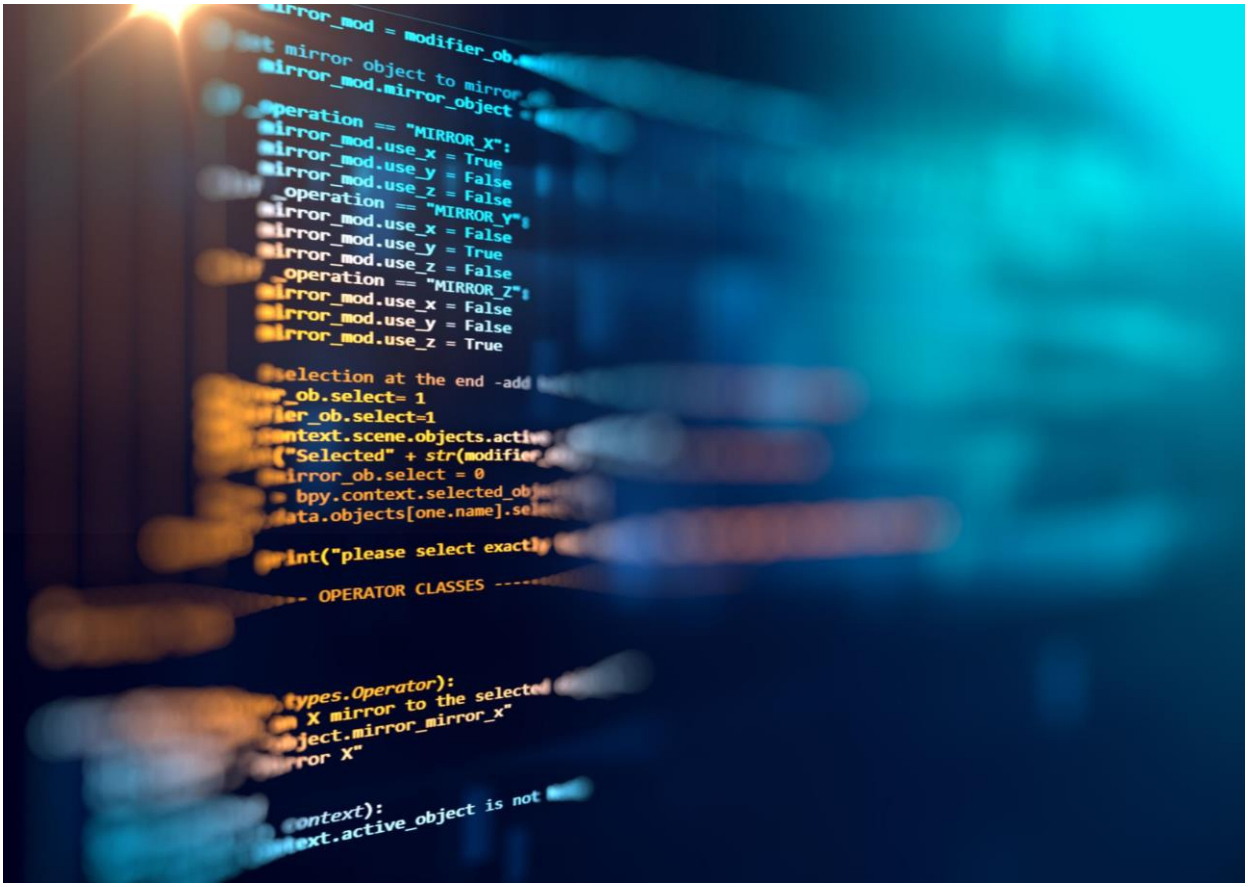
LBYCPA1

Programming Logic and Design Laboratory



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

Laboratory Module 5

Functions and Modules

By

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

INTRODUCTION

Fifth Module of this course will mainly focus on Functions and the modularity functions of Python. According in the Module 5 notes, Functions is a device or a system that groups a set of statements so they and be run more than once in a program. These functions are the most basic program structure of Python for programmers to maximize the reuse of codes and code redundancy. I used function to solve the Karel Bot problem from the previous modules because it can help me to organize and to maximize my coding space, so it won't be confusing to begin with. Module 5 also introduced us with built-in Python functions that can be used in the future such as round() – this function can round-off a float variable to a specific place decimal. Modules is a package containing a set of functions, it is like a zip file for Python that compiles a set of function and import it to another set of functions.

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.)

(a) Objectives

1. To understand the importance of modular programming
2. To define a function in Python and its scope
3. To familiarize with Python modules
4. To develop a simple Python module
5. To utilize functions in solving computational problems

What are the materials used for this module?

