



**FEU INSTITUTE OF TECHNOLOGY**

**COLLEGE OF COMPUTER STUDIES**

**IT0011**

**Integrative Programming and  
Technologies**

**EXERCISE**

---

**3**

**String and File Handling**

<b>Student Name:</b>	<b>Derek John E. Bantad</b>
<b>Section:</b>	<b>TC21</b>
<b>Professor:</b>	<b>Joseph Calleja</b>

## **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)
<b>Correctness</b>	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.
<b>Code Structure</b>	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.
<b>Documentation</b>	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.
<b>Coding Style</b>	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.
<b>Effort and Creativity</b>	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.
```

### CODE:

```
first_name = input("Enter your first name: ")
last_name = input("Enter your last name: ")
age = input("Enter your age: ")

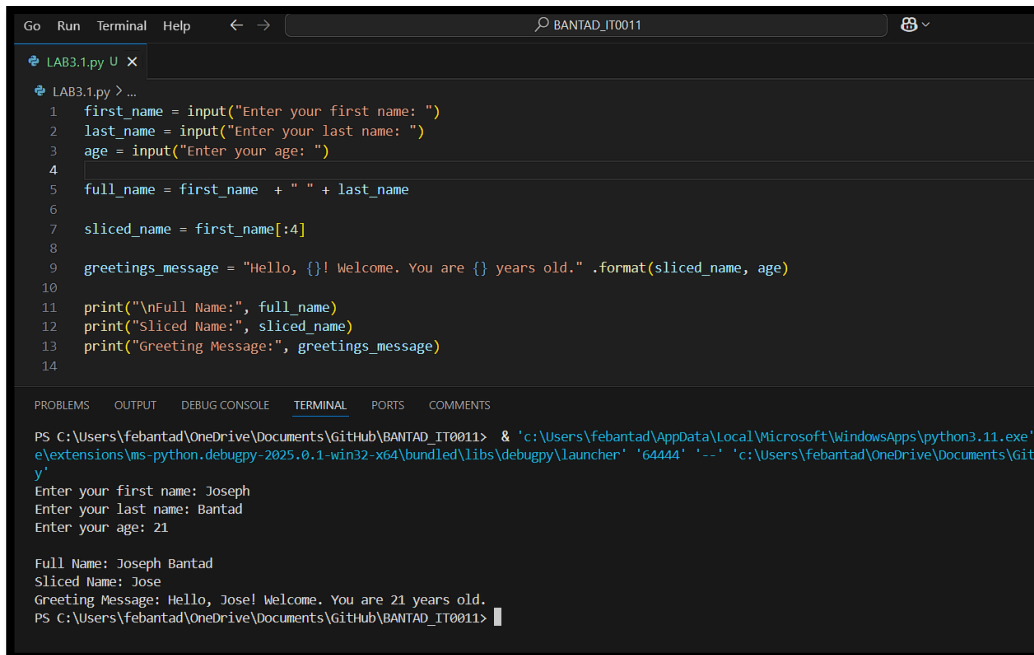
full_name = first_name + " " + last_name

sliced_name = first_name[:4]

greetings_message = "Hello, {}! Welcome. You are {} years old." .format(sliced_name, age)

print("\nFull Name:", full_name)
print("Sliced Name:", sliced_name)
print("Greeting Message:", greetings_message)
```

## OUTPUT:



The screenshot shows a Python IDE with a file named LAB3.1.py. The code in the editor is as follows:

```
1 first_name = input("Enter your first name: ")
2 last_name = input("Enter your last name: ")
3 age = input("Enter your age: ")
4
5 full_name = first_name + " " + last_name
6
7 sliced_name = first_name[:4]
8
9 greetings_message = "Hello, {}! Welcome. You are {} years old." .format(sliced_name, age)
10
11 print("\nFull Name:", full_name)
12 print("Sliced Name:", sliced_name)
13 print("Greeting Message:", greetings_message)
14
```

The terminal output at the bottom shows the execution of the program with the following input and output:

