FEU INSTITUTE OF TECHNOLOGY

COLLEGE OF COMPUTER STUDIES

IT0011
Integrative Programming and Technologies

EXERCISE

3

String and File Handling

Student Name:	Mary Ann B. Camacho
Section:	TB21
Professor:	Joseph Calleja

I. PROGRAM OUTCOME (PO) ADDRESSED

Analyze a complex problem and identify and define the computing requirements appropriate to its solution.

II. LEARNING OUTCOME (LO) ADDRESSED

Utilize string manipulation techniques and file handling in Python

III. INTENDED LEARNING OUTCOMES (ILO)

At the end of this exercise, students must be able to:

- Perform common string manipulations, such as concatenation, slicing, and formatting.
- Understand and use file handling techniques to read from and write to files in Python.
- Apply string manipulation and file handling to solve practical programming problems.

IV. BACKGROUND INFORMATION

String Manipulation:

String manipulation is a crucial aspect of programming that involves modifying and processing textual data. In Python, strings are versatile, and several operations can be performed on them. This exercise focuses on fundamental string manipulations, including concatenation (combining strings), slicing (extracting portions of strings), and formatting (constructing dynamic strings).

Common String Methods:

- len(): Returns the length of a string.
- lower(), upper(): Convert a string to lowercase or uppercase.
- replace(): Replace a specified substring with another.
- count(): Count the occurrences of a substring within a string.

File Handling:

File handling is essential for reading and writing data to external files, providing a way to store and retrieve information. Python offers straightforward mechanisms for file manipulation. This exercise introduces the basics of file handling, covering the opening and closing of files, as well as reading from and writing to text files.

Understanding File Modes:

- 'r' (read): Opens a file for reading.
- 'w' (write): Opens a file for writing, overwriting the file if it exists.
- 'a' (append): Opens a file for writing, appending to the end of the file if it exists.

Understanding string manipulation and file handling is fundamental for processing and managing data in Python programs. String manipulations allow for the transformation and extraction of information from textual data, while file handling enables interaction with external data sources. Both skills are essential for developing practical applications and solving real-world programming challenges. The exercises in this session aim to reinforce these concepts through hands-on practice and problem-solving scenarios.

V. GRADING SYSTEM / RUBRIC

Criteria	Excellent (5)	Good (4)	Satisfactory (3)	Needs Improvement (2)	Unsatisfactory (1)
Correctness	Code functions correctly and meets all requirements.	Code mostly functions as expected and meets most requirements.	Code partially functions but may have logical errors or missing requirements.	Code has significant errors, preventing proper execution.	Code is incomplete or not functioning.
Code Structure	Code is well-organized with clear structure and proper use of functions.	Code is mostly organized with some room for improvement in structure and readability.	Code lacks organization, making it somewhat difficult to follow.	Code structure is chaotic, making it challenging to understand.	Code lacks basic organization.
Documentation	Comprehensive comments and docstrings provide clarity on the code's purpose.	Sufficient comments and docstrings aid understanding but may lack details in some areas.	Limited comments, making it somewhat challenging to understand the code.	Minimal documentation, leaving significant gaps in understanding.	No comments or documentation provided.
Coding Style	Adheres to basic coding style guidelines, with consistent and clean practices.	Mostly follows coding style guidelines, with a few style inconsistencies.	Style deviations are noticeable, impacting code readability.	Significant style issues, making the code difficult to read.	No attention to coding style; the code is messy and unreadable.
Effort and Creativity	Demonstrates a high level of effort and creativity, going beyond basic requirements.	Shows effort and creativity in addressing most requirements.	Adequate effort but lacks creativity or exploration beyond the basics.	Minimal effort and creativity evident.	Little to no effort or creativity apparent.

