

**COLLEGE OF COMPUTER STUDIES**

**IT0011**

**Integrative Programming and Technologies**

**EXERCISE**

**3**

**String and File Handling**

|  |  |
| --- | --- |
| **Student Name:** | Raymond Gonzales Jr |
| **Section:** | TW24 |
| **Professor:** | Sr. 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**

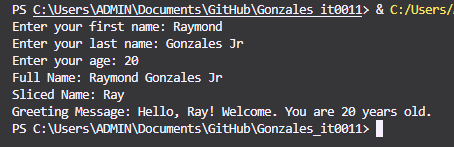
Objective: To perform common and practical string manipulations in Python.

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

****

****

first\_name = input("Enter your first name: ")

last\_name = input("Enter your last name: ")

age = *int*(input("Enter your age: "))

full\_name = *f*"{first\_name} {last\_name}"

sliced\_name = first\_name[:3]

greeting\_message = *f*"Hello, {sliced\_name}! Welcome. You are {age} years old."

print(*f*"Full Name: {full\_name}")

print(*f*"Sliced Name: {sliced\_name}")

print(*f*"Greeting Message: {greeting\_message}")

**3.2 Activity for Performing String Manipulations**

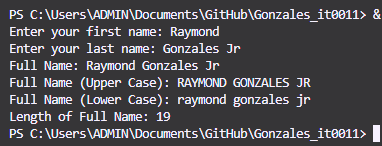
Objective: To perform common and practical string manipulations in Python.

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

****

****

first\_name = input("Enter your first name: ")

last\_name = input("Enter your last name: ")

full\_name = *f*"{first\_name} {last\_name}"

full\_name\_upper = full\_name.upper()

full\_name\_lower = full\_name.lower()

name\_length = len(full\_name)

print(*f*"Full Name: {full\_name}")

print(*f*"Full Name (Upper Case): {full\_name\_upper}")

print(*f*"Full Name (Lower Case): {full\_name\_lower}")

print(*f*"Length of Full Name: {name\_length}")

**3.3. Practical Problem Solving with String Manipulation and File Handling**

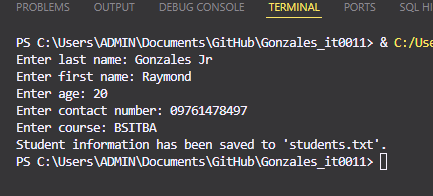
Objective: Apply string manipulation and file handling techniques to store student information in a file.

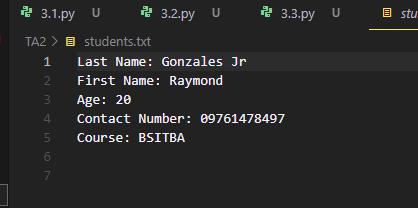
Task: Write a Python program that does the following:

* 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 = input("Enter last name: ")

first\_name = input("Enter first name: ")

age = input("Enter age: ")

contact\_number = input("Enter contact number: ")

course = input("Enter course: ")

student\_info = *f*"Last Name: {last\_name}\nFirst Name: {first\_name}\nAge: {age}\nContact Number: {contact\_number}\nCourse: {course}\n\n"

with open("students.txt", "a") as file:

    file.write(student\_info)

print("Student information has been saved to 'students.txt'.")

**3.4 Activity for Reading File Contents and Display**

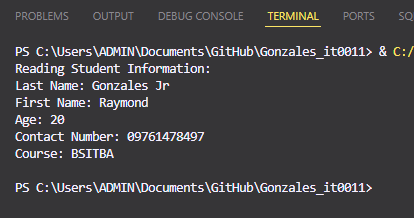
Objective: Apply file handling techniques to read and display student information from a file.

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

****

****

import os

script\_dir = os.path.dirname(os.path.abspath(\_\_file\_\_))

file\_path = os.path.join(script\_dir, "students.txt")

try:

    with open(file\_path, "r") as file:

        lines = file.readlines()

    print("Reading Student Information:")

    for line in lines:

        print(line.strip())

except *FileNotFoundError*:

    print(*f*"Error: The file 'students.txt' was not found in {script\_dir}. Please check the file location.")

**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() function helps combine variables with text by replacing placeholders {} with values. It makes string formatting easier and more readable.

1. Explain the basic difference between opening a file in 'read' mode ('r') and 'write' mode ('w') in Python. When would you use each

* 'r' mode is for reading a file and gives an error if the file doesn’t exist. 'w' mode is for writing and creates a new file or overwrites an existing one. Use 'r' to read data and 'w' to write new content.

1. Describe what string slicing is in Python. Provide a basic example of extracting a substring from a larger string.

* String slicing extracts a portion of a string using index positions. It allows selecting specific parts of a string by defining a start and end position.

1. 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.

* 'a' mode appends new data to a file without deleting existing content, while 'w' overwrites everything. Use 'a' to add information without losing previous data.

1. 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?

* To open and read a file, use 'r' mode, and to handle missing files, use a try-except block. This prevents errors if the file is not found.