(b) Materials and Tools

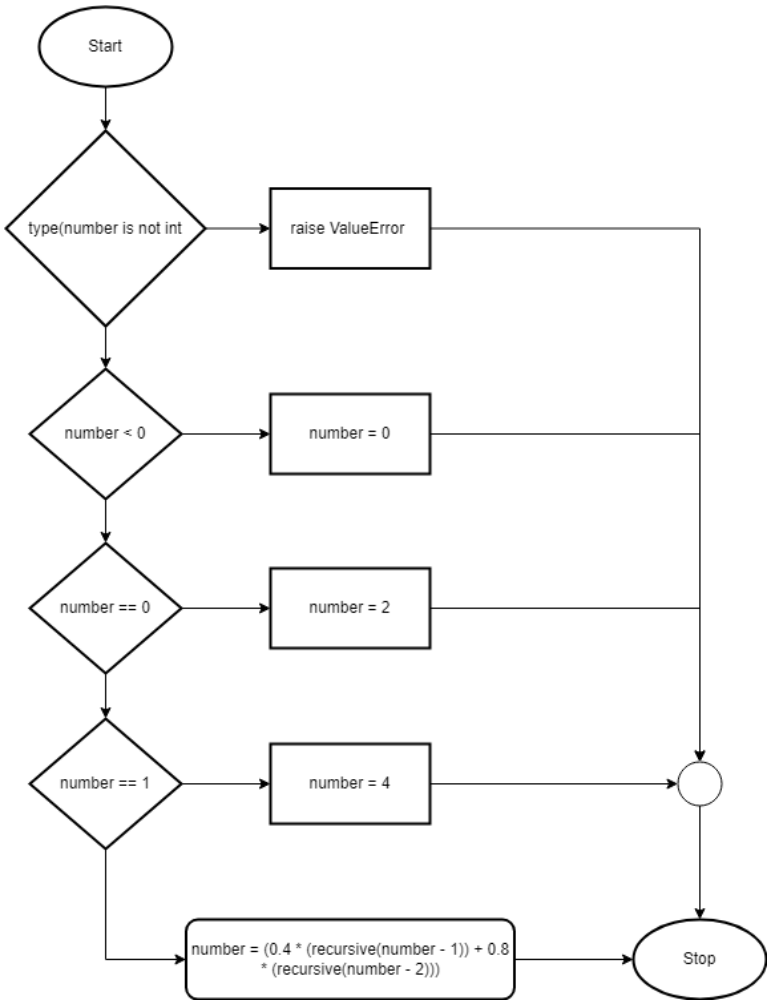
1. Instructor's lecture notes
2. Jupyter Notebook
3. Diagrams.net
4. Karel the Bot
5. Pycharm.edu
6. Google Browser

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.

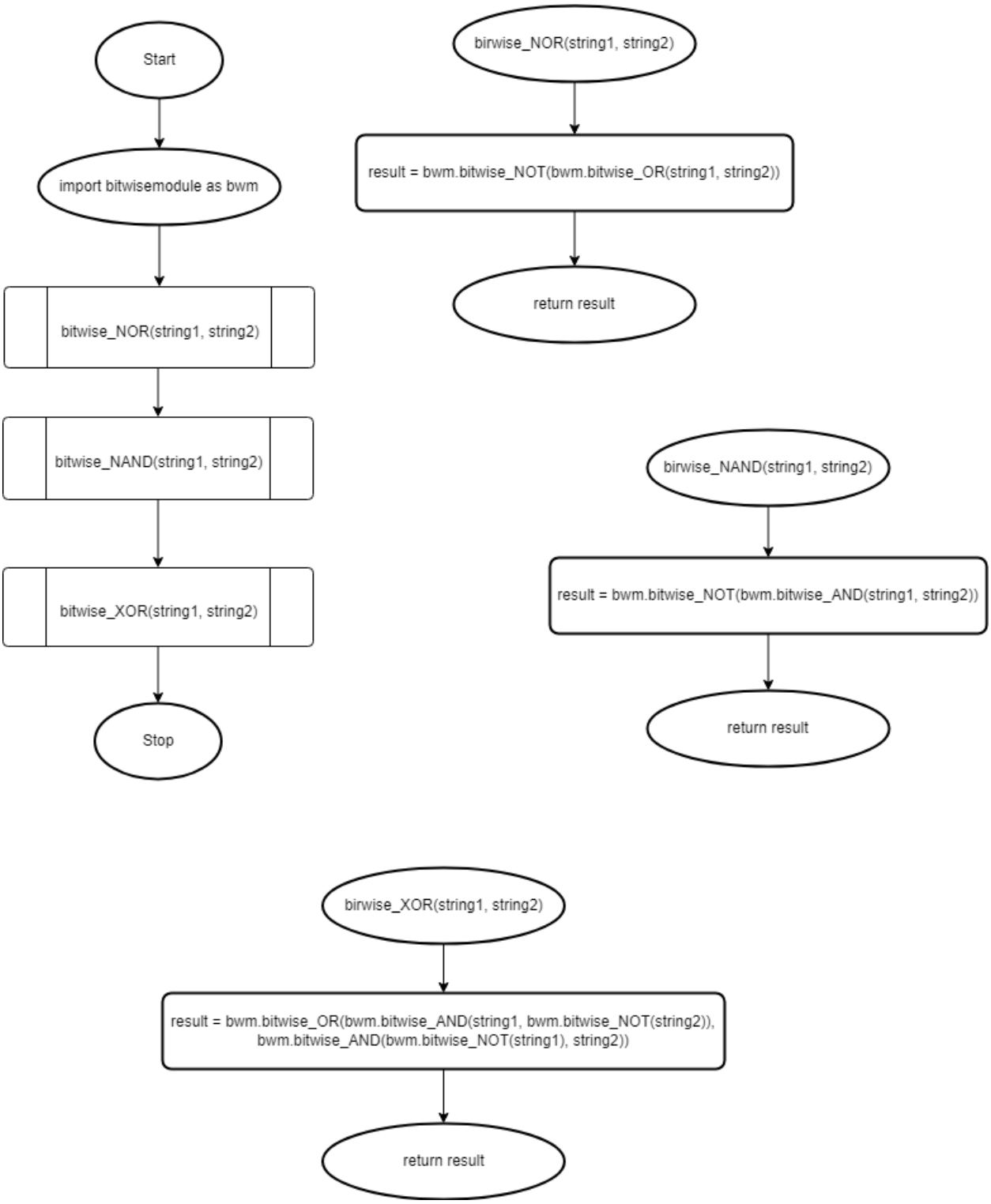
1. **Familiarization Exercise 1 - Recursion**

