CHARLES DARWIN UNIVERSITY

HIT137 SOFTWARE NOW

ASSESSMENT 02

Group Name: DAN/EXT 41

Group Members:

Roshan Neupane - 395086
Navodya Piumanthi- 396303
Aashish Sharma - 396419
Nayan Babu Sapkota - 393069

Introduction

This assignment has a total of three questions and the codes associated with them are attached within the submission along with the GitHub link of the project and the output files. This documentation provides a thorough explanation of the codes and the outputs. We have also attached screenshots of the codes and outputs in the explanation, so that the viewer can easily relate to them.

Common Code Structure

All codes are consistent and follow functional programming concepts. The entry code is the main() function. The if __name__ == "__main__" construct as shown in figure 1 makes sure the script is the top level module and allows the code to run only when the script is executed (Breuss, 2024).

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

figure 1: if __name__ == "__main__" construct

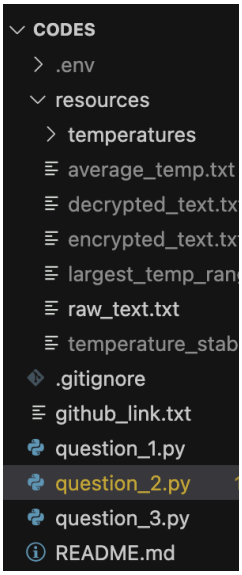

figure 2: Project tree

The above figure 2 shows the project tree. We have three python files question_1.py, question_2.py and question_3.py that correspond to the three consecutive questions. The .env folder is for the python environment and requirements.txt has the list of all the external libraries installed for this project. The **resources** folder is where we are keeping the output generated from the codes along with the raw text given for question 1 and temperatures given for question 2.

Question 1

This question is about encryption and here, we are using a shifting algorithm, a version of Caesar Cipher (Wikipedia contributors, 2025). We are trying to encrypt a text, then decrypt it with accuracy.

The main concern of this question is that it applies differently to lowercase and uppercase letters, which is acceptable. However, the algorithm applied to the lower and upper half (i.e. a-m, n-z or A-M, N-Z) are not mutually exclusive, since rotating on the whole system can't let us know which half it belonged to originally, during decryption. So, what we did is, we didn't let the half change for any letter.

The main() function has an exception handling block that handles value error, file not found error and other exceptions if raised. At first, we input two shift values namely shift1 and shift2 by splitting the input separated by space and mapping them to integers. We have used three important functions:

A. encryption(shift1, shift2)
   Here, the shift1 and shift2 are provided as arguments. As shown in figure 4, it first opens the resources/raw_text.txt file in read mode using with statement. Here, we are joining each letter we get by calling encrypt_letter function and then writing the overall output in resources/encrypted_text.txt file.

   Let's dive into the encrypt_letter(letter, shift1, shift2) function. It takes letter, shift1 and shift2 as arguments. We have three cases here:
   a. For lowercase letters i.e. 'a' <= letter <= 'z'
      i. If letter <= 'm'
         Here, we first get the relative position of the letter for the half (example: for letter c, it is 3 using ord of the letter minus the ord of 'a'. Then we shift it by +(shift1*shift2). Since we are making sure the letter remains in the same half, we module it by 13 (26/2=13). Finally, we add ord of 'a' to get the absolute position and use the chr function to get the

character.

ii. Else case
Just like in the above case, we use ord and chr functions. But here, we shift each letter by -(shift1+shift2) and use 'n' as the base character for the relative position.

b. For uppercase letters i.e. 'A' <= letter <= 'Z'
This is similar to the algorithm for lowercase letters. However, in this case, we shift first half letters by -shift1 and second half letters by +(shift2**2).

c. For other letters i.e. else case
These letters will remain the same as mentioned by the question.

B. decryption(shift1, shift2)
It also takes shift1 and shift2 as arguments. It first opens the resources/encrypted_text.txt file, then joins each letter we get by calling the decrypt_letter function and writes the overall output to resources/decrypted_text.txt file.