```
PS C:\Users\febantad\OneDrive\Documents\GitHub\BANTAD_IT0011> & 'c:\Users\febantad\AppData\Local\Microsoft\WindowsApps\python3.11.exe'
e\extensions\ms-python.debugpy-2025.0.1-win32-x64\bundled\libs\debugpy\launcher' '64444' '--' 'c:\Users\febantad\OneDrive\Documents\Giti
y'
Enter your first name: Joseph
Enter your last name: Bantad
Enter your age: 21

Full Name: Joseph Bantad
Sliced Name: Jose
Greeting Message: Hello, Jose! Welcome. You are 21 years old.
PS C:\Users\febantad\OneDrive\Documents\GitHub\BANTAD_IT0011>
```

### 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
```

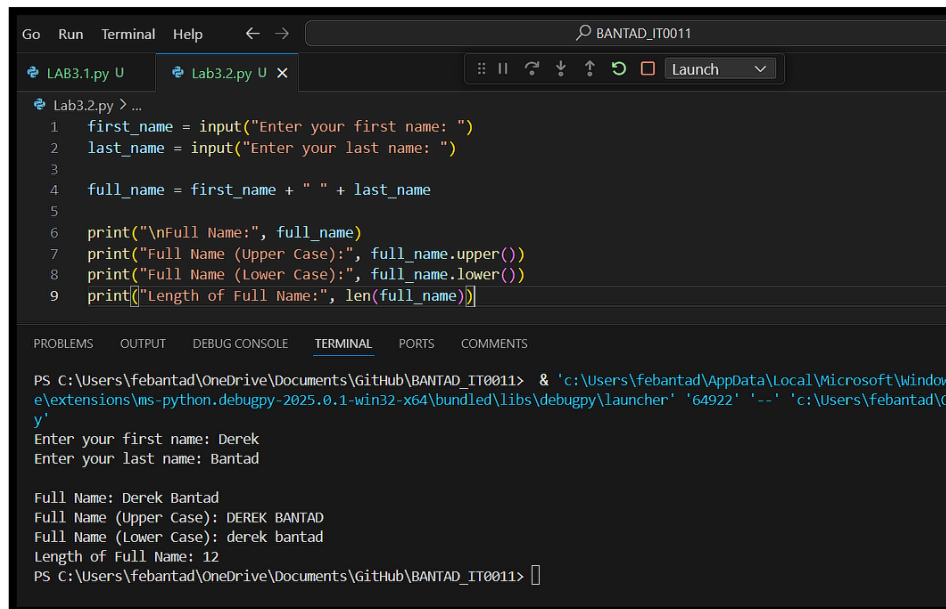
## CODE:

```
first_name = input("Enter your first name: ")
last_name = input("Enter your last name: ")

full_name = first_name + " " + last_name

print("\nFull Name:", full_name)
print("Full Name (Upper Case):", full_name.upper())
print("Full Name (Lower Case):", full_name.lower())
print("Length of Full Name:", len(full_name))
```

