SS25 HIT 137 SOFTWARE NOW

ASSIGNMENT 2
GROUP: SYD 16

GROUP MEMBERS:

APURBA PAUDEL- s398926
RAKSHA TIMILSINA- s398446
SUBEK SHARMA- s398321
SUJAN SIGDEL- s399205

TABLE OF CONTENTS

TABLE OF CONTENTS	**2**
1. INTRODUCTION	**3**
2. AI DECLARATION	**3**
3. QUESTION 1	**4**
4. QUESTION 2	**5**
5. QUESTION 3	**8**
6. TEAM COLLABORATION	**9**
7. INDIVIDUAL CONTRIBUTIONS	**10**
8. CONCLUSION	**10**
9. REFERENCES	**11**

1. INTRODUCTION

For this assignment, we had three real computing tasks with Python. The first challenge was to write and test a program for text-encryption, decryption, verification using new shift method. The second assignment asked us to process Australian temperature data spread across several CSV files, computing seasonal means and ranges and testing stability across stations. The third assignment was to construct a recursive geometric pattern with Python's Turtle graphics. We also utilized GitHub to work collaboratively, track individual contributions, and maintain version control. In this report, we describe how we addressed the tasks, cooperated as a team, responsibly used AI tools, and informally discussed our takeaways.

2. AI DECLARATION

The work in this assignment is solely our own and presents our own learning and effort. We did not utilize any AI tool to generate the final code, or complete any aspect of the work mentioned. All coding, debugging, testing, explanation and solution creation was performed by us as a group.

3. QUESTION 1

```
SEPARATOR = "<<<META_END>>>"  # used to split tag section and encrypted section


def encrypt_one_char(ch, shift1, shift2):


    # lowercase letters
    if 'a' <= ch <= 'z':
        if 'a' <= ch <= 'm':
            # tag 0 = lowercase first half (a-m)
            shift = shift1 * shift2
            pos = ord(ch) - ord('a')
            new_pos = (pos + shift) % 26
            return chr(new_pos + ord('a')), '0'
        else:
            # tag 1 = lowercase second half (n-z)
            shift = shift1 + shift2
            pos = ord(ch) - ord('a')
            new_pos = (pos - shift) % 26
            return chr(new_pos + ord('a')), '1'

    # uppercase letters
    if 'A' <= ch <= 'Z':
        if 'A' <= ch <= 'M':
            # tag 2 = uppercase first half (A-M)
            shift = shift1
            pos = ord(ch) - ord('A')
            new_pos = (pos - shift) % 26
            return chr(new_pos + ord('A')), '2'
        else:
            # tag 3 = uppercase second half (N-Z)
            shift = shift2 * shift2
            pos = ord(ch) - ord('A')
            new_pos = (pos + shift) % 26
            return chr(new_pos + ord('A')), '3'

    # other characters unchanged
    # tag 4 = other
    return ch, '4'


def decrypt_one_char(ch, tag, shift1, shift2):


    # tag 0: lowercase a-m was shifted forward by shift1*shift2, so reverse is backward
    if tag == '0':
        shift = shift1 * shift2
        pos = ord(ch) - ord('a')
        new_pos = (pos - shift) % 26
        return chr(new_pos + ord('a'))

    # tag 1: lowercase n-z was shifted backward by shift1+shift2, so reverse is forward
    if tag == '1':
        shift = shift1 + shift2
        pos = ord(ch) - ord('a')
        new_pos = (pos + shift) % 26
        return chr(new_pos + ord('a'))

    # tag 2: uppercase A-M was shifted backward by shift1, so reverse is forward
    if tag == '2':
        shift = shift1
        pos = ord(ch) - ord('A')
        new_pos = (pos + shift) % 26
        return chr(new_pos + ord('A'))

    # tag 3: uppercase N-Z was shifted forward by shift2^2, so reverse is backward
    if tag == '3':
        shift = shift2 * shift2
        pos = ord(ch) - ord('A')
        new_pos = (pos - shift) % 26
        return chr(new_pos + ord('A'))

    # tag 4: other character, unchanged
    return ch


def encrypt_file(shift1, shift2):
    f = open("raw_text.txt", "r", encoding="utf-8", newline="")
    text = f.read()
    f.close()

    tags = ""
    encrypted = ""

    # Encrypt character by character
    for ch in text:
        enc_ch, tag = encrypt_one_char(ch, shift1, shift2)
        encrypted += enc_ch
        tags += tag


    f = open("encrypted_text.txt", "w", encoding="utf-8", newline="")
    f.write(tags)
    f.write("\n" + SEPARATOR + "\n")
    f.write(encrypted)
    f.close()


def decrypt_file(shift1, shift2):
    f = open("encrypted_text.txt", "r", encoding="utf-8", newline="")
    content = f.read()
    f.close()


    parts = content.split("\n" + SEPARATOR + "\n")
    tags = parts[0]
    encrypted = parts[1]

    # Decrypt character by character using tags
    decrypted = ""
    for i in range(len(encrypted)):
        decrypted += decrypt_one_char(encrypted[i], tags[i], shift1, shift2)


    f = open("decrypted_text.txt", "w", encoding="utf-8", newline="")
    f.write(decrypted)
    f.close()


def verify_files():
    f = open("raw_text.txt", "r", encoding="utf-8", newline="")
    original = f.read()
    f.close()

    f = open("decrypted_text.txt", "r", encoding="utf-8", newline="")
    decrypted = f.read()
    f.close()

    if original == decrypted:
        print("Decryption successful! Files match.")
    else:
        print("Decryption failed! Files do not match.")


shift1 = int(input("Enter shift1: "))
shift2 = int(input("Enter shift2: "))

encrypt_file(shift1, shift2)
decrypt_file(shift1, shift2)
verify_files()
```

