```
#-----Statement of Authorship----------------------------------------#
#
#  This is an individual assessment task for QUT's teaching unit
#  IFB104, "Building IT Systems", Semester 1, 2023.  By submitting
#  this code I agree that it represents my own work.  I am aware of
#  the University rule that a student must not act in a manner
#  which constitutes academic dishonesty as stated and explained
#  in QUT's Manual of Policies and Procedures, Section C/5.3
#  "Academic Integrity" and Section E/2.1 "Student Code of Conduct".
#
#  Put your student number here as an integer and your name as a
#  character string:
#
student_number = 10583122
student_name   = 'Sharyn Tauro'
#
#  NB: Files submitted without a completed copy of this statement
#  will not be marked.  All files submitted will be subjected to
#  software plagiarism analysis using the MoSS system
#  (http://theory.stanford.edu/~aiken/moss/).
#
#--------------------------------------------------------------------#



#-----Assessment Task 1 Description----------------------------------#
#
#  This assessment task tests your skills at processing large data
#  sets, creating reusable code and following instructions to display
#  a complex visual image.  The incomplete Python program below is
#  missing a crucial function.  You are required to complete this
#  function so that when the program runs it fills a grid with various
#  symbols, using data stored in a list to determine which symbols to
#  draw and where.  See the online video instructions for
#  full details.
#
#  Note that this assessable assignment is in multiple parts,
#  simulating incremental release of instructions by a paying
#  "client".  This single template file will be used for all parts
#  and you will submit your final solution as this single Python 3
#  file only, whether or not you complete all requirements for the
#  assignment.
#
#  This file relies on other Python modules but all of your code
#  must appear in this file only.  You may not change any of the code
#  in the other modules and you should not submit the other modules
#  with your solution.  The markers will use their own copies of the
#  other modules to test your code, so your solution will not work
#  if it relies on changes made to any other files.
#
#--------------------------------------------------------------------#



#-----Preamble-------------------------------------------------------#
#
# This section imports necessary functions used to execute your code.
# You must NOT change any of the code in this section, and you may
# NOT import any non-standard Python modules that need to be
```

```
# downloaded and installed separately.
#

# Import standard Python modules needed to complete this assignment.
# You should not need to use any other modules for your solution.
# In particular, your solution must NOT rely on any non-standard
# Python modules that need to be downloaded and installed separately,
# because the markers will not have access to such modules.
from turtle import *
from math import *
from random import *
from sys import exit as abort
from os.path import isfile

# Confirm that the student has declared their authorship
if not isinstance(student_number, int):
    print('\nUnable to run: No student number supplied',
          '(must be an integer), aborting!\n')
    abort()
if not isinstance(student_name, str):
    print('\nUnable to run: No student name supplied',
          '(must be a character string), aborting!\n')
    abort()

# Import the functions for setting up the drawing canvas
if isfile('assignment_1_config.py'):
    print('\nConfiguration module found ...\n')
    from assignment_1_config import *
else:
    print("\nCannot find file 'assignment_1_config.py', aborting!\n")
    abort()

# Define the function for generating data sets, using the
# imported raw data generation function if available, but
# otherwise creating a dummy function that just returns an
# empty list
if isfile('assignment_1_data_source.py'):
    print('Data generation module found ...\n')
    from assignment_1_data_source import raw_data
    def data_set(new_seed = randint(0, 99999)):
        print('Using random number seed', new_seed, '...\n')
        seed(new_seed) # set the seed
        return raw_data() # return the random data set
else:
    print('No data generation module available ...\n')
    def data_set(dummy_parameter = None):
        return []

#
#-----------------------------------------------------------------#



#-----Student's Solution-------------------------------------------#
#
#  Complete the assignment by replacing the dummy function below with
#  your own function and any other functions needed to support it.
#  All of your solution code must appear in this section.  Do NOT put
#  any of your code in any other sections and do NOT change any of
```

```
#  the provided code except as allowed by the comments in the next
#  section.
#

# All of your code goes in, or is called from, this function.
# Make sure that your code does NOT call function data_set (or
# raw_data) because it's already called in the main program below.
def visualise_data(rename_me):


#-----My Solution----------------------------------------------#

# Task 1
    '''
    Draw six owls, three on each side of the grid, with each owl facing a different
direction.
