



**MAPÚA UNIVERSITY**

**SCHOOL OF ELECTRICAL, ELECTRONICS, AND COMPUTER ENGINEERING**

## Experiment 2:

# Strings, Lists, Tuples, and Dictionaries

CPE106L (Software Design Laboratory)

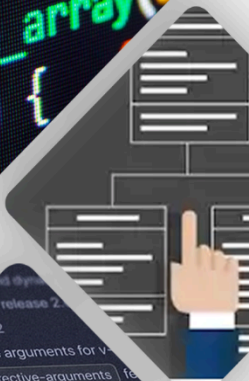
Member 1 (Cromuel Pangilinan)

Member 2 (Karl Ignatius G. Gavino)

Member 3 (Nicole Allyson B. Torres)

Member 3 (Xavier Alonzo)

Group No.: 9  
Section: B3





# PreLab

## Readings, Insights, and Reflection

- Book: Fundamentals of Python: First Programs (Chapter 4 & 5)  
Kenneth A. Lambert Edition 2  
ISBN: 9781337671019  
Cengage Learning US 2019
- Website: <https://docs.python.org/3/contents.html>  
<https://cplusplus.com/reference/list/list/?kw=list>

## Insights and Reflections

### ***Torres, Nicole Allyson B. ( Chapter 4 - strings and 5- lists and dictionaries)***

The book's fourth chapter comprehensively explains strings and their various properties. According to the chapter, a string is a sequence of characters, and the "len" function returns the number of characters in a string. Moreover, the chapter explains that the subscript operator "[" can access a character at a specific position in a string. The index inside the subscript operator must be an integer expression whose value is less than the length of the string. The chapter also discusses the concept of slicing, which enables us to extract a substring from a string using the subscript operator. When the subscript has the form [<start>:], the substring contains the characters from the start position to the end of the string.

Chapter 5 of the book explains lists and dictionaries, which are fundamental data structures in Python. The chapter defines a list as a sequence of elements of any type. Each list component occupies a specific position, ranging from 0 to the length of the list minus 1. Furthermore, the chapter explains that a list is a mutable data structure, meaning an element can be replaced with a new one, added to the list, or removed. The chapter also discusses the concept of dictionaries, which are key-value pairs. Finally, the chapter highlights pertinent files and websites that can be utilized to enhance our learning experience. We can better understand the Python programming language by studying and applying these concepts to programming problems.

### ***Gavino, Karl Ignatius G. ( Chapter 4 - strings and 5- lists and dictionaries)***

The basic ideas of control structures in Python, such as loops (while and for loops) and decision structures (if, else if, and else statements) are presented in Chapter 4 of the book. These structures let programmers efficiently iterate over data and regulate the program's flow based on circumstances. Learning these control structures equips students with the skills necessary to construct more dynamic programs. Control structures are one of the building blocks of any software application, and mastering them lays a solid foundation for further learning and exploration in Python. The author's approach to explaining these concepts is clear and structured, with plenty of examples and exercises to reinforce learning.

Using functions as a starting point, Chapter 5 delves into the Python notion. Functions are crucial for encouraging modularity and code efficiency by arranging code into reusable building parts.



The knowledge to define functions, pass parameters to them, and have them return values is gained in this chapter. Also, it discusses the idea of scope, explaining how variables can exist in different portions of a program and have varying lifetimes and visibility. These chapters emphasize the importance of writing clean, readable, and maintainable code. By following best practices in coding style and organization, one can cultivate good habits early on, setting themselves up for a good future in Python programming.

