latest stock price of a given company.

Expected Output:

```
Enter stock symbol (e.g., AAPL, TSLA): TSLA
Current price of TSLA: $690.32
```

2.0.26 Project 91: Text Summarizer

Objective: Generate a concise summary from a long paragraph.

Task:

- Take user input and generate a 3-sentence summary.

Expected Output:

```
Enter text to summarize: Lorem ipsum dolor sit amet ...
Summary: Lorem ipsum dolor sit amet ...
```

2.0.27 Project 92: Book Recommendation System

Objective: Suggest books based on the user's favorite genre.

Task:

- Recommend books based on predefined genre categories.

Expected Output:

```
Enter a genre (Fiction, Science, Technology): Science
Recommended Books: A Brief History of Time, The Selfish Gene
```

2.0.28 Project 93: File Encryption & Decryption

Objective: Encrypt and decrypt files securely.

Task:

- Encrypt a file and later decrypt it.

Expected Output:

```
File encrypted.
File decrypted.
```

2.0.29 Project 94: Two-Factor Authentication (2FA) Generator

Objective: Generate temporary authentication codes for security.

Task:

- Generate and verify a one-time 6-digit authentication code.

Expected Output:

```
Your 2FA Code: 573920
Enter the code: 573920
Access granted.
```

2.0.30 Project 95: Employee Database System

Objective: Manage employee records in a simple database.

Task:

- Add, remove, and list employees.

Expected Output:

```
1. Add Employee
2. View Employees
3. Exit
Enter choice: 1
Enter employee ID: E123
Enter employee name: John
Enter position: Developer
Employee added.
```

2.0.31 Project 96: Chatbot for FAQs

Objective: Create a simple chatbot that answers frequently asked questions.

Task:

- Store predefined questions and answers in a dictionary.
- Allow users to ask questions and receive responses.
- Provide a default response if the question is not recognized.

Expected Output:

```
Ask a question: What is Lkhibra Academy?
Lkhibra Academy is an online learning platform.
Ask a question: How do I enroll?
Visit our website and sign up for a course.
Ask a question: exit
```

2.0.32 Project 97: Automated Invoice Generator

Objective: Generate invoices automatically based on user input.

Task:

- Collect user input for customer name, items, and prices.
- Calculate the total and generate a formatted invoice.

- Save the invoice to a text file.

Expected Output:

```
Enter customer name: ACME Corp
Enter item (or 'done' to finish): Monitor
Enter price: 120.5
Enter item (or 'done' to finish): Keyboard
Enter price: 15
Enter item (or 'done' to finish): done
Invoice saved to invoice.txt
```

2.0.33 Project 98: Task Reminder with Notifications

Objective: Implement a basic task reminder system.

Task:

- Store tasks with due times.
- Notify users when a task is due.

Expected Output:

```
Task Reminder System Started!
(Reminder after 5 seconds...)
Reminder: Submit project is due now!
(Reminder after 10 seconds...)
Reminder: Meeting with team is due now!
```

2.0.34 Project 99: URL Shortener

Objective: Create a simple URL shortener.

Task:

- Allow users to input long URLs.
- Generate a short identifier.
- Store and retrieve shortened URLs.

Expected Output:

```
Enter URL to shorten: https://www.example.com/long/path
Shortened URL: https://short.ly/A1b2C3
```

2.0.35 Project 100: AI-Powered Sentiment Analyzer

Objective: Analyze the sentiment of user input using basic NLP.

Task:

- Take user input as a sentence.
- Determine if the sentiment is positive, neutral, or negative.
- Use a simple word-based approach.

Expected Output:

```
Enter a sentence: I love learning Python
Positive Sentiment
```

Chapter 3

Solutions

3.1 Basic Syntax & Variables

3.1.1 Solution 1: Declare and Print Variables

Solution:

```
instructor = "Alex"
students = 30
course = "Python"

print(f"The instructor is {instructor}, there are {students} students in the {course}
```

Explanation:

- Variables store data such as text or numbers.
- The `f""` (formatted string) allows inserting variables directly into a string.

3.1.2 Solution 2: Swap Two Variables Without a Third Variable

Solution:

```
students_morning = 15
students_evening = 25

# Swapping without a third variable
students_morning, students_evening = students_evening, students_morning

print(f"After Swap: Morning Batch = {students_morning}, Evening Batch = {students_even
```

Explanation:

- Python allows swapping variables using tuple unpacking: `a, b = b, a`.
- This avoids using an extra variable for storage.

3.1.3 Solution 3: Assign Multiple Variables in One Line

Solution:

```
python_students, java_students, ai_students = 25, 18, 12
print(f"Python = {python_students}, Java = {java_students}, AI = {ai_students}")
```

Explanation:

- Python allows multiple assignments in a single line.

3.1.4 Solution 4: Check the Type of a Variable

Solution:

```
age = 21
rating = 4.9
course_name = "Python Programming"

print(f"{age} is of type {type(age)}")
print(f"{rating} is of type {type(rating)}")
print(f"{course_name} is of type {type(course_name)}")
```

Explanation:

- The `type()` function is used to determine the type of a variable.

3.1.5 Solution 5: Concatenating Strings

Solution:

```
instructor = "The instructor"
academy = "Lkhibra Academy"
slogan = '"Learning Python is fun!"'

full_sentence = instructor + " at " + academy + " says: " + slogan
print(full_sentence)
```

Explanation:

- Strings can be concatenated using the '+' operator.
- Spaces must be manually added between words when concatenating.

3.2 Data Types & Conversions

3.2.1 Solution 6: Convert String to Integer and Vice Versa

Solution:

```
num_str = "100"
num_int = int(num_str)

num_int2 = 42
num_str2 = str(num_int2)

print(f"Integer value: {num_int}, Type: {type(num_int)}")
print(f"String value: {num_str2}, Type: {type(num_str2)}")
```

Explanation:

- `int()` converts a string to an integer.
- `str()` converts an integer to a string.
- `type()` is used to check the data type of a variable.

3.2.2 Solution 7: Convert Float to Integer and Vice Versa

Solution:

```
num_float = 9.75
num_int = int(num_float)

num_int2 = 50
num_float2 = float(num_int2)

print(f"Float to Int: {num_int}, Type: {type(num_int)}")
print(f"Int to Float: {num_float2}, Type: {type(num_float2)}")
```

Explanation:

- Converting a float to an integer **removes** the decimal part.
- Converting an integer to a float **adds** a decimal.