Fig: 3.1.1 Code

The use of this program is to encrypt, decrypt and verify the text information stored in external files. The program, at first reads the content or the raw ".txt" file and then processes the file in series, one character at a time. Each character is checked if it is uppercase, lowercase, or a non-alphabetic character. There are certain rules to be followed when shifting the characters. The lowercase letters are shifted depending on whether they fall in the range a to m or n to z. Similarly, the uppercase letters are shifted depending on whether they fall in the range A to M or N to Z. The non-alphabetic characters: spaces, numbers, and symbols are left changed however. To ensure the decryption is accurate, there is a small tag for each character stored in the program that indicates the encryption rule applied to the character. Then, these tags are saved together with the encrypted text file. For decrypting the message, the program has to read both the encrypted text and the corresponding tags in the encrypted text file. The correct operation for each character is then identified according to its tag and then the whole file is decrypted. There is a verification function which compares the original file and the decrypted file which confirms if the process was successful.[1]

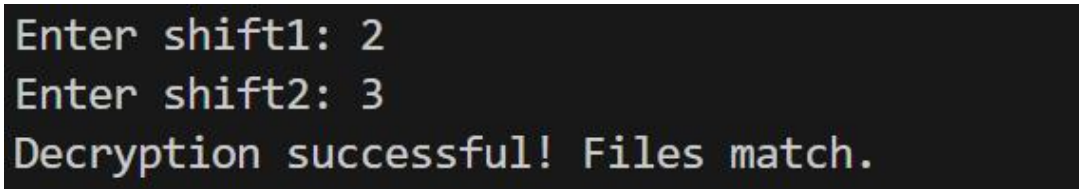


Fig: 3.1.2. Output

Results: The program successfully encrypted the text, decrypted it accurately and verified correctness with a success confirmation message.

4. QUESTION 2

To solve this question, first of all we need to combine all the csv files under temperature folder. There are many ways to combine the csv files, I personally used iterdir() and pandas concat() method[2][3]. After combining all the csv files now we can perform analysis on the single file. First we check if any of the fields is null value, if null value exists we need to

replace the null field with zero. The question has 3 sub parts.

```
final = []
folder = Path('temperatures')

for file in folder.iterdir():
    df = pd.read_csv(file)
    final.append(df)
```

```
final_df = pd.concat(final,ignore_index='False')
```

```
final_df.head()
```