### **Answers to Questions**

1. B
2. B
3. A
4. B
5. B
6. C
7. B
8. B
9. B
10. B

# InLab

## Objectives

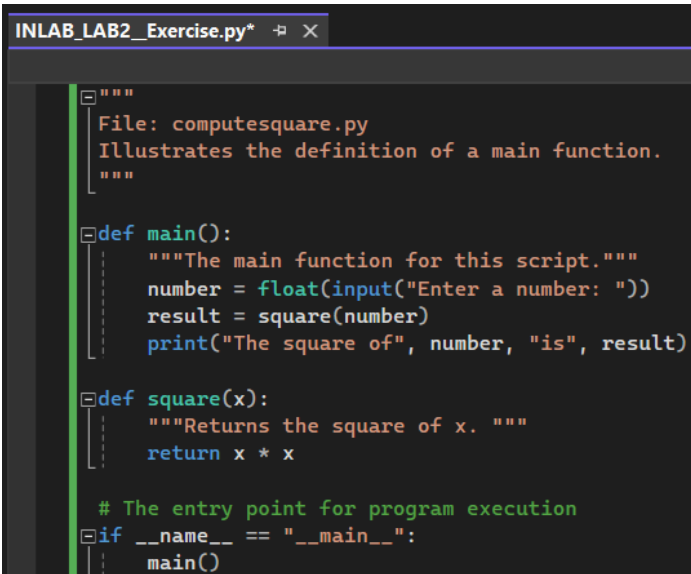
1. **Debug** the sample program computesquare.py.
2. **Using** Microsoft Visual Studio in running the program
3. **Verifying** the sample program computesqaure.py.
4. **Compare** (Discuss the difference) C++ and Python data structures.

## ● Tools Used

- Anaconda (or Python Virtual Environment)
- Ubuntu Linux Virtual Machine
- Microsoft Visual Studio Code 2022

## ● Procedure.

1. Creating sample program

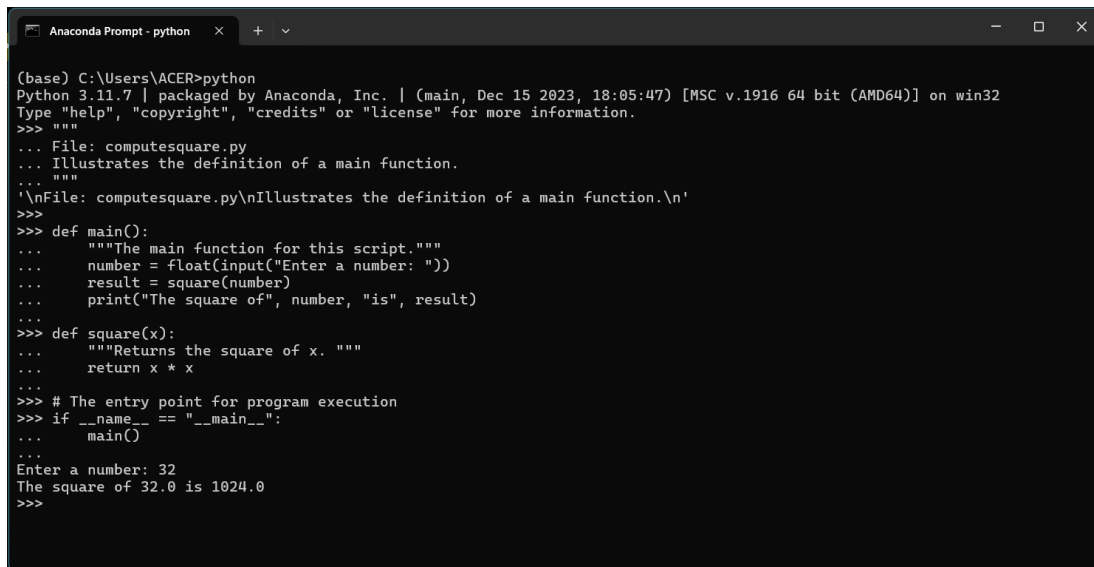


```
INLAB_LAB2_Exercise.py*  ↵  ×  
  
"""  
File: computesquare.py  
Illustrates the definition of a main function.  
"""  
  
def main():  
    """The main function for this script."""  
    number = float(input("Enter a number: "))  
    result = square(number)  
    print("The square of", number, "is", result)  
  
def square(x):  
    """Returns the square of x. """  
    return x * x  
  
# The entry point for program execution  
if __name__ == "__main__":  
    main()
```

Figure 1: Sample Source Code of Program: computesquare.py

Figure 1 displays the source file "computesquare.py," which was successfully executed without errors after debugging. The program's objective is to calculate the square of any number that the user enters. In this case, the user entered the number 32, resulting in the program computing the square of 2, equivalent to 1024.

