



Adamson University
College of Engineering
Computer Engineering Department



Linear Algebra

Laboratory Activity No. 1

Laboratory 1: Python Programming

Submitted by:

Casiño, Christian Carlos H.

Instructor:

Engr. Dylan Josh D. Lopez

February 5, 2021

I. Objectives

This laboratory activity aims to implement the principles and techniques of variables and data types, arithmetic, Inputs, and outputs, looping statements, flow control, and functions. It also aims to create a grade calculator that computes for the semestral grade of a course. Where students could type their names, the name of the course, then their prelim, midterm, and final grade. The program should print the semestral grade in 2 decimal points and should display the following emojis depending on the situation.

II. Methods

First, the programmer reviewed the basic principles and techniques which will give ideas for the flow of the program. Secondly, the programmer should manipulate the inputs, arithmetic, data types, conditional and looping statements, flow control, and functions for filling up the our program. Lastly, The programmer would create, demonstrate, and explain how this principles and techniques work . The deliverance of this activity is a program that interacts with the user, user friendly, has a unique design, and is very efficient to use.

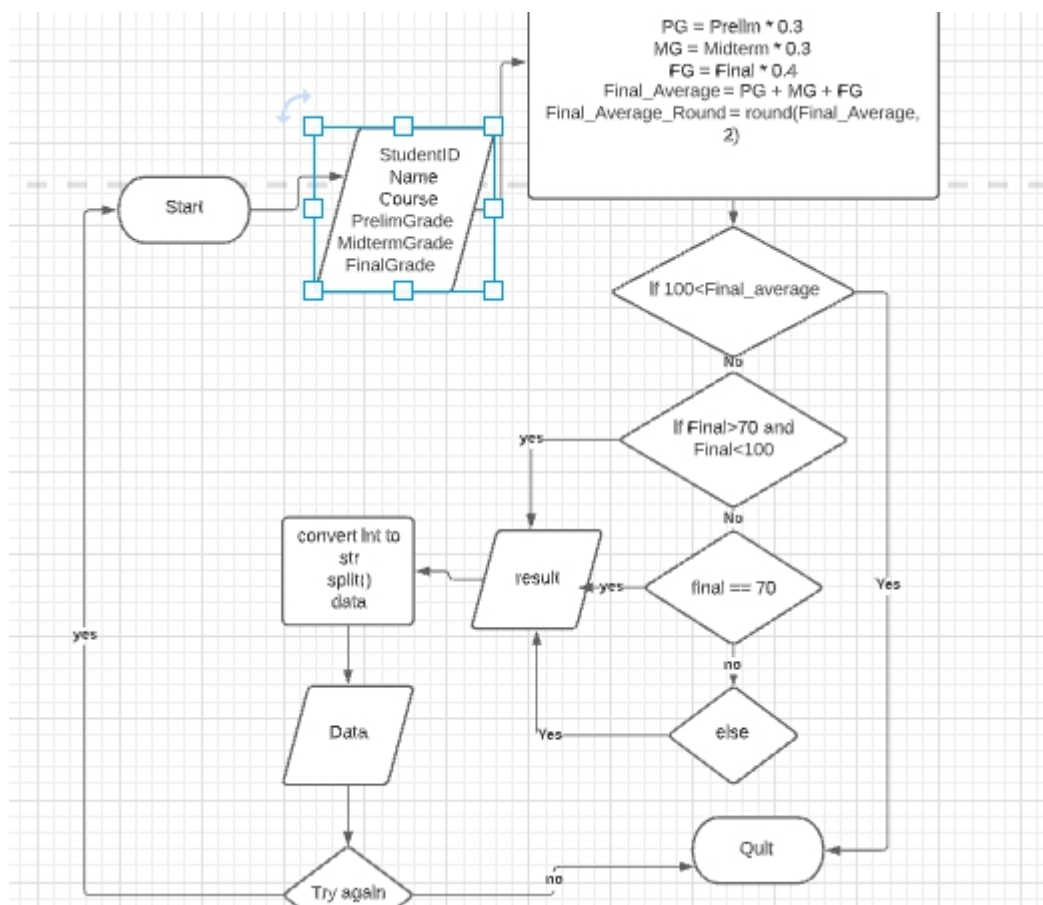


Figure 1 Flow Chart(Laboratory 1) by Casioño

III. Results

```
[1] xtian = 100
```

```
[2] type(xtian)
```

```
int
```

```
[3] ychan = 1.0  
type(ychan)
```

```
float
```

```
[4] xtian = float(x)  
type(xtian)
```

```
float
```

```
[8] x,t,i,a,n = "0", '1', 'one','3','2'  
type(x)
```

```
str
```

```
▶ chris_int = int(t)  
chris_int
```

```
1
```

Figure 1.1 Variable and Data Types by Casiño

In this Figure, every variable has its data type. The idea is that a variable is a value that can change, it can be represented as numerical values, characters, strings, or memory addresses. Variables have a huge role in the lifespan of a computer program because their purpose is to store everything like stored values, inputted values, and modified values. Each data type is unique depending on its variable. Different types of data have different uses, variables can be a string, integer, float, etc. Using it to manipulate the variable and data types for the activity will help in getting the ideal result. Additional information: there are two types of variables which are the global and local variables, the global variable can be accessed in any part of the program while the local variable only has scope within a specific program. For the activity, the programmer used 'data' as a global variable and used "name, course, prelim, midterm, final grades, final average, and result" as local variables.

```
[11] x,t,i,a,n = 2.0, 15, 0, -3, 22
```

```
[12] ### Addition  
answer = x + t  
answer
```

17.0

```
[13] ### Subtraction  
answer = i-a  