   In this program, we utilize the function of recursion (a DSCRTMATH lesson, that has a repeated application of a recursive procedure). Based in the instructions the function is defined that if the value is less than 0 then it will return 0, f(0) = 2, f(1) = 4. By following the given formula, it can satisfy the recurrence relation and returns the value needed.



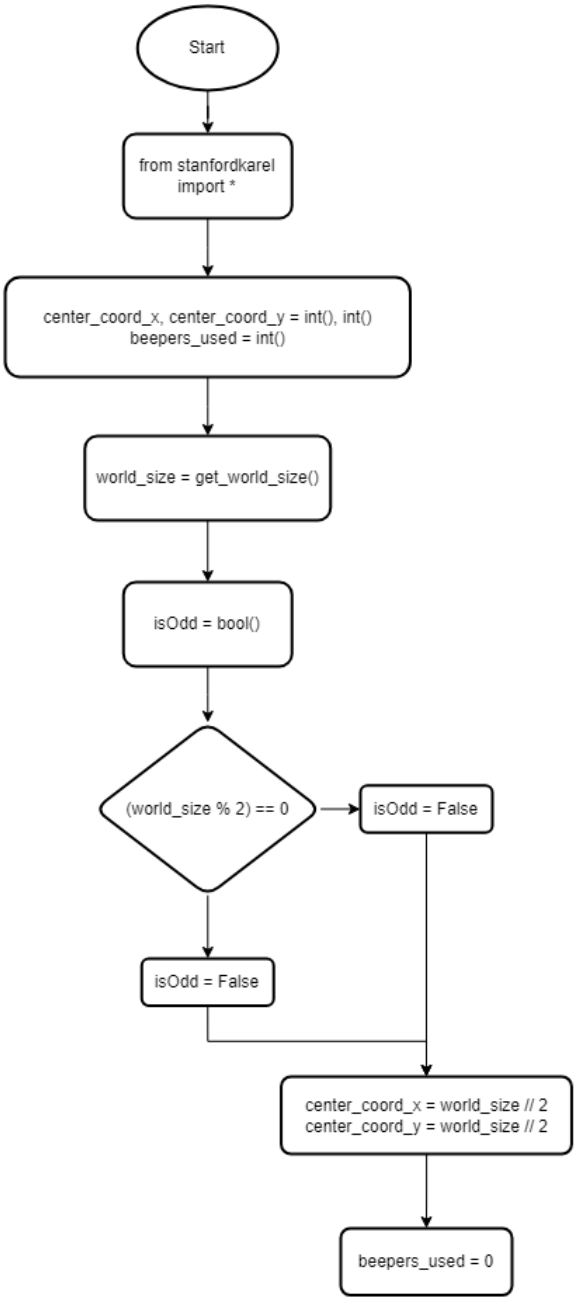
2. Familiarization Exercise 2 – Bitwise NOR, NAND, XOR

