



**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**

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

## **1. INTRODUCTION**

For this assignment, we had three real computing tasks to do 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

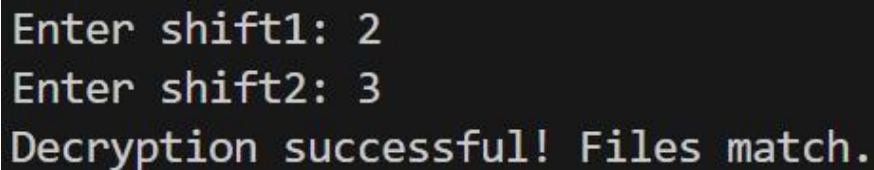
```

1 SEPARATOR = "<<META_END>>" # used to split tag section and encrypted section
2
3
4 def encrypt_one_char(ch, shift1, shift2):
5
6     # lowercase letters
7     if 'a' <= ch <= 'z':
8         # tag 0 = lowercase first half (a-m)
9         shift = shift1 + shift2
10        pos = ord(ch) - ord('a')
11        new_pos = (pos + shift) % 26
12        return chr(new_pos + ord('a'))
13    else:
14        # tag 1 = lowercase second half (n-z)
15        shift = shift1 + shift2
16        pos = ord(ch) - ord('a')
17        new_pos = (pos - shift) % 26
18        return chr(new_pos + ord('a'))
19
20 # uppercase letters
21 if 'A' <= ch <= 'Z':
22     # tag 2 = uppercase first half (A-M)
23     shift = shift1
24     pos = ord(ch) - ord('A')
25     new_pos = (pos + shift) % 26
26     return chr(new_pos + ord('A'))
27 else:
28     # tag 3 = uppercase second half (N-Z)
29     shift = shift1 + shift2
30     pos = ord(ch) - ord('A')
31     new_pos = (pos - shift) % 26
32     return chr(new_pos + ord('A'))
33
34 # other characters unchanged
35 # tag 4 = other
36 return ch, '4'
37
38 def decrypt_one_char(ch, tag, shift1, shift2):
39
40     # tag 0: lowercase a-m was shifted forward by shift1*shift2, so reverse is backward
41     if tag == '0':
42         shift = shift1 + shift2
43         pos = ord(ch) - ord('a')
44         new_pos = (pos - shift) % 26
45         return chr(new_pos + ord('a'))
46     # tag 1: lowercase n-z was shifted backward by shift1*shift2, so reverse is forward
47     if tag == '1':
48         shift = shift1 + shift2
49         pos = ord(ch) - ord('a')
50         new_pos = (pos + shift) % 26
51         return chr(new_pos + ord('a'))
52     # tag 2: uppercase A-M was shifted backward by shift1, so reverse is forward
53     if tag == '2':
54         shift = shift1
55         pos = ord(ch) - ord('A')
56         new_pos = (pos + shift) % 26
57         return chr(new_pos + ord('A'))
58     # tag 3: uppercase N-Z was shifted forward by shift2*2, so reverse is backward
59     if tag == '3':
60         shift = shift1 + shift2
61         pos = ord(ch) - ord('A')
62         new_pos = (pos - shift) % 26
63         return chr(new_pos + ord('A'))
64     # tag 4: other character, unchanged
65     return ch
66
67 def encrypt_file(shift1, shift2):
68     f = open("raw_text.txt", "r", encoding="utf-8", newline="")
69     text = f.read()
70     f.close()
71     tags = ""
72     encrypted = ""
73
74     # Encrypt character by character
75     for ch in text:
76         enc_ch, tag = encrypt_one_char(ch, shift1, shift2)
77         encrypted += enc_ch
78         tags += tag
79
80     f = open("encrypted_text.txt", "w", encoding="utf-8", newline="")
81     f.write(tags)
82     f.write("\n" + SEPARATOR + "\n")
83     f.write(encrypted)
84     f.close()
85
86 def decrypt_file(shift1, shift2):
87     f = open("encrypted_text.txt", "r", encoding="utf-8", newline="")
88     content = f.read()
89     f.close()
90
91     parts = content.split("\n" + SEPARATOR + "\n")
92     tags = parts[0]
93     encrypted = parts[1]
94
95     # Decrypt character by character using tags
96     decrypted = ""
97     for i in range(len(encrypted)):
98         decrypted += decrypt_one_char(encrypted[i], tags[i], shift1, shift2)
99
100    f = open("decrypted_text.txt", "w", encoding="utf-8", newline="")
101    f.write(decrypted)
102    f.close()
103
104 def verify_files():
105     f = open("raw_text.txt", "r", encoding="utf-8", newline="")
106     original = f.read()
107     f.close()
108
109     f = open("decrypted_text.txt", "r", encoding="utf-8", newline="")
110     decrypted = f.read()
111     f.close()
112
113     if original == decrypted:
114         print("Decryption successful! Files match.")
115     else:
116         print("Decryption failed! Files do not match.")
117
118 shift1 = int(input("Enter shift1: "))
119 shift2 = int(input("Enter shift2: "))
120
121 encrypt_file(shift1, shift2)
122 decrypt_file(shift1, shift2)
123 verify_files()
124
125
126