	STATION_NAME	STN_ID	LAT	LON	January	February	March	April	May	June	July	August	September	October	November	December
0	ADELAIDE-KENT-TOWN	23090	-34.92	138.62	31.48	31.37	28.12	24.81	21.28	17.92	17.20	17.87	20.54	22.98	26.65	28.38
1	ALBANY-AIRPORT-COMPARISON	9741	-34.94	117.80	25.24	26.03	25.45	23.50	20.55	18.00	16.95	17.02	18.34	19.52	21.85	23.75
2	ALICE-SPRINGS-AIRPORT	15590	-23.80	133.89	38.40	37.32	35.35	31.37	25.18	21.06	20.52	23.71	29.07	31.41	34.38	36.06
3	AMBERLEY-AMO	40004	-27.63	152.71	32.90	31.87	31.21	29.20	26.06	23.38	22.88	24.12	27.58	29.50	31.04	32.28
4	BARCALDINE-POST-OFFICE	36007	-23.55	145.29	38.03	36.21	35.41	31.73	27.46	25.21	24.64	26.46	30.75	34.13	35.93	37.41

In the first part, first we need to divide the months into seasons. We divide each month into 4 seasons namely summer, spring, autumn and winter. Now, we compute mean of each season using mean() method. For summer we have 3 months which are December, January and February . final_df[summer].mean() calculates the mean of individual columns December, January and February and again using the .mean() method gives mean of those three columns (December, January and February), finally giving the total mean of summer . Similarly we do this for other 3 seasons. We save the result of all 4 seasons into a dictionary. After that we create a file named 'average_temp.txt' using write method[4] and access the dictionary key value using .items() method and write it onto the file(the seasons and the averages) to produce the intended result.

```
summer = ['December','January','February']
autumn = ['March','April','May']
spring = ['June','July','August']
winter = ['September','October','November']
```

```
seasonal_avg = {}
seasonal_avg['Summer'] = final_df[summer].mean().mean()
print(seasonal_avg['Summer'])
```

```
32.103751488095234
```

```
seasonal_avg['Autumn'] = final_df[autumn].mean().mean()
seasonal_avg['Spring'] = final_df[spring].mean().mean()
seasonal_avg['Winter'] = final_df[winter].mean().mean()
```

```
with open('average_temp.txt', 'w') as file:
    for season, avg in seasonal_avg.items():
        file.write(f'{season}:{round(avg,2)}°C\n')
```

Fig 4.1.1: Season Averages

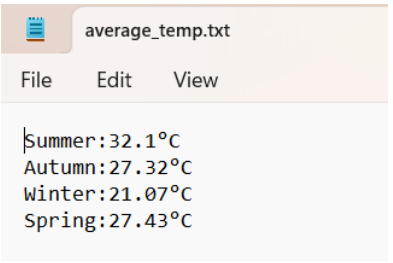


Fig 4.1.2: Part 1 Output

In the second part, we need to find the station with maximum temperature range for which we first need to calculate maximum and minimum temperature of each station across all years. Since we need to find the max and min of station across all years we first need to group the station using groupby() method. Then we specify [month] to provide the actual column on which to perform aggregation. Then we use .max() method to find the maximum across all method. Performing final_df.groupby('STATION_NAME')[month].max [5] gives

the maximum temperature of all stations across all twelve months. But we want to find maximum across all 12 months so we once again use .max() method but this time specifying the axis = 1 since we want to find max across all 12 month i.e across horizontally. Similarly, we do the same using .min() method to find the minimum. Now, since we have both min and max we can now simply calculate range of all station by subtracting min from max. Then, we find out the maximum range using .max(). Now we have the max range, we now need the index of stations with that maximum range. idxmax() method gives index of only first occurrence if there is tie then we need to combine it with filtering[6], this gives us the indexes of stations that match the maximum range value. Then like in part one, we create a file named 'largest_temp_range_station.txt' using write method and access the maximum, minimum and range of station using the index returned above and write onto the file to produce the intended result.

```
month = final_df.columns[4:16]
month
```

```
Index(['January', 'February', 'March', 'April', 'May', 'June', 'July',
       'August', 'September', 'October', 'November', 'December'],
      dtype='object')
```

```
max_temp_month = final_df.groupby('STATION_NAME')[month].max()
max_temp_month.head()
```