Let's dive into the decrypt_letter(letter, shift1, shift2) function. It takes letter, shift1 and shift2 as arguments. We are reversing the encryption algorithm here. We already know which half the character belongs and our logic makes sure the character remains in the same half. So, we just reverse the encryption operation for decryption. Just like in decrypt_letter function, we have three cases here:

a. For lowercase letters i.e. 'a' <= letter <= 'z'
This is similar to the encryption algorithm for lowercase letters. However, in this case, we shift first half letters by -(shift1*shift2) and second half letters by +(shift1+shift2).

b. For uppercase letters i.e. 'A' <= letter <= 'Z'
This is similar to the encryption algorithm for uppercase letters. However, in this case, we shift first half letters by +shift1 and second half letters by -(shift2**2).

c. For other letters i.e. else case
These letters will remain the same as encryption doesn't change them either.

C. verification(debug)
This function takes a debug argument, which is a boolean. Since we are comparing the encrypted and decrypted texts, if the debug is true, we break where the comparison goes wrong to let the user know where it went wrong. Here, we open both

resources/encrypted_text.txt and resources/decrypted_text.txt files and compare them using equality operator. The debugging is done by enumerating the pair each character of those two texts made by using zip function, which makes sure the user knows where it went wrong.

```python
def main():
    try:
        shift1, shift2 = map(int, input("Provide two shift values separated by space: ").split())
        encryption(shift1, shift2)
        decryption(shift1, shift2)
        verification(debug=True)
    except ValueError:
        print("Invalid input. Enter two integers separated by a space.")
    except FileNotFoundError:
        print("File not found. Make sure the file exists in resources folder.")
    except Exception as e:
        print("Error:", e)
```

figure 3: Q1 Main function

```python
def encryption(shift1, shift2):
    with open("resources/raw_text.txt", "r", encoding="utf-8") as file:
        text = file.read()
    encrypted_text = ''.join(encrypt_letter(c, shift1, shift2) for c in text)
    with open("resources/encrypted_text.txt", "w", encoding="utf-8") as file:
        file.write(encrypted_text)


def decryption(shift1, shift2):
    with open("resources/encrypted_text.txt", "r", encoding="utf-8") as file:
        text = file.read()
    decrypted_text = ''.join(decrypt_letter(c, shift1, shift2) for c in text)
    with open("resources/decrypted_text.txt", "w", encoding="utf-8") as file:
        file.write(decrypted_text)


def verification(debug=False):
    with open("resources/raw_text.txt", "r", encoding="utf-8") as file:
        raw_text = file.read()
    with open("resources/decrypted_text.txt", "r", encoding="utf-8") as file:
        decrypted_text = file.read()

    if raw_text == decrypted_text:
        print("Encryption and decryption are consistent.")
    else:
        print("Mismatch found between original and decrypted texts.")
        if debug:
            # help you see where it breaks
            for i, (a, b) in enumerate(zip(raw_text, decrypted_text)):
                if a != b:
                    print(f"First mismatch at pos {i}: '{a}' -> '{b}'")
                    break
```

figure 4: Q1 Encryption, Decryption and Verification

```python
def encrypt_letter(letter, shift1, shift2):
    # lowercase
    if 'a' <= letter <= 'z':
        if letter <= 'm':  # a–m : forward by shift1*shift2 within a–m
            return chr(ord('a') + ((ord(letter) - ord('a') + (shift1 * shift2)) % 13))
        else:              # n–z : backward by shift1+shift2 within n–z
            return chr(ord('n') + ((ord(letter) - ord('n') - (shift1 + shift2)) % 13))
    # uppercase
    elif 'A' <= letter <= 'Z':
        if letter <= 'M':  # A–M : backward by shift1 within A–M
            return chr(ord('A') + ((ord(letter) - ord('A') - shift1) % 13))
        else:              # N–Z : forward by shift2^2 within N–Z
            return chr(ord('N') + ((ord(letter) - ord('N') + (shift2 ** 2)) % 13))
    else:
        return letter  # spaces, tabs, newlines, digits, symbols unchanged


def decrypt_letter(letter, shift1, shift2):
    # exact inverse of above (same halves, opposite directions)
    if 'a' <= letter <= 'z':
        if letter <= 'm':  # a–m : backward by shift1*shift2 within a–m
            return chr(ord('a') + ((ord(letter) - ord('a') - (shift1 * shift2)) % 13))
        else:              # n–z : forward by shift1+shift2 within n–z
            return chr(ord('n') + ((ord(letter) - ord('n') + (shift1 + shift2)) % 13))
    elif 'A' <= letter <= 'Z':
        if letter <= 'M':  # A–M : forward by shift1 within A–M
            return chr(ord('A') + ((ord(letter) - ord('A') + shift1) % 13))
        else:              # N–Z : backward by shift2^2 within N–Z
            return chr(ord('N') + ((ord(letter) - ord('N') - (shift2 ** 2)) % 13))
    else:
        return letter
```

