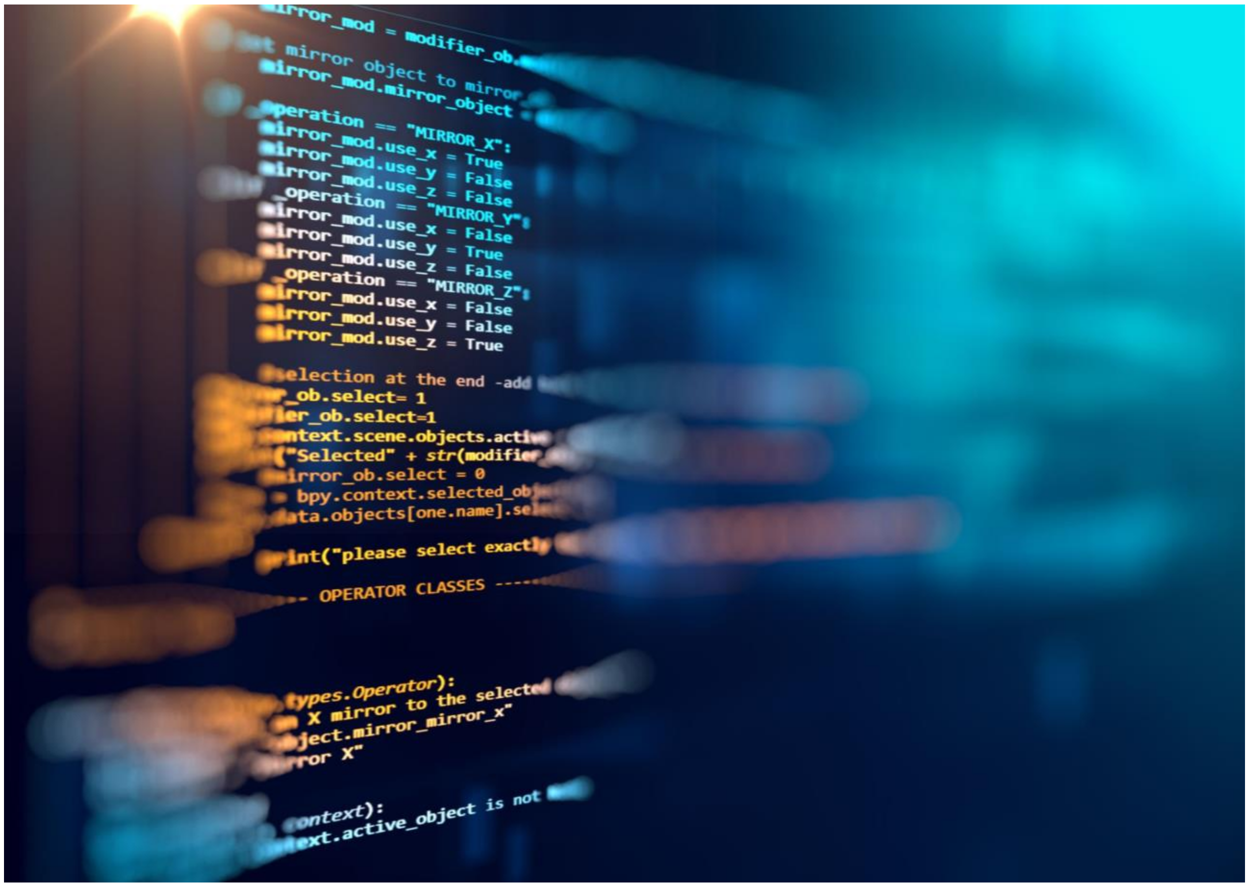
LBYCPA1

Programming Logic and Design Laboratory



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

Laboratory Module 4

Control Structure – Loops

By

John Carlo Theo S. Dela Cruz | LBYCPA1 | EQ1

INTRODUCTION

Fourth Module of this course will mainly focus on Control Structures more specifically looping statements. According in the module 4, Looping statements is a flow control statements that are always present in every aspect in the programming field; it is widely used and incredibly important. Looping from the word loop, it executes the same code within a specific condition or number of times. A basic differentiation between while and for loop is: While loop statement loops a certain code within a certain condition is satisfied or is true, and For loop statement repeats the code within a certain range. In this module, we will be focusing ourselves on how we can use the looping statements and use these statements to create a program with the knowledges from our past modules.

The main objective of every laboratory report is to provide opportunity for learning and to provide challenging experience in the field of programming despite the hindrance of the Pandemic. Through the process of programming alongside with the method of planning; algorithms, pseudocodes and flowcharts that provides systematic process on how the program works. Python is a very powerful and flexible language, and we are responsible to plan how the program will work. This module would be a big challenge for us aspiring software developers since this module would challenge us to further use the loop statements.

<u>What do you think are the main objectives for this module?</u> (Enumerate as many as you can.)

(a) Objectives

1. To familiarize with the use of FOR and WHILE Looping statements
2. To generate arithmetic progression using the range () function
3. To solve computational problems using looping statements
4. To use and adapt the knowledge in Looping Statements with the use of Karel the Bot.