## 2. Running the program



```
(base) C:\Users\ACER>python
Python 3.11.7 | packaged by Anaconda, Inc. | (main, Dec 15 2023, 18:05:47) [MSC v.1916 64 bit (AMD64)] on win32
Type "help", "copyright", "credits" or "license" for more information.
>>> """
... File: computesquare.py
... Illustrates the definition of a main function.
... """
'\nFile: computesquare.py\nIllustrates the definition of a main function.\n'
>>>
>>> def main():
...     """The main function for this script."""
...     number = float(input("Enter a number: "))
...     result = square(number)
...     print("The square of", number, "is", result)
...
>>> def square(x):
...     """Returns the square of x. """
...     return x * x
...
>>> # The entry point for program execution
>>> if __name__ == "__main__":
...     main()
...
Enter a number: 32
The square of 32.0 is 1024.0
>>>
```

Figure 2: Sample Running the Program: *computesquare.py* in Anaconda Prompt

After running the Python file in the Anaconda prompt, no errors were found, and it was successfully debugged, as shown in Figure 2.

## 3. Verifying the output

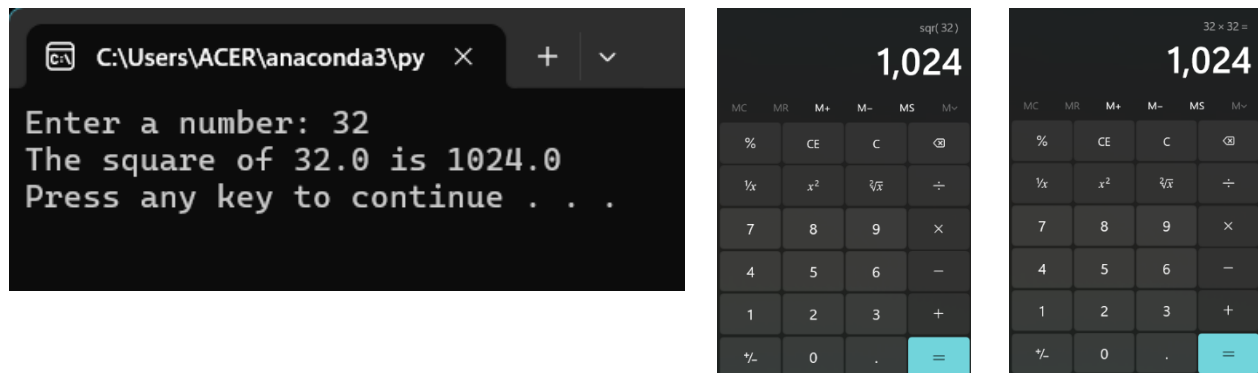


Figure 3: Verifying the Output of the Program: *computesquare.py*

Figure 3 proves that the output of the code is correct.



#### 4. *Compare (Discuss the difference) C++ and Python data structures.*

C++ and Python have different approaches to creating data structures. C++ uses classes and structures to build data structures, while Python prioritizes simplicity and readability. C++ has numerous built-in data structures, allowing developers to create custom structures using classes and templates. C++ provides various in-built data structures such as arrays, vectors, stacks, queues, and maps. It also offers better control over memory management and enables low-level manipulation of data structures. It provides more control over memory management and allows for low-level manipulation of data structures.

Conversely, Python is a programming language with built-in dynamic data structures that are easy to use and understand. It is designed to simplify coding with dynamic typing and automatic memory management. Its rich data structures, like lists, tuples, dictionaries, and sets, are highly abstracted, making them easy to use and comprehend. This makes Python popular among developers who want to write code faster and with fewer errors.