figure 5: Q1 Letter Encryption and Decryption

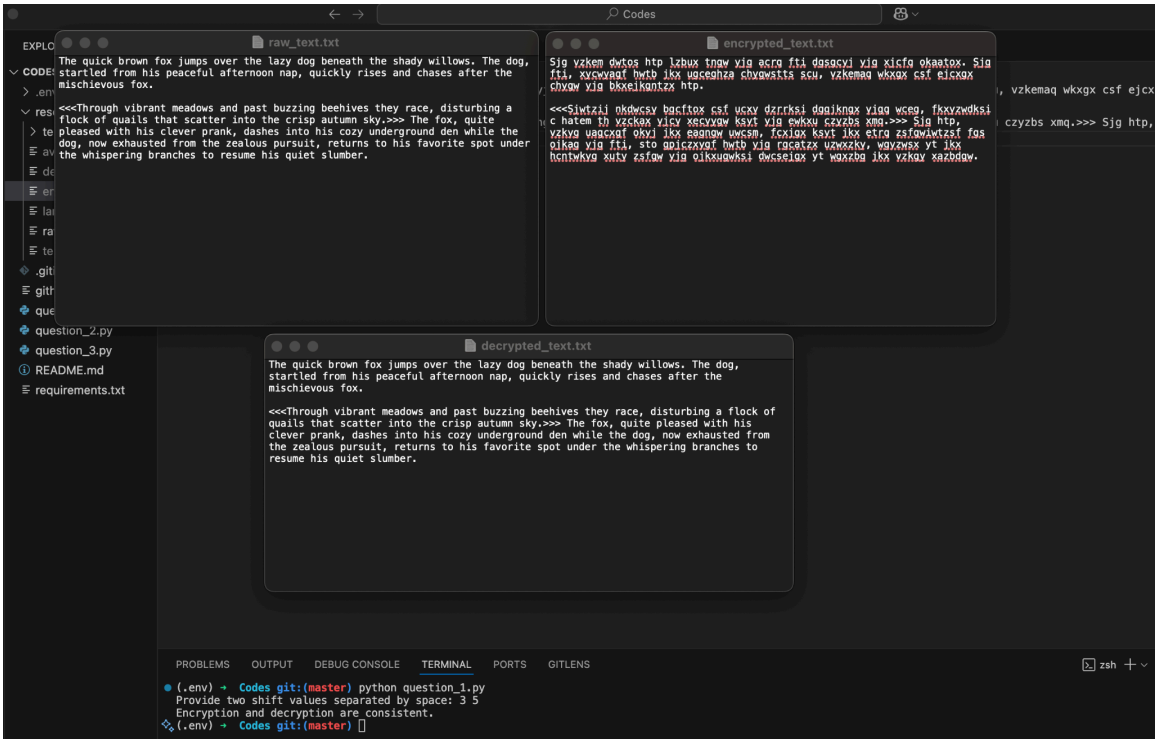

figure 6: Q1 Example Output

The figure 6 above shows an example output using two different shifts 3 and 5. We can see the encryption and decryption were consistent.

Question 2

This question is about analysing temperature data collected from different weather stations. The datasets are stored as multiple CSV files, each containing monthly temperature readings. Our aim was to combine this data and then calculate seasonal averages, identify the station with the largest annual temperature range, and find out which stations had the most stable and most variable temperatures.

In the beginning, the program loads all the temperature files. For this, we use the function load_temperature_data(), which collects all CSV files and reads them using Pandas. After that, it combines them into a single dataset. If no files are found, it simply returns an empty DataFrame. This dataset is then used as input for three main functions to perform the calculations.

