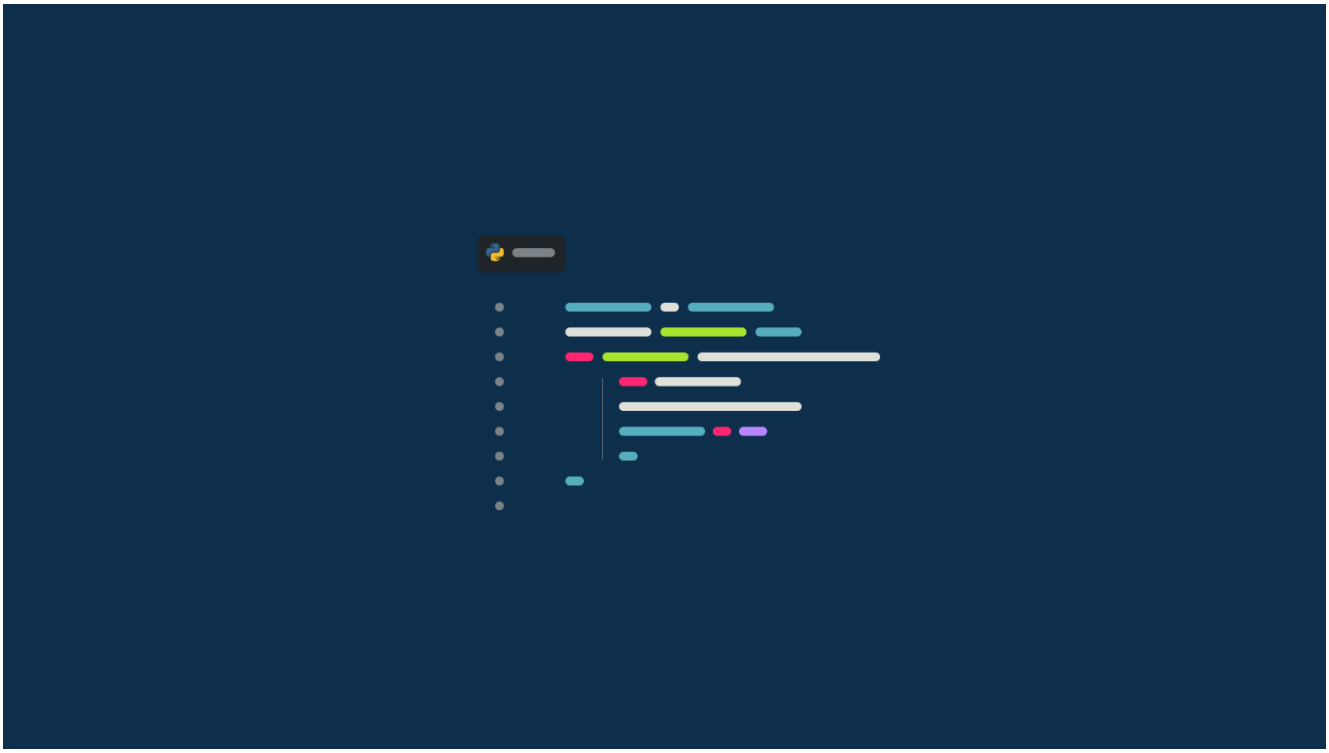
LBYCPA1

Programming Logic and Design Laboratory



Laboratory Module 8

Collection Array: Sets and Dictionaries

By

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

INTRODUCTION

The seventh module of this course will mainly focus on Collection Array, particularly Sets and Dictionaries. According from the notes of Module 7, Set is classified as an unordered collection that gets the unique elements, meaning it has no duplicate elements and does not repeat a certain element. Sets can support mathematical operations such as Union and Intersection which will be dealt in the problem sets later and differences. Then there is this another collection array called Dictionary. It is a mutable collection of many values, but in dictionaries its indexes have 2 pairs - Keys and Value. These indexes are separated with semicolons and these data types are flexible, example (Keys = string: Value = int). Whenever the word mapping is observant in the problem then dictionaries are the way to go, due to the fact that it is a type of mapping that can store data objects by key instead by the relative positions. The rule of thumb of every laboratory report is to provide opportunity for learning and provide a strong foundation in programming, molded by experience. Through this module, the students will reflect on how data analysis and organization will be a big role in programming. Despite the hindrances we face in our reality today, we utilize and appreciate the utilities that is provided by the world wide web to provide us students various information about programming. Through the process of programming alongside with the method of planning; algorithms, pseudocodes, and flowcharts that provides process on how the program executes. This laboratory report will be helpful for aspiring web developers who are studying Python right now.

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

(a) Objectives

1. To familiarize with the set and dictionary collection arrays
2. To know the different methods for sets and dictionaries
3. To use sets and dictionaries for effective storage of data array
4. To utilize sets and dictionaries in solving computational problems

What are the materials used for this module?