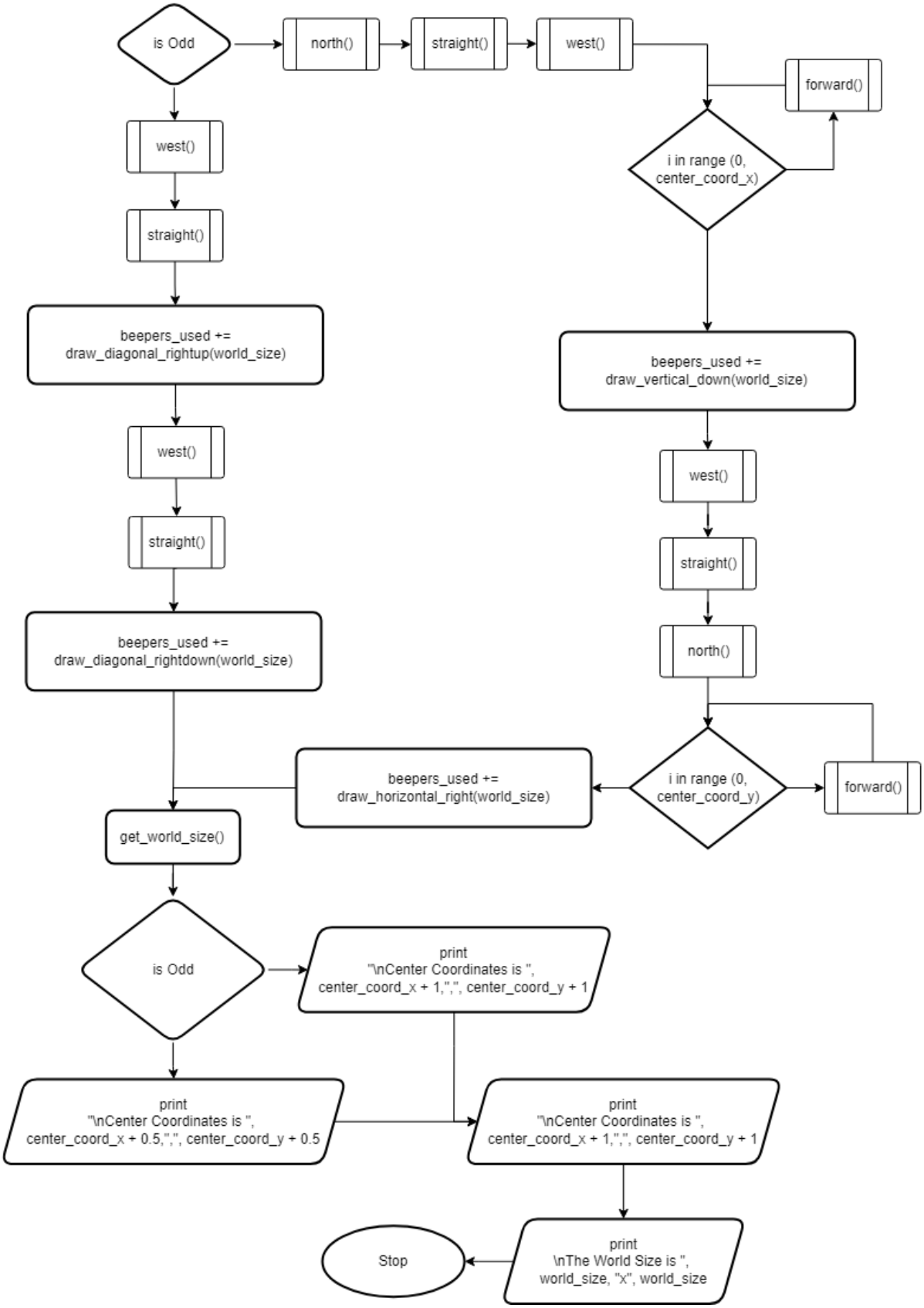
In this program, we utilize the function of importing (Module). Importing a python file of the previous activity in the notebook file, and using the function and utilizing it to implement the operations of NOR, NAND, and XOR


Start
import bitwisemodule as bwm
bitwise_NOR(string1, string2)
bitwise_NAND(string1, string2)
bitwise_XOR(string1, string2)
Stop
birwise_NOR(string1, string2)
result = bwm.bitwise_NOT(bwm.bitwise_OR(string1, string2))
return result
birwise_NAND(string1, string2)
result = bwm.bitwise_NOT(bwm.bitwise_AND(string1, string2))
return result
birwise_XOR(string1, string2)
result = bwm.bitwise_OR(bwm.bitwise_AND(string1, bwm.bitwise_NOT(string2)),
bwm.bitwise_AND(bwm.bitwise_NOT(string1), string2))
return result