Here, we use the glob module in Python to find all files that match a specific pattern in a folder (Glob — Unix Style Pathname Pattern Expansion, n.d.). In this program, we want to process all CSV files containing temperature data, but we don't know their exact names. By using glob with "*.csv", it automatically finds all files that end with .csv. We applied this approach because we have a large number of temperature files .

This dataset is then used as input for three main functions to perform the calculations.

a. Seasonal Average Calculation

   Here, It takes the full dataset as the argument. We use the function calculate_seasonal_average(data) to calculate the average temperature for each season, considering all years and all stations. We mapped the months to seasons according to the Australian seasonal system as follows

   Summer → December, January, February

   Autumn → March, April, May

   Winter → June, July, August

   Spring → September, October, November

   First, we cleaned the data by removing missing values for each month. Next, we added the valid numbers to the corresponding season. After that, we calculated the average by dividing the total sum by the number of valid values.

$$\text{Average} = \frac{Total\ Sum\ of\ Valid\ Temperatures}{Number\ of\ Valid\ Temperatures}$$

Finally, the seasonal averages were calculated and written to the output file resources/average_temp.txt, with the Celsius symbol included for clarity.

b. Largest Temperature Range

It takes full dataset data as an argument. The function calculate_largest_temp_range(data) identifies the station with the largest annual range, meaning the difference between its highest and lowest recorded monthly temperatures. For each row (station) in the dataset, we convert the monthly temperatures to numeric using pd.to_numeric() and remove missing values (NaN) with .dropna(). We calculate the temperature range when the station has valid temperatures.

$$\text{Range} = \text{Maximum Temperature} - \text{Minimum Temperature}$$

We found the maximum range among all stations and then wrote to the file /largest_temp_range_station.txt, along with its max, min, and range values in °C.

c. Temperature Stability

The function calculate_temperature_stability(data) checks how stable the temperatures are at each station. This was done by calculating the standard deviation of the monthly readings.

$$\text{StdDev} = \sqrt{\frac{\Sigma\left(x_i - \bar{x}\right)^2}{n}}$$

- The station with the smallest standard deviation is the most stable (little variation throughout the year).
- The station with the largest standard deviation is the most variable (big swings in temperatures).

The results are saved into /temperature_stability_stations.txt.

For each output, such as averages, ranges, or stability, we write the results to .txt files using the Unicode for the degree symbol (\u00B0) to display temperatures in °C.

Main Function Flow

The main function ties everything together. It first loads the data using load_temperature_data(). If no data is found, it prints a warning message. Otherwise, it runs the three calculations in order: seasonal averages, largest temperature range, and temperature stability.

```python
def main():
    try:
        data = load_temperature_data("resources/temperatures")
        if data.empty:
            print("No data found in resources/temperatures folder.")
        else:
            calculate_seasonal_average(data)
            calculate_largest_temp_range(data)
            calculate_temperature_stability(data)
    except Exception as e:
        print("Error:", e)
```

figure 7: Q2 Main function

```python
# Process ALL .csv files in the temperatures folder
def load_temperature_data(folder_path="resources/temperatures"):
    all_files = glob.glob(os.path.join(folder_path, "*.csv"))

    df_list = []
    for file in all_files:
        df = pd.read_csv(file)
        df_list.append(df)

    if df_list:
        combined_df = pd.concat(df_list, ignore_index=True)
        return combined_df
    else:
        return pd.DataFrame()
```

figure 8: Q2 Main function

```python
# Calculate the average temperature for each season across ALL stations and ALL years
def calculate_seasonal_average(data):
    seasonal_sum = {"Summer":0,"Autumn":0,"Winter":0,"Spring":0}
    seasonal_count = {"Summer":0,"Autumn":0,"Winter":0,"Spring":0}

    month_to_season = {1:"Summer",2:"Summer",12:"Summer",
                       3:"Autumn",4:"Autumn",5:"Autumn",
                       6:"Winter",7:"Winter",8:"Winter",
                       9:"Spring",10:"Spring",11:"Spring"}

    for month_index, month in enumerate(MONTH_COLUMNS, start=1):
        month_data = pd.to_numeric(data[month], errors='coerce')
        valid_values = month_data.dropna()
        season = month_to_season[month_index]
        seasonal_sum[season] += valid_values.sum()
        seasonal_count[season] += valid_values.count()

    with open("resources/average_temp.txt", "w", encoding="utf-8") as f:
        for season in ["Summer","Autumn","Winter","Spring"]:
            avg = seasonal_sum[season] / seasonal_count[season] #avg = total / count
            f.write(f"{season}: {avg:.1f}\u00B0C\n")
```