<u>What are the materials used for this module?</u>

(b) Materials and Tools

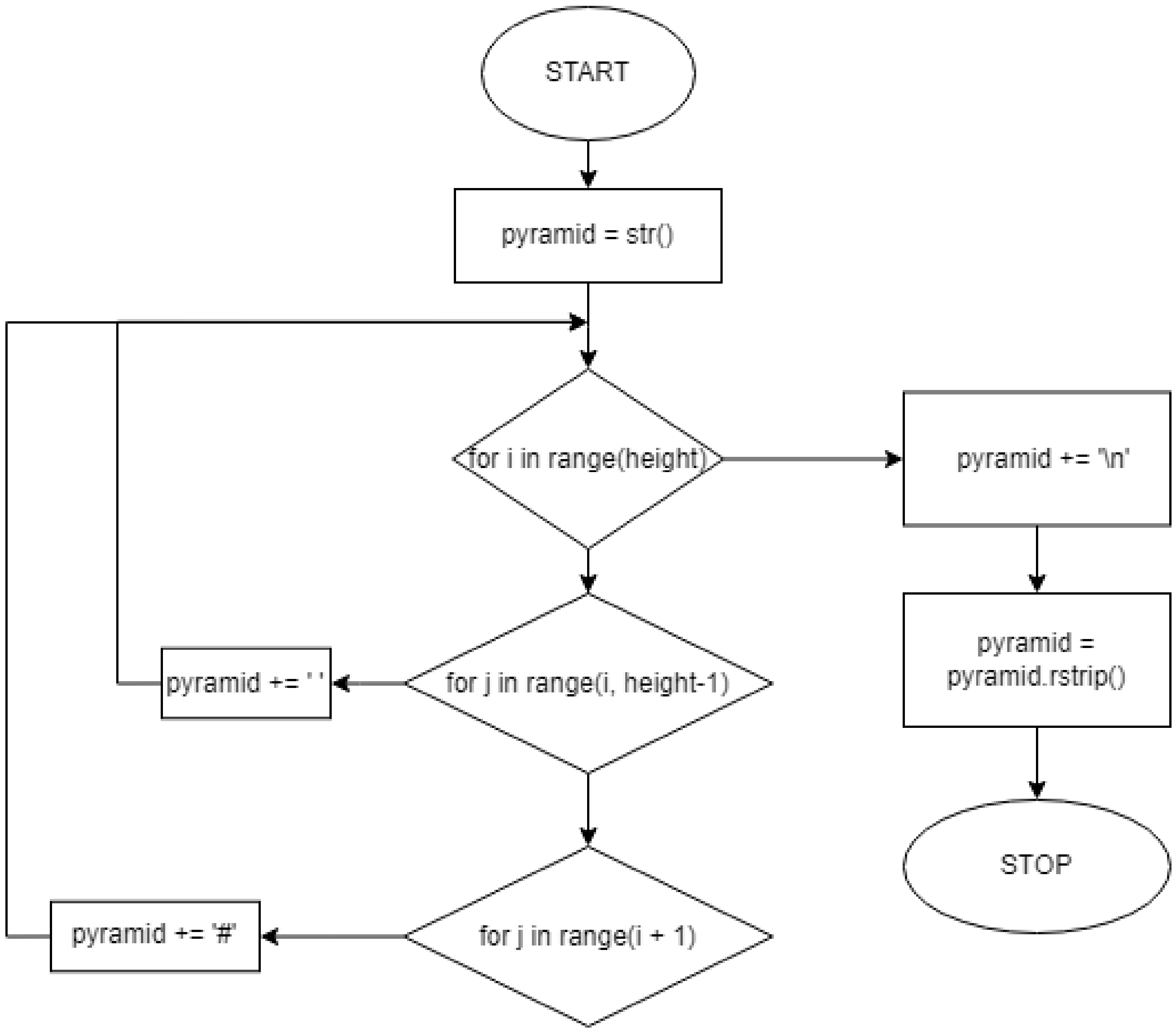
1. Instructor's lecture notes
2. Jupyter Notebook
3. Diagrams.net
4. Karel the Bot
5. Pycharm.edu
6. Google Browser

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.

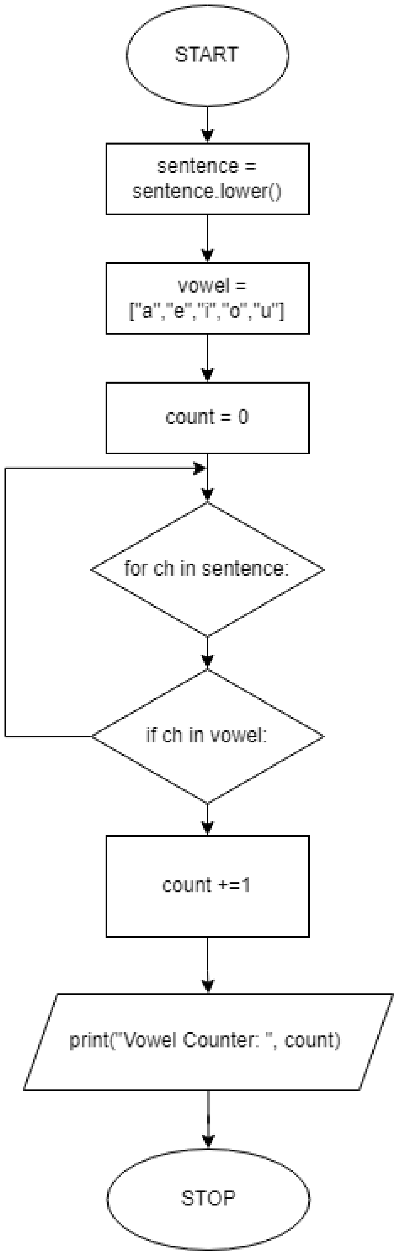
1. Familiarization Exercise 1 # Stairs:

In this program of creating a stair using the character of '#' just like the pyramid bricks in the Mario game. The problem said that the input or the character of # is a string and we need to concatenate. In this program we created a for loop within a for loop.



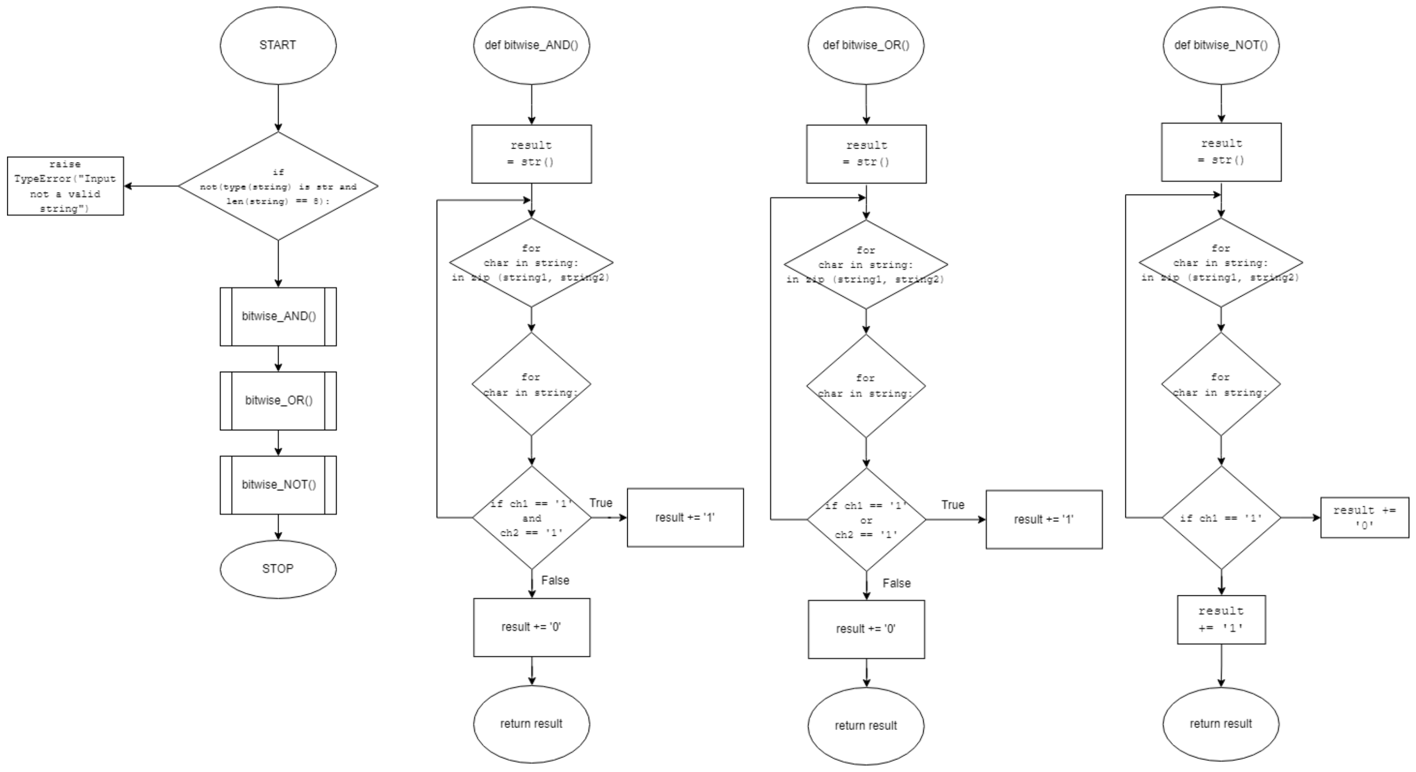
2. Familiarization Exercise 2 Vowel Counter:

This program will accept a string and count the number of vowels by using a mutable collection of values or a list []. Inside that list are each letter of the vowels and by using a for loop and an if statement that in every sentence there is a vowel included the value of count will increment and add 1 to the counter and print it.


START
sentence = sentence.lower()
vowel = ["a","e","i","o","u"]
count = 0
for ch in sentence:
if ch in vowel:
count +=1
print("Vowel Counter: ", count)
STOP

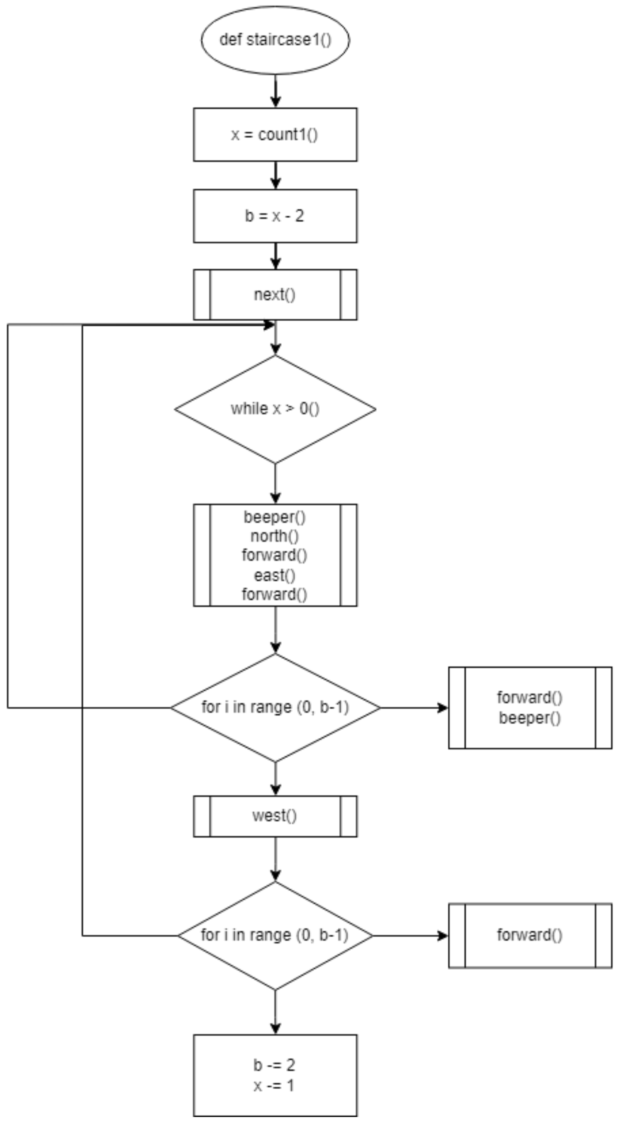
3. Familiarization Exercise 3 AND, OR, and NOT:

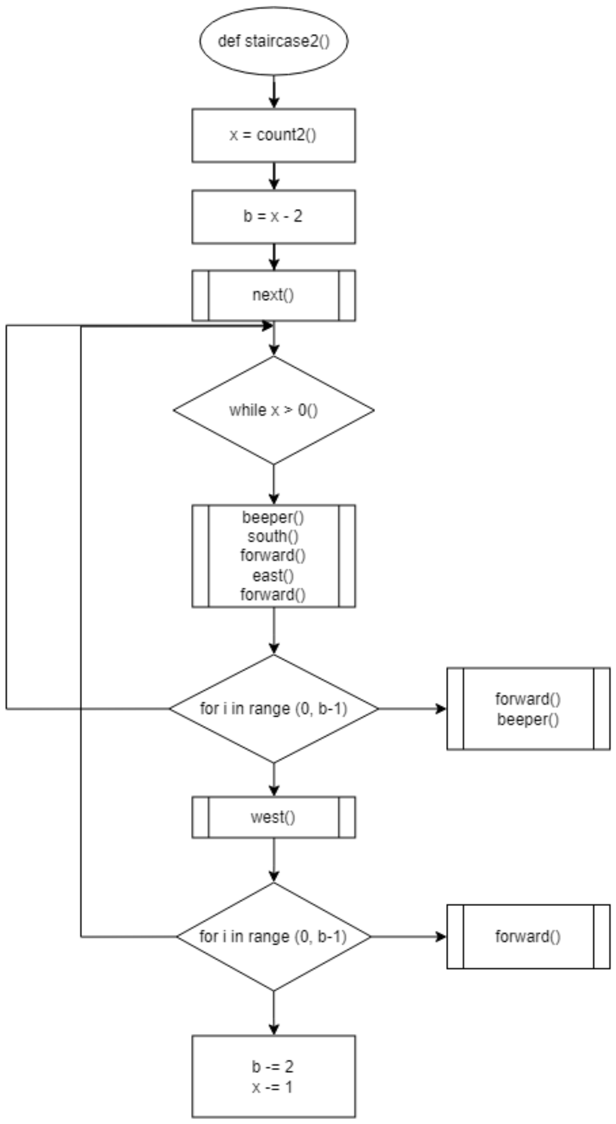
This program is a simulator of a string that is composed with three bitwise operations of NOT, OR, and NOT; also composed of characters of “1” and “0” as True and False. By using the Truth table, the solutions were based of that table as a reference on how these bitwise operations works.