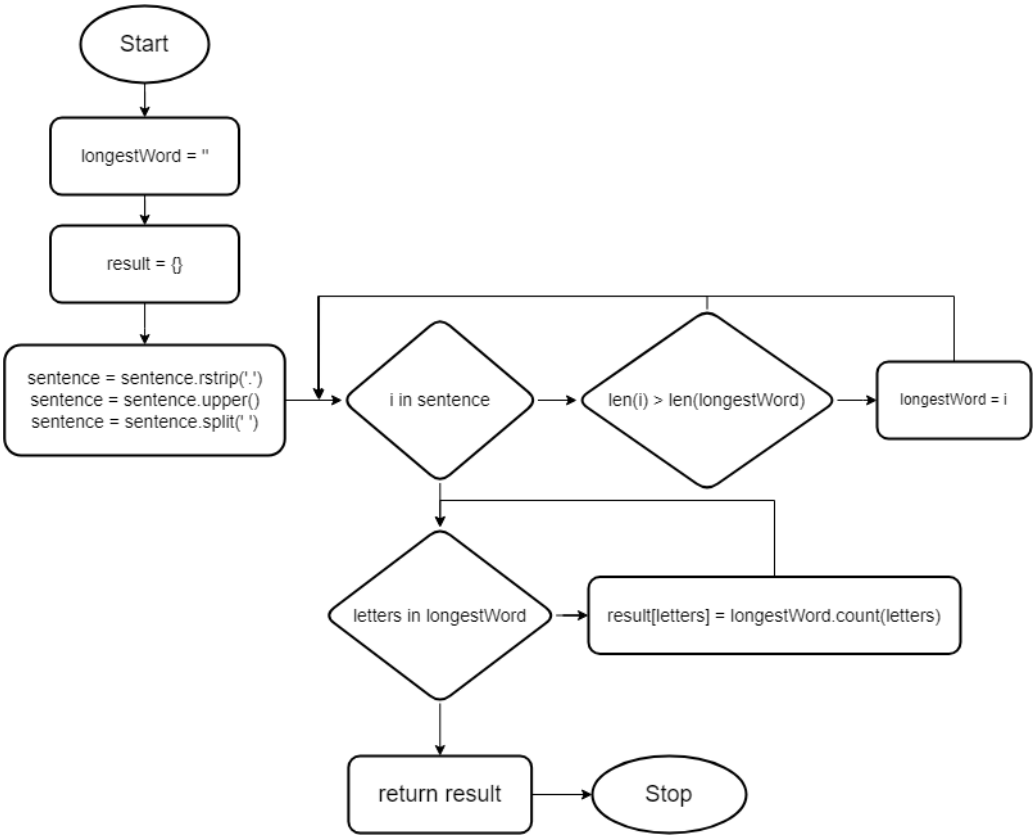
(b) Materials and Tools

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

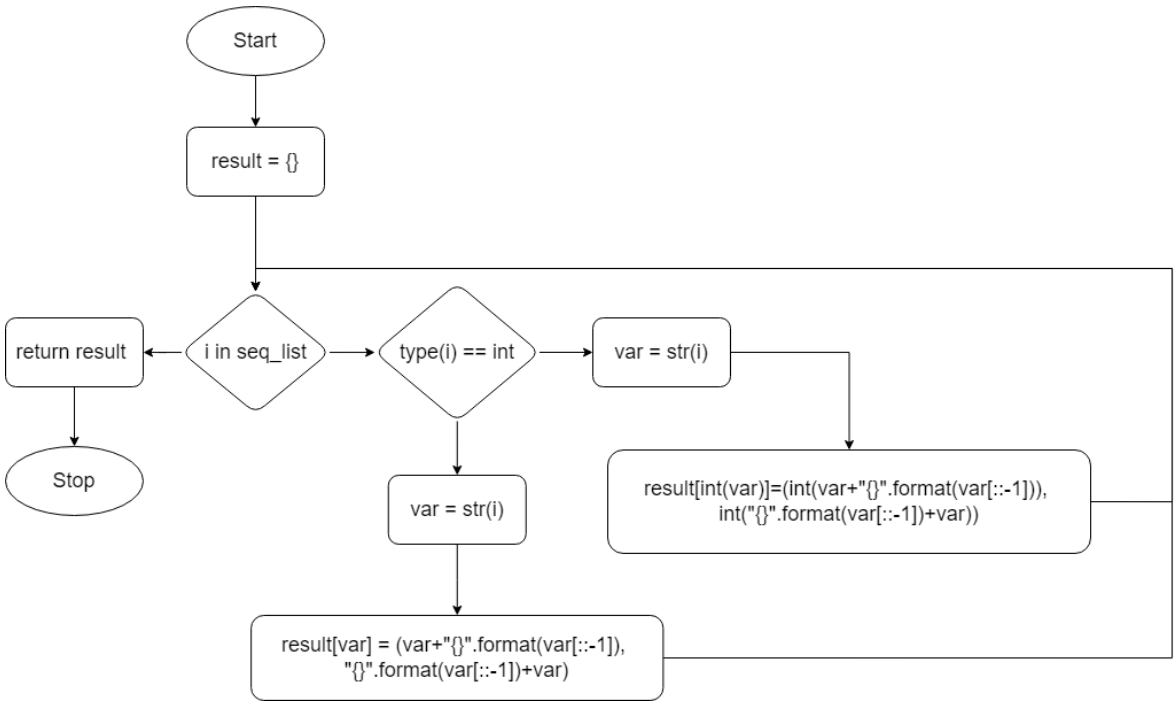
PROCEDURES (*Individual*) / EXPERIMENTAL PLAN

In every experimental plan of every Laboratory Report we always place our initial plans before performing executing our codes, in programming we always use Algorithms and flowcharts, as a visual representation of how we can show the separate steps of each process in a sequential manner. The overview.

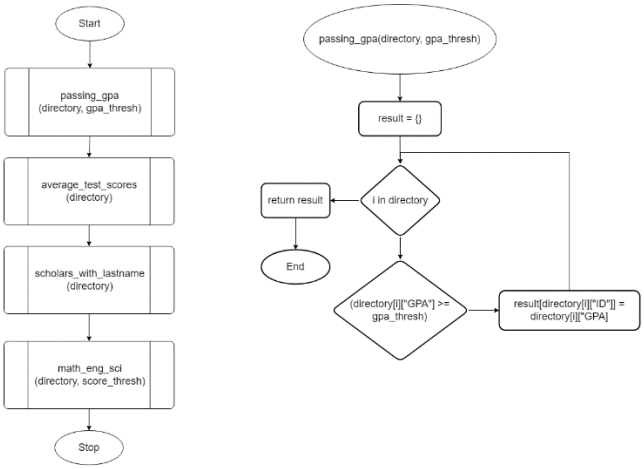
1. In Familiarization Exercise 1, its required output should be the longest word according to the sentence, first is to remove all punctuation marks and setting the sentence to upper, to make it all capitalized. Lastly, I need to do is to create a function to get the longest word and get its length value of the dictionary.