## OUTPUT:



The screenshot shows a Python IDE window titled 'BANTAD\_IT0011'. The editor displays a Python script (Lab3.2.py) that takes user input for first and last names, concatenates them with a space, and then prints the full name in its original case, uppercase, lowercase, and its length. The terminal output shows the program being run from a command prompt, with the user entering 'Derek' for the first name and 'Bantad' for the last name. The resulting output is: 'Full Name: Derek Bantad', 'Full Name (Upper Case): DEREK BANTAD', 'Full Name (Lower Case): derek bantad', and 'Length of Full Name: 12'.

```
Go Run Terminal Help < -> BANTAD_IT0011
LAB3.1.py U Lab3.2.py U X
Lab3.2.py > ...
1 first_name = input("Enter your first name: ")
2 last_name = input("Enter your last name: ")
3
4 full_name = first_name + " " + last_name
5
6 print("\nFull Name:", full_name)
7 print("Full Name (Upper Case):", full_name.upper())
8 print("Full Name (Lower Case):", full_name.lower())
9 print("Length of Full Name:", len(full_name))

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS COMMENTS
PS C:\Users\febantad\OneDrive\Documents\GitHub\BANTAD_IT0011> & 'c:\Users\febantad\AppData\Local\Microsoft\Windows\extensions\ms-python.debugpy-2025.0.1-win32-x64\bundle\libs\debugpy\launcher' '64922' '--' 'c:\Users\febantad\O
y'
Enter your first name: Derek
Enter your last name: Bantad

Full Name: Derek Bantad
Full Name (Upper Case): DEREK BANTAD
Full Name (Lower Case): derek bantad
Length of Full Name: 12
PS C:\Users\febantad\OneDrive\Documents\GitHub\BANTAD_IT0011> 
```

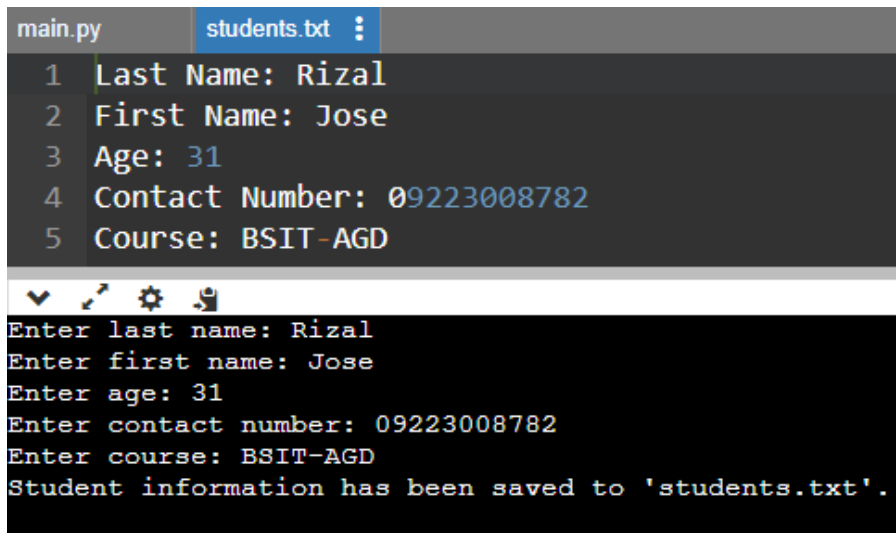
### 3.3. Practical Problem Solving with String Manipulation and File Handling

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



The screenshot shows a code editor with two tabs: 'main.py' and 'students.txt'. The 'main.py' tab is active, displaying a Python script with five lines of code. The 'students.txt' tab is also visible, showing the output of the script. The output is a text file containing the student information entered during the script's execution.

```
main.py students.txt
1 Last Name: Rizal
2 First Name: Jose
3 Age: 31
4 Contact Number: 09223008782
5 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'.
```