STATION_NAME	January	February	March	April	May	June	July	August	September	October	November	December
ADELAIDE-KENT-TOWN	32.91	33.13	30.28	26.83	23.29	19.62	18.35	19.45	22.38	25.02	29.38	31.40
ALBANY-AIRPORT-COMPARISON	27.64	28.16	28.01	25.73	22.49	19.44	18.95	19.29	20.74	22.39	24.51	26.75
ALICE-SPRINGS-AIRPORT	42.69	40.99	38.85	34.63	28.62	24.96	23.66	28.19	33.56	36.69	39.05	40.75
AMBERLEY-AMO	36.22	35.43	34.34	32.13	28.96	26.63	26.43	27.63	31.68	34.24	35.00	35.61
BARCALDINE-POST-OFFICE	40.41	39.94	39.79	35.80	32.26	28.78	28.18	30.86	35.44	39.20	40.15	40.33

```
max_temp_df = max_temp_month.max(axis=1)
```

```
max_temp_df
```

```
STATION_NAME
ADELAIDE-KENT-TOWN              33.13
ALBANY-AIRPORT-COMPARISON       28.16
ALICE-SPRINGS-AIRPORT           42.69
AMBERLEY-AMO                    36.22
BARCALDINE-POST-OFFICE          40.41
                                ...
```

Fig 4.2.1: Use of groupby() method

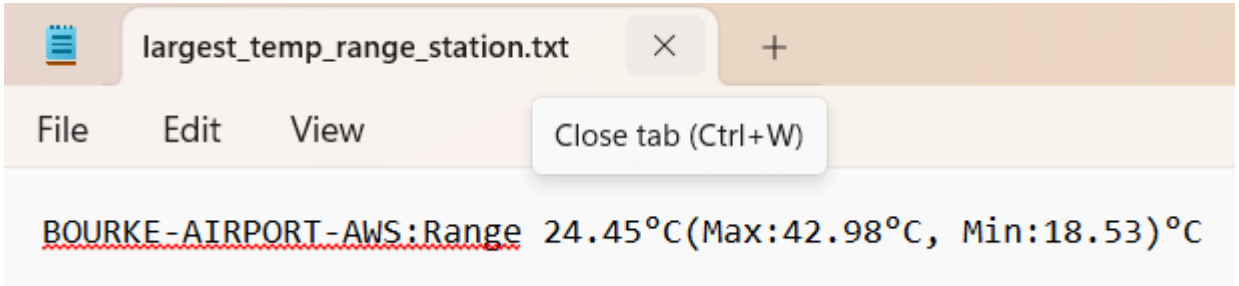


Fig 4.2.2: Part 2 Output

In the last part, we need to find the station with most stable and most variable temperatures for which we need to calculate the standard deviation. First we calculate standard deviation across 12 months for each row and average this standard deviation across all years for each station.Then, we find the min and max standard deviation and combine the idxmax() with filtering to give the indexes of stations that match the maximum and minimum temperature. Then, we create a file named 'temperature_stability_stations.txt' using write method and

access the station with maximum and minimum temperature from the returned indexes and write onto the file to produce the intended result.

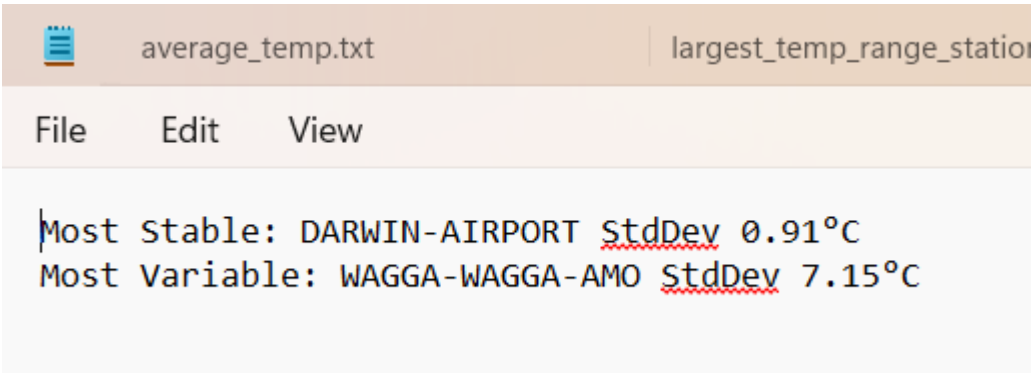


Fig 4.3: Part 3 Output

Results: The program successfully processed multi-year, multi-station data and generated accurate statistical climate results in required text formats.