answer
```

3

```
[14] ### Multiplication  
answer = a * n  
answer
```

-66

```
[15] ### Division  
answer = x/t  
answer
```

0.13333333333333333

Figure 1.2 Arithmetic by Casio

```
[17] ### Floor Division  
answer = n//a  
answer
```

-8

```
[19] ### Exponentiation  
answer = x**x  
answer
```

4.0

```
### Modulo  
answer = t%x  
answer
```

1.0

Figure 1.3 Arithmetic by Casio

In Figures 1.2 and 1.3 , Arithmetic operators are used to perform mathematical operations like addition, subtraction, multiplication and division. There are seven arithmetic operators in Python which are Addition, Subtraction, Multiplication, Division, Modulus, Exponentiation, and Floor division. In the figure 1.2, the programmer have declared different numerical values for the execution of each arithmetic operators. The first four arithmetic operations are the most basic operations while floor division, modulus, and power will be focused in the programmers explanation. Floor division is used to find the floor of the quotient when first operand is divided by the second. Modulus is used to return the remainder when first operand is divided by the second. Power is used to multiply itself depending on the value of the exponent.

```
[22] c,h,r,s = 1,2,3,4

[23] c += h
c

3

[24] h -= r
h

-10

[25] s *= 2
s

8

[26] r **= 2
r

9
```

Figure 1.4 Assignment Operations by Casiño

In this Figure, Assignment operators are used for assigning numerical values to variables. The values in the figure assigned were incremented or decremented depending on

the assignment operator used. The Assignment operators used above are explained in the table below.

Operator	Description
=	Assign value of right side of expression to left side operand
+=	Add and Assign: Add right side operand with left side operand and then assign to left operand.
-=	Subtract AND: Subtract right operand from left operand and then assign to left operand: True if both operands are equal
*=	Multiply AND: Multiply right operand with left operand and then assign to left operand
**=	Exponent AND: Calculate exponent(raise power) value using operands and assign value to left operand

Table 1 Assignment Operators according to <https://www.geeksforgeeks.org/assignment-operators-in-python/>

▼ Comparators

```
[34] chris_1, chris_2, chris_3 = 4,3,5  
     true_value = 4
```

```
[35] ## Equality  
     chris_1 == true_value
```

True

```
[36] ## Non-equality  
     chris_2 != true_value
```

True

```
▶ ## Inequality  
inequality1 = chris_1 > chris_2  
inequality2 = chris_1 < chris_2/2  
inequality3 = chris_1 >= chris_2/2  
inequality4 = chris_1 <= chris_2  
inequality1
```

True

Figure 1.5 Comparators by Casiño

In this Figure, It shows that using comparators help compare and contrast variables on each sides . The examples shown in the figure are equality, non-equality and inequality. In the table below is the explanation of each comparators used by the programmer.

