LBYCPA1

**Programming Logic and Design Laboratory**



*(Note: you can change the image to your own liking)*

**Laboratory Module 6**

Strings and Formatting

By

John Carlo Theo S. Dela Cruz || LBYCPA1 - EQ1

# INTRODUCTION

The Sixth Module of this course will mainly focus on String Manipulation and Formatting. According in the notes of Module 6 that Strings is a text or one of the most common forms of data that a program will handle. In this module, we are required to fully understand how to manipulate these data in the form of strings, such as an indexing, concatenation and many more that are presented in the module. We utilize the strip and split or what we call the slicing in a list or in a string. This module will introduce us on how to encrypt messages, create an emulator, and many more. The main objective of every laboratory report is to provide opportunity for learning and provide a strong foundation in programming, molded by experience. Despite the hindrance of the Pandemic, we can maximize the utility of the Internet to provide us more information about the topic. Through the process of programming alongside with the method of planning; algorithms, pseudocodes, and flowcharts that provides systematic process on how the program run. Python is a very powerful and flexible language, and this is a big opportunity for aspiring software developers to learn more about it.

**What do you think are the main objectives for this module? (Enumerate as many as you can.)**

1. Objectives
2. To familiarize with string representations in Python including raw strings and f-strings
3. To understand basic operations on strings, such as concatenation, repetition, indexing, and slicing
4. To use different string data type methods to achieve the desired string format
5. To develop programs that can parse string data

**What are the materials used for this module?**

1. Materials and Tools
2. Instructor's lecture notes
3. Jupyter Notebook
4. Flowchart Software (Diagrams.net)
5. Snipping Tool
6. Pycharm.edu
7. Google Browser

# PROCEDURES (*Individual*) / EXPERIMENTAL PLAN

In every experimental plan of every Laboratory Report we always place our initial plans before performing executing our codes, in programming we always use Pseudocodes / Algorithms or Flowcharts, as a representation of how we can show the separate steps of each process in a sequential manner. The overview of each exercise will be provided as well.

1. Familiarization Exercise 1 – Encrypting Message

In this program, we utilize the function of encryption. Based in the instructions that the encryption method will be like a Caesar cipher substitution is to be implemented, a text will be reversed to an inverse value of that string. Example: 9 to 0, and A to Z.

Diagram

Description automatically generated

1. Familiarization Exercise 2 – XNOR

In this program, we utilize the bitwise logical operation XNOR (exclusive-NOR), which takes two strings and compares each character in their respective position. Using the Truth Table XNOR will be the main basis of this program.

Diagram

Description automatically generated

1. Familiarization Exercise 3 – Border

In this program, utilizing the for loop to create and print a specific character and side length that creates a border or a rectangle shaped model.

Diagram

Description automatically generated

1. Familiarization Exercise 4 – Email Address

In this program, we emulate the DLSU email address generating system that the input is a list of names in a string that is separated by commas. The output must be a string of comma-separated (,) and place the email addresses.