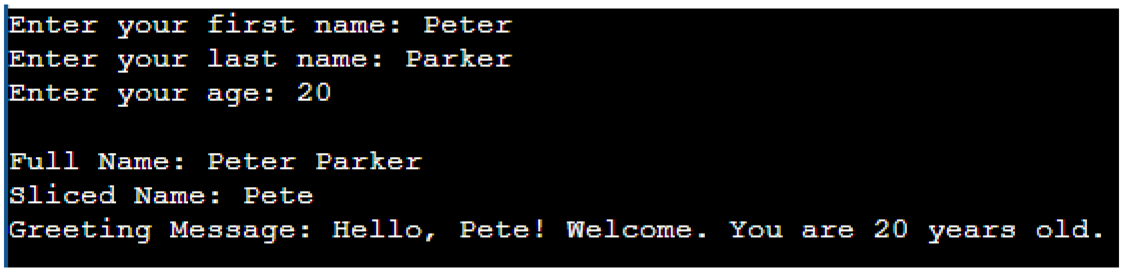
VI. LABORATORY ACTIVITY

INSTRUCTIONS:
Copy your source codes to be pasted in this document as well as a screen shot of your running output.

3.1. Activity for Performing String Manipulations
Objective: To perform common and practical string manipulations in Python.

Task: Write a Python program that includes the following string manipulations:
- Concatenate your first name and last name into a full name.
- Slice the full name to extract the first three characters of the first name.
- Use string formatting to create a greeting message that includes the sliced first name

Sample Output

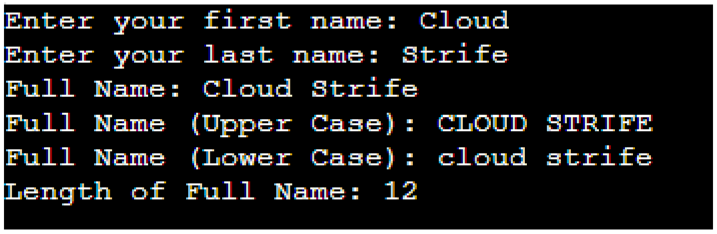


```
Enter your first name: Peter
Enter your last name: Parker
Enter your age: 20

Full Name: Peter Parker
Sliced Name: Pete
Greeting Message: Hello, Pete! Welcome. You are 20 years old.
```

3.2 Activity for Performing String Manipulations
Objective: To perform common and practical string manipulations in Python.

Task: Write a Python program that includes the following string manipulations:
- Input the user's first name and last name.
- Concatenate the input names into a full name.
- Display the full name in both upper and lower case.
- Count and display the length of the full name

Sample Output

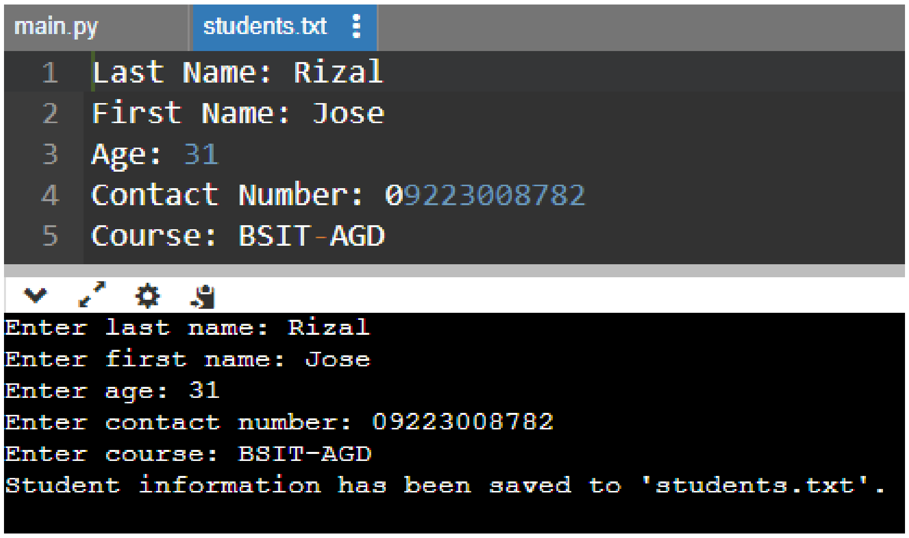


```
Enter your first name: Cloud
Enter your last name: Strife
Full Name: Cloud Strife
Full Name (Upper Case): CLOUD STRIFE
Full Name (Lower Case): cloud strife
Length of Full Name: 12
```

3.3. Practical Problem Solving with String Manipulation and File Handling
Objective: Apply string manipulation and file handling techniques to store student information in a file.