3.2.3 Solution 8: Convert a Boolean to an Integer

Solution:

```
bool_true = True
bool_false = False

print(f"True as an integer: {int(bool_true)}")
print(f"False as an integer: {int(bool_false)}")
```

Explanation:

- In Python, `True` is equivalent to `1`.

- `False` is equivalent to `0`.
- Boolean values are often used in conditional checks.

3.2.4 Solution 9: Convert List to a String and Back

Solution:

```
word_list = ["Python", "is", "amazing"]
joined_string = ", ".join(word_list)

split_list = joined_string.split(", ")

print(f"List to String: {joined_string}")
print(f"String to List: {split_list}")
```

Explanation:

- `join()` converts a list into a string with separators.
- `split()` breaks a string back into a list.

3.2.5 Solution 10: Convert Dictionary Keys and Values to Lists

Solution:

```
academy_info = {"name": "Lkhibra Academy", "age": 5, "language": "Python"}

keys_list = list(academy_info.keys())
values_list = list(academy_info.values())

print(f"Keys: {keys_list}")
print(f"Values: {values_list}")
```

Explanation:

- `keys()` extracts dictionary keys.
- `values()` extracts dictionary values.
- The `list()` function converts them into lists.

3.3 Operators & Expressions

3.3.1 Solution 11: Perform Arithmetic Operations

Solution:

```
a = 10
b = 5

print(f"Addition: {a + b}")
print(f"Subtraction: {a - b}")
print(f"Multiplication: {a * b}")
print(f"Division: {a / b}")
print(f"Modulus: {a % b}")
```

Explanation:

- +, -, *, /, and % are arithmetic operators.
- Modulus % returns the remainder of division.

3.3.2 Solution 12: Use Comparison Operators

Solution:

```
a = 10
b = 5

print(f"{a} > {b}: {a > b}")
print(f"{a} < {b}: {a < b}")
print(f"{a} == {a}: {a == a}")
print(f"{a} != {b}: {a != b}")
print(f"{a} >= {b}: {a >= b}")
print(f"{a} <= {b}: {a <= b}")
```

Explanation:

- >, <, >=, <=, ==, and != are comparison operators.
- These operators return `True` or `False`.

3.3.3 Solution 13: Use Logical Operators

Solution:

```
a = True
b = False

print(f"True and False: {a and b}")
print(f"True or False: {a or b}")
print(f"Not True: {not a}")
```

Explanation:

- `and` returns `True` if both conditions are `True`.

- `or` returns `True` if at least one condition is `True`.
- `not` inverts a boolean value.

3.3.4 Solution 14: Use Assignment Operators

Solution:

```
x = 10
print(f"Initial Value: {x}")

x += 5
print(f"After += : {x}")

x -= 3
print(f"After -= : {x}")

x *= 2
print(f"After *= : {x}")

x /= 3
print(f"After /= : {x}")

x %= 2
print(f"After %= : {x}")
```

Explanation:

- +=, -=, *=, /=, and %= modify variables in place.
- These operators **update** the value of the variable.

3.3.5 Solution 15: Use Bitwise Operators

Solution:

```
a = 5  # Binary: 101
b = 3  # Binary: 011

print(f"{a} & {b} = {a & b}")  # AND
print(f"{a} | {b} = {a | b}")  # OR
print(f"{a} ^ {b} = {a ^ b}")  # XOR
print(f"{a} << 1 = {a << 1}")  # Left Shift
print(f"{a} >> 1 = {a >> 1}")  # Right Shift
```

Explanation:

- & (AND) keeps common bits.
- | (OR) combines all bits.

- $(XOR) keeps different bits. <<(Left Shift) multiplies by 2.$
- >> (Right Shift) divides by 2.

3.4 Control Flow (Conditionals & Loops)

3.4.1 Solution 16: Check if a Number is Even or Odd

Solution:

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

if num % 2 == 0:
    print(f"{num} is an even number.")
else:
    print(f"{num} is an odd number.")
```

Explanation:

- The modulus operator % is used to check if a number is divisible by 2.
- The `if-else` statement determines the result.

3.4.2 Solution 17: Find the Largest Number

Solution:

```
a, b, c = map(int, input("Enter three numbers: ").split())

if a >= b and a >= c:
    print(f"The largest number is {a}.")
elif b >= a and b >= c:
    print(f"The largest number is {b}.")
else:
    print(f"The largest number is {c}.")
```

Explanation:

- The program compares three numbers using `if-elif-else`.
- The largest number is determined and printed.

3.4.3 Solution 18: Check if a Year is a Leap Year

Solution:

```python
year = int(input("Enter a year: "))

if (year % 4 == 0 and year % 100 != 0) or (year % 400 == 0):
    print(f"{year} is a leap year.")
else:
    print(f"{year} is not a leap year.")
```

Explanation:

- A leap year is divisible by 4 but not by 100, except when divisible by 400.

3.4.4 Solution 19: Print Numbers from 1 to N

Solution:

```python
n = int(input("Enter a number: "))

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

Explanation:

- The `range()` function generates numbers from 1 to `N`.
- `end=" "` ensures output is printed on the same line.

3.4.5 Solution 20: Print Even Numbers up to N

Solution:

```python
n = int(input("Enter a number: "))

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

Explanation:

- The loop starts at 2 and increments by 2 to print even numbers.

3.4.6 Solution 21: Sum of First N Natural Numbers

Solution:

```python
n = int(input("Enter a number: "))
sum_n = 0

for i in range(1, n + 1):
    sum_n += i

print(f"Sum of first {n} numbers: {sum_n}")
```

Explanation:

- The loop iterates from 1 to `N`, adding each number to the sum.

3.4.7 Solution 22: Factorial of a Number

Solution:

```
n = int(input("Enter a number: "))
factorial = 1

for i in range(1, n + 1):
    factorial *= i

print(f"Factorial of {n} is {factorial}.")
```

Explanation:

- The loop multiplies numbers from 1 to `N` to compute factorial.

3.4.8 Solution 23: Reverse a Number

Solution:

```
num = int(input("Enter a number: "))
rev_num = 0

while num > 0:
    digit = num % 10
    rev_num = rev_num * 10 + digit
    num //= 10

print(f"Reversed number: {rev_num}")
```

Explanation:

- Extracts the last digit using `% 10` and builds the reversed number.

3.4.9 Solution 24: Multiplication Table

Solution:

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

for i in range(1, 11):
    print(f"{n} x {i} = {n * i}")
```

Explanation:

- The loop iterates from 1 to 10 and prints the multiplication result.

3.4.10 Solution 25: Count Digits in a Number

Solution:

```
num = int(input("Enter a number: "))
count = 0

while num > 0:
    num //= 10
    count += 1

print(f"The number has {count} digits.")
```

Explanation:

- The loop removes the last digit and increments the count until the number becomes 0.