3. Familiarization Exercise 3 - Karel Cross

In this program, we utilize the lesson of function in Karel Bot. There is a step-by-step procedure on how we can operate and execute the code in our IDE. If the world size is Odd, then it will execute a Vertical / Horizontal cross; and if it is Even, it will execute a diagonal cross.




is Odd
north()
straight()
west()
forward()
i in range (0, center_coord_x)
west()
straight()
beepers_used += draw_diagonal_rightup(world_size)
beepers_used += draw_vertical_down(world_size)
west()
straight()
west()
straight()
north()
beepers_used += draw_diagonal_rightdown(world_size)
beepers_used += draw_horizontal_right(world_size)
i in range (0, center_coord_y)
forward()
get_world_size()
is Odd
print "\nCenter Coordinates is ", center_coord_x + 1,",", center_coord_y + 1
print "\nCenter Coordinates is ", center_coord_x + 0.5,",", center_coord_y + 0.5
print "\nCenter Coordinates is ", center_coord_x + 1,",", center_coord_y + 1
Stop
print \nThe World Size is ", world_size, "x", world_size

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 - Recursion**

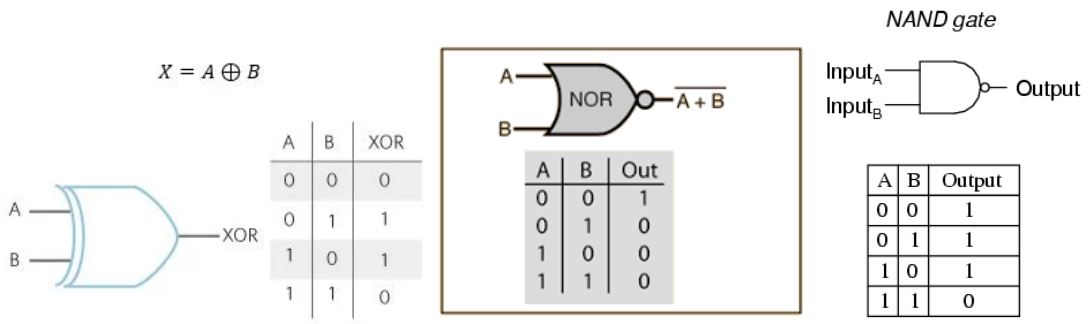
```
def recursive(number):
    # This function satisfies the recurrence relation f(x) = 0.4*f(x-1) + 0.8*f(x-2)
    if type(number) is not int:
        raise ValueError("Input value must be an integer")
    if (number < 0):
        number = 0
    elif (number == 0 ):
        number = 2
    elif (number == 1):
        number = 4
    else:
        number = (0.4 * (recursive(number-1)) + 0.8 * (recursive(number-2)))

    return number
```

Explanation:

This program requires the user to input an integer then the program solves the recursion value of the input. Recursion is a repetitive application of recursion, a real-life example of this is facing 2 mirrors at each other, they reflect to each other into small pieces of itself. As stated in the instructions that 0 is equals to 2 and 1 is equals to 4. If it is greater than those values, then the program will be using the formula stated to get the recursion value. As stated in the instructions we replace f into the function or definition of the recursion and follow the stated formula.

