## 

## 

## 

**Experiment 2:**

**Strings, Lists, Tuples, and Dictionaries**

CPE106L (Software Design Laboratory)

**Manzanero, Jan Adrian C.**

Group No.: **6**

Section: **B1**

## **PreLab**



|  |
| --- |
| **Readings, Insights, and Reflection** |

Learning Python for the first time seemed to be a bit scary for the students, who had only previously experienced coding in C++. However, with Python having a richer set of built-in data structures like lists, tuples, dictionaries, and strings, it was more efficient to use for students in programming problems. Understanding these data structures is essential to the student’s ability to quickly organize and manipulate information depending on the desired application.

The students have learned several data structures in Python: strings, lists, tuples, and dictionaries. Strings are sequences of characters that represent textual data. They are immutable character arrays, which makes them ideal for storing information such as names, descriptions, or addresses. Lists are basically collections of items, which can be of different data types. They are mutable, allowing the students to dynamically modify them for applications in storing student grades, shopping lists, and the like. Tuples are like lists, but they are immutable, which makes them good for storing data that should not be modified. Lastly, dictionaries are storages for data using key-value pairs: unique identifiers associated with values. This structure is particularly useful when fast access to information based on a specific key is required, making dictionaries ideal for scenarios like phonebook applications or student information databases.

It is surprising how these data structures can be molded to fit many applications in data handling with the right thinking, which just shows how versatile Python is when it comes to data management. Selecting the appropriate data structure depends on the specific needs of a program. The students are inspired to master these fundamental data structures to build well-organized and efficient Python programs.

**Answers to Questions**

1. 20
2. [20, 30]
3. 1
4. [10, 20, 30, 40, 50]
5. [10, 5, 30]
6. [10, 15, 20, 30]
7. ["name", "age"]
8. None
9. pop
10. strings and tuples

## **InLab**



**Objectives**

1. a
2. a

**Tools Used**

* GitHub
* Oracle Virtual Machine VirtualBox Manager
* Visual Studio Code

**Procedure**

**Part 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](https://mymailmapuaedu-my.sharepoint.com/:u:/g/personal/dapadilla_mapua_edu_ph/EWQI7kSWI-xFiYfm2KyofXoBBV_Zjex0RtarScWwn57pag?e=CD4HuL), [median.py](https://mymailmapuaedu-my.sharepoint.com/:u:/g/personal/dapadilla_mapua_edu_ph/EQ7kUrEOI0RIgplQhfEYnTABvArfAp403AaoreENtzVfgw?e=yo1hzy). 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.

A screenshot of a computer program

Description automatically generated

*Figure 1. Sample output for* stats.py *with input* numList = [1, 2, 3, 3, 4, 5]*.*

**Part 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.

A screenshot of a computer

Description automatically generated

*Figure 2. Sample output for* LR2\_2.py *with test file* myfile.txt*.*

**Part 3: generator\_modified.py**

Modify the sentence-generator program of *Case Study 5.3*: *METIS book: 9781337671019, page 150*, Python source code: [generator.py](https://mymailmapuaedu-my.sharepoint.com/:u:/g/personal/dapadilla_mapua_edu_ph/EWb2bLxiCAROunKE_BC2ux8BFjuKrS3b-6YYBLrs5zDObw?e=YCZWOK), so 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.)

A screenshot of a computer

Description automatically generated

*Figure 3. Sample output for* generator\_modified.py *with 5 sentences.*

## **PostLab**



[GitHub Repository Branch](https://github.com/mepue-mels/CPE106L_B1_Labs/tree/manzanero/Lab2)

**Programming Problem 1:**

def mean(numList):

return sum(numList)/len(numList)

def median(numList):

length = len(numList)

i = length // 2

# for odd list count = one median

if length % 2 == 1:

return(numList[i])

# for even list count = average of medians

return(sum(numList[i - 1: i + 1])/2)

def mode(numList):

numCount = {}

for i in numList:

if not i in numCount:

numCount[i] = 1

else:

numCount[i] += 1

modeList = [x for x,y in numCount.items() if y == max(numCount.values())]

if len(modeList) != 1:

return "No mode."

return modelist[0]

**Programming Problem 2:**

def main():

userFile = str(input("Enter file name: "))

userList = open(userFile, 'r').readlines()

while 1:

x = int(input("Select line to print, or type '0' to exit: "))

if x == 0:

break

elif x - 1 < len(userList):

print(userList[x - 1].strip())

else:

print("Out of range. Please try again.")

**Programming Problem 3:**

import random

def getWords(filename):

try:

f = open(filename, 'r')

wordList = []

while True:

line = f.readline()

if line == "":

break

line = line.strip().split()

word = line[0]

wordList.append(word)

return tuple(wordList)

except:

print("Invalid filename.")

raise SystemExit

def sentence(articles, nouns, verbs, prepositions):

return nounPhrase(articles, nouns) + " " + verbPhrase(articles, nouns, verbs, prepositions)

def nounPhrase(articles, nouns):

return random.choice(articles) + " " + random.choice(nouns)

def verbPhrase(articles, nouns, verbs, prepositions):

return random.choice(verbs) + " " + nounPhrase(articles, nouns) + " " + \ prepositionalPhrase(articles, nouns, prepositions)

def prepositionalPhrase(articles, nouns, prepositions):

return random.choice(prepositions) + " " + nounPhrase(articles, nouns)

def main():

articles = getWords("articles.txt")

nouns = getWords("nouns.txt")

verbs = getWords("verbs.txt")

prepositions = getWords("prepositions.txt")

number = int(input("Enter the number of sentences: "))

for count in range(number):

print(sentence(articles, nouns, verbs, prepositions))

if \_\_name\_\_ == "\_\_main\_\_":

main()