3.5 Functions & Scope

3.5.1 Solution 26: Define and Call a Function

Solution:

```
def greet():
    print("Hello, Lkhibra Academy!")

greet()
```

Explanation:

- The `def` keyword is used to define a function.
- The function is called by writing its name followed by parentheses.

3.5.2 Solution 27: Function with Parameters

Solution:

```
def welcome(name):
    print(f"Hello, {name}! Welcome to Lkhibra Academy!")

welcome("Sara")
welcome("Youssef")
```

Explanation:

- Functions can take parameters to make them more dynamic.

3.5.3 Solution 28: Function Returning a Value

Solution:

```
def square(num):
    return num * num

result = square(5)
print(f"The square of 5 is {result}.")
```

Explanation:

- The `return` statement allows a function to return a value.

3.5.4 Solution 29: Function with Default Parameter

Solution:

```
def greet(name="Student"):
    print(f"Hello, {name}!")

greet()
greet("Amine")
```

Explanation:

- Default parameters allow functions to be called without arguments.

3.5.5 Solution 30: Function Using Global Variable

Solution:

```
academy = "Lkhibra"

def show_academy():
    print(f"The best academy is {academy}!")

show_academy()
```

Explanation:

- Functions can access global variables defined outside them.

3.5.6 Solution 31: Function Using Local Variable

Solution:

```python
def say_something():
    message = "Lkhibra Rocks!"
    print(f"Inside function: {message}")

say_something()
print(message)  # This will cause an error
```

Explanation:

- Local variables exist only inside the function where they are declared.

3.5.7 Solution 32: Recursive Function

Solution:

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

print(f"Factorial of 5 is {factorial(5)}.")
```

Explanation:

- A recursive function calls itself to solve a problem.

3.5.8 Solution 33: Function with Multiple Return Values

Solution:

```python
def calculate(x, y):
    return x + y, x * y

sum_result, product_result = calculate(3, 5)
print(f"Sum: {sum_result}, Product: {product_result}")
```

Explanation:

- A function can return multiple values using a tuple.

3.6 Basic Data Structures

3.6.1 Solution 34: Manage a Shopping List

Solution:

```
shopping_list = ["Milk", "Eggs", "Bread"]
shopping_list.append("Tomatoes")

print(shopping_list)
```

Explanation:

- Lists allow dynamic updates, such as adding items using `append()`.

3.6.2 Solution 35: Sort a List of Student Scores

Solution:

```
scores = [88, 92, 75, 100, 89]
scores.sort()

print(scores)
```

Explanation:

- The `sort()` method arranges elements in ascending order.

3.6.3 Solution 36: Find the Most Expensive Product

Solution:

```
prices = [299, 159, 499, 120, 349]
highest_price = max(prices)

print(f"Most expensive product costs: {highest_price}")
```

Explanation:

- The `max()` function finds the highest value in a list.

3.6.4 Solution 37: Count Word Frequency in a Sentence

Solution:

```
sentence = "Python is great and Python is fun"
words = sentence.split()
word_count = {}
```

```
for word in words:
    word_count[word] = word_count.get(word, 0) + 1

print(word_count)
```

Explanation:

- The `split()` method breaks the sentence into words.
- The `get()` method helps count word occurrences.

3.6.5 Solution 38: Store Student Grades in a Dictionary

Solution:

```
grades = {"Sarah": "A", "John": "B", "Emily": "A+"}
print(f"Sarah's grade: {grades['Sarah']}")
```

Explanation:

- Dictionaries store key-value pairs, allowing quick lookups.

3.6.6 Solution 39: Find the Common Elements in Two Sets

Solution:

```
python_students = {"Alice", "Bob", "Charlie", "David"}
java_students = {"David", "Eve", "Alice", "Frank"}

common_students = python_students & java_students
print(common_students)
```

Explanation:

- The `&` operator finds common elements in two sets.

3.6.7 Solution 40: Identify Unique Words in a Text

Solution:

```
text = "Python is amazing and Python is powerful"
unique_words = set(text.split())

print(unique_words)
```

Explanation:

- Using a set removes duplicate words from the text.

3.6.8 Solution 41: Create a Menu Using a Dictionary

Solution:

```
menu = {"Burger": 9.99, "Pasta": 12.99, "Salad": 7.99}
print(f"Pasta costs: {menu['Pasta']}")
```

Explanation:

- A dictionary allows quick access to menu prices.

3.6.9 Solution 42: Track Inventory in a Warehouse

Solution:

```
inventory = {"Laptops": 15, "Monitors": 8, "Keyboards": 20}
print(f"Laptops in stock: {inventory['Laptops']}")
```

Explanation:

- Dictionaries allow quick access to stock levels.

3.6.10 Solution 43: Find the Most Frequent Word in a List

Solution:

```
reviews = ["great", "excellent", "great", "good", "excellent", "excellent"]
word_count = {}

for word in reviews:
    word_count[word] = word_count.get(word, 0) + 1

most_frequent = max(word_count, key=word_count.get)
print(f"Most frequent word: {most_frequent}")
```

Explanation:

- Uses a dictionary to count occurrences and `max()` to find the most common word.

3.6.11 Solution 44: Find the Unique Visitors on a Website

Solution:

```
visitors = ["192.168.1.1", "192.168.1.2", "192.168.1.1", "192.168.1.3"]
unique_visitors = len(set(visitors))

print(f"Unique visitors: {unique_visitors}")
```

Explanation:

- Sets automatically remove duplicate elements.

3.6.12 Solution 45: Filter Customers Who Purchased a Specific Product

Solution:

```
laptop_buyers = {"Sarah", "John", "Alex", "Nina"}
monitor_buyers = {"Alex", "Sarah", "Mike"}

both = laptop_buyers & monitor_buyers
print(f"Customers who bought both: {both}")
```

Explanation:

- The & operator finds common elements in two sets.

3.6.13 Solution 46: Find the Missing Student in a Class List

Solution:

```
registered_students = {"Ali", "Omar", "Lina", "Sofia"}
attended_students = {"Ali", "Sofia"}

absent_students = registered_students - attended_students
print(f"Absent students: {absent_students}")
```

Explanation:

- The - operator finds elements that are in one set but not in the other.

3.6.14 Solution 47: Merge Two Dictionaries

Solution:

```
stock_1 = {"Laptops": 10, "Monitors": 5}
stock_2 = {"Keyboards": 8, "Mice": 12}

merged_stock = {**stock_1, **stock_2}
print(merged_stock)
```

Explanation:

- The `{**dict1, **dict2}` syntax merges two dictionaries.

3.6.15 Solution 48: Remove Duplicates from a List

Solution:

```
product_ids = [101, 102, 103, 101, 104, 102]
unique_products = list(set(product_ids))

print(f"Unique product IDs: {unique_products}")
```

Explanation:

- Converting a list to a set removes duplicates automatically.

3.7 String Operations & Formatting

3.7.1 Solution 49: Extract the Domain from an Email

Solution:

```
email = "user@example.com"
domain = email.split("@")[1]

print(f"Domain: {domain}")
```

Explanation:

- The `split("@")` function extracts the domain from an email.

3.7.2 Solution 50: Count the Occurrences of a Word in a Review

Solution:

```
review = "The quality of this product is great. Quality matters. We love the quality!"
word_count = review.lower().split().count("quality")

print(f"The word 'quality' appears {word_count} times.")
```

3.7.3 Solution 51: Format an Invoice

Solution:

```
items = [("Laptop", 1200.99), ("Mouse", 25.50)]

print(f"{'Item':<12}{'Price':>10}")
print("-" * 22)

for item, price in items:
    print(f"{item:<12}${price:>9.2f}")
```

3.7.4 Solution 52: Reverse Words in a Sentence

Solution:

```
sentence = "Lkhibra Academy is great"
reversed_sentence = " ".join(sentence.split()[::-1])

print(reversed_sentence)
```

3.7.5 Solution 53: Extract Hashtags from a Social Media Post

Solution:

```
import re

post = "Loving #Python and #Coding at #LkhibraAcademy!"
hashtags = re.findall(r"#\w+", post)

print(f"Hashtags: {hashtags}")
```

3.7.6 Solution 54: Validate a Password Strength

Solution:

```
import re

password = "Secure@123"
is_valid = bool(re.match(r'^(?=.*[A-Za-z])(?=.*\d)(?=.*[@$!%*?&])[A-Za-z\d@$!%*?&]{8,}
print("Valid password" if is_valid else "Invalid password")
```

3.7.7 Solution 55: Remove Extra Spaces from a String

Solution:

```
import re

text = "  Hello     World  !  "
clean_text = re.sub(r'\s+', ' ', text).strip()

print(clean_text)
```

3.7.8 Solution 56: Convert a String to Title Case

Solution:

```
text = "lkhibra academy python training"
title_case_text = text.title()

print(title_case_text)
```

3.7.9 Solution 57: Replace Words in a Text

Solution:

```
text = "I love Python programming"
updated_text = text.replace("Python", "Java")

print(updated_text)
```

3.8 I/O & Error Handling

3.8.1 Solution 58: Read a File and Count Lines

Solution:

```
file_name = input("Enter file name: ")

try:
    with open(file_name, "r") as file:
        lines = file.readlines()
        print(f"Total lines: {len(lines)}")
except FileNotFoundError:
    print("Error: The file does not exist.")
```

Explanation:

- Uses `open(...)` in read mode and `.readlines()` to get all lines.
- Prints the length of the resulting list.
- Demonstrates basic file handling and `FileNotFoundError` exception handling.

3.8.2 Solution 59: Write a Sales Record to a File

Solution:

```
customer = input("Enter customer name: ")
item = input("Enter item: ")
price = input("Enter price: ")

record = f"Customer: {customer}, Item: {item}, Price: ${price}"
```

```
with open("sales.txt", "a") as file:
    file.write(record + "\n")

print("Record written to sales.txt")
```

Explanation:

- Demonstrates appending text to an existing file using `mode='a'`.
- Shows string formatting to generate a sales record entry.

3.8.3 Solution 60: Read a File and Handle Missing File Errors

Solution:

```
file_name = input("Enter file name: ")

try:
    with open(file_name, "r") as file:
        content = file.read()
        print(content)
except FileNotFoundError:
    print(f"Error: {file_name} not found.")
```

Explanation:

- Uses a `try-except` block to catch `FileNotFoundError`.
- Demonstrates graceful error handling for missing files.

3.8.4 Solution 61: Handle Division by Zero in a Calculator

Solution:

```
try:
    num1 = float(input("Enter first number: "))
    num2 = float(input("Enter second number: "))
    result = num1 / num2
    print(f"Result: {result}")
except ZeroDivisionError:
    print("Error: Division by zero is not allowed.")
```

Explanation:

- Converts user inputs to `float` for decimal division.
- Wraps division in a `try-except` to catch `ZeroDivisionError`.

3.8.5 Solution 62: Validate User Input as an Integer

Solution:

```
while True:
    try:
        num = int(input("Enter a number: "))
        print(f"You entered: {num}")
        break
    except ValueError:
        print("Invalid input. Please enter an integer.")
```

Explanation:

- Uses an infinite loop to repeatedly prompt until successful input.
- Catches `ValueError` if `int(...)` fails.

3.8.6 Solution 63: Log Errors to a File

Solution:

```
try:
    num1 = float(input("Enter numerator: "))
    num2 = float(input("Enter denominator: "))
    result = num1 / num2
    print(f"Result: {result}")
except ZeroDivisionError as e:
    with open("error_log.txt", "a") as f:
        f.write(f"Error: {str(e)}\n")
    print("Error logged to file.")
```

Explanation:

- Shows how to catch an exception, log it to a file, and give user feedback.
- The `ZeroDivisionError` object is converted to a string for logging.

3.8.7 Solution 64: Read and Process a CSV File

Solution:

```
import csv

print("Reading sales_data.csv...")

try:
    with open("sales_data.csv", "r") as file:
        reader = csv.reader(file)
        for row in reader:
```

```
            if row:
                product, amount = row
                print(f"Product: {product}, Amount: {amount}")
except FileNotFoundError:
    print("Error: CSV file not found.")
```

Explanation:

- Demonstrates reading CSV data row by row.
- Simple error handling for missing files.

3.8.8 Solution 65: Automatically Close a File Using a Context Manager

Solution:

```
print("File content from notes.txt:")

try:
    with open("notes.txt", "r") as file:
        content = file.read()
        print(content)
except FileNotFoundError:
    print("Error: notes.txt not found.")
```

Explanation:

- The `with` statement handles closing the file automatically.
- Demonstrates best practices in Python file I/O.

3.9 Mini Projects

3.9.1 Solution 66: Student Grade Management System

Solution:

```
students = {}

def add_student():
    name = input("Enter student name: ")
    grade = input("Enter grade: ")
    students[name] = grade
    print("Student added successfully.")

def update_grade():
    name = input("Enter student name: ")
    if name in students:
```