2. **Familiarization Exercise 2 – Bitwise NOR, NAND, XOR**

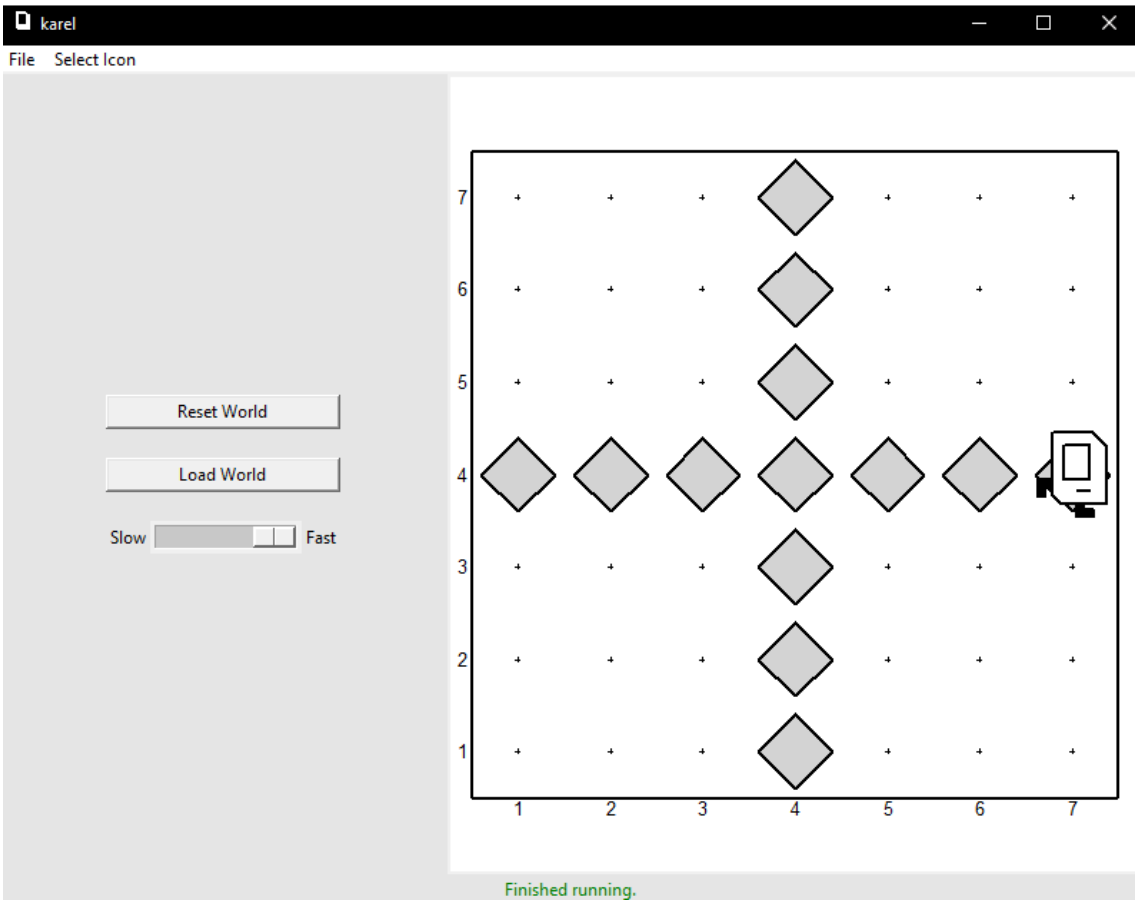
```
import bitwisemodule as bwm

def bitwise_NOR(string1, string2):

#     result = bwm.bitwise_OR(bwm.bitwise_NOT(string1),bwm.bitwise_NOT(string2))

    result = bwm.bitwise_NOT(bwm.bitwise_OR(string1,string2))
    return result

    # Do not forget the return statement for this function

def bitwise_NAND(string1, string2):
    # This function performs the logical bitwise NOR on two 8-character strings

    result = bwm.bitwise_NOT(bwm.bitwise_AND(string1,string2))
    return result
# Do not forget the return statement for this function

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

    result = bwm.bitwise_OR(bwm.bitwise_AND(string1,bwm.bitwise_NOT(string2)),bwm.bitwise_AND(bwm.bitwise_NOT(string1),string2))
    return result
```