    Write the direction of the owl below the hexagon that the owl is in.
    The owls should be drawn in the following order: North, North East, South East,
South, South West, North West.
    '''


### Step 1: Make constants for the directions of the owls so that values remain
unchanged throughout the program.

    OBJECT_DIRECTION = {
        'NORTH': 'North',
        'NORTH_EAST': 'North east',
        'SOUTH_EAST': 'South east',
        'SOUTH': 'South',
        'SOUTH_WEST': 'South west',
        'NORTH_WEST': 'North west',
    }


### Step 2: Function to draw a hexagon at the given co-ordinates.

    ''' @param x_1 and y_1 takes the co-ordinates on the canvas where the turtle
needs to be positioned before starting to draw the hexagon
    '''

    def drawHexagon(x_1:int,y_1:int):
        goto(x_1,y_1)
        setheading(120)
        penup()
        pencolor('black')
        forward(60)
        setheading(0)
        pendown()
        pencolor("grey")
        fillcolor("BlanchedAlmond")
        width(2)
        pendown()
        begin_fill()
        for i in range(6):
            forward(60)
            right(60)
        end_fill()
        penup()
```

Step 3: Function to position the turtle to draw the owl.

```
    '''
    ANGLE_1 is the angle at which the turtle needs to be positioned at the centre
of the hexagon
    After this the turtle will move forward 60 to position itself at the vertex of
the hexagon from which it needs to start drawing the owl
    ANGLE_2 is the angle at which the turtle needs to be turned towards to start to
draw the owl
    '''

    def setDirection(direction):
        NORTH = {
            'ANGLE_1': 120,
            'ANGLE_2': 270,
        }
        NORTH_EAST = {
            'ANGLE_1': 60,
            'ANGLE_2': 210,
        }
        SOUTH_EAST = {
            'ANGLE_1': 0,
            'ANGLE_2': 150,
        }
        SOUTH = {
            'ANGLE_1': 300,
            'ANGLE_2': 90,
        }
        SOUTH_WEST = {
            'ANGLE_1': 240,
            'ANGLE_2': 30,
        }
        NORTH_WEST = {
            'ANGLE_1': 180,
            'ANGLE_2': 330,
        }
        if (direction == OBJECT_DIRECTION['NORTH']):
            setheading(NORTH['ANGLE_1'])
            forward(60)
            setheading(NORTH['ANGLE_2'])
        elif (direction == OBJECT_DIRECTION['NORTH_EAST']):
            setheading(NORTH_EAST['ANGLE_1'])
            forward(60)
            setheading(NORTH_EAST['ANGLE_2'])
        elif (direction == OBJECT_DIRECTION['SOUTH_EAST']):
            setheading(SOUTH_EAST['ANGLE_1'])
            forward(60)
            setheading(SOUTH_EAST['ANGLE_2'])
        elif (direction == OBJECT_DIRECTION['SOUTH']):
            setheading(SOUTH['ANGLE_1'])
            forward(60)
            setheading(SOUTH['ANGLE_2'])
        elif (direction == OBJECT_DIRECTION['SOUTH_WEST']):
            setheading(SOUTH_WEST['ANGLE_1'])
            forward(60)
            setheading(SOUTH_WEST['ANGLE_2'])
        elif (direction == OBJECT_DIRECTION['NORTH_WEST']):
```

```
            setheading(NORTH_WEST['ANGLE_1'])
            forward(60)
            setheading(NORTH_WEST['ANGLE_2'])
        else:
            print('Invalid direction')


### Step 4: Function to draw the different parts of the Owl which include oblong
body, eyes (golden, white, black, white circles), beak and feet.