Operator	Meaning
>	Greater than
<	Less than
==	Equal to
!=	Not equal to
>=	Greater than or equal to
<=	Less than or equal to

Table 2 Comparators according to https://www.tutorialspoint.com/python/comparison_operators_example.htm

▼ I/O

```
[60] count = 1
      string = "Hello Christian"
      print(string, ", give me :", count, '$')
      count += 1
```

Hello Christian , give me : 1 \$

```
▶ print(f"{string}, give me : {count} $")
```

Hello Christian, give me : 2 \$

```
[56] sem_grade = 90
      name = "Christian"
      print("Hello {}, your grade is: {}".format(name, sem_grade))
```

Hello Christian, your grade is: 90

```
[59] prelim, midterm, final = 0.3, 0.3, 0.4
      print("The computation of your semestral grades are:\n\t{:.2%} for Prelims\n\t{:.2%} for Midterms, and\n\t{:.2%} for Finals.".format(prelim, midterm, final))
```

The computation of your semestral grades are:
30.00% for Prelims
30.00% for Midterms, and
40.00% for Finals.

Figure 1.6 Input and Output by Casañó

In this Figure, The programmer used the print function to output data along with the declared variable “string” and “count” to fill in the blanks that were missing in the code that the programmer have coded. The “,” in the print function acts as a separator when outputted acts as a space or blank. The “f{}” in the print function acts as a format where “{}” acts as place holders and also to specify the order being formatted.

▼ Logical

```
[38] chris_1 == true_value
```

True

```
[39] chris_1 is true_value
```

True

```
[40] chris_1 is not true_value
```

False

```
[41] T, F = True, False  
conjunction = T and F  
conjunction
```

False

```
[42]  
disjunction = T or F  
disjunction
```

True

Figure 1.7 Logical by Casio

```
[43]  
Nand = not(T and F)  
Nand
```



True



```
exclusiveor = (not T and F) or (T and not F)  
exclusiveor
```

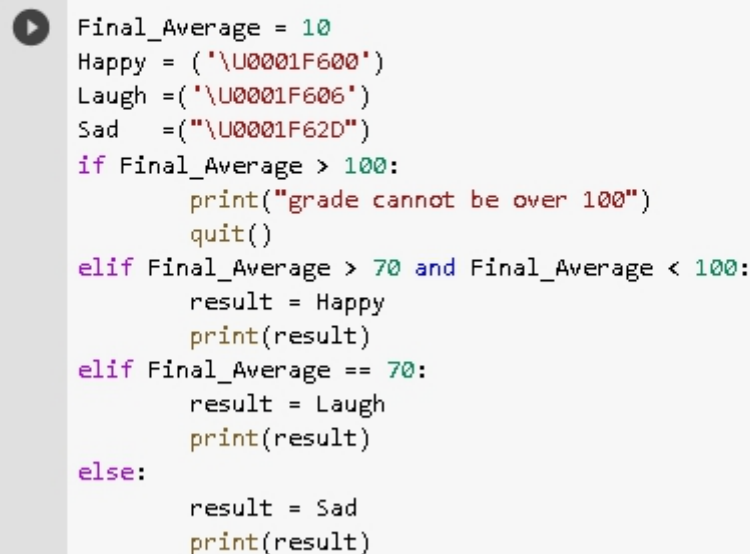
True

Figure 1.8 Logical by Casio

In Figures 1.7 and 1.8, Logical operators are used to manipulate logical operations in each of the variables values, we can figure it out if it is true or false depending on its conditions. The application of a truth table can be applied in logical operators because of the use of different kind of conditions can determine if two variables can be true or false . The three types of logical operators shown are the logical Not, logical And, and logical Or.