### CODE:

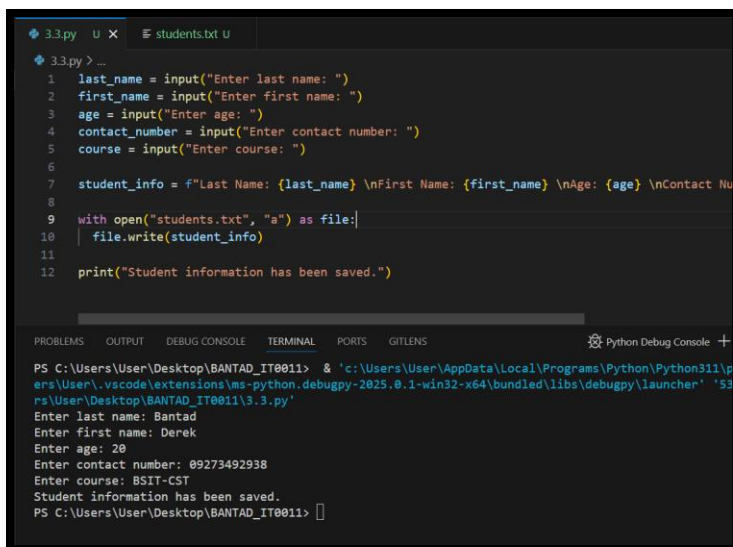
```
last_name = input("Enter last name: ")
first_name = input("Enter first name: ")
age = input("Enter age: ")
contact_number = input("Enter contact number: ")
course = input("Enter course: ")

student_info = f"Last Name: {last_name} \nFirst Name: {first_name} \nAge: {age} \nContact Number: {contact_number} \nCourse: {course} \n"

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

print("Student information has been saved.")
```

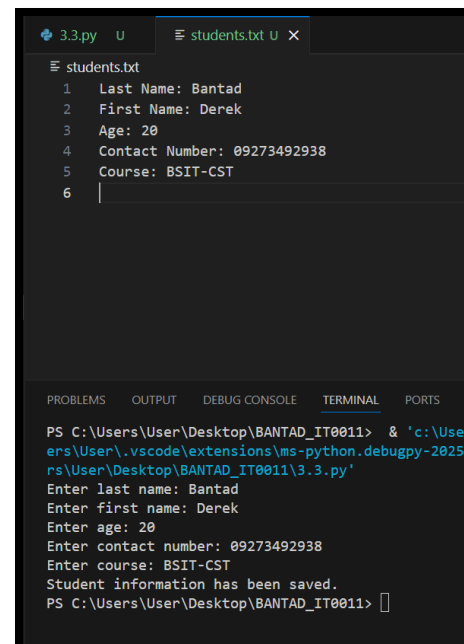
### OUTPUT:



The screenshot shows a code editor with two tabs: '3.3.py' and 'students.txt'. The '3.3.py' tab is active, displaying a Python script with 12 lines of code. The 'students.txt' tab is also visible, showing the output of the script. The output is a text file containing the student information entered during the script's execution.

```
3.3.py students.txt
1 last_name = input("Enter last name: ")
2 first_name = input("Enter first name: ")
3 age = input("Enter age: ")
4 contact_number = input("Enter contact number: ")
5 course = input("Enter course: ")
6
7 student_info = f"Last Name: {last_name} \nFirst Name: {first_name} \nAge: {age} \nContact Nu
8
9 with open("students.txt", "a") as file:
10 | file.write(student_info)
11
12 print("Student information has been saved.")

PS C:\Users\User\Desktop\BANTAD_IT0011> & 'c:\Users\User\AppData\Local\Programs\Python\Python311\p
ers\User\.vscode\extensions\ms-python.debugpy-2025.0.1-win32-x64\bundle\libs\debugpy\launcher' '53
rs\User\Desktop\BANTAD_IT0011\3.3.py'
Enter last name: Bantad
Enter first name: Derek
Enter age: 20
Enter contact number: 09273492938
Enter course: BSIT-CST
Student information has been saved.
PS C:\Users\User\Desktop\BANTAD_IT0011>
```



The screenshot shows a code editor with two tabs: '3.3.py' and 'students.txt'. The 'students.txt' tab is active, displaying the output of the script. The output is a text file containing the student information entered during the script's execution.

```
3.3.py students.txt
1 Last Name: Bantad
2 First Name: Derek
3 Age: 20
4 Contact Number: 09273492938
5 Course: BSIT-CST
6

PS C:\Users\User\Desktop\BANTAD_IT0011> & 'c:\Users\User\AppData\Local\Programs\Python\Python311\p
ers\User\.vscode\extensions\ms-python.debugpy-2025.0.1-win32-x64\bundle\libs\debugpy\launcher' '53
rs\User\Desktop\BANTAD_IT0011\3.3.py'
Enter last name: Bantad
Enter first name: Derek
Enter age: 20
Enter contact number: 09273492938
Enter course: BSIT-CST
Student information has been saved.
PS C:\Users\User\Desktop\BANTAD_IT0011>
```