### Step 4a: Function to draw the oblong body of the Owl.

    def drawOblong_brown():
        forward(67)
        pendown
        pensize(1)
        pencolor("brown4")
        fillcolor("brown4")
        begin_fill()
        circle(30,180)
        forward(30)
        circle(30,180)
        forward(40)
        end_fill()


### Step 4b: Function to draw the Golden Circles of the Owl eyes

    def drawCircle_golden():
        penup()
        backward(30)
        left(50)
        pencolor('DarkGoldenrod1')
        fillcolor('DarkGoldenrod1')
        begin_fill()
        circle(22)
        end_fill()
        left(33)
        forward(47)
        begin_fill()
        circle(22)
        end_fill()


### Step 4c: Function to draw the Large White Circles of the Owl eyes

    def drawCircle_white_1():
        forward(12)
        left(90)
        forward(22)
        pendown()
        pencolor('white')
        fillcolor('white')
        begin_fill()
        circle(15)
        end_fill()
        penup()
        left(64)
        forward(52.5)
        left(90)
```

```
        forward(22)
        pendown()
        begin_fill()
        circle(15)
        end_fill()
        penup()


### Step 4d: Function to draw the Black Circles of the Owl eyes

    def drawCircle_black():
        left(45)
        forward(10)
        pendown()
        pencolor('black')
        fillcolor('black')
        begin_fill()
        circle(10)
        end_fill()
        penup()
        left(65)
        forward(38)
        pendown()
        begin_fill()
        circle(10)
        end_fill()
        penup()


### Step 4e: Function to draw the Small White Circles of the Owl eyes

    def drawCircle_white_2():
        left(175)
        forward(27)
        pendown()
        pencolor('white')
        fillcolor('white')
        begin_fill()
        circle(2)
        end_fill()
        penup()
        right(170)
        forward(27)
        pendown()
        begin_fill()
        circle(2)
        end_fill()
        penup()


### Step 4f: Function to draw the Orange Triangles of the Owl legs

    def drawTriangle_orange_1():
        right(75)
        forward(52)
        pendown()
        pencolor('orange')
        fillcolor('orange')
        begin_fill()
```

```
        forward(15)
        right(120)
        forward(20)
        right(130)
        forward(20)
        end_fill()
        left(170)
        penup()
        forward(52)
        pendown()
        begin_fill()
        right(120)
        forward(18)
        right(120)
        forward(20)
        right(122)
        forward(21)
        end_fill()


### Step 4g: Function to draw the Orange Triangle of the Owl nose

    def drawTriangle_orange_2():
        right(120)
        penup()
        forward(51)
        right(72)
        forward(25)
        pendown()
        begin_fill()
        right(120)
        forward(18)
        right(120)
        forward(20)
        right(122)
        forward(21)
        end_fill()


### Step 5: Function to draw the text under hexagon.
    def text_under_hexagon(x,y,text):
        up()
        ht()
        goto(x,y)
        st()
        down()
        width(1)
        pencolor("grey")
        write(text,align = 'center', font = ('Arial', 23, 'normal'))
        penup()


### Step 6: Function to draw the owl inside the hexagon.
### we use config:any as we want config to be of any data type, in this case it is
a dictionary.

    def drawOwl(config:any):
        pos_1 = (config['x_1'])                 # x coordinate of the hexagon
        pos_2 = (config['y_1'])                 # y coordinate of the hexagon
```

```
        pos_3 = (config['x_2'])                  # x coordinate of the text
        pos_4 = (config['y_2'])                  # y coordinate of the text
        drawHexagon(pos_1,pos_2)
        right(60)
        forward(60)
        setDirection(config['OBJECT_DIRECTION']) # direction of the owl
        drawOblong_brown()
        drawCircle_golden()
        drawCircle_white_1()
        drawCircle_black()
        drawCircle_white_2()
        drawTriangle_orange_1()
        drawTriangle_orange_2()
        goto(pos_1,pos_2)
        setheading(0)

        # The below code is to check if the owl is being drawn inside the grid or
not. If it is not, then the text will be displayed.
        if (pos_1 <= -420 or pos_1 >= 420) or (pos_2 <= -300 or pos_2 >= 300):
            text_under_hexagon(pos_3,pos_4,config['text'])
        else:
            print("Invalid position for text")
        penup()

        # The below code is to reset the turtle to the position that the owl was
drawn in. This will help with Task 2.