```
        grade = input("Enter new grade: ")
        students[name] = grade
        print("Grade updated.")
    else:
        print("Student not found.")

def get_grade():
    name = input("Enter student name: ")
    print(f"{name}'s grade: {students.get(name, 'Not found')}")

def display_all():
    for name, grade in students.items():
        print(f"{name}: {grade}")

while True:
    print("\n1. Add Student\n2. Update Grade\n3. Get Grade\n4. Display All\n5. Exit")
    choice = input("Enter choice: ")

    if choice == "1":
        add_student()
    elif choice == "2":
        update_grade()
    elif choice == "3":
        get_grade()
    elif choice == "4":
        display_all()
    elif choice == "5":
        break
    else:
        print("Invalid choice. Try again.")
```

Explanation:

- We maintain a dictionary `students` to store each student's name and grade.
- Users can add or update a student's grade, retrieve a specific grade, and display all entries.
- This simple CLI menu provides a real-world scenario for basic data storage.

3.9.2 Solution 67: Expense Tracker

Solution:

```
expenses = {}

def add_expense():
    category = input("Enter category (Food, Transport, etc.): ")
```

```
    amount = float(input("Enter amount: "))
    expenses[category] = expenses.get(category, 0) + amount
    print("Expense added.")

def view_by_category():
    for category, total in expenses.items():
        print(f"{category}: \${total:.2f}")

def save_to_file():
    with open("expenses.txt", "w") as file:
        for category, total in expenses.items():
            file.write(f"{category}: \${total:.2f}\n")
    print("Expenses saved to file.")

while True:
    print("\n1. Add Expense\n2. View Expenses by Category\n3. Save & Exit")
    choice = input("Enter choice: ")

    if choice == "1":
        add_expense()
    elif choice == "2":
        view_by_category()
    elif choice == "3":
        save_to_file()
        break
    else:
        print("Invalid choice. Try again.")
```

Explanation:

- We store expenses in a dictionary, mapping categories to total amounts.
- The user can add a new expense, view totals by category, or save the data to a file.
- This project demonstrates basic file writing, dictionary usage, and floating-point operations.

3.9.3 Solution 68: Contact Book

Solution:

```
contacts = {}

def add_contact():
    name = input("Enter name: ")
    phone = input("Enter phone number: ")
    contacts[name] = phone
    print("Contact saved.")
```

```
def search_contact():
    name = input("Enter name: ")
    print(f"Phone: {contacts.get(name, 'Not found')}")

def display_contacts():
    for name, phone in contacts.items():
        print(f"{name}: {phone}")

def save_contacts():
    with open("contacts.txt", "w") as file:
        for name, phone in contacts.items():
            file.write(f"{name}: {phone}\n")
    print("Contacts saved.")

while True:
    print("\n1. Add Contact\n2. Search Contact\n3. Display Contacts\n4. Save & Exit")
    choice = input("Enter choice: ")

    if choice == "1":
        add_contact()
    elif choice == "2":
        search_contact()
    elif choice == "3":
        display_contacts()
    elif choice == "4":
        save_contacts()
        break
    else:
        print("Invalid choice. Try again.")
```

Explanation:

- A dictionary maps each contact name to a phone number.
- The user can add, search, and display contacts; results are saved to a file for persistence.

3.9.4 Solution 69: Temperature Converter

Solution:

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

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

choice = input("Convert to (C/F): ").upper()
```

```
temp = float(input("Enter temperature: "))

if choice == "C":
    print(f"Temperature in Celsius: {fahrenheit_to_celsius(temp):.2f}°C")
elif choice == "F":
    print(f"Temperature in Fahrenheit: {celsius_to_fahrenheit(temp):.2f}°F")
else:
    print("Invalid choice.")
```

Explanation:

- Demonstrates arithmetic operations and user input.
- The user decides which conversion to perform.

3.9.5 Solution 70: Number Guessing Game

Solution:

```
import random

secret_number = random.randint(1, 100)
attempts = 0

while True:
    guess = int(input("Guess a number (1-100): "))
    attempts += 1
    if guess < secret_number:
        print("Too low! Try again.")
    elif guess > secret_number:
        print("Too high! Try again.")
    else:
        print(f"Congratulations! You guessed it in {attempts} tries.")
        break
```

Explanation:

- Uses random number generation from `random`.
- Showcases while loops, conditionals, and counters.

3.9.6 Solution 71: To-Do List App

Solution:

```
tasks = []

def add_task():
    task = input("Enter task: ")
```

```
    tasks.append(task)
    print("Task added.")

def remove_task():
    task = input("Enter task to remove: ")
    if task in tasks:
        tasks.remove(task)
        print("Task removed.")
    else:
        print("Task not found.")

def display_tasks():
    print("\nTo-Do List:")
    for index, task in enumerate(tasks, start=1):
        print(f"{index}. {task}")

while True:
    print("\n1. Add Task\n2. Remove Task\n3. View Tasks\n4. Exit")
    choice = input("Enter choice: ")

    if choice == "1":
        add_task()
    elif choice == "2":
        remove_task()
    elif choice == "3":
        display_tasks()
    elif choice == "4":
        break
    else:
        print("Invalid choice. Try again.")
```

Explanation:

- Demonstrates list operations (append/remove).
- A straightforward approach to building a CLI-based to-do manager.

3.9.7 Solution 72: Banking System

Solution:

```
balance = 0

def deposit():
    global balance
    amount = float(input("Enter deposit amount: "))
    balance += amount
    print(f"Deposited \${amount:.2f}. New balance: \${balance:.2f}")
```

```
def withdraw():
    global balance
    amount = float(input("Enter withdrawal amount: "))
    if amount > balance:
        print("Insufficient funds.")
    else:
        balance -= amount
        print(f"Withdrew \${amount:.2f}. New balance: \${balance:.2f}")

while True:
    print("\n1. Deposit\n2. Withdraw\n3. Check Balance\n4. Exit")
    choice = input("Enter choice: ")

    if choice == "1":
        deposit()
    elif choice == "2":
        withdraw()
    elif choice == "3":
        print(f"Current balance: \${balance:.2f}")
    elif choice == "4":
        break
    else:
        print("Invalid choice. Try again.")
```

Explanation:

- Showcases function scope and a global balance variable.
- Basic error handling for insufficient funds.