Diagram

Description automatically generated

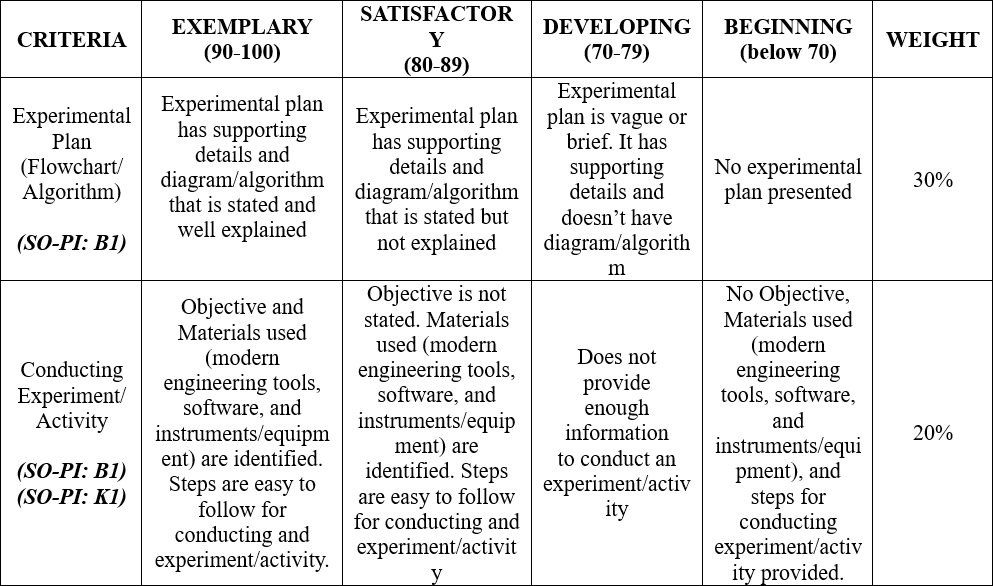
1. Familiarization Exercise 5 – Compute Average

In this program, there are 2 main functions – 1st function compute\_average, in which this functions get the list of integers that is separated by comma and combine them to compute for the average, and the 2nd main function combine\_averages, in which this function that splits the list and separate the name and the integer data inputted as well as inputting the 1st function to automatically compute the average.

Diagram

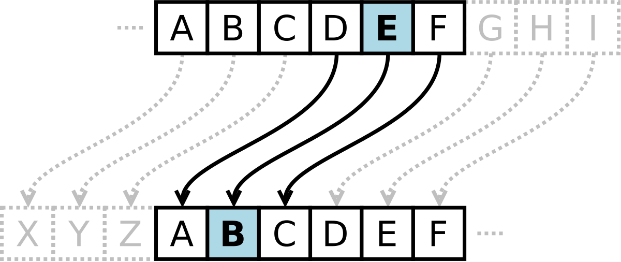
Description automatically generated

**The Introduction together with individual Procedures and plan comprises the Experimental Plan and Conducting Experiment/ Activity criteria in the Final Laboratory Report Rubric**:



# RESULTS AND DISCUSSION/COMPUTATIONS (*Include the program output screenshots, and discussions per problem solution*)

**1. Familiarization Exercise 1 - Encryption:**



Text

Description automatically generated

Text, letter

Description automatically generated

**Explanation:**

This program’s main function is to create an encryption method that is similar to a Caesar cipher substitution is to be implemented, that will inverse a positioning of a strings of alpha and decimals. By basing on ASCII, we subtract the ASCII value of string alpha z and get the inverse of the value; after getting the inverse value is that the position will move from A onwards. It is also the formula or method on how to get the inverse value of the decimal.

**2. Familiarization Exercise 2 - XNOR:**

Text

Description automatically generated

A screenshot of a computer

Description automatically generated with medium confidence

**Explanation:**

This program requires us to review and revisit again the Truth Table of AND, OR, and NOT function to understand furtherly the concept of XNOR, in which it is an exclusive NOR logical function. XNOR is a combination XOR followed by an inverter, in which if an output is “True” and if the input is the same, and “False” if the inputs aren’t similar.

**3. Familiarization Exercise 3 - Border:**

Text

Description automatically generated

Graphical user interface, text

Description automatically generated

**Explanation:**

This program is similar with the previous modules in which we print a specific number or size of an object. In this case, we are required to printout a border-like graphic. By converting the border into string and by utilizing the for-loop statements. The 1st for loop will be for the characters, 2nd loop will be for the input of \n in every end of the loop, 3rd loop will be the basis of the width and adding the char of space instead of the character to create a space between the borders. It will loop per line according to the user’s input of how many sides will they input.