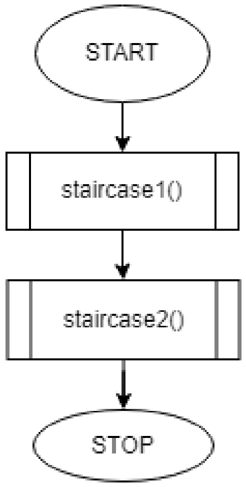


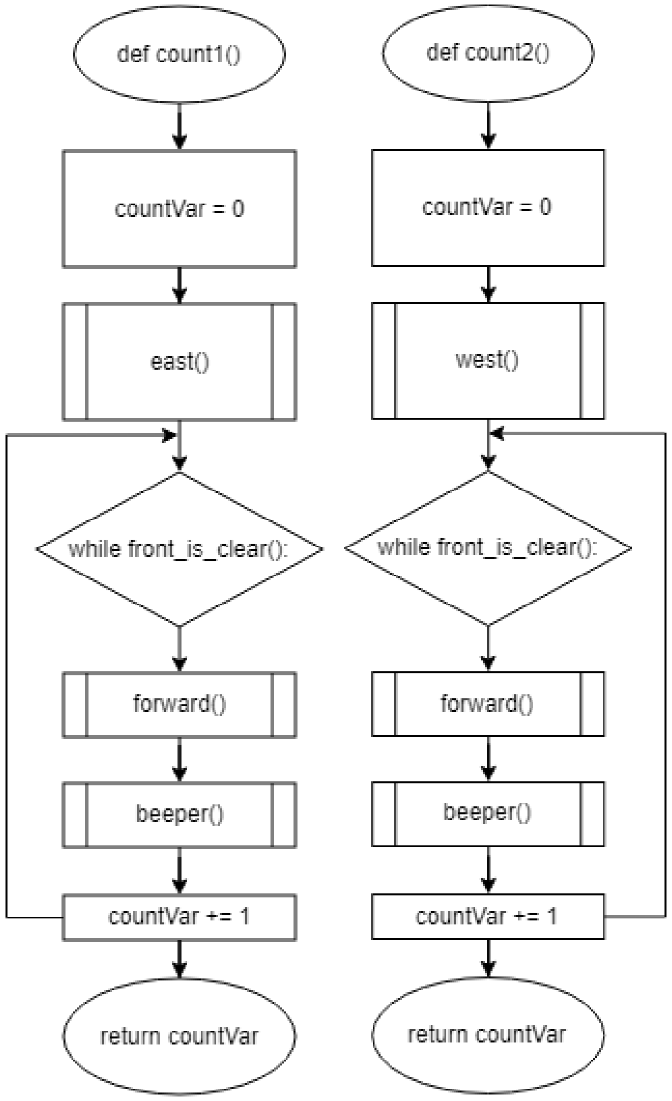
4. Familiarization Exercise 4 Hourglass using Karel Bot:

This program using Karel the bot, to create a simple hourglass in any kind of environment in the world of Karel. In my approach, I defined all the moves that Karel can do; to be able to execute the code with just the definition variable to avoid lengthy codes.






START
staircase1()
staircase2()
STOP

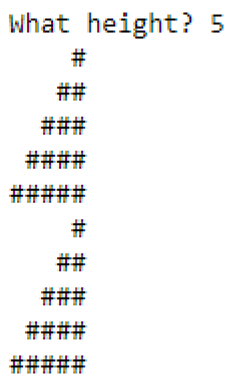

def count1()
countVar = 0
east()
while front_is_clear():
forward()
beeper()
countVar += 1
return countVar
def count2()
countVar = 0
west()
while front_is_clear():
forward()
beeper()
countVar += 1
return countVar

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

CRITERIA	**EXEMPLARY (90-100)**	**SATISFACTORY (80-89)**	**DEVELOPING (70-79)**	**BEGINNING (below 70)**	**WEIGHT**
Experimental Plan (Flowchart/ Algorithm) ***(SO-PI: B1)***	Experimental plan has supporting details and diagram/algorithm that is stated and well explained	Experimental plan has supporting details and diagram/algorithm that is stated but not explained	Experimental plan is vague or brief. It has supporting details and doesn't have diagram/algorithm	No experimental plan presented	30%
Conducting Experiment/ Activity ***(SO-PI: B1)*** ***(SO-PI: K1)***	Objective and Materials used (modern engineering tools, software, and instruments/equipment) are identified. Steps are easy to follow for conducting and experiment/activity.	Objective is not stated. Materials used (modern engineering tools, software, and instruments/equipment) are identified. Steps are easy to follow for conducting and experiment/activity	Does not provide enough information to conduct an experiment/activity	No Objective, Materials used (modern engineering tools, software, and instruments/equipment), and steps for conducting experiment/activity provided.	20%

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

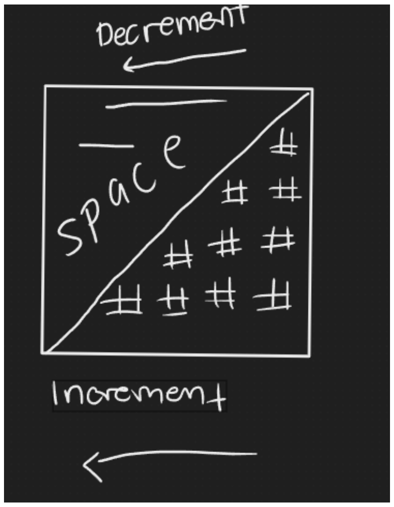
1. Familiarization Exercise 1 # Stairs:



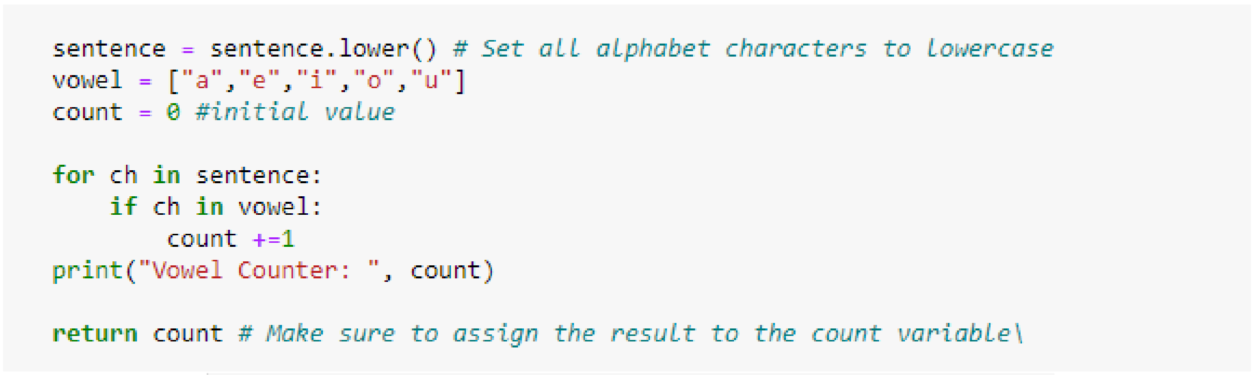
```
What height? 5
    #
   ##
  ###
 ####
#####
    #
   ##
  ###
 ####
#####
```

```python
for i in range(height):
    for j in range(i, height-1):
        pyramid += ' ' #concatenation, putting the char to the string
    for j in range(i+1):
        pyramid += '#'
    pyramid += '\n'
pyramid = pyramid.rstrip() #Remove Newline From String in Python
#Reference:
#https://www.delftstack.com/howto/python/python-remove-newline-from-string/
print(pyramid)

return pyramid # Make sure to assign the result to the pyramid variable
```