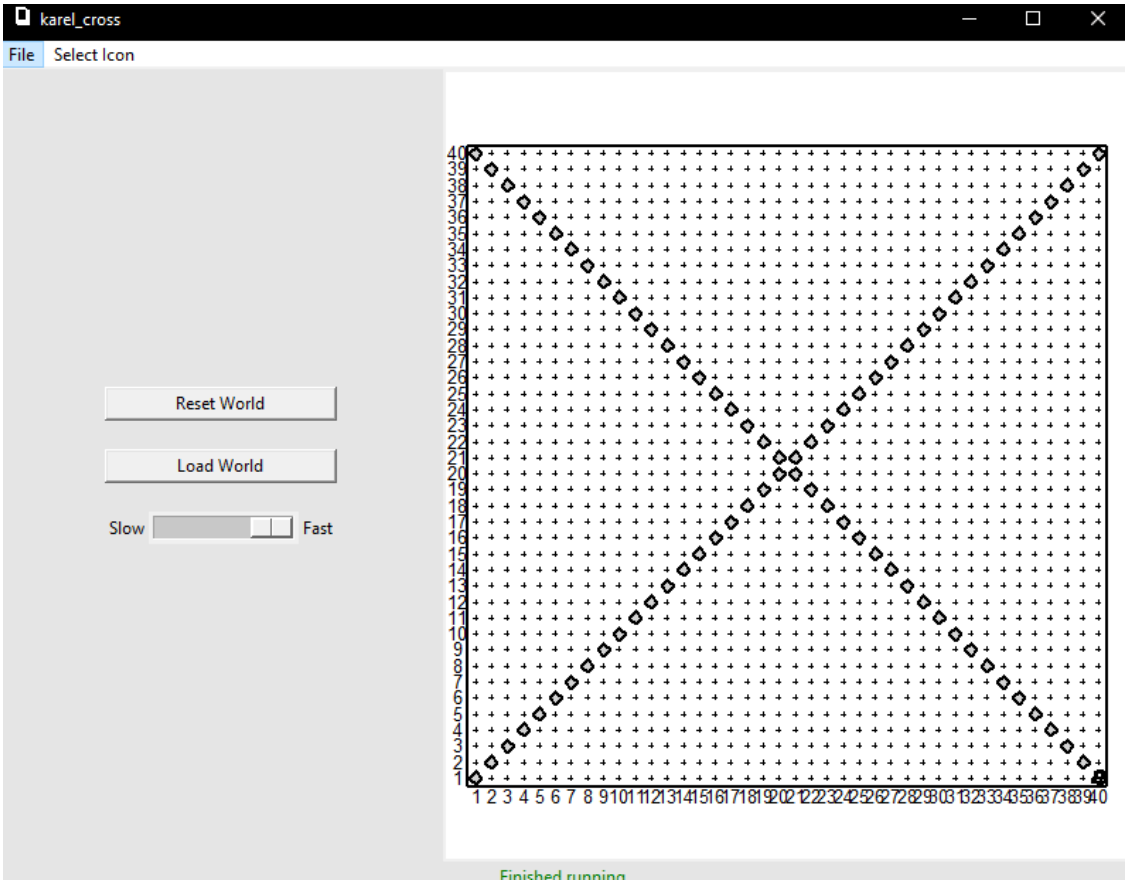


Explanation:

This program requires us to review again the Truth Table of AND, OR, and NOT function to understand NOR, NAND, and XOR, because it is just a sum or combination of 2 bitwise functions. After getting the base bitwise functions we apply the NOT function; NOT function's result is always the reverse of the result. Both NOR and NAND are applicable in those method. Lastly, XOR is a combination of the three bitwise. In executing this in the Python, we use the imported definitions/functions to evaluate and implement the rules of NOR, NAND, and XOR.

3. Familiarization Exercise 3 - Karel Cross





Explanation:

In this program, we utilize the lesson of function in Karel Bot. There is a step-by-step procedure on how we can operate and execute the code in our IDE. If the world size is Odd, then it will execute a Vertical / Horizontal cross; and if it is Even, it will execute a diagonal cross. The cross codes are already provided so we need to put them in the main function for them to execute. First is declaring variables to count the world size, after reading the program will now move and create the cross according to the world size.

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 5th Module of LBYCPA1 mainly focuses on the Functions and Module of Python. We revisited the lessons and knowledge of the past modules, and we used those codes to implement another program. We were introduced to a DSCRTMATH Lesson which is Recursion, and revisited the bitwise function to create XOR, NAND, and NOR. Lastly students were tasked to create a Karel Program that reads its environment and draw a cross.

Utilizing and understanding the functions is the key point of this Module; we understand how important it is to organize and maximize the reuse and redundancy of codes. I learned a lot of things in this module most especially the importance of functions, because all programmers use this to create a simple and to have a few lines of codes. I achieved everything in this module including debugging and making sure to have no Assertion Errors. My recommendations to students who are having a hard time in this module is that they should memorize and understand the all the syntax and its concept by heart. Typical difficulties I have encountered are merely logical flaws. This error is a learning opportunity for me and a piece of advice for students who will be taking this program in the near future. The final piece of advice is to continue practicing, as this builds a stronger basis for 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/

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

Familiarization Exercise 1:

```
def recursive(number):
    # This function satisfies the recurrence relation f(x) = 0.4*f(x-1) +
0.8*f(x-2)
    if type(number) is not int:
        raise ValueError("Input value must be an integer")
    if (number < 0):
        number = 0
    elif (number == 0 ):
        number = 2
    elif (number == 1):
        number = 4
    else:
        number = (0.4 * (recursive(number-1)) + 0.8 * (recursive(number-2)))

    return number
```

Familiarization Exercise 2:

```
import bitwisemodule as bwm

def bitwise_NOR(string1, string2):

#     result =
bwm.bitwise_OR(bwm.bitwise_NOT(string1),bwm.bitwise_NOT(string2))

    result = bwm.bitwise_NOT(bwm.bitwise_OR(string1,string2))
    return result

    # Do not forget the return statement for this function

def bitwise_NAND(string1, string2):
    # This function performs the logical bitwise NOR on two 8-character
strings

    result = bwm.bitwise_NOT(bwm.bitwise_AND(string1,string2))
    return result
# Do not forget the return statement for this function

def bitwise_XOR(string1, string2):
    # This function performs the logical bitwise XOR on two 8-character
```

```
strings

    result =
bwm.bitwise_OR(bwm.bitwise_AND(string1,bwm.bitwise_NOT(string2)),bwm.bitwise_
AND(bwm.bitwise_NOT(string1),string2))
    return result

    # Do not forget the return statement for this function
```