```

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 of 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 unchanged 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.



```
Enter shift1: 2
Enter shift2: 3
Decryption successful! Files match.
```

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[1][2]. 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[3] 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')
```

```
average_temp.txt

File Edit View

Summer:32.1°C
Autumn:27.32°C
Winter:21.07°C
Spring:27.43°C
```

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 [4] 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[5], 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()
```

	January	February	March	April	May	June	July	August	September	October	November	December
STATION_NAME												
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

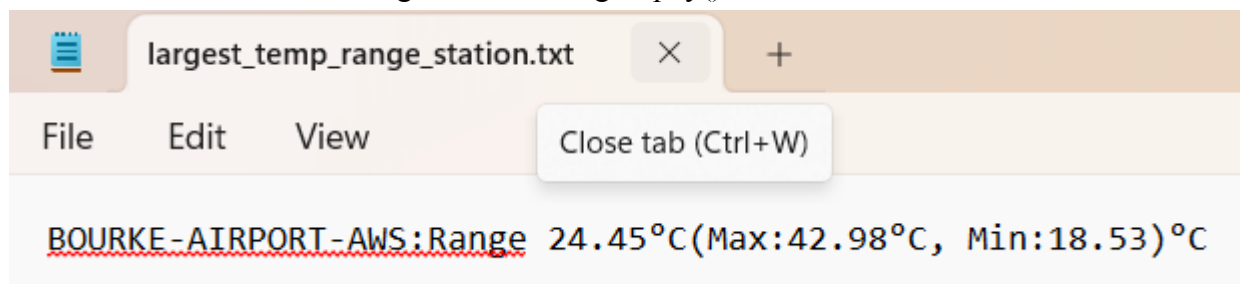


Fig 4.2.2: Part 2 Output

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 month for each row and 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.