        # For an owl drawn in the North direction, the turtle will be reset to the
North direction and so on.
        setheading(config['TURTLE_DIRECTION'])
        goto(pos_1,pos_2)


### Step 7: Function to draw 6 Owls in 6 different directions as required in the
task.

### Drawing 6 Owls in 6 different directions

    def initial_display():
        North: any = {"x_1": -550,"y_1": 210, "OBJECT_DIRECTION":
OBJECT_DIRECTION['NORTH'],"x_2": -550,"y_2": 110,"text":"North",
"TURTLE_DIRECTION": 90}
        drawOwl(North)

        North_East: any = {"x_1":-550,"y_1": 0, "OBJECT_DIRECTION":
OBJECT_DIRECTION['NORTH_EAST'],"x_2": -550,"y_2": -100, "text":"North East",
"TURTLE_DIRECTION": 60}
        drawOwl(North_East)

        South_East: any = {"x_1":-550,"y_1":-210, "OBJECT_DIRECTION":
OBJECT_DIRECTION['SOUTH_EAST'],"x_2": -550,"y_2": -310, "text":"South East",
"TURTLE_DIRECTION": 300}
        drawOwl(South_East)

        South: any = {"x_1":550,"y_1":210, "OBJECT_DIRECTION":
OBJECT_DIRECTION['SOUTH'],"x_2": 550,"y_2": 110, "text":"South",
"TURTLE_DIRECTION": 270}
        drawOwl(South)

        South_West: any = {"x_1":550,"y_1":0, "OBJECT_DIRECTION":
```

```
OBJECT_DIRECTION['SOUTH_WEST'], "x_2": 550,"y_2": -100,"text":"South West",
"TURTLE_DIRECTION": 180}
        drawOwl(South_West)

        North_West: any = {"x_1":550,"y_1":-210, "OBJECT_DIRECTION":
OBJECT_DIRECTION['NORTH_WEST'], "x_2": 550,"y_2": -310,"text":"North West",
"TURTLE_DIRECTION": 120}
        drawOwl(North_West)



### Step 8: Calling the function initial_display() to draw the 6 owls in 6
different directions.

    initial_display()


# Task 2: Function to draw the owls in the grid in directions specified by the
randon moves list .
    '''
    The first owl will always be drawn at (x=0,y=0). The number of moves to make
will be supplied by the energy level and the next position and
    direction to move in will be supplied by the random moves list. The direction
of the owl will be updated after each move.
    '''

### Step 1: Make constants for the moves to make that is being supplied by the
random moves list.

    OBJECT_MOVE_TO_MAKE = {
        'MF': 'Move forward',
        'MTR': 'Move & turn right',
        'MTL': 'Move & turn left'}


### Step 2: Write the function to draw owls in all 6 directions at any given co-
ordinates for x,y in the hexagonal grid.
# x_1 and y_1 are the co-ordinates of the hexagon and x_2 and y_2 are the co-
ordinates of the text under the hexagon.
# As these owls are being drawn in the grid, the text under the hexagon does not
need to be written hence.