Familiarization Exercise 3:

```
%%writefile karel_cross.py

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 get_world_size():
    # This function will let Karel survey the world to determine its size. It
will return with the size of the world.
    wsize = int(1)
    east()
    while front_is_clear():
        forward()
        wsize += 1
    return wsize

def draw_diagonal_rightdown(nbeepers):
    # This function will generate a downward diagonal line of beepers. Karel
will move right then down as beepers are placed.
    # It will not put a beeper if a corner has already a beeper in it. Karel
will stop if blocked ahead.
```

```
    used_beepers = 0
    while not_facing_east():
        turn_left()
    for i in range(nbeepers):
        if no_beepers_present():
            put_beeper()
            used_beepers += 1
        if i < nbeepers - 1:
            move()
            while not_facing_south():
                turn_left()
            if front_is_blocked():
                break
            move()
        while not_facing_east():
            turn_left()
        if front_is_blocked():
            break
    if nbeepers > 0 and no_beepers_present():
        put_beeper()
        used_beepers += 1
    return used_beepers

def draw_diagonal_rightup(nbeepers):
    # This function will generate a upward diagonal line of beepers. Karel
will move right then up as beepers are placed.
    # It will not put a beeper if a corner has already a beeper in it. Karel
will stop if blocked ahead.

    used_beepers = 0
    while not_facing_east():
        turn_left()
    for i in range(nbeepers):
        if no_beepers_present():
            put_beeper()
            used_beepers += 1
        if i < nbeepers - 1:
            move()
            while not_facing_north():
                turn_left()
            if front_is_blocked():
                break
            move()
        while not_facing_east():
            turn_left()
        if front_is_blocked():
            break
    if nbeepers > 0 and no_beepers_present():
        put_beeper()
        used_beepers += 1
    return used_beepers

def draw_horizontal_right(nbeepers):
    # This function will generate a horizontal line of beepers. Karel will
move right as beepers are placed.
    # It will not put a beeper if a corner has already a beeper in it. Karel
```

```
will stop if blocked ahead.

    used_beepers = 0
    while not_facing_east():
        turn_left()
    for i in range(nbeepers):
        if no_beepers_present():
            put_beeper()
            used_beepers += 1
        if i < nbeepers - 1:
            move()
        if front_is_blocked():
            break
    if nbeepers > 0 and no_beepers_present():
        put_beeper()
        used_beepers += 1
    return used_beepers

def draw_vertical_down(nbeepers):
    # This function will generate a vertical line of beepers. Karel will move
down as beepers are placed.
    # It will not put a beeper if a corner has already a beeper in it. Karel
will stop if blocked ahead.

    used_beepers = 0
    while not_facing_south():
        turn_left()
    for i in range(nbeepers):
        if no_beepers_present():
            put_beeper()
            used_beepers += 1
        if i < nbeepers - 1:
            move()
        if front_is_blocked():
            break
    if nbeepers > 0 and no_beepers_present():
        put_beeper()
        used_beepers += 1
    return used_beepers

def main():
    # Step 0: Initialize variables
    center_coord_x, center_coord_y = int(), int()
    beepers_used = int()

    # # Step 1: Determine world size
    world_size = get_world_size() # you have to implement the function above

    # # Step 2: Check if the world size is odd or even
    isOdd = bool()
    if (world_size % 2) == 0:
        isOdd = False
    else:
        isOdd = True

    # # Step 3: Compute for the center coordinates of the world
    center_coord_x = world_size // 2
```

```
    center_coord_y = world_size // 2

    #
    # # Step 4: Position Karel and generate the beeper lines
    beepers_used = 0
    if isOdd:
        north()
        straight()
        west()
        for i in range (0, center_coord_x):
            forward()
        beepers_used += draw_vertical_down(world_size)
        west()
        straight()
        north()
        for i in range (0, center_coord_y):
            forward()
        beepers_used += draw_horizontal_right(world_size)
    else:
        west()
        straight()
        beepers_used += draw_diagonal_rightup(world_size)
        west()
        straight()
        beepers_used += draw_diagonal_rightdown(world_size)

    #
    # # Step 5: Print the center coordinates and the number of beepers used
    get_world_size()
    if isOdd:
        print("\nCenter Coordinates is ", center_coord_x + 1,",",
center_coord_y + 1)
    else:
        print("\nCenter Coordinates is ", center_coord_x + 0.5 ,",",
center_coord_y + 0.5)
    print("Beepers Used is ", beepers_used)
    print("\nThe World Size is ", world_size, "x", world_size)



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