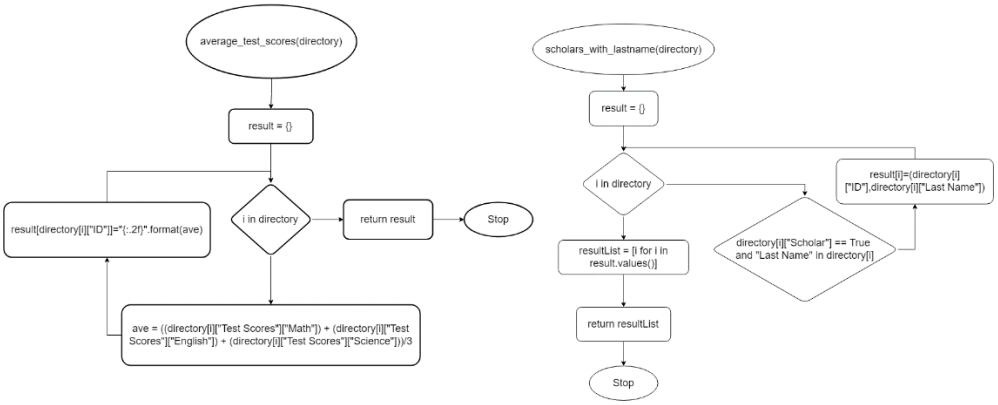


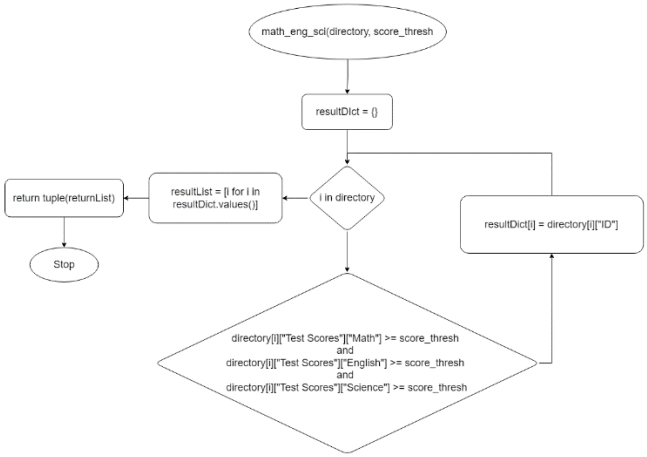
2. In Familiarization Exercise 2, the procedure in exercise 1 is somehow similar because we'll be creating an empty dictionary and creating a loop that creates a format in which the word or the numbers are in a reversed position. In which the output should be in a dictionary and its key is the input and its values is the palindrome of a word or number.



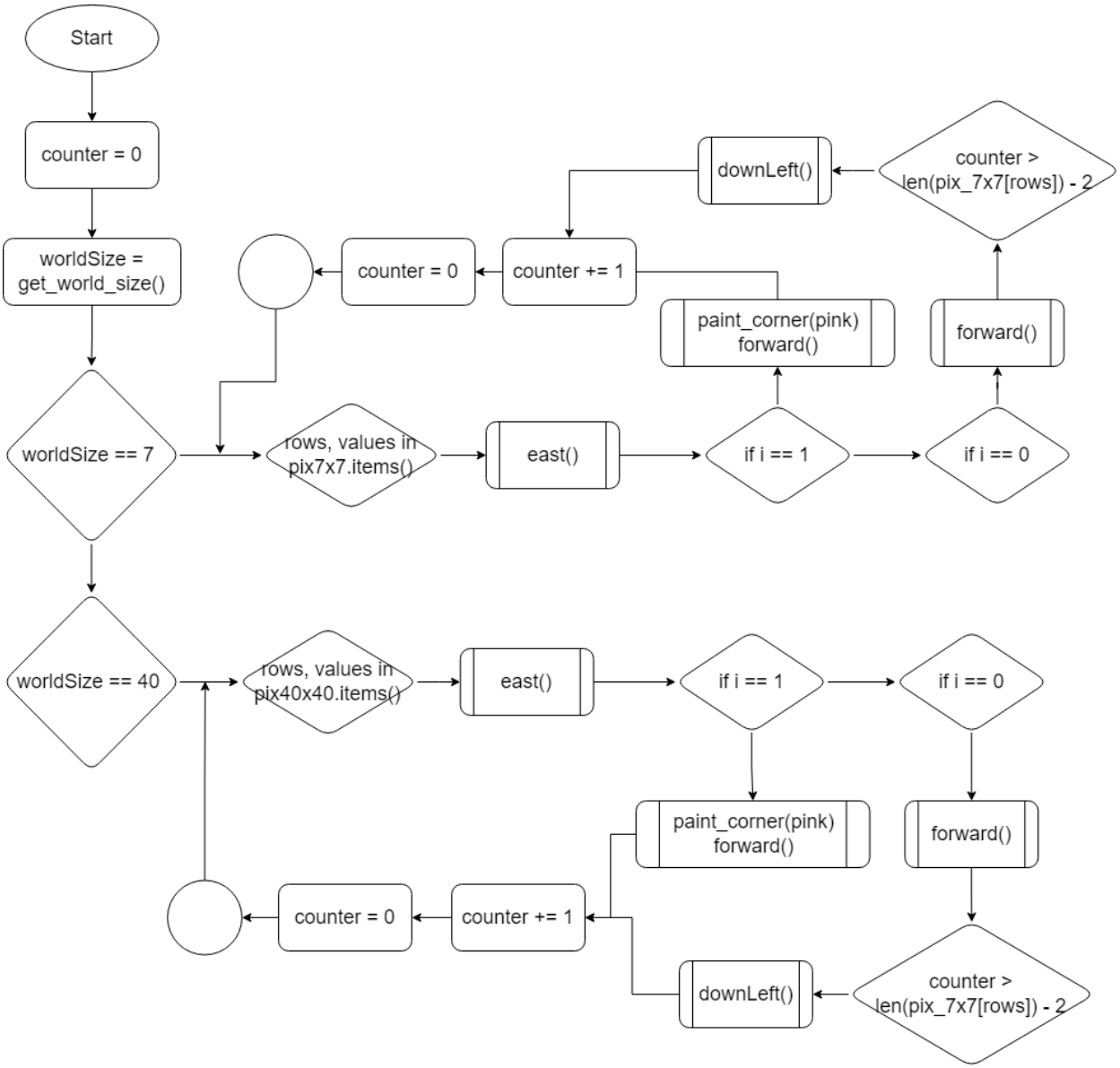
3. In Familiarization Exercise 3, there is a given student directory, and we are tasked to develop a function to get a specific key and value of each index. I dissect each directory according to the required output. Like the first function which is the `passing_gpa`, the program verifies each index in the according to the output needed, verify that the GPA is greater or equal than the threshold and so on.