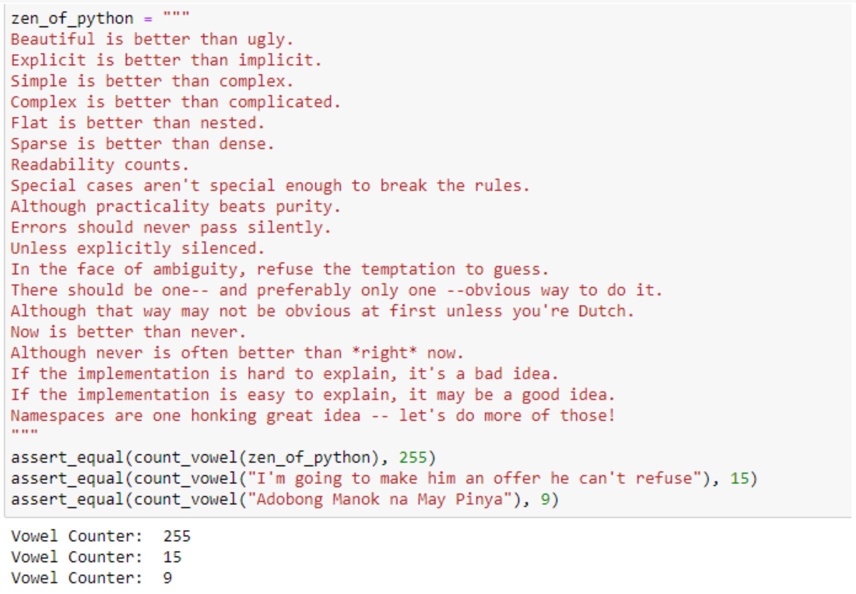
Explanation: This program requires us to create stairs just like in Mario World. The process in this program is to do it with a For loop within a for loop. 1st condition is the overall height and inside that function we will plug in the characters that we will be needing and plugging in which is an empty space and a hashtag '#'. (i, height – 1) we decremented that value of the height to make sure that the space will go in reverse or thru the left and concatenating the char to the string. Lastly end the loop with a newline. The problem I encountered in this program is that the last value should not have a newline in the end, I found a solution which is a special syntax called r. strip, and that syntax removes the newline from the string. Lastly, I printed out the output to test it out.



2. Familiarization Exercise 2 Vowel Counter:



```
sentence = sentence.lower() # Set all alphabet characters to lowercase
vowel = ["a","e","i","o","u"]
count = 0 #initial value

for ch in sentence:
    if ch in vowel:
        count +=1
print("Vowel Counter: ", count)

return count # Make sure to assign the result to the count variable\
```

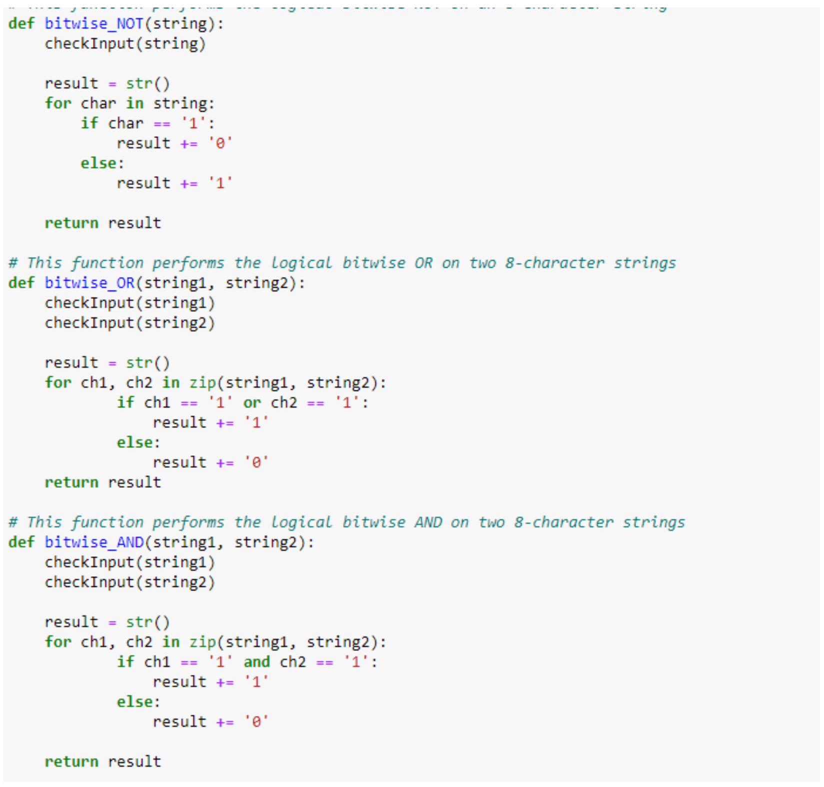


```
zen_of_python = """
Beautiful is better than ugly.
Explicit is better than implicit.
Simple is better than complex.
Complex is better than complicated.
Flat is better than nested.
Sparse is better than dense.
Readability counts.
Special cases aren't special enough to break the rules.
Although practicality beats purity.
Errors should never pass silently.
Unless explicitly silenced.
In the face of ambiguity, refuse the temptation to guess.
There should be one-- and preferably only one --obvious way to do it.
Although that way may not be obvious at first unless you're Dutch.
Now is better than never.
Although never is often better than *right* now.
If the implementation is hard to explain, it's a bad idea.
If the implementation is easy to explain, it may be a good idea.
Namespaces are one honking great idea -- let's do more of those!
"""
assert_equal(count_vowel(zen_of_python), 255)
assert_equal(count_vowel("I'm going to make him an offer he can't refuse"), 15)
assert_equal(count_vowel("Adobong Manok na May Pinya"), 9)
```

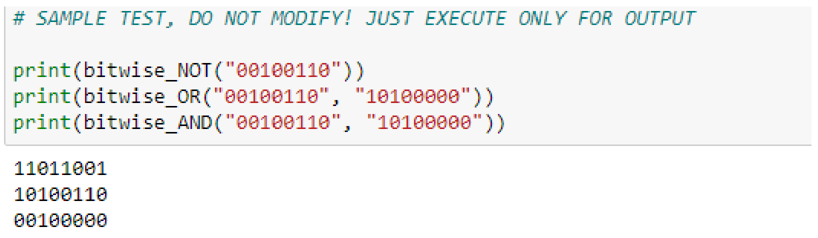
```
Vowel Counter:  255
Vowel Counter:  15
Vowel Counter:  9
```

Explanation: This program is a Vowel Counter, in which the user is required to enter a sentence, and locates every vowel letter each word in the sentences. By declaring the sentence variable and setting it all the alphabet characters to lowercase; following by a variable vowel that contains a list of the characters of the vowel, which is "a, e, i, o, u", lastly creating the last variable which is count and initialize it to a value of 0. To create this program, we will use a for loop and an if statement. The loop's condition is in the value containing the sentence and inside the If statement is that in every sentence, we will be counting down how many vowels are inside the sentence. Every time the program detects a vowel the count will add 1 value.