3.9.8 Solution 73: BMI Calculator

Solution:

```
def calculate_bmi(weight, height):
    return weight / (height ** 2)

weight = float(input("Enter weight in kg: "))
height = float(input("Enter height in m: "))

bmi = calculate_bmi(weight, height)
print(f"Your BMI is: {bmi:.2f}")

if bmi < 18.5:
    print("You are underweight.")
elif 18.5 <= bmi < 25:
    print("You are in the Normal weight range.")
elif 25 <= bmi < 30:
```

```
    print("You are overweight.")
else:
    print("You are obese.")
```

Explanation:

- Demonstrates the use of a custom function `calculate_bmi()`.
- The BMI interpretation uses basic conditional checks.
- Showcases floating-point operations and function usage.

3.9.9 Solution 74: Unit Converter

Solution:

```
def km_to_miles(km):
    return km * 0.621371

def kg_to_lbs(kg):
    return kg * 2.20462

print("1. KM to Miles")
print("2. KG to Lbs")
choice = input("Enter choice: ")

if choice == "1":
    km = float(input("Enter value in KM: "))
    print(f"Miles: {km_to_miles(km):.2f}")
elif choice == "2":
    kg = float(input("Enter value in KG: "))
    print(f"Lbs: {kg_to_lbs(kg):.2f}")
else:
    print("Invalid choice.")
```

Explanation:

- Each conversion is encapsulated in its own function.
- Demonstrates how to structure code for various units.

3.9.10 Solution 75: PDF Merger

Solution:

```
from PyPDF2 import PdfMerger

def merge_pdfs(pdf_list, output):
    merger = PdfMerger()
    for pdf in pdf_list:
```

```
            merger.append(pdf.strip())
    merger.write(output)
    merger.close()

pdfs = input("Enter PDFs to merge (comma-separated): ").split(",")
print("Merging PDF files...\n")
merge_pdfs(pdfs, "merged.pdf")
print("PDF files merged and saved as merged.pdf")
```

Explanation:

- Uses the `PyPDF2` library (`PdfMerger`) to append multiple PDF files.
- Demonstrates how to parse user input, handle file paths, and create a combined PDF.

3.9.11 Solution 76: IP Location Finder

Solution:

```
import requests

def get_ip_info(ip):
    url = f"http://ip-api.com/json/{ip}"
    response = requests.get(url)
    data = response.json()
    if data["status"] == "success":
        print(f"City: {data['city']}")
        print(f"Region: {data['regionName']}")
        print(f"Country: {data['country']}")
    else:
        print("Could not retrieve IP info.")

ip_address = input("Enter IP address: ")
get_ip_info(ip_address)
```

Explanation:

- Leverages the free `ip-api.com` service to retrieve location data.
- Demonstrates JSON handling, HTTP requests, and console-based data display.

3.9.12 Solution 77: Basic Maze Generator

Solution:

```
import random

def generate_maze(width, height):
    maze = []
```

```
    for _ in range(height):
        row = ["#" if random.random() < 0.2 else " "] * width
        maze.append(row)

    # Ensure outer boundary is walls
    for x in range(width):
        maze[0][x] = "#"
        maze[height-1][x] = "#"
    for y in range(height):
        maze[y][0] = "#"
        maze[y][width-1] = "#"

    return maze

def print_maze(maze):
    for row in maze:
        print("".join(row))

w = int(input("Enter width: "))
h = int(input("Enter height: "))
maze = generate_maze(w, h)
print("Maze:")
print_maze(maze)
```

Explanation:

- Randomly places walls (”#”) vs. open spaces (” ”) with a 20 percentage chance.
- Outer boundary is forced to be walls, ensuring a closed perimeter.
- Shows the basics of 2D list manipulation in Python.

3.9.13 Solution 78: File Duplicate Finder

Solution:

```
import os, hashlib

def get_file_md5(filepath):
    hasher = hashlib.md5()
    with open(filepath, 'rb') as f:
        buf = f.read()
        hasher.update(buf)
    return hasher.hexdigest()

def find_duplicates(directory):
    hash_map = {}
```

```
    for root, dirs, files in os.walk(directory):
        for file in files:
            path = os.path.join(root, file)
            file_hash = get_file_md5(path)
            hash_map.setdefault(file_hash, []).append(path)
    return hash_map

dir_path = input("Enter directory path: ")
result = find_duplicates(dir_path)

duplicates_found = False
for h, paths in result.items():
    if len(paths) > 1:
        duplicates_found = True
        print(", ".join(paths))

if not duplicates_found:
    print("No duplicates found.")
```

Explanation:

- Recursively walks a directory to compute MD5 hashes for each file.
- Groups files by hash, printing only entries that appear more than once.
- Demonstrates file I/O, hashing, and directory traversal.

3.9.14 Solution 79: Student Attendance System

Solution:

```
attendance = {}

def mark_attendance():
    name = input("Enter student name: ")
    attendance[name] = attendance.get(name, 0) + 1
    print(f"{name}'s attendance marked.")

def view_attendance():
    if not attendance:
        print("No attendance records.")
    else:
        for student, count in attendance.items():
            print(f"{student}: {count} days")

def reset_attendance():
    attendance.clear()
    print("All attendance records reset.")
```

```
while True:
    print("\n1. Mark Attendance\n2. View Attendance\n3. Reset Attendance\n4. Exit")
    choice = input("Enter choice: ")

    if choice == "1":
        mark_attendance()
    elif choice == "2":
        view_attendance()
    elif choice == "3":
        reset_attendance()
    elif choice == "4":
        break
    else:
        print("Invalid choice. Try again.")
```

Explanation:

- Demonstrates using a dictionary to accumulate attendance counts.
- Each student is a key, incremented whenever attendance is marked.

3.9.15 Solution 80: Basic Chat Application

Solution:

```
# server.py
import socket

server_socket = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
server_socket.bind(("localhost", 12345))
server_socket.listen(1)
print("Waiting for client...")

conn, addr = server_socket.accept()
print("Connection established.")

while True:
    data = conn.recv(1024)
    if not data:
        break
    print("Client:", data.decode())
    reply = input("Server: ")
    conn.sendall(reply.encode())

conn.close()
server_socket.close()

# client.py
```

```
import socket

client_socket = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
client_socket.connect(("localhost", 12345))
print("Connected to server.")

while True:
    message = input("You: ")
    if not message:
        break
    client_socket.sendall(message.encode())
    data = client_socket.recv(1024)
    print("Server:", data.decode())

client_socket.close()
```

Explanation:

- Demonstrates a minimal TCP client-server model with sockets in Python.
- The server waits for incoming connections; the client sends messages.
- Both read and write from the socket until the user quits.