▼ Flow Control

▼ Condition Statements



```
Final_Average = 10
Happy = ('\U0001F600')
Laugh = ('\U0001F606')
Sad = ('\U0001F62D')
if Final_Average > 100:
    print("grade cannot be over 100")
    quit()
elif Final_Average > 70 and Final_Average < 100:
    result = Happy
    print(result)
elif Final_Average == 70:
    result = Laugh
    print(result)
else:
    result = Sad
    print(result)
```

Figure 1.9 Condition Statements by Casin o

Conditional Statement in Python is used to perform different computations or command depending on whether a specific variable constraint evaluates to true or false. The program can be explained in a analogy where the programmer used the “Final_Average” as a specific key to determine what door will open when the value of the key is compared to a set of if statements. Decisions and options are given because we are finding that specific answer to that condition and if the specific answer is not met then the outcome would present as false.

▼ Looping Statements

▼ While

```
[65] ## while loops
      activity, score = 0, 5
      while(activity<=score):
          print(f"{activity}\t|\t{score}")
          activity+=1
```

0		5
1		5
2		5
3		5
4		5
5		5

▼ For

```
[68] for i in range(10):
      print(i)
```

0
1
2
3
4
5
6
7
8
9

Figure 1.10 Looping Statements by Casiño

In this Figure, looping statements shows that the system would loop until range or the given variable has ended. The range default value is zero which is the starting point and if given a parameter let's say it is ten then the sequence would loop ten times until it reaches the number nine(Counting from the start which is zero). To why is it adding up to nine, its because there is "i" which acts as the variable with the value depends on the range then increments when it loops.

▼ Functions

```
[ ] def try_again():
    print('#' * 66)
    answer = input('Do you want to tryn again?:Yes/No ')
    print('#' * 66)
    if answer == 'Yes' or answer == "yes" or answer == "YES":
        try_again()
    elif answer == "No" or answer == "no" or answer == "NO":
        quit()

try_again()

#####
Do you want to try again?:Yes/No yes
#####
Do you want to try again?:Yes/No no
#####
```

Figure 1.11 Functions by Casio

In this figure, it shows that functions are the blueprint of the code. It helps separate different actions for the program. Functions only runs when it is called like in the figure. It also acts as a messenger of data, because it is given its own specific data that may be sent to other def function.

▼ Lambda Functions

```
[2] xtian = 4

[3] def f(xtian):
    return 2*(xtian*xtian)-2
    f(xtian)

30

▶ casio = lambda xtian: 2*(xtian*xtian)-1
print(casio(xtian))

31
```

Figure 1.12 Lambda by Casio

In this figure, it shows that lambda can take any number of arguments but it should have a single expression. Its analogy can be compared into a linear equation where the expression for an equation is just a straight line.

```

1  data = {}
2
3  def register():
4      #Register
5      Student_ID = int(input("Enter the Student Number :"))
6      Name = input("Enter the name :")
7      Name_Full = Name.replace(' ', '_')
8      Course = input("Enter the course of {} :".format(Name))
9      Course_Full = Course.replace(' ', '_')
10     Prelim = float(input("Enter Prelim Grade: "))
11     Midterm = float(input("Enter Midterm Grade: "))
12     Final = float(input("Enter Final Grade: "))
13     print(('#' * 27 )+ 'DATABASE' + ('#' * 27))
14     #Average
15     PG = Prelim * 0.3
16     MG = Midterm * 0.3
17     FG = Final * 0.4
18     Final_Average = PG + MG + FG
19     Final_Average_Round = round(Final_Average, 2)
20
21     #Results
22     Happy = ('\U0001F600')
23     Laugh = ('\U0001F606')
24     Sad = ('\U0001F62D')
25     if Final_Average > 100:
26         print("grade cannot be over 100")
27         quit()
28     elif Final_Average > 70 and Final_Average < 100:
29         result = Happy
30     elif Final_Average == 70:
31         result = Laugh
32     else:
33         result = Sad