4. In Familiarization Exercise 4, we're going to revisit Karel once again and creating a script that translate a dictionary of tuples. After translating the dictionary, I created a script in which Karel will first read the size of the environment and proceeds on creating a condition that if the world size is 7 or 40, It will get the items of each dictionary. In every 1, in the dictionary the bot will paint the corner while the bot will ignore the value of 0.



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 Result:

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

    longestWord = ''
    result = {}
    sentence = sentence.rstrip('.')
    sentence = sentence.upper()
    word = sentence.split(' ')
    for i in sentence:
        if len(i) > len(longestWord):
            longestWord = i
    for letters in longestWord:
        result[letters] = longestWord.count(letters)
    return result
```

```
# DO NOT MODIFY, THIS IS UNIT TESTING! [1 pt]
from nose.tools import assert_equal

assert_equal(wordStatistics("The ethics of xenotransplantation are relatively unworrying."),
             {'X': 1, 'N': 4, 'O': 2, 'I': 1, 'P': 1, 'S': 1, 'R': 1, 'E': 1, 'L': 1, 'A': 3, 'T': 3})
assert_equal(wordStatistics("Her pulchritudinous likeness adorns everything from bestselling books to ski boards."),
             {'N': 1, 'I': 2, 'O': 1, 'P': 1, 'H': 1, 'D': 1, 'S': 1, 'R': 1, 'C': 1, 'T': 1, 'L': 1, 'U': 3})
assert_equal(wordStatistics("Floccinaucinihilipilification"),
             {'N': 3, 'I': 9, 'O': 2, 'P': 1, 'H': 1, 'F': 2, 'C': 4, 'T': 1, 'L': 3, 'A': 2, 'U': 1})
assert_equal(wordStatistics("Pseudopseudohypoparathyroidism needs no treatment, but genetic counselling is recommended."),
             {'O': 4, 'I': 2, 'P': 4, 'D': 3, 'H': 2, 'M': 1, 'S': 3, 'R': 2, 'E': 2, 'Y': 2, 'T': 1, 'A': 2, 'U': 2})
assert_equal(wordStatistics("Pneumonoultramicroscopicsilicovolcanoconiosis"),
             {'N': 4, 'O': 9, 'I': 6, 'P': 2, 'V': 1, 'M': 2, 'S': 4, 'R': 2, 'E': 1, 'C': 6, 'T': 1, 'L': 3, 'A': 2, 'U': 2})
```

Explanation:

The program's required output would be a dictionary that gets the longest word and its length to be counted in each letter. First the declaration of an empty string and an empty dictionary, after declaring is the I removed the punctuations using the rstrip('.'), I specifically used rstrip because in every sentences it has a period indicated, and changed its case into an upper-cased since it is the expected output. After changing cases, we split the spaces then following a loop that if the length of I is greater than the length of the longest word, until this condition satisfies, the longest word will be altered by the word present. After checking the longest word, it will now loop to check the dissected word and count the repeating letters. Lastly we return the result.

2. Familiarization Exercise 2 Result:

```
result = {}
for i in seq_list:
    if type(i) == int:
        var = str(i)
        result[int(var)]=(int(var+"{}".format(var[::-1])), int("{}".format(var[::-1])+var))
    else:
        var = str(i)
        result[var] = (var+"{}".format(var[::-1]), "{}".format(var[::-1])+var)
return result
```

```
# DO NOT MODIFY, THIS IS UNIT TESTING! [1 pt]
from nose.tools import assert_equal

assert_equal(toPalindromes(["cover", 123, "hi"]),
             {'cover': ('coverrevoc', 'revoccover'), 123: (123321, 321123), 'hi': ('hiih', 'ihhi')})
assert_equal(toPalindromes([235, "car"]),
             {235: (235532, 532235), 'car': ('carrac', 'raccar')})
assert_equal(toPalindromes(["and", "or", "not", 314]),
             {'and': ('anddna', 'dnaand'), 'or': ('orro', 'roor'), 'not': ('notton', 'tonnot'), 314: (314413, 413314)})
```

Explanation:

The program's required output would be a palindrome, which reads the same backwards and forward. It can be either a word or a number. First creating an empty dictionary and followed by a loop to verify the data type of the given decimal, whether it is a float or an integer. Using string comprehensions, to make the palindrome by adding the given by the reversed sequence of string, after adding, it will be converted again to an integer since it is the required output from the program (int -> string -> int), then the program will proceed to add the given palindrome.

3. Familiarization Exercise 3 Result:

```
def passing_gpa(directory, gpa_thresh):
    if type(directory) is not dict:
        raise TypeError("First argument must be a dictionary")
    if type(gpa_thresh) is not float:
        raise TypeError("Second argument must be a float")

    result = {}
    for i in directory:
        if directory[i]["GPA"] >= gpa_thresh :
            result[directory[i]["ID"]]=directory[i]["GPA"]
    return result

def average_test_scores(directory):
    if type(directory) is not dict:
        raise TypeError("Argument must be a dictionary")
    result = {}
    for i in directory:
        ave = ((directory[i]["Test Scores"]["Math"]) + (directory[i]["Test Scores"]["English"]) + (directory[i]["Test Scores"]["Science"]
        result[directory[i]["ID"]]="{:.2f}".format(ave)
    return result

def scholars_with_lastname(directory):
    if type(directory) is not dict:
        raise TypeError("Argument must be a dictionary")

    result = {}
    for i in directory:
        if directory[i]["Scholar"] == True and "Last Name" in directory[i]:
            result[i]=(directory[i]["ID"],directory[i]["Last Name"])
    resultList = [i for i in result.values()]
    return resultList

def math_eng_sci(directory, score_thresh):
    if type(directory) is not dict:
        raise TypeError("First argument must be a dictionary")
    if type(score_thresh) is not float:
        raise TypeError("Second argument must be a float")
    # YOUR CODE HERE
    resultDict = {}
    for i in directory:
        if directory[i]["Test Scores"]["Math"] >= score_thresh and directory[i]["Test Scores"]["English"] >= score_thresh and directory[i
            resultDict[i] = directory[i]["ID"]
    resultList = [i for i in resultDict.values()]
    return tuple(resultList)
```