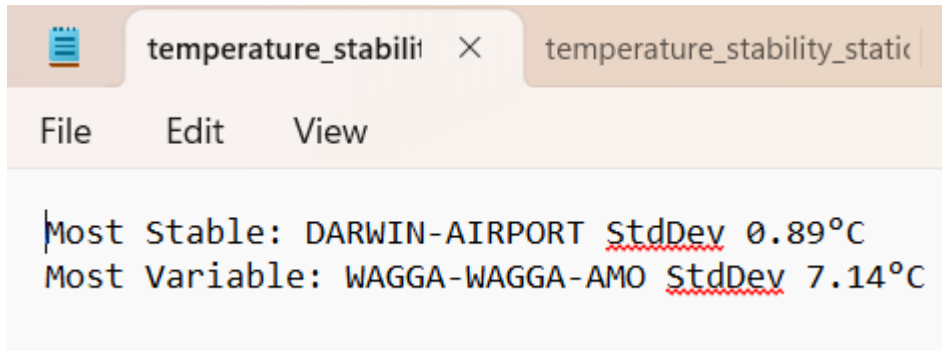


Fig 4.3: Part 3 Output

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

## 5. QUESTION 3

```

1  import turtle
2
3  def draw_edge(length, depth):
4      if depth == 0:
5          turtle.forward(length)
6      else:
7          length /= 3
8
9          draw_edge(length, depth - 1)
10
11         turtle.left(60)
12         draw_edge(length, depth - 1)
13
14         turtle.right(120)
15         draw_edge(length, depth - 1)
16
17         turtle.left(60)
18         draw_edge(length, depth - 1)
19
20
21  def draw_polygon(sides, length, depth):
22      angle = 360 / sides
23      for _ in range(sides):
24          draw_edge(length, depth)
25          turtle.left(angle)
26
27
28  #Main Program
29  sides = int(input("Enter the number of sides: "))
30
31  depth = int(input("Enter the recursion depth: "))
32
33  #Turtle setup
34  turtle.speed(0)
35  turtle.hideturtle()
36  turtle.penup()
37  turtle.goto(0, 0)
38  turtle.setheading(0)
39  turtle.pendown()
40
41  draw_polygon(sides, length, depth)
42
43  turtle.done()

```

This Python program uses turtle graphics 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 ( $60^\circ$ ,  $120^\circ$ , and then  $60^\circ$  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. Balancing face-to-face time with online collaboration, we could divvy up the work, keep tabs on each other's progress and stay open about everything. In that way we managed to work better together, solved problems more easily and every single member of the group was active and participating.

If there is anything you guys feel missing out in report please do let know. After that we will finalise the assessment.



RT Reply

Last read

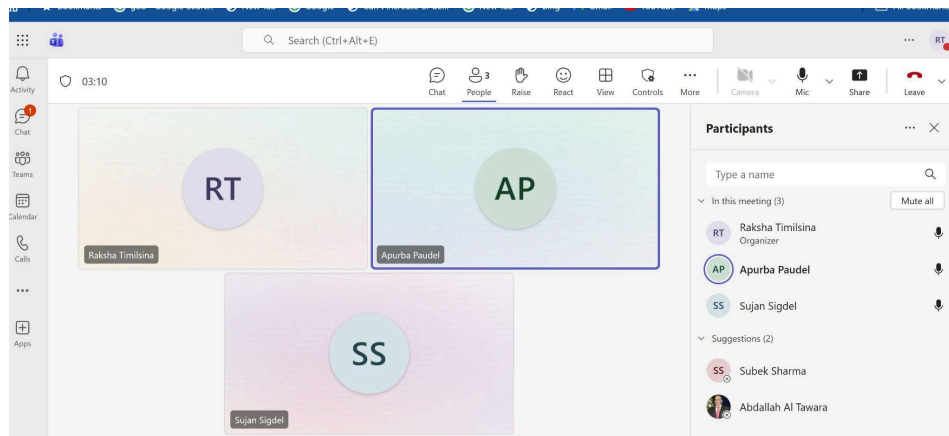


AP Apurba Paudel 1:48 AM

New

Question 1 explanation:

This program is run to encrypt, decrypt and verify text stored in external files. The programs first reads the content of `raw_text.txt` and processes it one character at a time. During encryption, each character is checked to determine whether it is a lowercase letter, uppercase letter, or a non-alphabetic character. Based on the



## 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 handled Question 2 by reading several CSV files, cleaning up the temperature data, organizing it properly, figuring out the average temperatures for each season, finding which stations had the biggest temperature differences, and calculating standard deviations to see how stable the temperatures were at each station.

**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 what the assignment asked for.

**Raksha Timilsina (s398446)** - I prepared the final report, gathered all the outputs and screenshots, made sure the documentation was easy to understand and neatly arranged, and prepared the report in the way it needed to be submitted.

## 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. We also 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] 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].
- [2] Stackoverflow, ‘Combining separate daily CSVs in pandas,’ [Online]. Available: <https://stackoverflow.com/questions/60923806/combining-separate-daily-csvs-in-pandas> [Accessed: 6-Jan-2026].
- [3] 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].
- [4] B. Solomon, ‘pandas groupby: Your Guide to grouping data in Python,’ [Online]. Available: <https://realpython.com/pandas-groupby/> [Accessed: 11-Jan-2026].
- [5] 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].