**4. Familiarization Exercise 4 – Email Address:**

Text

Description automatically generated

Text

Description automatically generated

**Explanation:**

This program will emulate the DLSU email address generating system according to the list of names in a string that is separated by commas. First thing to do is lower all the cases of the string, because in formatting the email, upper case is not allowed. Next is by creating a format in which we will place the first and last name inside the curly brackets (placeholder) followed by the format of an email address. Splitting the name list, since it is in list form; we are splitting the list with the character of the comma. By splitting the list and turning the list into an array I created a variable to strip the spaces in the array and split the string in the array to separate the first and last name of the individual persons included in the list. Lastly, we use formatting and getting the 1st (First Name) and 2nd (Last Name) index of each array to generate the email, and joining the result to the function with the character ‘,’

**5. Familiarization Exercise 5 – Compute Average:**

Text, letter

Description automatically generated

Text, letter

Description automatically generated

**Explanation:**

This program is separated with 2 functions:

1st function is to accept a string with three comma-separated grades and returns the average as a float. We also utilize the strip and split function to remove and to convert a specific string to an array. Then in every string we transform them into string to be able to get the value of the average. Lastly, we round off and add the 3 index and divide them by 3 to get the average value.

2nd function is to combine all student average scores into a string and calling the 1st function to be able to calculate the average. Usual method the split and strip to create an array and stripping the comma and \n. After those methods I joined the index of 2-4 which is the numbers and calling the function of compute\_average to create solve every average in the list. Get the first index and format them again to rearrange [Name:Average\n].

Text

Description automatically generated

# CONCLUSION:

This 6th Module of LBYCPA1 mainly focuses on String Manipulation and Formatting using the language of Python. We revisited the lessons and knowledge of the past modules and utilized them to fully master each function. We were introduced to Ciphering and Encryption with the method of Caesar Cipher. Re-introduced again with the bitwise logical operations and truth table; XNOR an exclusive-NOR. Printing border made up of characters with a specified side length. The last 2 exercises are all about list and indexes and how to manipulate these data to create an emulating DLSU email address and computing averages.

Slicing, Indexing, and other String methods are very useful for manipulating any data in an array or list. Formatting is used instead of string concatenations for better code legibility. Using the functions was also a one of the methods we used (previous module) to create and to have a organize and to have a maximize the reuse and redundancy of codes. In the last 2 exercises I used a method called List comprehension, in python a function or a list comprehension offers a shorter syntax when you want to create a new list based on the values of an existing list. To create a for loop or an if-else statement in one line of code. The downfalls of my work in this Lab Activity were me having an incorrect syntax or errors and my recommendation in this module is to discover new things and view other syntaxes that python provides for programmers for us to utilize and we will take advantage of those assets that are offered in this language. As programmers we should always review our notebooks / module / and reliable sources.

**The rest of the rubric criteria are as follows:**

**Text

Description automatically generated with medium confidence**

# REFERENCES (*Enumerate references in APA format*)

W3school (n.d.). Retrieved from: <https://www.w3schools.com/python/python_lists_comprehension.asp>

1. Familization Exercise 1:

def encryptMessage(message):  
 # This function encrypts a message such that each character is substituted with its equivalent reversed alphabet ordering.  
  
 if type(message) is not str:  
 raise TypeError("Input a string only")  
   
 encrypt = str()  
 for i in message:  
 if i.isalpha():  
 num = ord(i)  
 if i.isupper():  
 num = (ord('Z')-num) + ord('A')  
 encrypt += chr(num)  
 elif i.islower:  
 num = (ord('z')-num) + ord('a')  
 encrypt += chr(num)  
 elif i.isdecimal():  
 num = ord(i)  
 num = (ord('9')-num) + ord ('0')  
 encrypt += chr(num)  
 else:  
 num = ord(i)  
 encrypt += chr(num)  
 return encrypt