3.9.16 Solution 81: Library Management System

Solution:

```
books = {"Python Basics": True, "Data Science": True, "AI Fundamentals": True}

def display_books():
    print("\nAvailable Books:")
    for book, available in books.items():
        if available:
            print(f"- {book}")

def borrow_book():
    book = input("Enter book name: ")
    if books.get(book, False):
        books[book] = False
        print("Book borrowed successfully!")
    else:
        print("Book not available.")

def return_book():
    book = input("Enter book name: ")
    if book in books:
        books[book] = True
```

```
        print("Book returned successfully!")

while True:
    print("\n1. View Books\n2. Borrow Book\n3. Return Book\n4. Exit")
    choice = input("Enter choice: ")

    if choice == "1":
        display_books()
    elif choice == "2":
        borrow_book()
    elif choice == "3":
        return_book()
    elif choice == "4":
        break
    else:
        print("Invalid choice. Try again.")
```

Explanation:

- Uses a dictionary to map each book to a boolean (available or borrowed).
- Conditionals and basic logic implement borrowing and returning.

3.9.17 Solution 82: Voting System

Solution:

```
votes = {}
voted_users = set()

def vote():
    user = input("Enter your ID: ")
    if user in voted_users:
        print("You have already voted!")
    else:
        candidate = input("Vote for (Alice/Bob/Charlie): ")
        votes[candidate] = votes.get(candidate, 0) + 1
        voted_users.add(user)
        print("Vote recorded!")

def results():
    print("\nVoting Results:")
    for candidate, count in votes.items():
        print(f"{candidate}: {count} votes")

while True:
    print("\n1. Vote\n2. Show Results\n3. Exit")
    choice = input("Enter choice: ")
```

```
    if choice == "1":
        vote()
    elif choice == "2":
        results()
    elif choice == "3":
        break
    else:
        print("Invalid choice. Try again.")
```

Explanation:

- The set voted$_{u}sers prevents duplicate voting. The dictionary votes tallies the votes for each candidate.$

3.9.18 Solution 83: Resume Formatter

Solution:

```
name = input("Enter your name: ")
skills = input("Enter your skills (comma-separated): ").split(", ")
experience = input("Enter years of experience: ")

resume = f"""
----------------------------
    {name.upper()}
----------------------------
Skills: {", ".join(skills)}
Experience: {experience} years
"""
print(resume)
```

Explanation:

- Demonstrates string formatting, uppercase conversion, and list manipulation.
- A practical example of converting user data into a neatly formatted resume.

3.9.19 Solution 84: Palindrome Checker

Solution:

```
def is_palindrome(text):
    text = text.replace(" ", "").lower()
    return text == text[::-1]

word = input("Enter a word or phrase: ")
print("Palindrome" if is_palindrome(word) else "Not a palindrome")
```

Explanation:

- Demonstrates simple string manipulation and slicing.
- Removing spaces and converting to lowercase ensures consistent checks.

3.9.20 Solution 85: CSV File Analyzer

Solution:

```
import csv

try:
    with open("sales.csv", "r") as file:
        reader = csv.reader(file)
        total_sales = sum(float(row[1]) for row in reader if row)
        print(f"Total Sales: \${total_sales:.2f}")
except FileNotFoundError:
    print("Error: CSV file not found.")
```

Explanation:

- CSV reading is done with the built-in `csv` module.
- Summation of the second column in each row (index `[1]`).
- Demonstrates file I/O, error handling, and floating-point arithmetic.

3.9.21 Solution 86: Password Manager

Solution:

```
from cryptography.fernet import Fernet

key = Fernet.generate_key()
cipher = Fernet(key)

def save_password():
    site = input("Enter website: ")
    password = input("Enter password: ")
    encrypted_pw = cipher.encrypt(password.encode())

    with open("passwords.txt", "a") as file:
        file.write(f"{site}: {encrypted_pw.decode()}\n")
    print("Password saved!")

def view_passwords():
    with open("passwords.txt", "r") as file:
        print(file.read())

while True:
```

```
    print("\n1. Save Password\n2. View Passwords\n3. Exit")
    choice = input("Enter choice: ")

    if choice == "1":
        save_password()
    elif choice == "2":
        view_passwords()
    elif choice == "3":
        break
    else:
        print("Invalid choice. Try again.")
```

Explanation:

- Uses the `cryptography` library to generate a key and encrypt passwords.
- Stores the encrypted passwords in a text file; the key must be kept safe.

3.9.22 Solution 87: Random Password Generator

Solution:

```
import random
import string

def generate_password(length=12):
    characters = string.ascii_letters + string.digits + string.punctuation
    password = ''.join(random.choice(characters) for _ in range(length))
    return password

print("Generated Password:", generate_password())
```

Explanation:

- Demonstrates randomization with `random.choice`.
- Uses a combination of ASCII letters, digits, and punctuation to produce a secure password.

3.9.23 Solution 88: Email Validator

Solution:

```
import re

def validate_email(email):
    pattern = r'^[a-zA-Z0-9._%+-]+@[a-zA-Z0-9.-]+\.[a-zA-Z]{2,}$'
    return bool(re.match(pattern, email))
```

```
email = input("Enter an email: ")
print("Valid" if validate_email(email) else "Invalid")
```

Explanation:

- Uses a simple regex pattern for basic email validation.
- Ensures an @ symbol and a valid domain structure.

3.9.24 Solution 89: Weather Forecast App

Solution:

```
import requests

API_KEY = "your_api_key"  # Replace with your OpenWeatherMap API Key
BASE_URL = "http://api.openweathermap.org/data/2.5/weather?"

city = input("Enter city name: ")
url = f"{BASE_URL}q={city}&appid={API_KEY}&units=metric"

response = requests.get(url)
data = response.json()

if data["cod"] != "404":
    weather = data["main"]
    temperature = weather["temp"]
    description = data["weather"][0]["description"]
    print(f"Temperature: {temperature}°C")
    print(f"Weather: {description.capitalize()}")
else:
    print("City not found.")
```

Explanation:

- Demonstrates simple HTTP requests with the `requests` library.
- Uses JSON parsing to extract temperature and description.

3.9.25 Solution 90: Stock Price Tracker

Solution:

```
import requests

API_KEY = "your_api_key"  # Replace with Alpha Vantage API Key
BASE_URL = "https://www.alphavantage.co/query?"

symbol = input("Enter stock symbol (e.g., AAPL, TSLA): ")
```

```
params = {"function": "GLOBAL_QUOTE", "symbol": symbol, "apikey": API_KEY}

response = requests.get(BASE_URL, params=params)
data = response.json()

if "Global Quote" in data:
    price = data["Global Quote"]["05. price"]
    print(f"Current price of {symbol}: \${price}")
else:
    print("Stock symbol not found.")
```

Explanation:

- Shows how to handle query parameters with the `requests` library.
- Parses JSON data to extract stock prices from a real-time API.