3. Familiarization Exercise 3 AND, OR, and NOT:



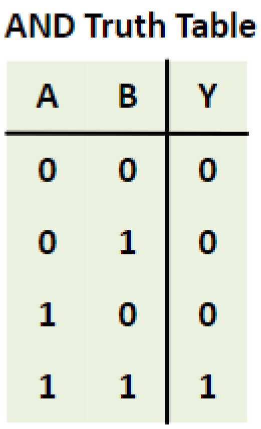
```python
# This function performs the logical bitwise NOT on an 8-character string
def bitwise_NOT(string):
    checkInput(string)

    result = str()
    for char in string:
        if char == '1':
            result += '0'
        else:
            result += '1'

    return result

# This function performs the logical bitwise OR on two 8-character strings
def bitwise_OR(string1, string2):
    checkInput(string1)
    checkInput(string2)

    result = str()
    for ch1, ch2 in zip(string1, string2):
            if ch1 == '1' or ch2 == '1':
                result += '1'
            else:
                result += '0'
    return result

# This function performs the logical bitwise AND on two 8-character strings
def bitwise_AND(string1, string2):
    checkInput(string1)
    checkInput(string2)

    result = str()
    for ch1, ch2 in zip(string1, string2):
            if ch1 == '1' and ch2 == '1':
                result += '1'
            else:
                result += '0'

    return result
```



```python
# SAMPLE TEST, DO NOT MODIFY! JUST EXECUTE ONLY FOR OUTPUT

print(bitwise_NOT("00100110"))
print(bitwise_OR("00100110", "10100000"))
print(bitwise_AND("00100110", "10100000"))
```

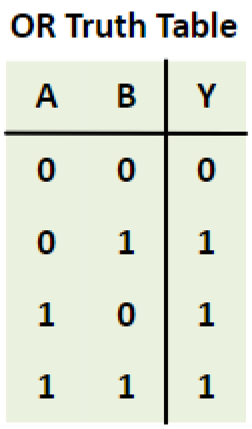
```
11011001
10100110
00100000
```

Explanation: This program requires us to review the Truth table of AND, OR, and NOT functions. In the NOT function, it is self-explanatory; in whatever value you put the function will inverse the output. For the function of AND, and OR we used the zip function instead of range, since the output is only 1 and 0, the values can be interchanged; it can be 1,1 – 1,0 etc.