2. Familization Exercise 2:

def bitwise\_XNOR(string1, string2):  
 # This function performs the logical bitwise XNOR on two n-character strings  
   
 checkInput(string1)  
 checkInput(string2)  
  
 if len(string1) != len(string2):  
 raise ValueError("String inputs must have same lengths")  
 result = str()  
 for char1, char2 in zip(string1, string2):  
 if char1 == '0' and char2 == '0':   
 result += '1'  
 elif char1 == '1' and char2 =='1':  
 result +='1'  
 else:  
 result +='0'  
   
 return result  
def checkInput(string):  
 if type(string) is not str:  
 raise TypeError("Input not a valid string")  
 for char in string:  
 if not (char == "0" or char == "1"):  
 raise ValueError("Strings must contain a '0' or '1' characters only")

3. Familization Exercise 3:

def printBorder(width, char='\*'):  
 # Prints a border of specified width using a single character  
   
 if type(char) is not str:  
 raise TypeError("Input character must be a string")  
 if type(width) is not int:  
 raise TypeError("Input width must be an integer")  
 if len(char) < 1 or len(char) > 1:  
 raise ValueError("Specify single character input")  
 if width < 0:  
 raise ValueError("Specify a non-negative integer input")  
   
 border = str()  
 for char1 in char:  
 for i in range(width):  
 for j in range(width):  
 if i == 0 or i == width - 1:  
 border += char1  
 else:  
 if j == 0 or j == width - 1:  
 border += char1  
 else:   
 border +=' '  
 border +='\n'  
   
 return border

4. Familization Exercise 4:

def namesToEmails(names):  
 # Emulates the DLSU email address generation system and returns a list of email addresses as strings  
   
 names = names.lower   
 email = "{}\_{}@dlsu.edu.ph"   
 nameslist = names.split(',')   
 namesspace = [x.strip() for x in nameslist] nameslist2 = [x.split(' ') for x in namesspace]   
 namestoEmails = [email.format (x[0], x[1]) for x in nameslist2]   
 #for x in nameslist2:   
 # namestoEmails.append(email.format(x[0], x[1]))  
   
 result = ', '.join(namestoEmails) #join list   
   
 return result

5. Familization Exercise 5:

# DO NOT MODIFY, THIS IS UNIT TESTING! [1 pt]  
student\_scores = \  
"""Anna,60,68,89  
Darcey,80,78,90  
Dylan,90,78,95  
Eva,78,76,67  
Layla,89,88,95  
Lewis,78,88,89  
Lucas,50,65,45  
Theodore,88,86,87  
Victoria,55,65,45  
Zoe,94,67,96  
"""  
# "60, 68, 69"  
# average  
def compute\_average(grade\_string):  
 # This function accepts a string with three comma-separated grades and returns the average as a float  
   
 if type(grade\_string) is not str:  
 raise TypeError("Input must be a string")  
 grade\_string = grade\_string.strip(',') # tanggal yung specific chuchu  
 grade\_string = grade\_string.split(',') # string to array  
 integers = [int(x) for x in grade\_string]  
 average = round(((integers[0] + integers [1] + integers[2]) / 3),1)  
 return average   
  
# student\_scores = string  
# name, compute\_average  
def combine\_averages(student\_table):  
 # This function combines all student average scores into a string.   
 # It should call the compute\_average() function to calculate the average  
   
 if type(student\_table) is not str:  
 raise TypeError("Input must be a string")  
   
 result = "{}: {}"  
 student\_table = student\_table.strip('\n')   
 student\_table = student\_table.split('\n')   
 student\_table\_space = [x.strip(',') for x in student\_table]  
 combine = [x.split(',') for x in student\_table\_space]  
 # avgList = [x.split(',') for x in combine]  
 List1 = [compute\_average(','.join([x[1],x[2],x[3]]))for x in combine]  
 List2 = [x[0] for x in combine]  
 averageTable = [result.format (List2[x],List1[x]) for x in range (0,len(combine))]   
   
   
 end ='\n'.join(averageTable)  
   
 return end + '\n'