```

Figure 2.1 Codes by Casiño

Figure 2.1 codes show the dict, functions, variables and data types, inputs and outputs, and flow control. The arithmetic help determine the final average with the help of conditional statements which would give the result needed.

```

35     #Split and string
36     Prelim_str = str(Prelim).split()
37     Midterm_str= str(Midterm).split()
38     Final_str  = str(Final).split()
39     Final_Average_str = str(Final_Average_Round).split()
40     Student_ID_str = str(Student_ID).split()
41     Name_Full_str = str(Name_Full).split()
42     Course_Full_str = str(Course_Full).split()
43
44     #Dict
45     Student_ID_key = Student_ID_str[0]
46     Name_key = Name_Full_str[0]
47     Course_key = Course_Full_str[0]
48     Prelim_key = Prelim_str[0]
49     Midterm_key = Midterm_str[0]
50     Final_key = Final_str[0]
51     Final_Average_key = Final_Average_str[0]
52     emoji_key = result[0]
53
54     data["1 : Student Number:      " + Student_ID_key] = \
55     {
56
57         2:"Name:                  " + Name_key.upper(),
58         3:"Course:                " + Course_key.upper(),
59         4:"Prelim Grade:          " + Prelim_key,
60         5:"Midterm Grade:         " + Midterm_key,
61         6:"Final Grade:           " + Final_key,
62         7:"Final Average:         " + Final_Average_key,
63         8:'Result:                ' + emoji_key,
64         0:'#' * 66
65
66     }

```

Figure 2.2 Codes by Casiño

Figure 2.2 codes show that the float and int data types are converted into a string so that it can be contained in the container. The string would be taken as an element with an index of 0 and to avoid having a stand-alone string we have used the split function. The programmer used the container “dict” because its better to save the data that are inputted every time “tryagain()” is use to repeat the program.

```

67         for tab in data:
68             print(tab)
69             for tabs in data[tab]:
70                 print(tabs, ":", data[tab][tabs])
71         try_again()
72
73     def try_again():
74         print('#' * 66)
75         answer = input('Do you want to calculate again?:Yes/No ')
76         print('#' * 66)
77         if answer == 'Yes' or answer == "yes" or answer == "YES":
78             register()
79         elif answer == "No" or answer == "no" or answer == "NO":
80             quit()
81
82     register()

```

Figure 2.3 Codes by Casíño

Figure 2.3 codes show that the for looping statement is used for avoiding the outputs to be printed in a whole straight line. It also shows the function “try again” so that the user can repeatedly use the program. The question is answered by yes or no, if yes then it would go back to the first function and if no then it would quit the program.

```

Enter the Student Number :1
Enter the name :christian
Enter the course of christian :cpe
Enter Prelim Grade: 99.9999
Enter Midterm Grade: 92.534
Enter Final Grade: 93.12312
#####DATABASE#####
1 : Student Number:      1
2 : Name:                CHRISTIAN
3 : Course:              CPE
4 : Prelim Grade:        99.9999
5 : Midterm Grade:       92.534
6 : Final Grade:         93.12312
7 : Final Average:       95.01
8 : Result:              😊
0 : #####
#####
Do you want to calculate again?:Yes/No yes

```

Figure 3.1 Input and Output by Casiño

Figure 3.1 show the inputs from the user and outputs from the program. The inputs are all saved in the container for future references. The condition in the activity where “Students could type their names, the name of the course, then their prelim, midterm, and final grade” and “The program should print the semestral grade in 2 decimal points and should display the following emojis depending on the situation” is satisfied. The condition of the result is also satisfied when the grade is higher than 70 it should print happy emoji.


```
#####DATABASE#####
1 : Student Number:      1
2 : Name:                CHRISTIAN
3 : Course:              CPE
4 : Prelim Grade:        99.9999
5 : Midterm Grade:       92.534
6 : Final Grade:         93.12312
7 : Final Average:       95.01
8 : Result:              😊
0 : #####
1 : Student Number:      2
2 : Name:                AERON
3 : Course:              CPE
4 : Prelim Grade:        70.0
5 : Midterm Grade:       70.0
6 : Final Grade:         70.0
7 : Final Average:       70.0
8 : Result:              😞
0 : #####
1 : Student Number:      3
2 : Name:                JOBERT
3 : Course:              CPE
4 : Prelim Grade:        10.0
5 : Midterm Grade:       20.0
6 : Final Grade:         30.0
7 : Final Average:       21.0
8 : Result:              😞
0 : #####
#####
```

Figure 3.3 Input and Output by Casíño

Figure 3.3 show that all three conditions have been satisfied when the grade is below 70 the emoji printed shall be sad. The inputs are also saved in database.

```
#####
Do you want to calculate again?:Yes/No no
#####

Process finished with exit code 0
|
```

Figure 3.4 Input and Output by Casíño

Figure 3.4 shows that the quit function works and the program exited smoothly.