```
# DO NOT MODIFY, THIS IS UNIT TESTING! [1 pt]
from nose.tools import assert_equal
from nose.tools import assert_is_instance

assert_equal(passing_gpa(student_directory, 1.5),
             {'12198': 3.4, '12199': 1.5, '12200': 3.9, '12202': 4.0, '12204': 3.7, '12205': 2.8})
assert_equal(passing_gpa(student_directory, 3.0),
             {'12198': 3.4, '12200': 3.9, '12202': 4.0, '12204': 3.7})
assert_equal(passing_gpa(student_directory, 3.5),
             {'12200': 3.9, '12202': 4.0, '12204': 3.7})
assert_is_instance(passing_gpa(student_directory, 1.0), dict)
assert_is_instance(passing_gpa(student_directory, 1.0)['12198'], float)
```

```
# DO NOT MODIFY, THIS IS UNIT TESTING! [1 pt]
from nose.tools import assert_equal
from nose.tools import assert_is_instance

assert_equal(average_test_scores(student_directory),
             {'12198': '90.67', '12199': '79.33', '12200': '89.33', '12201': '84.00',
              '12202': '89.00', '12203': '90.67', '12204': '89.33', '12205': '82.67'})
assert_equal(len(average_test_scores(student_directory)['12198']), 5)
assert_is_instance(average_test_scores(student_directory)['12200'], str)
```

```
# DO NOT MODIFY, THIS IS UNIT TESTING! [1 pt]
from nose.tools import assert_equal

assert_equal(scholars_with_lastname(student_directory),
             [('12198', 'Santa Cruz'), ('12201', 'Delfino'), ('12204', 'Legarda'), ('12205', 'Ramos')])
assert_is_instance(scholars_with_lastname(student_directory), list)
assert_is_instance(scholars_with_lastname(student_directory)[0], tuple)
```

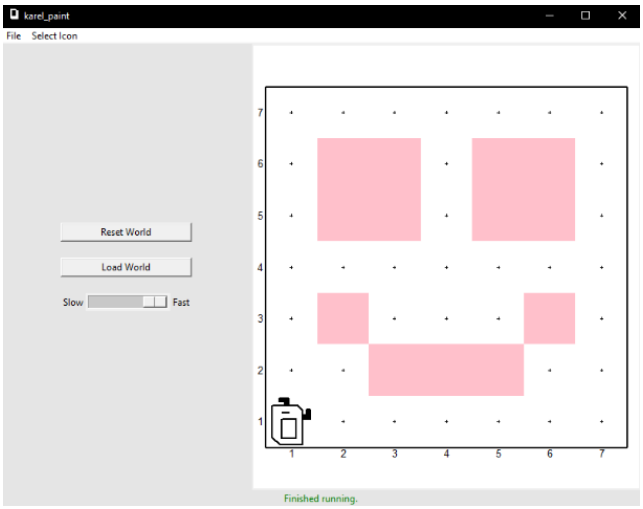
```
# DO NOT MODIFY, THIS IS UNIT TESTING! [1 pt]
from nose.tools import assert_equal
from nose.tools import assert_is_instance

assert_equal(math_eng_sci(student_directory, 80.0),
             ('12198', '12200', '12201', '12202', '12203', '12204', '12205'))
assert_equal(math_eng_sci(student_directory, 85.0),
             ('12198', '12200', '12202', '12203'))
assert_equal(math_eng_sci(student_directory, 87.0),
             ('12198', '12202'))
assert_is_instance(math_eng_sci(student_directory, 75.0), tuple)
assert_is_instance(math_eng_sci(student_directory, 75.0)[0], str)
```

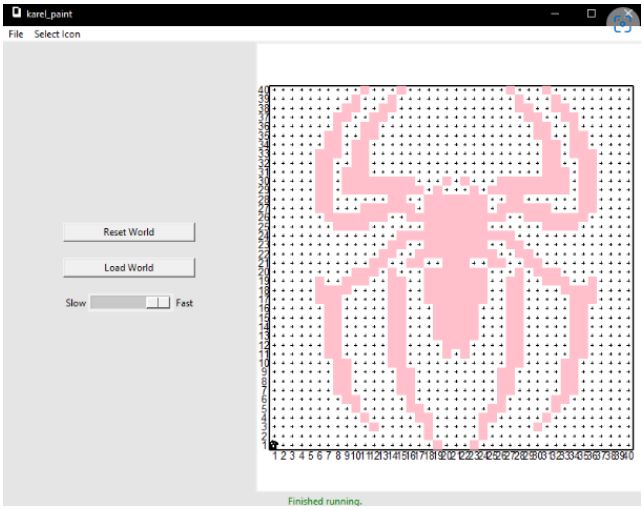
Explanation:

This program provides a directory of information about students and develop a function to verify each function with their respective outputs.

a. Passing_gpa – As usual, creating an empty dictionary and followed by a loop to check the directory, and proceeds on creating a conditional statement in which if the directory specifically ‘GPA’ is greater than or equals than the threshold. If the grades are satisfied according to the condition, its ID number and GPA will be added to the empty dictionary to be used as a placeholder of the data. Lastly return the dictionary variable.
b. Average_test_scores – Same as the creation of an empty dictionary and the verification of checking the directory, then gets the average value of the student’s data, getting 3 data divided by 3 and setting a format that specifies a float format (.2f).
c. Scholars_with_lastname – Same as the creation of an empty dictionary and the verification of checking the directory, proceeds on checking if there is a value of ‘Scholar’ in the directory and if it is true and the conditions are met then it will get the Last Name and the Id value in the directory. Lastly appending and creating a list of tuples of the required values.
d. Math_eng_sci – Same as the creation of an empty dictionary and the verification of checking the directory, then verifies the students with all Math, English and Science scores greater than or equal to the threshold. Lastly appending and creating a list of tuples of the required values (same in the 3rd function).

4. Familiarization Exercise 4 Result:





```
counter = 0
worldSize = get_world_size()
print(worldSize)

if worldSize == 7:
   for rows, values in pix_7x7.items():
        for i in values:
            east()
            if i == 1:
                paint_corner(PINK)
                forward()
            elif i == 0:
                forward()
                if counter > len(pix_7x7[rows]) - 2:
                    downLeft()
            counter+=1
        counter = 0

elif worldSize == 40:
    for rows, values in pix_40x40.items():
        # print("==",len(pix_40x40[rows]))
        for i in values:
            east()
            if i == 1:
                paint_corner(PINK)
                forward()
            elif i == 0:
                forward()
                if counter > len(pix_40x40[rows]) - 2:
                    downLeft()
            print("--",counter)
            counter+=1
        counter = 0
```

Explanation:

This part of the activity will let us revisit Karel and we'll see a new feature called ***paint_corner***, in which the bot will paint the block on where the bot is standing there are many colors of choice and I ended up choosing the pink for this output to be unique. First is I copy pasted the functions I created in the previous activities with karel that automatically call out a specific direction and move, after calling the function is that I created a counter variable and the world size in which the bot looks north and goes straight forward and that counts the size of the Karel's environment. After identifying the value of the size, I proceeded on creating a conditional statement that if the world size is 7 or 40 it will get the items of the specific dictionary, creating a loop to get the value of each key and value inside the dictionary and each 1 inside the value it will paint a corner then moves forward, else if it is 0 then it goes forward. Getting the length of the rows – 2, after this gets satisfied then the function downleft() will execute (downleft = looks down, move, then looks left and go straight forward), I printed the counter to see the value, so I can visualize when the bot should do the function of downleft to reset the loop and proceeds on drawing a certain figure provided in

the dictionary.

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 8th Module of LBYCPA1 mainly focuses on Collection Arrays specifically sets and dictionaries, this module taught me the basic things about these Collection arrays. This module serves as a training ground for us aspiring software engineers, because it helped me to improvise and utilize the functions of sets and dictionaries, as well as using other type of array like lists and tuples. We were re-introduced with Palindromes and Karel Bot once again, but in this module, we discovered that we could paint a block in the environment of Karel bot. This module played a big role in my journey on learning about programming, because it helped me understand and utilize the storage of data arrays using sets and dictionaries effectively. This module also helped me identify what are the difference between all the given collection array that can be used in Python. I learned various things in this module, and I also learned here dimensional arrays since they are observable in the problems 1-3, and dimensional arrays is observable in other language. I don't have many downfalls in this activity, but one challenged I faced is time; time is gold and LBYCPA1 is just one of the few subjects I take in this term. Procrastination is an unhealthy way to deal with our academics, a student will face its consequences if they do. I recommend to people who are stressed with their workloads is to create a timetable of what will you do in a specific time and day, so you can track your goals. As a programmer Time management will be playing a big role because it will help us in the future to do our duties and responsibilities in time.

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*)

1. W3school (n.d.). Retrieved from: https://www.w3schools.com/python/python_dictionaries.asp
2. Karel (n.d.) Extra Features. Retrieved from: https://compedu.stanford.edu/karel-reader/docs/python/en/chapter9.html

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

1. Familization Exercise 1:

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

    longestWord = ''
    result = {}
    sentence = sentence.rstrip('.')
    sentence = sentence.upper()
    word = sentence.split(' ')
    for i in sentence:
        if len(i) > len(longestWord):
            longestWord = i
    for letters in longestWord:
        result[letters] = longestWord.count(letters)
    return result
```

2. Familization Exercise 2:

```
def toPalindromes(seq_list):
    if type(seq_list) is not list:
        raise TypeError("Input not a valid list")

    result = {}
    for i in seq_list:
        if type(i) == int:
            var = str(i)
            result[int(var)]=(int(var+"{}".format(var[::-1])),
int("{}".format(var[::-1])+var))
```

```
        else:
            var = str(i)
            result[var] = (var+"{}".format(var[::-1]), "{}".format(var[::-
1])+var)
    return result
```