### 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

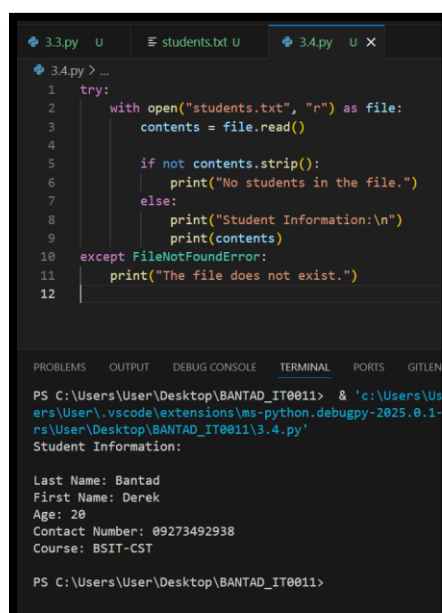
```
Reading Student Information:
Last Name: Rizal
First Name: Jose
Age: 31
Contact Number: 09223008782
Course: BSIT-AGD
```

CODE:

```
try:
    with open("students.txt", "r") as file:
        contents = file.read()

    if not contents.strip():
        print("No students in the file.")
    else:
        print("Student Information:\n")
        print(contents)
except FileNotFoundError:
    print("The file does not exist.")
```

OUTPUT:



The screenshot shows a code editor with a Python file named 3.4.py. The code is as follows:

```
1 try:
2     with open("students.txt", "r") as file:
3         contents = file.read()
4
5     if not contents.strip():
6         print("No students in the file.")
7     else:
8         print("Student Information:\n")
9         print(contents)
10 except FileNotFoundError:
11     print("The file does not exist.")
12
```

Below the code editor is a terminal window. The command executed is:

```
PS C:\Users\User\Desktop\BANTAD_IT0011> & 'c:\Users\User\OneDrive\Desktop\BANTAD_IT0011\3.4.py'
```

The output of the program is:

```
Student Information:

Last Name: Bantad
First Name: Derek
Age: 20
Contact Number: 09273492938
Course: BSIT-CST
```



## QUESTION AND ANSWER:

1. How does the format() function help in combining variables with text in Python? Can you provide a simple example?

For example, the curly braces {} serve as placeholders for the variables name and age. When using format(), variables can be passed through as arguments in the function by order as they should appear in the string.

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

Difference is 'r' in read mode, is opened in a file for reading only and 'w' in write mode, is opened in a file for writing only. When using for 'r' is it only needs to read the contents of a file and 'w' is when you want to write in a file.

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

String slicing is when a statement extracts its portion of a string by ranging from the statement. In this example, string[7:12] it extracts the characters from index 7 to 11 from the string.

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.

The 'a' mode in python file handling is used to append data to the end of the file, without overwriting the existing content.

Example: Suppose you have a file called "data.txt" with the following content:

Line 1

Line 2

If you use the 'w' mode to write to the file:

with open('data.txt', 'w') as file:

file.write("New line")

The content of "data.txt" will be overwritten, and it will only contain:

New line

However, if you use the 'a' mode to append to the file:

with open('data.txt', 'a') as file:

file.write("\nAppended line")

The content of "data.txt" will be:

Line 1

Line 2

Appended line

In this case, the new data is added to the end of the file without affecting the existing content.

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?

```
try:  
    with open('data.txt', 'r') as file:  
        data = file.read()  
except FileNotFoundError:  
    print("The file does not exist.")
```