IV. Conclusion

The principles and techniques of variables and data types, arithmetic, Inputs, and outputs, looping statements, flow control, and functions combined and applied together can create a whole new program depending on its usage and application. These principles and techniques are essential for beginners since it helps discover the answers to what creates a program in python, what basic syntaxes are used in order to achieve these feats, how to handle the different conditions and logics for its application, how to maintain the style of coding, what extra ordinary functions can be used with the basics and how to handle the errors of the program. The programmer discovered how to confront the variable naming style so that the code is readable and understandable, how to use conditional statements more efficiently, the roles of there techniques and its application, and manipulation of several datas. The programmer created a semestral grade calculator with the use of all the principles and techniques because these are taught might as well use it so that it can show the relation of each of the python's syntaxes for this labarotory activity. The programmer knows that by even if the only requirements in this study is to create a semestral calculator where students could type their names, the name of the course, then their prelim, midterm, and final grade with its limit to a two decimal points and results in a emoji is that we should still apply all the python's syntaxes that were taught because the professor worked hard to teach it and that means the programmers should work hard for this task, thats why the programmer created a complex set of codes where the application is still user friendly.

References

- [1] J. Lopez, “dyjdllopez/linearAlgebra2021,” *GitHub*, 03-Feb-2021. [Online]. Available: https://github.com/dyjdllopez/linearAlgebra2021/blob/main/Week%201%20-%20Programming%20Fundamentals/CpE_Programming_101_b_Python_Fundamentals.ipynb. [Accessed: 06-Feb-2021].
- [2] *Python Assignment Operators*. [Online]. Available: https://www.w3schools.com/python/gloss_python_assignment_operators.asp. [Accessed: 13-Mar-2021].
- [3] “Assignment Operators in Python,” *GeeksforGeeks*, 29-Aug-2020. [Online]. Available: <https://www.geeksforgeeks.org/assignment-operators-in-python/>. [Accessed: 13-Mar-2021].
- [4] “Python Comparison Operators Example,” *Tutorialspoint*. [Online]. Available: https://www.tutorialspoint.com/python/comparison_operators_example.htm. [Accessed: 13-Mar-2021].
- [5] “Python Input, Output and Import,” *Programiz*. [Online]. Available: <https://www.programiz.com/python-programming/input-output-import>. [Accessed: 13-Mar-2021].
- [6] “Python Operators: Arithmetic, Logical, Comparison, Assignment, Bitwise & Precedence,” *Guru99*. [Online]. Available: <https://www.guru99.com/python-operators-complete-tutorial.html#:~:text=Logical%20Operators%20in%20Python%20are,logical%20OR%20and%20logical%20NOT>. [Accessed: 13-Mar-2021].
- [7] “Python Conditional Statements: IF...Else, ELIF & Switch Case,” *Guru99*. [Online]. Available: <https://www.guru99.com/if-loop-python-conditional-structures.html#:~:text=What%20are%20Conditional%20Statements%20in,by%20IF%20statements%20in%20Python>. [Accessed: 13-Mar-2021].
- [8] *Python For Loops*. [Online]. Available: https://www.w3schools.com/python/python_for_loops.asp. [Accessed: 13-Mar-2021].
- [9] *Python Functions*. [Online]. Available: https://www.w3schools.com/python/python_functions.asp. [Accessed: 13-Mar-2021].
- [10] C. C. H. Casiño, “ChristianCasino/Linear,” *GitHub*, 13-Mar-2021. [Online]. Available: <https://github.com/ChristianCasino/Linear/blob/main/lab1/Activity%201.py>. [Accessed: 13-Mar-2021].