Task: Write a Python program that does the following:

- Accepts input for the last name, first name, age, contact number, and course from the user.
- Creates a string containing the collected information in a formatted way.
- Opens a file named "students.txt" in append mode and writes the formatted information to the file.
- Displays a confirmation message indicating that the information has been saved.

Sample Output


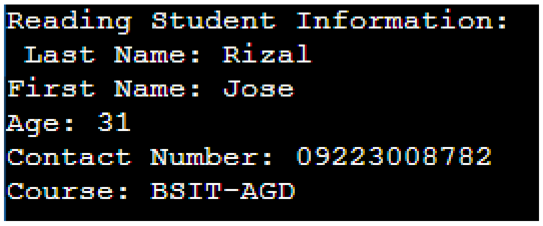
```
Last Name: Rizal
First Name: Jose
Age: 31
Contact Number: 09223008782
Course: BSIT-AGD
```

```
Enter last name: Rizal
Enter first name: Jose
Enter age: 31
Enter contact number: 09223008782
Enter course: BSIT-AGD
Student information has been saved to 'students.txt'.
```

3.4 Activity for Reading File Contents and Display
Objective: Apply file handling techniques to read and display student information from a file.

Task: Write a Python program that does the following:

- Opens the "students.txt" file in read mode.
- Reads the contents of the file.
- Displays the student information to the user

Sample Output


```
Reading Student Information:
 Last Name: Rizal
First Name: Jose
Age: 31
Contact Number: 09223008782
Course: BSIT-AGD
```

QUESTION AND ANSWER:

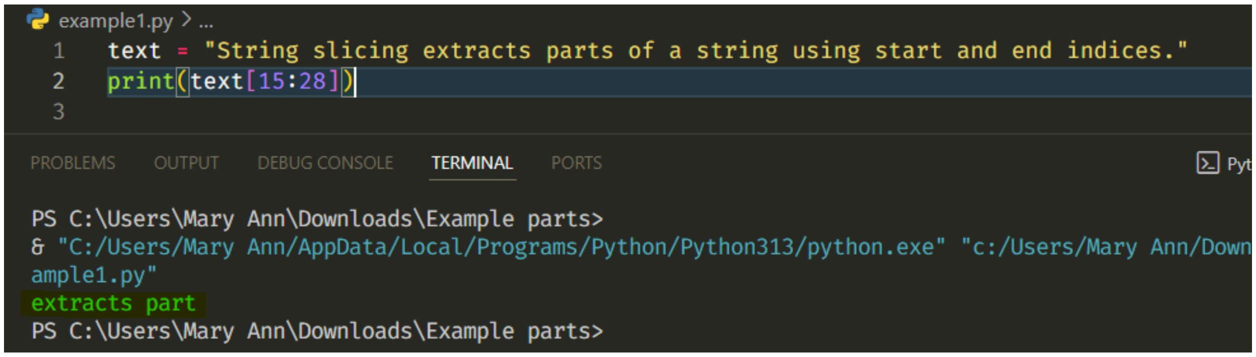
1. How does the format() function help in combining variables with text in Python? Can you provide a simple example?
The format() method allows values to be dynamically placed into a string, which facilitates the combination of variables with text. It offers a legible and organized method for inserting values into placeholders {}.

2. Explain the basic difference between opening a file in 'read' mode ('r') and 'write' mode ('w') in Python. When would you use each
When you open a file in'read' mode ('r'), you may access and read its contents without changing anything. An error will be returned if the file does not exist. This mode helps when you simply need to retrieve data from a file and not change it. When you open a file in 'write' mode ('w'), you can either create a new file or modify an existing one. If the file had previously existed, its existing contents will be removed. This mode is suitable for storing new info in a file, ensures that only the latest information is stored.

3. Describe what string slicing is in Python. Provide a basic example of extracting a substring from a larger string.
Python's string slicing feature lets you define a range of indexes to retrieve specific elements of a more expansive text. The syntax is string[start:end], where start is the index at which slicing starts and end (without including the end index) is the index at which slicing finishes. Substrings can be retrieved with this method without changing the original string.