    def draw_owl_based_on_direction(x1: int, y1: int, direction: str):
        if direction == OBJECT_DIRECTION['NORTH']:
                North: any = {"x_1": x1,"y_1": y1, "OBJECT_DIRECTION":
OBJECT_DIRECTION['NORTH'],"x_2": 0,"y_2": 0,"text":"North", "TURTLE_DIRECTION": 90}
                drawOwl(North)
        elif direction == OBJECT_DIRECTION['NORTH_EAST']:
            North_East: any = {"x_1": x1,"y_1": y1, "OBJECT_DIRECTION":
OBJECT_DIRECTION['NORTH_EAST'],"x_2": 0,"y_2": 0,"text":"North",
"TURTLE_DIRECTION": 30}
            drawOwl(North_East)
        elif direction == OBJECT_DIRECTION['SOUTH_EAST']:
            South_East: any = {"x_1": x1,"y_1": y1, "OBJECT_DIRECTION":
OBJECT_DIRECTION['SOUTH_EAST'],"x_2": 0,"y_2": 0,"text":"North",
"TURTLE_DIRECTION": 330}
            drawOwl(South_East)
        elif direction == OBJECT_DIRECTION['SOUTH']:
            South: any = {"x_1": x1,"y_1": y1, "OBJECT_DIRECTION":
```

```
OBJECT_DIRECTION['SOUTH'],"x_2": 0,"y_2": 0,"text":"North", "TURTLE_DIRECTION":
270}
            drawOwl(South)
        elif direction == OBJECT_DIRECTION['SOUTH_WEST']:
            South_West: any = {"x_1": x1,"y_1": y1, "OBJECT_DIRECTION":
OBJECT_DIRECTION['SOUTH_WEST'],"x_2": 0,"y_2": 0,"text":"North",
"TURTLE_DIRECTION": 210}
            drawOwl(South_West)
        elif direction == OBJECT_DIRECTION['NORTH_WEST']:
            North_West: any = {"x_1": x1,"y_1": y1, "OBJECT_DIRECTION":
OBJECT_DIRECTION['NORTH_WEST'],"x_2": 0,"y_2": 0,"text":"North",
"TURTLE_DIRECTION": 150}
            drawOwl(North_West)
        else:
                print("Invalid direction")



### Step 3: Write the functions to provide the next co-ordinates (x,y) the owl
needs to be drawn in. This function is for the move forward part of
#  the random moves list.
    '''
To find the height of an equilateral triangle with all its sides equal we need to
draw an altitude from one vertex to the midpoint of
the opposite side. This divides the equilateral triangle into two 30-60-90 right
triangles. The altitude will be the height (h) of the
equilateral triangle. In a 30-60-90 triangle, the sides are in the ratio 1 : √3 :
2. Since the side length of the equilateral triangle
is 60, the hypotenuse will be equal to the side length (60). The side opposite the
60-degree angle (height) will be half of the side length
(30) times √3.
Therefore
h = 0.5 * side_length * √3
This comes to rounded number 52 (51.999) and since we have to move the height
distance of two triangles the forward distance equals 104.
    '''
    def calculate_next_position(x:int, y:int, angle:int):
        # distance to move forward calculated based on the height of the
equilateral triangle
        distance = 104
        # angle at which the turtle needs to move forward. This will be the same as
the direction of the previously drawn owl.
        # round it as the answer is a float with fractional parts resulting in
error in the co-ordinates for further calculations.
        angle_radians = radians(angle)
        new_x = round(x + distance * cos(angle_radians))
        new_y = round(y + distance * sin(angle_radians))
        return (new_x, new_y)


    # calculate the next position coordinates (X,Y) based on the current position
coordinates.
    def get_next_position_coordinates(x:int, y:int, direction:str):
        if direction == OBJECT_DIRECTION['NORTH']:
            return calculate_next_position(x, y, 90)
        elif direction == OBJECT_DIRECTION['NORTH_EAST']:
            return calculate_next_position(x, y, 30)
        elif direction == OBJECT_DIRECTION['SOUTH_EAST']:
            return calculate_next_position(x, y, 330)
```

```
        elif direction == OBJECT_DIRECTION['SOUTH']:
            return calculate_next_position(x, y, 270)
        elif direction == OBJECT_DIRECTION['SOUTH_WEST']:
            return calculate_next_position(x, y, 210)
        elif direction == OBJECT_DIRECTION['NORTH_WEST']:
            return calculate_next_position(x, y, 150)
        else:
            print("Invalid direction")


### Step 4: Write the function to provide the next direction the owl needs to be
drawn in. This function is for turn right or left part of
#  the random moves list.
    '''
    In random moves list, only the first list has the direction the owl needs to be
drawn in.