figure 9: Q2 Calculate Seasonal Average

```python
# Find station with largest annual temperature range
def calculate_largest_temp_range(data):
    ranges = []

    for idx, row in data.iterrows():
        temps = pd.to_numeric(row[MONTH_COLUMNS], errors='coerce').dropna()
        if not temps.empty:
            temp_range = temps.max() - temps.min()
            ranges.append((row["STATION_NAME"], temp_range, temps.max(), temps.min()))

    # Write the station with max range to file
    if ranges:
        max_range = max(ranges, key=lambda x: x[1])[1]
        with open("resources/largest_temp_range_station.txt", "w", encoding="utf-8") as f:
            for station, r, t_max, t_min in ranges:
                if r == max_range:
                    f.write(f"Station {station}: Range {r:.1f}\u00B0C (Max: {t_max:.1f}\u00B0C, Min: {t_min:.1f}\u00B0C)\n")
```

figure 10: Q2 Calculate Largest Temperature Range

```python
def calculate_temperature_stability(data):
    stability = []

    for idx, row in data.iterrows():
        temps = pd.to_numeric(row[MONTH_COLUMNS], errors='coerce').dropna()
        if not temps.empty:
            std_dev = temps.std() # Calculate standard deviation of the temperatures
            stability.append((row["STATION_NAME"], std_dev))

    # Write seasonal averages to file
    if stability:
        min_std = min(stability, key=lambda x: x[1])[1] # Find smallest standard deviation (most stable)
        max_std = max(stability, key=lambda x: x[1])[1] # Find largest standard deviation (most variable)

        with open("resources/temperature_stability_stations.txt", "w", encoding="utf-8") as f:
            for station, std in stability:
                if std == min_std:
                    f.write(f"Most Stable: Station {station}: StdDev {std:.1f}\u00B0C\n")


            for station, std in stability:
                if std == max_std:
                    f.write(f"Most Stable: Station {station}: StdDev {std:.1f}\u00B0C\n")
```