# PostLab

---

## Programming Problems

### 1. stats.py

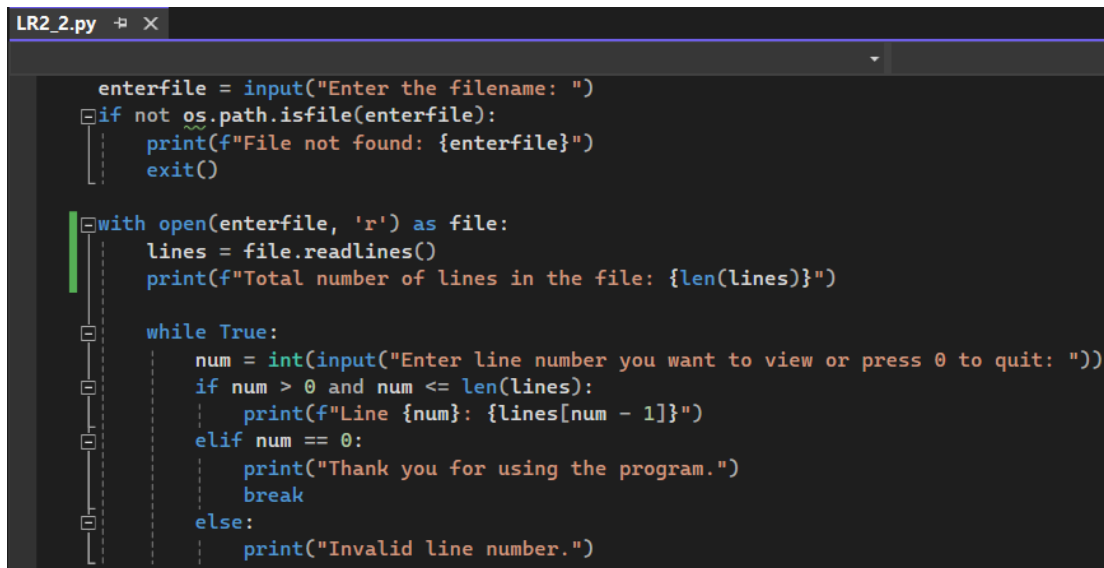
A group of statisticians at a local college has asked you to create a set of functions that compute the median and mode of a set of numbers, as defined in the below sample programs: ·

- mode.py
- median.py

Define these functions in a module named stats.py. Also include a function named mean, which computes the average of a set of numbers. Each function should expect a list of numbers as an argument and return a single number. Each function should return 0 if the list is empty. Include a main function that tests the three statistical functions with a given list.

## 2. LR2\_2.py

Write a program that allows the user to navigate the lines of text in a file. The program should prompt the user for a filename and input the lines of text into a list. The program then enters a loop that prints the number of lines in the file and prompts the user for a line number. Actual line numbers range from 1 to the number of lines in the file. If the input is 0, the program quits. Otherwise, the program prints the line associated with that number.



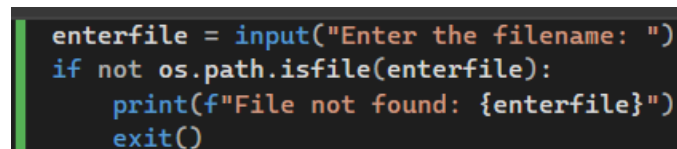
```
LR2_2.py  [icon] [x]

enterfile = input("Enter the filename: ")
if not os.path.isfile(enterfile):
    print(f"File not found: {enterfile}")
    exit()

with open(enterfile, 'r') as file:
    lines = file.readlines()
    print(f"Total number of lines in the file: {len(lines)}")

    while True:
        num = int(input("Enter line number you want to view or press 0 to quit: "))
        if num > 0 and num <= len(lines):
            print(f"Line {num}: {lines[num - 1]}")
        elif num == 0:
            print("Thank you for using the program.")
            break
        else:
            print("Invalid line number.")
```

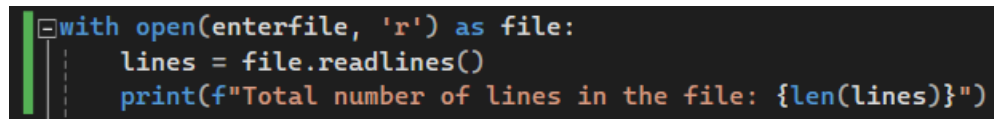
Figure 1. Complete breakdown of code



```
enterfile = input("Enter the filename: ")
if not os.path.isfile(enterfile):
    print(f"File not found: {enterfile}")
    exit()
```