5. QUESTION 3

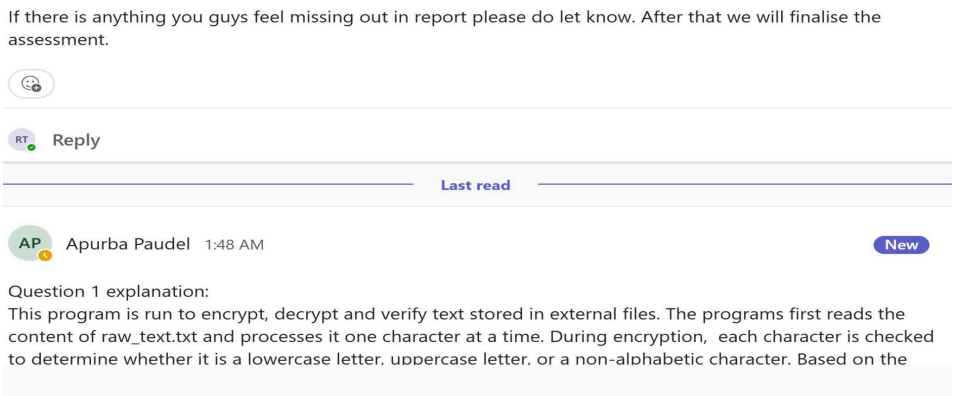
```
import turtle

def draw_edge(length, depth):
    if depth == 0:
        turtle.forward(length)
    else:
        length /= 3

        draw_edge(length, depth - 1)

        turtle.left(60)
        draw_edge(length, depth - 1)

        turtle.right(120)
        draw_edge(length, depth - 1)

        turtle.left(60)
        draw_edge(length, depth - 1)


def draw_polygon(sides, length, depth):
    angle = 360 / sides
    for _ in range(sides):
        draw_edge(length, depth)
        turtle.left(angle)


#Main Program
sides = int(input("Enter the number of sides: "))
depth = int(input("Enter the recursion depth: "))

#Turtle setup
turtle.speed(0)
turtle.hideturtle()
turtle.penup()
turtle.goto(0, 0)
turtle.setheading(0)
turtle.pendown()

draw_polygon(sides, length, depth)

turtle.done()
```

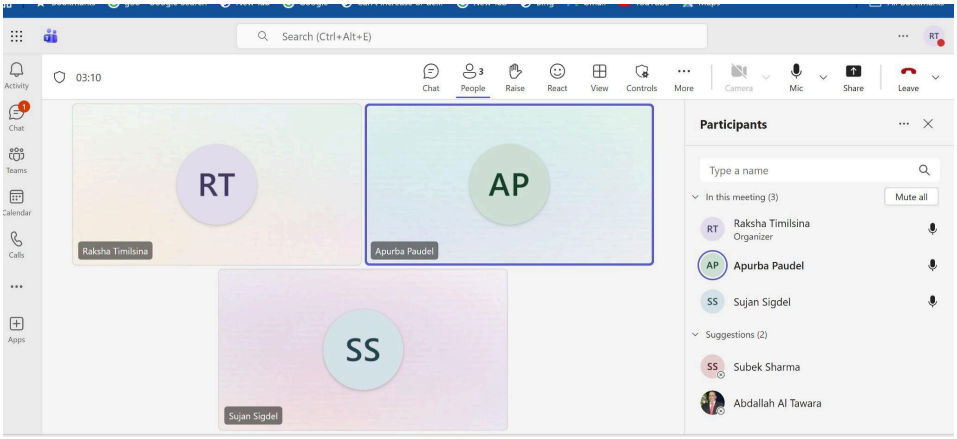
This Python program uses turtle graphics [7] and recursion to create a fractal shape. The function draw_edge(length, depth) draws one side of the fractal. If the recursion depth is 0, it just draws a straight line. But if the depth is more than 0, it divides the line into three parts and draws four smaller lines in a left-right-left pattern with fixed turns (60°, 120°, and then 60° again), which creates a bump-like shape similar to the Koch curve. It calls itself again with a lower depth. The function draw_polygon(sides, length, depth) uses draw_edge to create one side of the fractal and then repeats this process for the number of sides specified. After each side, the turtle turns by 360 divided by the number of sides to form a closed shape. The program then asks for the number of sides, the length of each side, and the recursion depth. It sets up the turtle graphics by centering the pen, hiding the cursor, and speeding up the drawing. It calls draw_polygon to start drawing and then shows the finished fractal.