figure 11: Q2 Calculate Temperature Stability

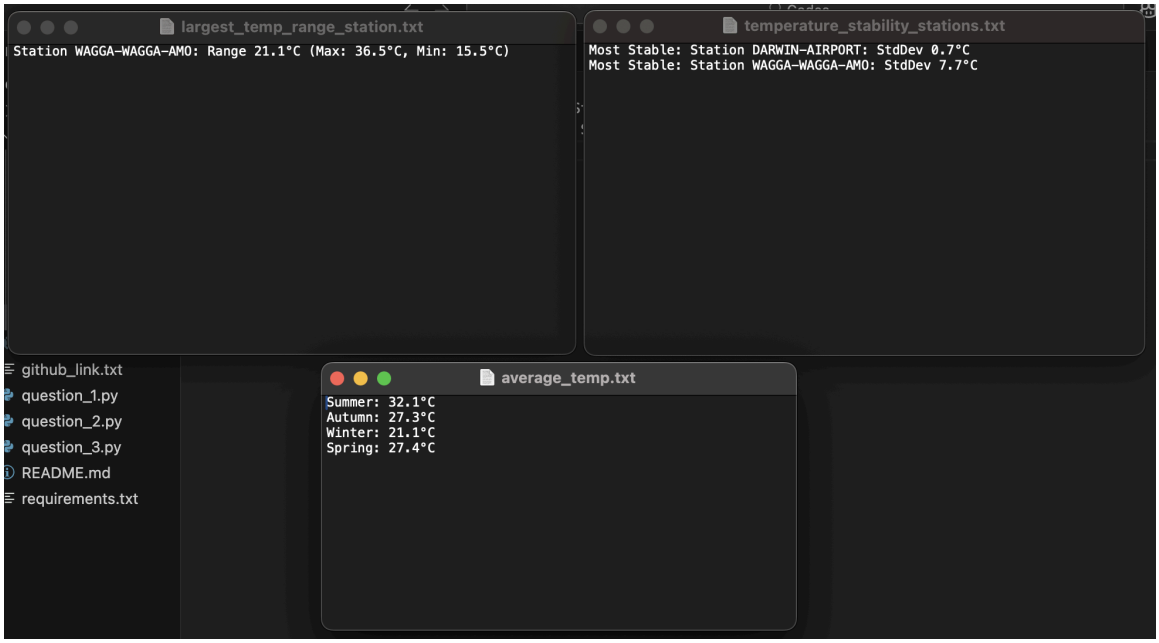

figure 12: Q2 Example Output

Question 3

This question is a bit tricky and covers two main topics viz. Recursion and Turtle Graphics. It asks us to create patterns in a regular polygon, recursively. The number of sides, length of each side and the depth of recursion are to be inputted from the user.

Let's first understand what we did in the main function. We asked for three inputs from the user as mentioned above. We then initialized a Turtle with visible=False. We shall come back to this later. The important maths behind this is the exterior angle formula. Since, we are dealing with regular polygons, we can easily calculate the exterior angle using the formula below:

ext_angle = 360 / no_of_sides (Third Space Learning, 2024)

Now, another logic we implemented here is by starting the turtle from (- length/2, - length/2). To understand this, we need to acknowledge the Turtle graphics window coordinate system. The center is the origin and what we did by this is, we pushed the turtle to the -x and -y axes just to make sure the final output lies at the center (since we are moving to the positive axes while drawing). Now, let us get back to visible=False, which is because we are pushing the turtle and we don't want to show this animation at first. We show the turtle after this using the showturtle function. We used the fastest speed of the turtle.

We have two recursive functions:

a. make_polygon(t, no_of_sides, length, angle, rec_depth)
   It takes the turtle object, no_of_sides, length, angle and rec_depth as arguments. This function basically runs for the no_of_sides times. The base case no_of_sides <= 0, makes it clear that this is like a loop for no_of_sides times. However, what we're doing here is calling the make_edge function and turning left with the exterior angle of the polygon.

b. make_edge(t, length, rec_depth)
   This function makes each edge recursively. For each edge, this function runs rec_depth + 1 times, as given by the base case. The last time it runs, it moves forward. For every other time, we cut the line into three halves i.e. length/3. But, it is to be noted that the second half is further divided into two halves, both 1/3rd of length.

   We do four steps here:
   i. Call make_edge with length/3 and rec_depth - 1 (1st segment)
   ii. Rotate left 60 degrees and call make_edge with length/3 and rec_depth - 1 (2nd segment, 1st part)

iii. Rotate right 120 degrees and call make_edge with length/3 and rec_depth - 1 (2nd segment, 2nd part)

iv. Rotate left 60 degrees and call make_edge with length/3 and rec_depth - 1 (3rd segment)

This is a bit tricky and recursion for complex patterns can't be understood easily unless practically applied.

```
def main():
    try:
        no_of_sides = int(input("Enter the number of sides of the polygon: "))
        length = float(input("Enter the length of each side: "))
        rec_depth = int(input("Enter the recursion depth: "))

        # we are making turtle invisible for the offset
        t = Turtle(visible=False)
        t.screen.title('Geometric Pattern - Polygon')
        ext_angle = 360 / no_of_sides

        # making sure we draw at the center, so we offset it to the left and down by half the length of a side
        # while moving the pen to the left, we don't want it to draw, so we are doing penup
        t.penup()
        t.goto(-(length / 2), -(length / 2))
        t.pendown()
        t.showturtle()

        # using the fastest speed
        t.speed(0)
        # we are drawing to the upper side, so we always use left turn
        make_polygon(t, no_of_sides, length, ext_angle, rec_depth)
        t.screen.mainloop()

    except Exception as e:
        print(f"Error: {e}")
```

