

**COLLEGE OF COMPUTER STUDIES**

**IT0011**

**Integrative Programming and Technologies**

**EXERCISE**

**3**

**String and File Handling**

|  |  |
| --- | --- |
| **Student Name:** | Banzuela, Norwood Shane A. |
| **Section:** | TW23 |
| **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

Source Code:

firstName = **input**('Enter your first name: ')

lastName = **input**('Enter your last name: ')

age = **input**('Enter your age: ')

slicedName = firstName[0:3]

formattedText = 'Hello, {}! Welcome. You are {} years old.'

**print**('Full Name: ' + firstName + ' ' + lastName)

**print**('Sliced Name: ' + slicedName)

**print**(formattedText.**format**(slicedName, age))

Output:

**A screenshot of a computer

Description automatically generated**

**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

Source Code:

firstName = **input**('Enter your first name: ')

lastName = **input**('Enter your last name: ')

fullName = firstName + ' ' + lastName

**print**('Full Name: ' + firstName + ' ' + lastName)

**print**('Full Name (Upper Case): ' + firstName.**upper**() + ' ' + lastName.**upper**())

**print**('Full Name (Lower Case): ' + firstName.**lower**() + ' ' + lastName.**lower**())

**print**('Length of Full Name: ' + **str**(**len**(fullName)))

**S**

Output:

**A screenshot of a computer

Description automatically generated**

**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.

Source Code:

firstName = input('Enter first name: ')

lastName = input('Enter last name: ')

age = input('Enter age: ')

conNum = input('Enter contact number: ')

course = input('Enter course: ')

formattedText = ('Last Name: {} \nFirst Name: {} \nAge: {} \nContact Number: {} \nCourse: {} \n')

f = open('TFA2/students.txt', 'a+')

f.write(formattedText.format(lastName, firstName, age, conNum, course))

line = f.readline()

f.close()

print('Information has been saved to \'students.txt\'.')

Output:

**A screenshot of a computer

Description automatically generated**

**A screenshot of a computer

Description automatically generated**

**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

Source Code:

**print**('Reading Student Information:')

f = **open**('TFA2/students.txt', 'r')

for content **in** f:

**print**(content)

f.**close**()

Output:

A screen shot of a computer

Description automatically generated

**QUESTION AND ANSWER:**

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

Using format() function helps make a string that can contain variables inside. Instead of using “+” symbol every time you put a variable inside a string or print output, you can use the format() function instead to bypass long string concatenation.

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

Opening a file in read mode only allows you to read a certain file without having the permission to append or put any text on that file. On the other hand, the write mode enables the user to write or put a text in a certain file, but not having the permission to read the file. Both can be used at the same time; I can write some text with write mode and then read the written text after with read mode.

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

By slicing a string in Python, it returns a specific part of the string or value. For example, I want to return the first three letters of a word. I will write a code that looks something like this to return or extract the first three letters.

word = ‘hello’

slicedWord = word[0:3]

this returns “hel”.

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.

In simplest explanation, write mode creates a new file and writes text on it. Meanwhile, append mode adds text to a file with or without pre-existing text.

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?

I did a little bit of research. I think it would be like this. If Python catches an error file not found. It outputs an error message to the user.

f = open(‘data.txt’, 'r')

while True:

try:

line = next(f)

print = (line)

except StopIteration:

break;

except FileNotFoundError:

print(‘An error occurred. File is not found.’)

else:

break;