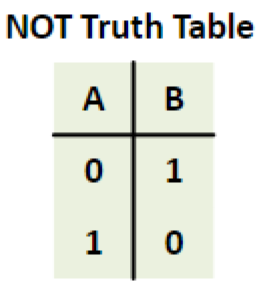
AND Truth Table

A	B	Y
0	0	0
0	1	0
1	0	0
1	1	1



OR Truth Table

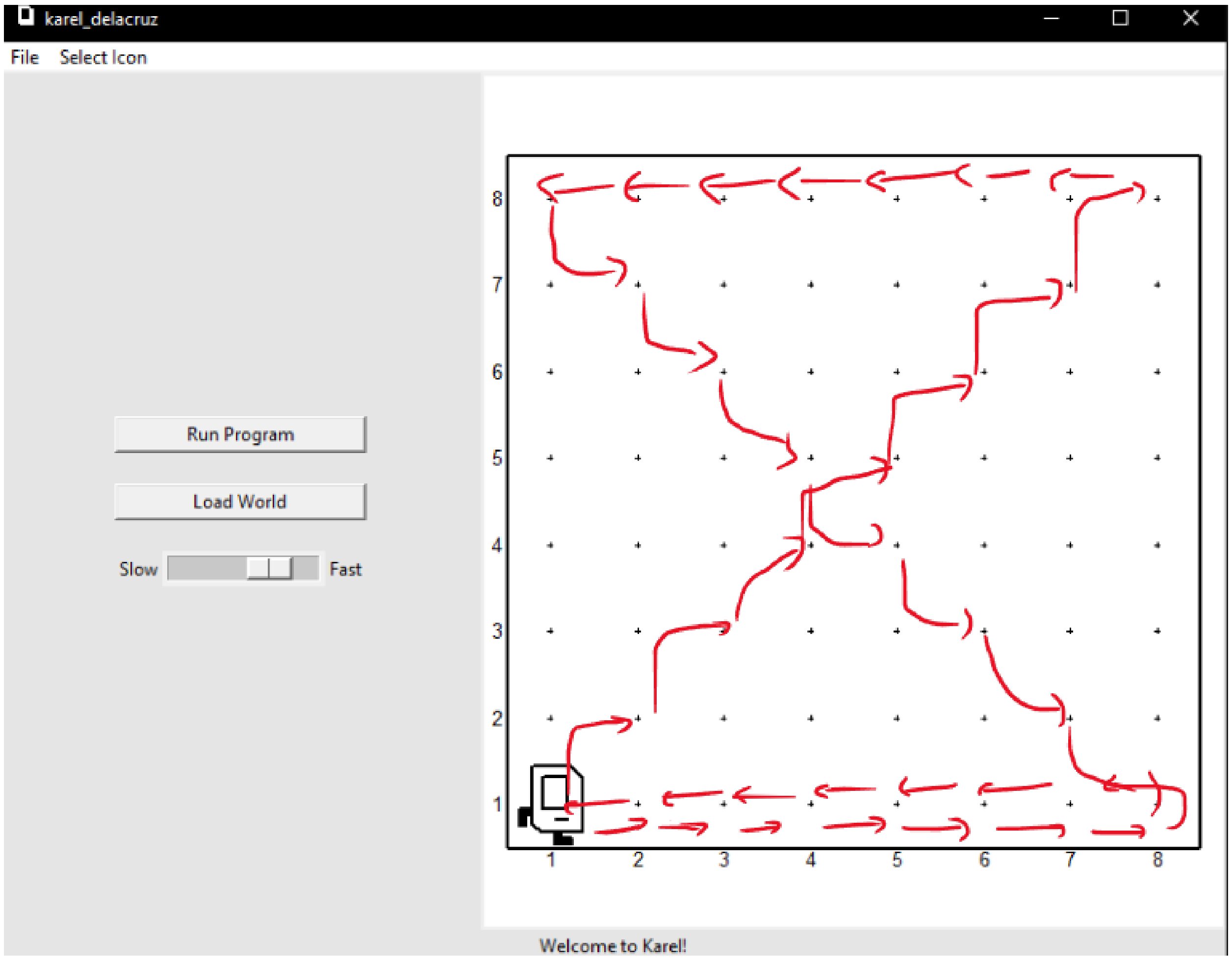
A	B	Y
0	0	0
0	1	1
1	0	1
1	1	1



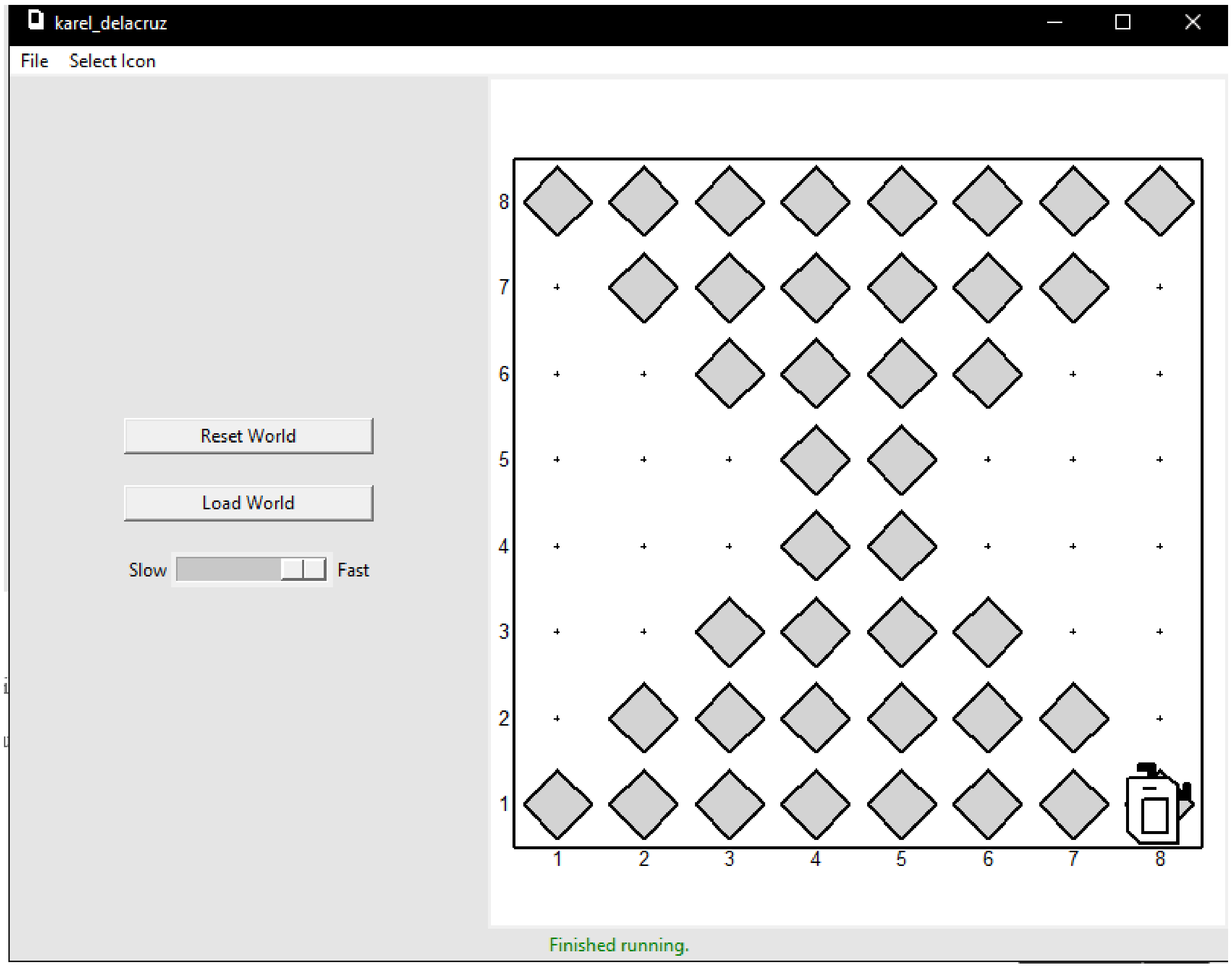
NOT Truth Table

A	B
0	1
1	0

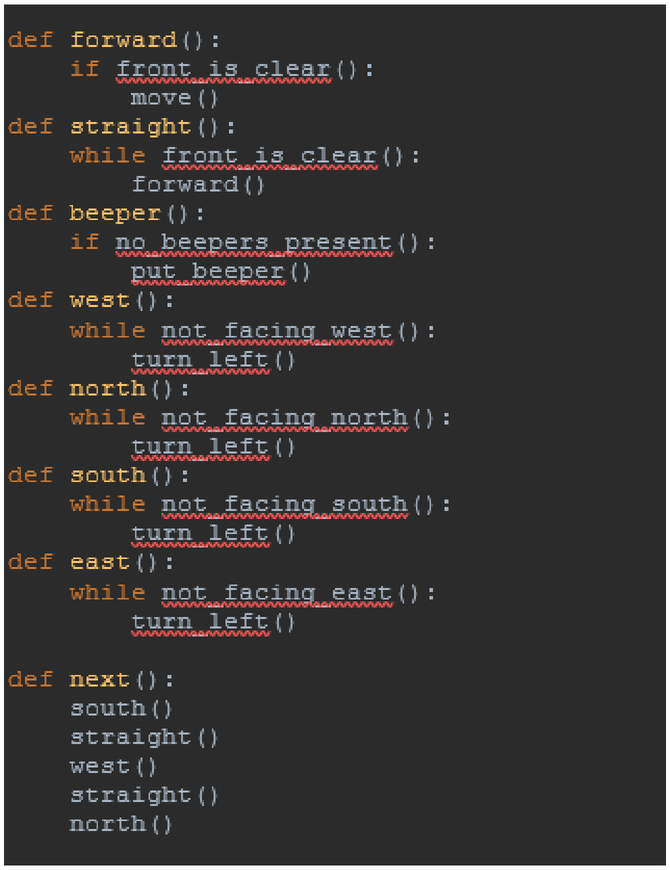
4. Familiarization Exercise 4 Hourglass using Karel Bot:



Welcome to Karel!