3.9.26 Solution 91: Text Summarizer

Solution:

```
import nltk
from nltk.tokenize import sent_tokenize

nltk.download("punkt")

text = input("Enter text to summarize: ")
sentences = sent_tokenize(text)

summary = " ".join(sentences[:3]) if len(sentences) > 3 else text
print("Summary:", summary)
```

Explanation:

- Uses NLTK's sentence tokenizer to split text into sentences.
- Returns only the first three sentences as a naive summary approach.

3.9.27 Solution 92: Book Recommendation System

Solution:

```
books = {
    "Fiction": ["The Great Gatsby", "To Kill a Mockingbird"],
    "Science": ["A Brief History of Time", "The Selfish Gene"],
    "Technology": ["The Pragmatic Programmer", "Clean Code"]
}

genre = input("Enter a genre (Fiction, Science, Technology): ")
```

```
recommendations = books.get(genre, ["No recommendations available."])

print("Recommended Books:", ", ".join(recommendations))
```

Explanation:

- Demonstrates dictionary lookups to match a genre to a list of titles.
- Basic example of a recommendation system using static data.

3.9.28 Solution 93: File Encryption & Decryption

Solution:

```
from cryptography.fernet import Fernet

key = Fernet.generate_key()
cipher = Fernet(key)

def encrypt_file(filename):
    with open(filename, "rb") as file:
        data = file.read()
    encrypted_data = cipher.encrypt(data)
    with open(f"{filename}.enc", "wb") as file:
        file.write(encrypted_data)
    print("File encrypted.")

def decrypt_file(filename):
    with open(filename, "rb") as file:
        encrypted_data = file.read()
    decrypted_data = cipher.decrypt(encrypted_data)
    with open("decrypted.txt", "wb") as file:
        file.write(decrypted_data)
    print("File decrypted.")

encrypt_file("example.txt")
decrypt_file("example.txt.enc")
```

Explanation:

- Uses the `cryptography` library for file encryption.
- Demonstrates reading/writing binary data and generating secure keys.

3.9.29 Solution 94: Two-Factor Authentication (2FA) Generator

Solution:

```python
import random

def generate_2fa_code():
    return random.randint(100000, 999999)

code = generate_2fa_code()
print(f"Your 2FA Code: {code}")

user_input = int(input("Enter the code: "))
if user_input == code:
    print("Access granted.")
else:
    print("Invalid code.")
```

Explanation:

- A random 6-digit integer serves as a basic 2FA code.
- Real-world 2FA solutions typically use time-based one-time passwords (TOTP).

3.9.30 Solution 95: Employee Database System

Solution:

```python
employees = {}

def add_employee():
    id = input("Enter employee ID: ")
    name = input("Enter employee name: ")
    position = input("Enter position: ")
    employees[id] = {"Name": name, "Position": position}
    print("Employee added.")

def list_employees():
    for id, details in employees.items():
        print(f"{id}: {details['Name']} - {details['Position']}")

while True:
    print("\n1. Add Employee\n2. View Employees\n3. Exit")
    choice = input("Enter choice: ")
    if choice == "1":
        add_employee()
    elif choice == "2":
        list_employees()
    elif choice == "3":
        break
    else:
        print("Invalid choice. Try again.")
```

Explanation:

- Stores employee info in a dictionary, mapping an ID to a sub-dictionary.
- Demonstrates nested data structures and a simple CLI menu.

3.9.31 Solution 96: Chatbot for FAQs

Solution:

```
faqs = {
    "What is Lkhibra Academy?": "Lkhibra Academy is an online learning platform.",
    "What courses are available?": "Python, Data Science, AI, and more!",
    "How do I enroll?": "Visit our website and sign up for a course."
}

while True:
    question = input("Ask a question: ")
    if question.lower() == "exit":
        break
    print(faqs.get(question, "Sorry, I don't understand that question."))
```

Explanation:

- A dictionary maps specific questions to predefined answers.
- If a question doesn't match, a default response is returned.

3.9.32 Solution 97: Automated Invoice Generator

Solution:

```
customer = input("Enter customer name: ")
items = []
while True:
    item = input("Enter item (or 'done' to finish): ")
    if item.lower() == "done":
        break
    price = float(input("Enter price: "))
    items.append((item, price))

total = sum(price for _, price in items)

invoice = f"Invoice for {customer}\n" + "-" * 30 + "\n"
for item, price in items:
    invoice += f"{item}: \${price:.2f}\n"
invoice += f"Total: \${total:.2f}\n"

with open("invoice.txt", "w") as file:
```

```
    file.write(invoice)

print("Invoice saved to invoice.txt")
```

Explanation:

- Gathers multiple item/price pairs from the user.
- Calculates a grand total and saves a formatted invoice to a file.

3.9.33 Solution 98: Task Reminder with Notifications

Solution:

```
import time

tasks = {"Submit project": 5, "Meeting with team": 10}
print("Task Reminder System Started!")

for task, delay in tasks.items():
    time.sleep(delay)  # Simulates waiting until the task is due
    print(f"Reminder: {task} is due now!")
```

Explanation:

- Maps each task to a delay (in seconds). In a real system, scheduling libraries or OS-level notifications can be used.
- Demonstrates basic time-based reminders.

3.9.34 Solution 99: URL Shortener

Solution:

```
import random
import string

urls = {}

def shorten_url(long_url):
    short = ''.join(random.choices(string.ascii_letters + string.digits, k=6))
    urls[short] = long_url
    return short

long_url = input("Enter URL to shorten: ")
short_url = shorten_url(long_url)
print(f"Shortened URL: https://short.ly/{short_url}")
```

Explanation:

- Generates a random 6-character string to serve as the short link.
- In a production system, you'd store these mappings in a database.

3.9.35 Solution 100: AI-Powered Sentiment Analyzer

Solution:

```
positive_words = {"happy", "great", "fantastic", "excellent", "love"}
negative_words = {"sad", "bad", "terrible", "awful", "hate"}

def analyze_sentiment(text):
    words = set(text.lower().split())
    if words & positive_words:
        return "Positive Sentiment"
    elif words & negative_words:
        return "Negative Sentiment"
    else:
        return "Neutral Sentiment"

text = input("Enter a sentence: ")
print(analyze_sentiment(text))
```

Explanation:

- Demonstrates a very naive approach to sentiment analysis by intersecting words with sets of positive or negative terms.
- Real-world implementations often rely on advanced NLP libraries.