3. Familization Exercise 3:

```
student_directory = {
    "Student 1" : {
        "ID" : "12198",
        "Last Name" : "Santa Cruz",
        "Gender" : 'Male',
        "GPA" : 3.4,
        "Scholar" : True,
        "Test Scores" : {"Math" : 90, "English" : 95, "Science" : 87}
    },
    "Student 2" : {
        "ID" : "12199",
        "Gender" : 'Male',
        "GPA" : 1.5,
        "Scholar" : True,
        "Test Scores" : {"Math" : 80, "English" : 75, "Science": 83}
    },
    "Student 3" : {
        "ID" : "12200",
        "Gender" : 'Female',
        "GPA" : 3.9,
        "Scholar" : False,
        "Test Scores" : {"Math" : 88, "English" : 85, "Science": 95}
    },
    "Student 4" : {
        "ID" : "12201",
        "Last Name" : "Delfino",
        "Gender" : 'Female',
        "GPA" : 1.4,
        "Scholar" : True,
        "Test Scores" : {"Math" : 82, "English" : 82, "Science" : 88}
    },
    "Student 5" : {
        "ID" : "12202",
        "Gender" : 'Female',
        "GPA" : 4.0,
        "Scholar" : True,
        "Test Scores" : {"Math" : 87, "English" : 91, "Science": 89}
    },
    "Student 6" : {
        "ID" : "12203",
        "Gender" : 'Male',
        "GPA" : 1.0,
        "Scholar" : False,
        "Test Scores" : {"Math" : 86, "English" : 92, "Science" : 94}
    },
```

```
    "Student 7" : {
        "ID" : "12204",
        "Last Name" : "Legarda",
        "Gender" : 'Female',
        "GPA" : 3.7,
        "Scholar" : True,
        "Test Scores" : {"Math" : 90, "English" : 95, "Science": 83}
    },
    "Student 8" : {
        "ID" : "12205",
        "Last Name" : "Ramos",
        "Gender" : 'Male',
        "GPA" : 2.8,
        "Scholar" : True,
        "Test Scores" : {"Math" : 80, "English" : 82, "Science" : 86}
    }
}

def passing_gpa(directory, gpa_thresh):
    if type(directory) is not dict:
        raise TypeError("First argument must be a dictionary")
    if type(gpa_thresh) is not float:
        raise TypeError("Second argument must be a float")

    result = {}
    for i in directory:
        if directory[i]["GPA"] >= gpa_thresh :
            result[directory[i]["ID"]]=directory[i]["GPA"]
    return result

def average_test_scores(directory):
    if type(directory) is not dict:
        raise TypeError("Argument must be a dictionary")
    result = {}
    for i in directory:
        ave = ((directory[i]["Test Scores"]["Math"]) + (directory[i]["Test
Scores"]["English"]) + (directory[i]["Test Scores"]["Science"]))/3
        result[directory[i]["ID"]]="{:.2f}".format(ave)
    return result

def scholars_with_lastname(directory):
    if type(directory) is not dict:
        raise TypeError("Argument must be a dictionary")

    result = {}
    for i in directory:
        if directory[i]["Scholar"] == True and "Last Name" in directory[i]:
            result[i]=(directory[i]["ID"],directory[i]["Last Name"])
    resultList = [i for i in result.values()]
    return resultList

def math_eng_sci(directory, score_thresh):
    if type(directory) is not dict:
```

```
        raise TypeError("First argument must be a dictionary")
    if type(score_thresh) is not float:
        raise TypeError("Second argument must be a float")
    # YOUR CODE HERE
    resultDict = {}
    for i in directory:
        if directory[i]["Test Scores"]["Math"] >= score_thresh and
directory[i]["Test Scores"]["English"] >= score_thresh and directory[i]["Test
Scores"]["Science"] >= score_thresh:
            resultDict[i] = directory[i]["ID"]
    resultList = [i for i in resultDict.values()]
    return tuple(resultList)
```