figure 13: Q3 Main Function

```python
def make_polygon(t, no_of_sides, length, angle, rec_depth):
    if no_of_sides <= 0:
        return
    make_edge(t, length, rec_depth)
    t.left(angle)
    make_polygon(t, no_of_sides - 1, length, angle, rec_depth)

def make_edge(t, length, rec_depth):
    if rec_depth < 1:
        t.forward(length)
        return
    else:
        length /= 3 # we divide the line into 3 equal segments
        # for first segment
        make_edge(t, length, rec_depth - 1)
        # for second segment - first part
        t.left(60)
        make_edge(t, length, rec_depth - 1)
        # for second segment - second part
        t.right(120)
        make_edge(t, length, rec_depth - 1)
        # for third segment
        t.left(60)
        make_edge(t, length, rec_depth - 1)
```

figure 14: Q3 Make Polygon and Edge

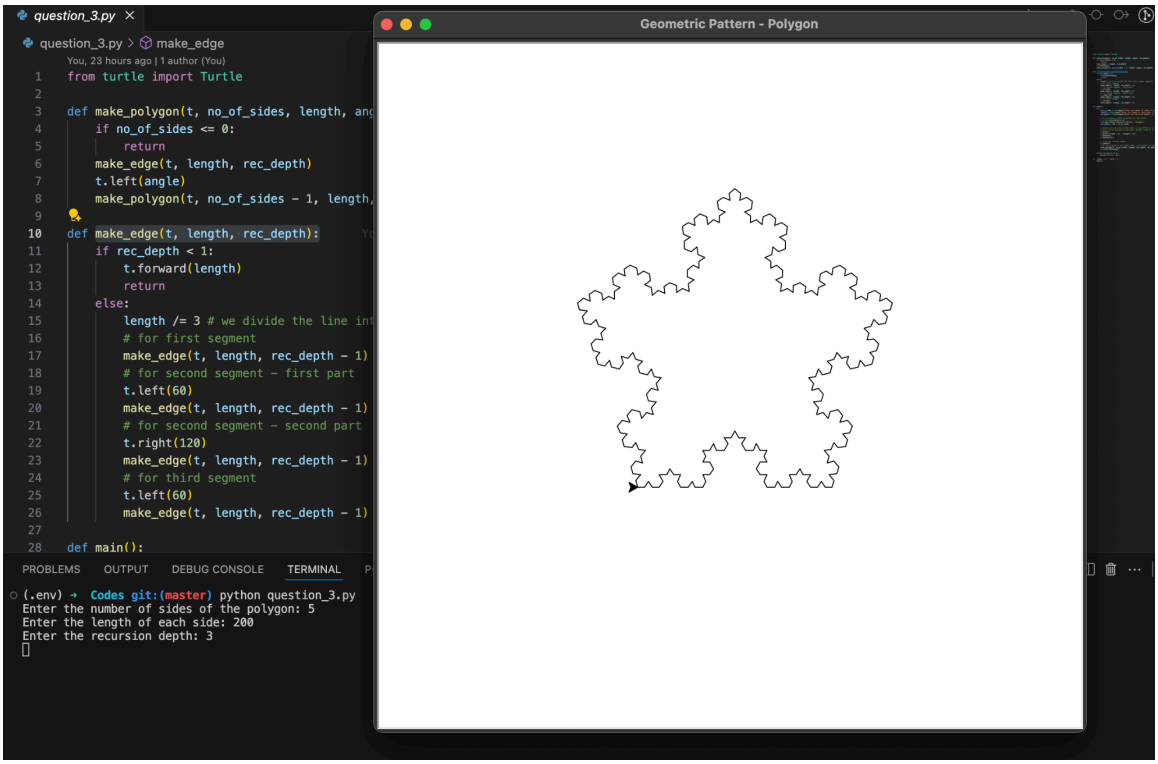

figure 15: Q3 Example Output

The above example output shows the pattern for a pentagon with side length 200 and recursion depth 3

References

Breuss, M. (2024, November 30). *What Does if __name__ == "__main__" Do in Python?* https://realpython.com/if-name-main-python/

Wikipedia contributors. (2025, July 17). *Caesar cipher*. Wikipedia. https://en.wikipedia.org/wiki/Caesar_cipher

glob — Unix style pathname pattern expansion. (n.d.). Python Documentation. https://docs.python.org/3/library/glob.html

Third Space Learning. (2024, July 17). *Exterior Angles of a Polygon - GCSE Maths - Revision Guide*. https://thirdspacelearning.com/gcse-maths/geometry-and-measure/exterior-angles/