Results: The program successfully generated visually appealing recursive geometric patterns on the basis of user input. The increase in the depth resulted in more complex and detailed designs.

6. TEAM COLLABORATION

The assignment was a collaborative project, with all the planning and talk done through face-to-face consultation. Once we had settled on a plan and divided the work, everyone contributed their portion and updated their progress regularly onto an open-source GitHub site. This website allowed us to see what everyone was up to, look at the updates, and manage different versions of our work. In that way we managed to work better together, solved problems more easily and every single member of the group was active and participating.





7. INDIVIDUAL CONTRIBUTIONS

Each group member contributed meaningfully to different parts of the assignment. Individual roles were divided in order to make the assignment better and ensure equal learning.

Apurba Poudel (s398926) - I contributed to Question 1 by creating the encryption and decryption algorithms, implementing the verification process, managing file input and output, and carefully testing the program to ensure it functioned correctly.

Sujan Sigdel (s399205) - I completed Question 2 by carefully analyzing the CSV files, combining multiple CSVs into one. I then referred to relevant references to assist in solving the problem which involved calculating the seasonal averages, the maximum and minimum temperature to find out the range and the standard deviation. The approach I followed while solving this problem is detailed in Question 2 part.

Subek Sharma (s398321) - I focused on Question 3, where I created and set up the recursive Turtle graphics pattern. I handled the user input parameters, tried out various levels of recursion, and checked that the resulting images matched according to the ask of the assignment.

Raksha Timilsina (s398446) - I prepared the final report, gathered all the outputs and screenshots, made sure the documentation was easy to understand and well arranged, and created the final draft to be submitted. Overall, I worked collaboratively with the team to achieve a desirable goal.

8. CONCLUSION

This assignment helped us get better at using Python in real situations. We practiced working with files, editing text, encrypting and decrypting data, analyzing information, and handling datasets that had missing entries. Additionally, we learned about writing programs that call themselves, called recursion, and creating visual patterns. Through working together on coding projects and using GitHub to manage our code changes, we improved

our teamwork skills. Overall, the assignment showed us how programming can solve real-life problems, made us better at solving challenges, and helped us become more effective at working with others.

9. REFERENCES

[1] "Caesar Cipher in Cryptography," GeeksforGeeks. [Online]. Available: https://www.geeksforgeeks.org/ethical-hacking-caesar-cipher-in-cryptography/ . [Accessed: 14 Jan. 2026].

[2] Stackoverflow, 'What is the difference between pathlib glob('*') and iterdir?,' [Online]. Available: https://stackoverflow.com/questions/63001429/what-is-the-difference-between-pathlib-glob-and-iterdir [Accessed: 6-Jan-2026].

[3] Stackoverflow, 'Combining separate daily CSVs in pandas,' [Online]. Available: https://stackoverflow.com/questions/60923806/combining-separate-daily-csvs-in-pandas [Accessed: 6-Jan-2026].

[4] Stackoverflow, 'How to create a new text file using Python,' [Online]. Available: https://stackoverflow.com/questions/48959098/how-to-create-a-new-text-file-using-python [Accessed: 7-Jan-2026].

[5] B. Solomon, 'pandas groupby: Your Guide to grouping data in Python,' [Online]. Available: https://realpython.com/pandas-groupby/ [Accessed: 11-Jan-2026].

[6] SparkCodeHub, 'Mastering the idxmax method in pandas: A comprehensive guide to finding maximum value indices,' [Online].
Available: https://www.sparkcodehub.com/pandas/data-analysis/idxmax-index [Accessed: 11-Jan-2026].

[7] Python Software Foundation, "turtle — Turtle graphics", [Online]. Available: https://docs.python.org/3/library/turtle.html
[Accessed: 15-Jan-2026].