    The next direction is calculated based on the current direction and the move to
make provided by the random moves list.
    Based on the current direction the owl is in and the move it needs to make
supplied by the values in the list generated by random moves,
    the next direction is calculated.
    '''
    def get_next_direction(current_direction:str, move_to_make:str):
        if current_direction == OBJECT_DIRECTION['NORTH']:
            if move_to_make == OBJECT_MOVE_TO_MAKE['MF']:
                return OBJECT_DIRECTION['NORTH']
            elif move_to_make == OBJECT_MOVE_TO_MAKE['MTR']:
                return OBJECT_DIRECTION['NORTH_EAST']
            elif move_to_make == OBJECT_MOVE_TO_MAKE['MTL']:
                return OBJECT_DIRECTION['NORTH_WEST']
            else:
                print("Invalid direction")
        elif current_direction == OBJECT_DIRECTION['NORTH_EAST']:
            if move_to_make == OBJECT_MOVE_TO_MAKE['MF']:
                return OBJECT_DIRECTION['NORTH_EAST']
            elif move_to_make == OBJECT_MOVE_TO_MAKE['MTR']:
                return OBJECT_DIRECTION['SOUTH_EAST']
            elif move_to_make == OBJECT_MOVE_TO_MAKE['MTL']:
                return OBJECT_DIRECTION['NORTH']
            else:
                print("Invalid direction")
        elif current_direction == OBJECT_DIRECTION['SOUTH_EAST']:
            if move_to_make == OBJECT_MOVE_TO_MAKE['MF']:
                return OBJECT_DIRECTION['SOUTH_EAST']
            elif move_to_make == OBJECT_MOVE_TO_MAKE['MTR']:
                return OBJECT_DIRECTION['SOUTH']
            elif move_to_make == OBJECT_MOVE_TO_MAKE['MTL']:
                return OBJECT_DIRECTION['NORTH_EAST']
            else:
                print("Invalid direction")
        elif current_direction == OBJECT_DIRECTION['SOUTH']:
            if move_to_make == OBJECT_MOVE_TO_MAKE['MF']:
                return OBJECT_DIRECTION['SOUTH']
            elif move_to_make == OBJECT_MOVE_TO_MAKE['MTR']:
                return OBJECT_DIRECTION['SOUTH_WEST']
            elif move_to_make == OBJECT_MOVE_TO_MAKE['MTL']:
                return OBJECT_DIRECTION['SOUTH_EAST']
            else:
                print("Invalid direction")
```

```
        elif current_direction == OBJECT_DIRECTION['SOUTH_WEST']:
            if move_to_make == OBJECT_MOVE_TO_MAKE['MF']:
                return OBJECT_DIRECTION['SOUTH_WEST']
            elif move_to_make == OBJECT_MOVE_TO_MAKE['MTR']:
                return OBJECT_DIRECTION['NORTH_WEST']
            elif move_to_make == OBJECT_MOVE_TO_MAKE['MTL']:
                return OBJECT_DIRECTION['SOUTH']
            else:
                print("Invalid direction")
        elif current_direction == OBJECT_DIRECTION['NORTH_WEST']:
            if move_to_make == OBJECT_MOVE_TO_MAKE['MF']:
                return OBJECT_DIRECTION['NORTH_WEST']
            elif move_to_make == OBJECT_MOVE_TO_MAKE['MTR']:
                return OBJECT_DIRECTION['NORTH']
            elif move_to_make == OBJECT_MOVE_TO_MAKE['MTL']:
                return OBJECT_DIRECTION['SOUTH_WEST']
            else:
                print("Invalid direction")
        else:
            print("Invalid direction")


### Step 5: Write the function for displaying text above the grid for the different
conditions
    def display_message(message):
     goto(0, 300)
     write(message, align='center', font=('Arial', 23, 'normal'))


### Step 6: Write the function to draw the owl at the position and in the direction
calculated in the previous steps.