Explanation: This is one of the most complicated problems I've encountered. Since the Karel should have an output of a Hourglass in every environment. First step is that I defined all the moves that I will be using to execute the program, to easily put my codes easily, I created a function that counts the environment, its x and y. While the Bot is moving forward and counting, the bot also places its beepers along the way. After reading the world value, it goes back to the origin. Create a staircase first, by repeating a set of codes within a certain range of the x value or the world value. While it creates a stair, I made sure that it fills the void of the hourglass. Variable b is the beeper count, and it decrements 2 values each time the loop is executing. After creating a half of the hourglass, I copy-pasted the same line of codes to create another function that corresponds to the side of the bot, and it does the same thing from top to bottom. **x = world count; b = beeper count**



```
def forward():
    if front_is_clear():
        move()
def straight():
    while front_is_clear():
        forward()
def beeper():
    if no_beepers_present():
        put_beeper()
def west():
    while not_facing_west():
        turn_left()
def north():
    while not_facing_north():
        turn_left()
def south():
    while not_facing_south():
        turn_left()
def east():
    while not_facing_east():
        turn_left()

def next():
    south()
    straight()
    west()
    straight()
    north()
```

The Results and Discussions constitutes the data criteria in the Lab Report Evaluation Rubric:

Codes/Data/ Program	Data is well utilized in the program. Program codes are easy to read. Program output has no error. Questions are answered completely and correctly	Data is somewhat utilized in the program. Program code are easy to read. Program output has an output but logically incorrect. Some questions are answered completely and correctly	Data is not utilized in the program. It has a missing significant code/syntax in the program.	No program presented	30%

CONCLUSION:

This 4th Module of LBYCPA1 mainly focuses on the Control Structure, more specifically the For and While loops. We revisited the lessons and knowledge that were introduced last 2 modules which is the ***not, and, or,*** and ***if and else.*** This module tested our limits as a student. The expected outputs of this module were easy, but the process was complicated; the most complicated one in my opinion, is the Karel

Bot: Hourglass, because in this problem we can do whatever we want, any approach just to create an hourglass in every environment world of Karel.

I learned that we could do several things with the Loop statements, and I appreciated those features and structures as I go through this module. I achieved everything in this module including debugging and making sure that there is no error in the Assertion Error. I also learned and discovered that I should not look on the whole problem first, because it will lead me to stress and fear of completing the module; instead of worrying, I first think of a solution step-by-step. What I want to declare first, and what I want to do after it; as I do the step-by-step, always make sure that you are in a Code-state to be more focused and to be less distracted. Common pitfalls that I have encountered is just purely logic errors. I didn't read and understand the problems first and that mistake is a learning experience for me and some advice for those students who will also take this module soon. The last advice is to continue practicing because as time goes by, it creates a better foundation for us future programmers.

The rest of the rubric criteria are as follows:

Grammar, logical presentation, and format ***(SO-PI: G1)***	The report was grammatically correct, logically presented and used the required format.	The report had minimal grammatical errors and somewhat presented logically. The required format was used.	The report had a lot of grammatical errors and not logically presented; the required format was barely used.	The report had a lot of grammatical errors, was not logically presented and the required format was not used.	20%

REFERENCES (*Enumerate references in APA format*)

w3school (n.d.). Retrieved from. https://www.w3schools.com/python/python_intro.asp

stackOverflow (n.d.). Retrieved from. https://stackoverflow.com/

DelftStack (May 9, 2021). Retrieved from. https://www.delftstack.com/howto/python/python-remove-newline-from-string/

APPENDIX (*Attach all the source codes here per problem category*)

1. Familization Exercise 1:

```
def rightPyramid(height):
    if not type(height) is int:
        raise TypeError("Input not a valid integer")
    pyramid = str() # declare an empty string

    for i in range(height):
        for j in range(i, height-1):
            pyramid += ' ' #concatenation, putting the char to the string
        for j in range(i+1):
            pyramid += '#'
        pyramid += '\n'
    pyramid = pyramid.rstrip() #Remove Newline From String in Python
    #Reference:
    #https://www.delftstack.com/howto/python/python-remove-newline-from-
string/
    print(pyramid)

    return pyramid # Make sure to assign the result to the pyramid variable
```

2. Familiarization Exercise 2:

```
def count_vowel(sentence):
    if not type(sentence) is str:
        raise TypeError("Input not a string")

    sentence = sentence.lower() # Set all alphabet characters to lowercase
    vowel = ["a","e","i","o","u"]
    count = 0 #initial value

    for ch in sentence:
        if ch in vowel:
            count +=1
    print("Vowel Counter: ", count)

    return count # Make sure to assign the result to the count variable\
```

3. Familiarization Exercise 3:

```
def checkInput(string):
    if not(type(string) is str and len(string) == 8):
        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")

# This function performs the logical bitwise NOT on an 8-character string
def bitwise_NOT(string):
    checkInput(string)

    result = str()
    for char in string:
        if char == '1':
            result += '0'
        else:
            result += '1'

    return result

# This function performs the logical bitwise OR on two 8-character strings
```

```
def bitwise_OR(string1, string2):
    checkInput(string1)
    checkInput(string2)

    result = str()
    for ch1, ch2 in zip(string1, string2):
            if ch1 == '1' or ch2 == '1':
                result += '1'
            else:
                result += '0'
    return result

# This function performs the logical bitwise AND on two 8-character strings
def bitwise_AND(string1, string2):
    checkInput(string1)
    checkInput(string2)

    result = str()
    for ch1, ch2 in zip(string1, string2):
            if ch1 == '1' and ch2 == '1':
                result += '1'
            else:
                result += '0'

    return result
```

4.Familiarization Exercise 4:

```
# KAREL HOURGLASS
# Dela Cruz -- 12195758
from stanfordkarel import *

def forward():
    if front_is_clear():
        move()
def straight():
    while front_is_clear():
        forward()
def beeper():
    if no_beepers_present():
        put_beeper()
def west():
    while not_facing_west():
        turn_left()
def north():
    while not_facing_north():
        turn_left()
def south():
    while not_facing_south():
        turn_left()
def east():
    while not_facing_east():
        turn_left()

def next():
    south()
    straight()
    west()
    straight()
    north()

def count2():
    countVar = 0
    west()
    while front_is_clear():
        forward()
        beeper()
        countVar += 1
    # print(countVar)
    return countVar

def count1():
    countVar = 0
    east()
    while front_is_clear():
        forward()
        beeper()
        countVar += 1
    # print(countVar)
    return countVar

def staircase1():
    x = count1()
    b = x - 2
    next()
    # -- Start of Staircase --
    while x >= 0:
        beeper()
        north()
        forward()
        east()
        forward()
        for i in range (0,b-1):
            forward()
            beeper()
        west()
        for i in range (0,b-1):
            forward()
        b -= 2  #b -= 2
        x -= 1

def staircase2():
    x = count2()
    b = x - 2
    # -- Start of Staircase --
    while x >= 0:
        beeper()
        south()
        forward()
        east()
        forward()
        for i in range (0,b-1):
            forward()
            beeper()
        west()
        for i in range (0,b-1):
            forward()
        b -= 2
        x -= 1

def main():
    """ Karel code goes here! """
    staircase1()
    staircase2()

if __name__ == "__main__":
    run_karel_program()
```