Figure 2. File Input

The line `enterfile = input("Enter the filename: ")` prompts the user to enter a filename, and `os.path.isfile()` function checks if the file exists at the specified path. If the file doesn't exist, the program prints an error message with the filename and terminates using the `exit()` code. This code block is crucial in ensuring that the program can read from the file entered by the user. By checking if the file exists before continuing, we can ensure that the program will run smoothly.



```
with open(enterfile, 'r') as file:
    lines = file.readlines()
    print(f"Total number of lines in the file: {len(lines)}")
```

Figure 3. Line count

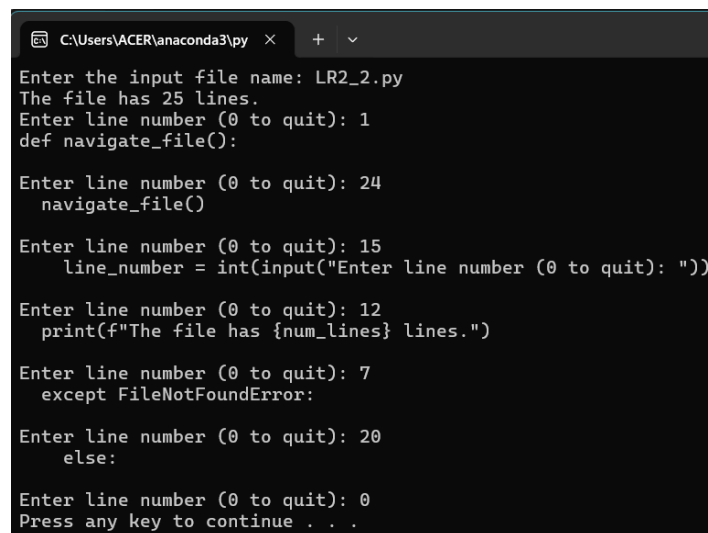


This code reads a file specified by the 'enterfile' variable and calculates the number of lines in the file. It opens the file in read mode, reads all the lines using the 'readlines()' method, calculates the list length using the 'len()' function, and displays the total number of lines. The time and space complexity of the code are both  $O(n)$ , where  $n$  is the number of lines in the file.

*Figure 4. Print User chosen Line*

```
while True:
    num = int(input("Enter line number you want to view or press 0 to quit: "))
    if num > 0 and num <= len(lines):
        print(f"Line {num}: {lines[num - 1]}")
    elif num == 0:
        print("Thank you for using the program.")
        break
    else:
        print("Invalid line number.")
```

This code allows the user to view specific lines of a file by entering a line number. The program will prompt the user until 0 is entered to exit the loop. The line's contents are displayed with its number if the input is valid. If the input is invalid, "Invalid line number" is displayed. The input must be a number, or else an error will occur.



```
C:\Users\ACER\anaconda3\py x + v
Enter the input file name: LR2_2.py
The file has 25 lines.
Enter line number (0 to quit): 1
def navigate_file():

Enter line number (0 to quit): 24
    navigate_file()

Enter line number (0 to quit): 15
    line_number = int(input("Enter line number (0 to quit): "))

Enter line number (0 to quit): 12
    print(f"The file has {num_lines} lines.")

Enter line number (0 to quit): 7
    except FileNotFoundError:

Enter line number (0 to quit): 20
    else:

Enter line number (0 to quit): 0
Press any key to continue . . .
```

*Figure 5. Output of the program*

After executing the Python file in the Anaconda prompt, Figure 5 indicates a successful debugging process without errors.



### 3. Generator\_modified.py

Modify the sentence-generator program of Case Study 5.3:

- METIS book: 9781337671019, page 150.0020
- Python source code: generator.py

so that it inputs its vocabulary from a set of text files at startup. The filenames are nouns.txt, verbs.txt, articles.txt, and prepositions.txt. (Hint: Define a single new function, getWords. This function should expect a filename as an argument. The function should open an input file with this name, define a temporary list, read words from the file, and add them to the list. The function should then convert the list to a tuple and return this tuple. Call the function with an actual filename to initialize each of the four variables for the vocabulary.)