    def moves_to_visualise(data_set: list[list]):
        # for the first element of the random moves list, the x and y coordinates
are 0 as we are always starting at the centre of the grid
        # the 'counter' is 0 and 'energy level' will be provided in the 0th element
of the random moves list
        # the 'initial direction' is the direction in which the owl needs to be
drawn and this will be provided in the 0th element of the
        #       random moves list
        # 'move to make' is the move the turtle needs to make and this will be
provided from the 1st element onwards of the random moves list
        #      (move and turn right or move and turn left))
        # 'next direction' is the direction the turtle needs to be in after the
move is made and this will be calculated based on the current
        #       direction using the function get_next_direction
        # the 1st seven lines we are making the variables
        x = 0
        y = 0
        counter = 0
        energy_consumed = 0
        energy_level = 0
        new_energy_level = 0
        initial_direction = None
        move_to_make = None
        next_direction = None
        for i in range(len(data_set)):
            print(f"i is {i} data_set[{i}] is {data_set[i]}")
            if (i == 0):
                x = 0
```

```
                y = 0
                counter = data_set[i][0]
                energy_level = data_set[i][1]
                initial_direction = data_set[i][2]
                next_direction = initial_direction

                print(f"counter: {counter} move_to_make: {move_to_make}
next_direction: {next_direction}")

                draw_owl_based_on_direction(x,y,next_direction)

# The below is to account for the cases where energy level is 0 right at the start
and the owl is exhausted after the initialization step.
                if energy_level == 0:
                    display_message(f'The Owl was exhausted after move {counter}')
                    break

            else:
                counter = data_set[i][0]
                move_to_make = data_set[i][1]
                (x,y) = get_next_position_coordinates(x,y,next_direction)
                print(f"x = {x} y = {y}")

# The below is to account for the cases where the next owl co-oridnates are outside
the grid and the owl has flown away.
                if (x <= -420 or x >= 420) or ((y <= -300 or y >= 300) or (y ==
260.0 or y == -260.0)):
                    display_message(f'Owl has flown away in move {counter}')
                    break
                else:
                    next_direction = get_next_direction(next_direction,
move_to_make)
                    draw_owl_based_on_direction(x,y,next_direction)

# The below is to account for the cases where the owl has reached the special co-
ordinates for home cells and the owl has reached home.
                    target_coordinates = [(270,52), (-90, 156), (-360, -104)]
                    if (x,y) in target_coordinates:
                        display_message(f'Owl has reached home in move {counter}!')
                        break
                    else:
# The below is to account for all the other cases where the owl is still on the
grid and has not reached home and eventually exhausted when energy
# is consumed in successive steps till it reaches 0 and the owl is exhausted.
                          if energy_level == counter:
                            display_message(f'The Owl was exhausted after move
{counter}')
                            break
                print(f"counter: {counter} move_to_make: {move_to_make}
next_direction: {next_direction}")




    moves_to_visualise(rename_me)
```

```
#---------------------------------------------------------------------#
#---------------------------------------------------------------------#



#-----Main Program to Run Student's Solution-------------------------#
#
# This main program configures the drawing canvas, calls the student's
# function and closes the canvas.  Do NOT change any of this code
# except as allowed by the comments below.  Do NOT put any of
# your solution code in this section.
#

# Configure the drawing canvas
# ***** You can change the background and line colours, and choose
# ***** whether or not to draw the grid and other elements, by
# ***** providing arguments to this function call
create_drawing_canvas(canvas_title = "Wise Owl The More You See The Less You Talk",
                      bg_colour = 'light grey',
                      line_colour = 'slate grey',
                      draw_grid = True,
                      write_instructions = False)

# Call the student's function to process the data set
# ***** While developing your program you can call the
# ***** "data_set" function with a fixed seed below for the
# ***** random number generator, but your final solution must
# ***** work with "data_set()" as the function call,
# ***** i.e., for any random data set that can be returned by
# ***** the function when called with no seed
visualise_data(data_set()) # <-- no argument for "data_set" when assessed

# Exit gracefully
# ***** Do not change this function call
release_drawing_canvas(student_name)

#
#---------------------------------------------------------------------#
```