To extract "extract part" from the text "String slicing extracts parts of a string using start and end indices.", we must first identify the required substring's beginning and ending indices.

```
text = "String slicing extracts parts of a string using start and end indices."
print(text[15:28])
```



example1.py > ...

```
text = "String slicing extracts parts of a string using start and end indices."
print(text[15:28])
```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS

```
PS C:\Users\Mary Ann\Downloads\Example parts>
& "C:/Users/Mary Ann/AppData/Local/Programs/Python/Python313/python.exe" "c:/Users/Mary Ann/Down
ample1.py"
extracts part
PS C:\Users\Mary Ann\Downloads\Example parts>
```

4. When saving information to a file in Python, what is the purpose of using the 'a' mode instead of the 'w' mode? Provide a straightforward example.

When storing data in a file, using the 'a' (append) mode rather than the 'w' (write) option guarantees that new information is appended to the current content rather than overwritten. When you need to preserve a record of several entries, such when logging data or keeping a list, the 'a' mode is helpful.

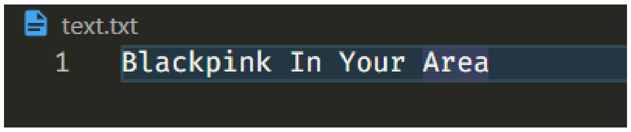
Before:

--Manual Version--

```
# Define a string variable containing text information
Textinfo = "Blackpink in your area"

# Open the file in append mode ('a') to add new data without deleting existing content
# Note: Change the file path to your preferred location if needed.
f = open("C:\\Users\\Mary Ann\\Downloads\\Example parts\\text.txt", "a")

# Write the content of Textinfo to the file
f.writelines(Textinfo)

# Close the file to ensure changes are saved
f.close()
```

--Automatic Version—

```
# Define a string variable containing text information
Textinfo = "Blackpink in your area"


# Open a file named students.txt in append mode and write the formatted info to the file
with open("text.txt ", "a") as f:
    f.write(Textinfo)
```



```
text.txt
Blackpink In Your Area
```

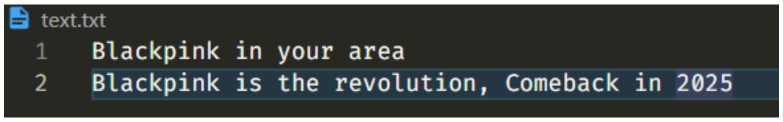
After:

--Manual Version--

```
# Open the file again in append mode to add more text on a new line
# Note: If you want to store this file elsewhere, modify the path accordingly.
f = open("C:\\Users\\Mary Ann\\Downloads\\Example parts\\text.txt", "a")

# Write a new sentence with a newline character to move to the next line
f.write("\nBlackpink is the revolution, Comeback in 2025")

# Close the file to save the changes
f.close()
```

--Automatic Version—

```
# Open a file named students.txt in append mode and write the formatted info to the file
with open("text.txt ", "a") as f:

# Write a new sentence with a newline character to move to the next line
      f.write("\nBlackpink is the revolution, Comeback in 2025")
```



```
Blackpink in your area
Blackpink is the revolution, Comeback in 2025
```

5. Write a simple Python code snippet to open and read a file named "data.txt." How would you handle the case where the file might not exist?

--Manual Version--

```
# Open the file in read mode
# Note: it depends where you want to put the file
f = open("C:\\Users\\Mary Ann\\Downloads\\Example parts\\data.txt", "r")

# Read and print the file content
print(f.read())

# Close the file
f.close()
```

--Automatic Version—

```
 # Import os module for handling file paths
import os

# Automatically locate the file in the same directory as the script
file_path = os.path.join(os.getcwd(), "data.txt")

# Open the file in read mode
f = open(file_path, "r")

# Read and print the file content
print(f.read())

# Close the file
f.close()
```

To handle the case where a file might not exist when opening and reading it, you can check for its existence before opening or use error handling to prevent crashes. One way is to use os.path.exists() or pathlib.Path.exists() to verify if the file exists before attempting to open it. Another approach is to open the file in a mode that ensures it gets created if it doesn't exist, such as 'a' (append mode). Additionally, using try-except with FileNotFoundError can gracefully catch the error and display a message instead of crashing the program. These methods ensure smooth file handling without unexpected interruptions.