4. Familization Exercise 4:

```
from
stanfordkarel
import *

                def forward():
                    if front_is_clear():
                        move()
                def straight():
                    while front_is_clear():
                        forward()
                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 downLeft():
                    south()
                    forward()
                    west()
                    straight()
                def get_world_size():
                    wsize = int(1)
                    north()
                    while front_is_clear():
```

```
        forward()
        wsize += 1
    return wsize

def main():
    # Dictionary data for a 7x7 world
    pix_7x7 = {
        'Row 1': (0,0,0,0,0,0,0),
        'Row 2': (0,1,1,0,1,1,0),
        'Row 3': (0,1,1,0,1,1,0),
        'Row 4': (0,0,0,0,0,0,0),
        'Row 5': (0,1,0,0,0,1,0),
        'Row 6': (0,0,1,1,1,0,0),
        'Row 7': (0,0,0,0,0,0,0)
    }

    # Dictionary data for a 40x40 world
    pix_40x40 = {
        'Row 1':  (0,0,0,0,0,0,0,0,0,0,1,0,0,0,1,0,0,0,0,0,0,0,0,0,0,0,1,0,0,0,1,0,0,0,0,0,0,0,0,0),
        'Row 2':  (0,0,0,0,0,0,0,0,0,1,0,0,0,1,0,0,0,0,0,0,0,0,0,0,0,0,1,0,0,0,1,0,0,0,0,0,0,0,0,0),
        'Row 3':  (0,0,0,0,0,0,0,0,1,1,0,0,1,1,0,0,0,0,0,0,0,0,0,0,0,0,1,1,0,0,1,1,0,0,0,0,0,0,0,0),
        'Row 4':  (0,0,0,0,0,0,0,1,1,0,0,1,1,0,0,0,0,0,0,0,0,0,0,0,0,0,1,1,0,0,1,1,0,0,0,0,0,0,0,0),
        'Row 5':  (0,0,0,0,0,0,0,1,0,0,1,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1,1,0,0,1,0,0,0,0,0,0,0,0,0),
        'Row 6':  (0,0,0,0,0,0,1,1,0,0,1,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1,1,0,0,1,1,0,0,0,0,0,0,0,0),
        'Row 7':  (0,0,0,0,0,0,1,1,0,1,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1,1,0,1,1,0,0,0,0,0,0,0,0,0),
        'Row 8':  (0,0,0,0,0,1,1,1,0,1,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1,1,0,1,1,1,0,0,0,0,0,0,0,0),
        'Row 9':  (0,0,0,0,0,1,1,0,0,1,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1,1,0,0,1,1,0,0,0,0,0,0,0,0),
        'Row 10': (0,0,0,0,0,1,1,0,1,1,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1,1,1,0,1,1,0,0,0,0,0,0,0,0),
        'Row 11': (0,0,0,0,0,1,1,0,1,1,1,1,1,1,1,1,1,0,0,0,1,0,1,0,0,0,1,1,1,1,1,1,1,1,0,1,1,0,0,0,0,0),
        'Row 12': (0,0,0,0,0,1,1,0,1,1,1,1,1,1,1,1,1,1,0,1,0,0,0,1,0,1,1,1,1,1,1,1,1,1,0,1,1,0,0,0,0,0),
        'Row 13': (0,0,0,0,0,1,1,0,0,0,0,0,0,1,1,0,0,1,1,1,1,1,1,1,0,0,1,1,0,0,0,0,0,0,1,1,0,0,0,0,0,0),
        'Row 14': (0,0,0,0,0,1,1,0,0,0,1,1,1,1,1,1,1,0,1,1,1,1,1,1,1,0,1,1,1,1,1,1,1,0,0,0,1,1,0,0,0,0),
        'Row 15': (0,0,0,0,0,0,1,1,1,1,1,1,1,1,0,0,1,1,1,1,1,1,1,1,1,1,1,0,0,1,1,1,1,1,1,1,1,0,0,0,0,0),
        'Row 16': (0,0,0,0,0,0,0,1,1,1,1,1,0,0,0,0,0,0,0,1,1,1,1,1,1,1,1,0,0,0,0,0,0,1,1,1,1,0,0,0,0,0),
        'Row 17': (0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1,1,1,1,1,1,1,1,1,1,1,1,0,0,0,0,0,0,0,0,0,0,0,0,0),
        'Row 18': (0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1,1,0,0,1,1,1,1,1,1,1,1,0,0,1,1,0,0,0,0,0,0,0,0,0,0,0),
        'Row 19': (0,0,0,0,0,0,0,0,0,0,0,0,0,1,1,0,0,1,1,1,1,1,1,1,1,1,1,0,0,1,1,0,0,0,0,0,0,0,0,0,0,0),
        'Row 20': (0,0,0,0,0,0,0,0,0,0,0,0,1,1,0,0,1,1,0,0,1,1,1,1,0,0,1,1,0,0,1,1,0,0,0,0,0,0,0,0,0,0),
        'Row 21': (0,0,0,0,0,0,0,0,0,0,1,1,1,0,1,0,0,0,1,1,1,1,1,1,1,1,0,0,0,1,0,1,1,1,0,0,0,0,0,0,0,0),
        'Row 22': (0,0,0,0,0,1,0,0,1,1,1,0,0,1,1,0,0,1,1,1,1,1,1,1,1,0,0,1,1,0,0,1,1,1,1,0,0,1,0,0,0,0),
        'Row 23': (0,0,0,0,0,1,1,1,1,1,0,0,0,0,1,1,0,0,1,1,1,1,1,1,1,1,0,0,1,1,0,0,0,0,0,1,1,1,1,1,0,0,0,0),
        'Row 24': (0,0,0,0,0,1,1,1,1,0,0,0,0,0,1,1,0,0,1,1,1,1,1,1,1,1,0,0,1,1,0,0,0,0,0,1,1,1,1,0,0,0,0,0),
        'Row 25': (0,0,0,0,0,0,1,1,0,0,0,0,0,1,1,0,0,0,1,1,1,1,1,1,0,0,0,1,1,0,0,0,0,0,1,1,0,0,0,0,0,0),
```

```
    'Row 26': (0,0,0,0,0,0,1,1,0,0,0,0,0,1,1,0,0,0,1,1,1,1,1,0,0,0,1,1,0,0,0,0,0,1,1,0,0,0,0,0),
    'Row 27': (0,0,0,0,0,0,1,1,0,0,0,0,0,1,1,0,0,0,1,1,1,1,1,0,0,0,1,1,0,0,0,0,0,1,1,0,0,0,0,0),
    'Row 28': (0,0,0,0,0,0,1,1,0,0,0,0,0,1,1,0,0,0,1,1,1,1,1,0,0,0,1,1,0,0,0,0,0,1,1,0,0,0,0,0),
    'Row 29': (0,0,0,0,0,0,1,1,0,0,0,0,0,1,1,0,0,0,0,1,1,1,1,0,0,0,0,1,1,0,0,0,0,0,1,1,0,0,0,0,0),
    'Row 30': (0,0,0,0,0,0,1,1,0,0,0,0,0,1,1,0,0,0,0,1,0,1,0,0,0,0,1,1,0,0,0,0,0,1,1,0,0,0,0,0),
    'Row 31': (0,0,0,0,0,0,0,1,1,0,0,0,0,1,1,0,0,0,0,0,0,0,0,0,0,0,1,1,0,0,0,0,1,1,0,0,0,0,0,0),
    'Row 32': (0,0,0,0,0,0,0,1,1,0,0,0,0,0,1,1,0,0,0,0,0,0,0,0,0,1,1,0,0,0,0,0,1,1,0,0,0,0,0,0),
    'Row 33': (0,0,0,0,0,0,0,1,1,0,0,0,0,0,1,1,0,0,0,0,0,0,0,0,0,1,1,0,0,0,0,0,1,1,0,0,0,0,0,0),
    'Row 34': (0,0,0,0,0,0,0,1,1,0,0,0,0,0,1,1,0,0,0,0,0,0,0,0,0,1,1,0,0,0,0,0,1,1,0,0,0,0,0,0),
    'Row 35': (0,0,0,0,0,0,0,0,1,1,0,0,0,0,0,1,0,0,0,0,0,0,0,0,0,1,0,0,0,0,0,1,1,0,0,0,0,0,0,0),
    'Row 36': (0,0,0,0,0,0,0,0,0,1,0,0,0,0,0,1,1,0,0,0,0,0,0,0,1,1,0,0,0,0,0,1,0,0,0,0,0,0,0,0),
    'Row 37': (0,0,0,0,0,0,0,0,0,0,1,0,0,0,0,0,1,1,0,0,0,0,0,1,1,0,0,0,0,0,1,0,0,0,0,0,0,0,0,0),
    'Row 38': (0,0,0,0,0,0,0,0,0,0,0,1,0,0,0,0,0,1,0,0,0,0,0,1,0,0,0,0,0,1,0,0,0,0,0,0,0,0,0,0),
    'Row 39': (0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1,0,0,0,0,0,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0),
    'Row 40': (0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1,0,0,0,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0)
}
counter = 0
worldSize = get_world_size()
print(worldSize)

if worldSize == 7:
  for rows, values in pix_7x7.items():
      for i in values:
          east()
          if i == 1:
              paint_corner(PINK)
              forward()
          elif i == 0:
              forward()
              if counter > len(pix_7x7[rows]) - 2:
                  downLeft()
          counter+=1
      counter = 0

elif worldSize == 40:
    for rows, values in pix_40x40.items():
        # print("==",len(pix_40x40[rows]))
        for i in values:
            east()
            if i == 1:
                paint_corner(PINK)
                forward()
            elif i == 0:
```

```
                forward()
                if counter > len(pix_40x40[rows]) - 2:
                    downLeft()
            print("--",counter)
            counter+=1
        counter = 0

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