ASSIGNMENT 2

SOFTWARE NOW
HIT 137

Summer Semester

2025

Group Name: SYDN 18

Group Members:

ASMITA PANTA - S398448

NASLA MAHARJAN - S398425

NAYANDEEP SHRESTHA - S397732

RAJAT SHRESTHA - S398548

Contents

Questions

Question 1

Create a program that reads the text file "raw_text.txt", encrypts its contents using a simple encryption method, and writes the encrypted text to a new file "encrypted_text.txt". Then create a function to decrypt the content and a function to verify the decryption was successful.

Requirements

The encryption should take two user inputs (shift1, shift2), and follow these rules:

- For lowercase letters:
  - If the letter is in the first half of the alphabet (a-m): shift forward by shift1 * shift2 positions
  - If the letter is in the second half (n-z): shift backward by shift1 + shift2 positions
- For uppercase letters:
  - If the letter is in the first half (A-M): shift backward by shift1 positions
  - If the letter is in the second half (N-Z): shift forward by $shift2^2$ positions (shift2 squared)
- Other characters:
  - Spaces, tabs, newlines, special characters, and numbers remain unchanged

Main Functions to Implement:

Encryption function: Reads from "raw_text.txt" and writes encrypted content to "encrypted_text.txt".

Decryption function: Reads from "encrypted_text.txt" and writes the decrypted content to "decrypted_text.txt".

Verification function: Compares "raw_text.txt" with "decrypted_text.txt" and prints whether the decryption was successful or not.

Program Behavior:

When run, your program should automatically:

1. Prompt the user for shift1 and shift2 values
2. Encrypt the contents of "raw_text.txt"
3. Decrypt the encrypted file
4. Verify the decryption matches the original

Code: [1] [2] [3] [4]

```
import os

# //////////  ENCRYPTION FUNCTION //////////
def encryption(shift1, shift2):
    # opens raw file in read mode and creates (or opens) encrypted file in write mode
    with open("raw_text.txt", "r", encoding="utf-8") as raw_file, \
        open("encrypted_text.txt", "w", encoding="utf-8") as encrypted_file:

        for char in raw_file.read():
            # Encryption logic for lowercase letters
            if char.islower():
                if 'a' <= char <= 'm':
                    # %13 for wrapping around first half of alphabet
                    shift = (shift1 * shift2) % 13
                    # Applying forward shift with wrap around
                    new_char = chr((ord(char) - ord('a') + shift) % 13 + ord('a'))
                else:
                    shift = (shift1 + shift2) % 13
                    # Applying backward shift with wrap around
                    new_char = chr((ord(char) - ord('n') - shift) % 13 + ord('n'))

            # Encryption logic for uppercase letters
            elif char.isupper():
                if 'A' <= char <= 'M':
                    shift = shift1 % 13
                    new_char = chr((ord(char) - ord('A') - shift) % 13 + ord('A'))
                else:
                    shift = (shift2 ** 2) % 13
                    new_char = chr((ord(char) - ord('N') + shift) % 13 + ord('N'))

            # If character is not a letter, no change
            else:
                new_char = char

            encrypted_file.write(new_char)

    print(f"Encryption successfull")

# //////////  DECRYPTION FUNCTION //////////
def decryption(shift1, shift2):
    with open("encrypted_text.txt", "r", encoding = "utf-8") as encrypted_file, \
        open("decrypted_text.txt", "w", encoding = "utf-8") as decrypted_file:
```

```
        for char in encrypted_file.read():
            # Decryption logic for lowercase letters
            if char.islower():
                if 'a' <= char <= 'm':
                    shift = (shift1 * shift2) % 13
                    new_char = chr((ord(char) - ord('a') - shift) % 13 + ord('a'))
                else:
                    shift = (shift1 + shift2) % 13
                    new_char = chr((ord(char) - ord('n') + shift) % 13 + ord('n'))

            # Decryption logic for uppercase letters
            elif char.isupper():
                if 'A' <= char <= 'M':
                    shift = shift1 % 13
                    new_char = chr((ord(char) - ord('A') + shift) % 13 + ord('A'))
                else:
                    shift = (shift2 ** 2) % 13
                    new_char = chr((ord(char) - ord('N') - shift) % 13 + ord('N'))

            # If character is not a letter, no change
            else:
                new_char = char

            decrypted_file.write(new_char)

    print(f"Decryption successfull")

# //////////  VERIFICATION FUNCTION //////////
def verification():
    with open("raw_text.txt", "r", encoding="utf-8") as raw_file, \
        open("decrypted_text.txt", "r", encoding="utf-8") as decrypted_file:

        if raw_file.read() == decrypted_file.read():
            print(f"Verification successfull")
        else:
            print(f"Verification failed")

# //////////  MAIN FUNCTION //////////
def main():
    if not os.path.exists("raw_text.txt"):
        print("File not found for processing")
        return
    shift1 = int(input("Enter value for shift1: "))
    shift2 = int(input("Enter value for shift2: "))
```

```
    encryption(shift1, shift2)
    decryption(shift1, shift2)
    verification()

main()
```

Output:

```
dell@Nayan MINGW64 /d/CDU/Sum_SEM25/Software Now/assessment_2_HIT137/Q1 (Q1)
$ python3 Q1.py
Enter value for shift1: 1
Enter value for shift2: 2
Encryption successfull
Decryption successfull
Verification successfull

dell@Nayan MINGW64 /d/CDU/Sum_SEM25/Software Now/assessment_2_HIT137/Q1 (Q1)
$
```

Question 2

Create a program that analyses temperature data collected from multiple weather stations in Australia. The data is stored in multiple CSV files under a "temperatures" folder, with each file representing data from one year. Process ALL .csv files in the temperatures folder. Ignore missing temperature values (NaN) in calculations.

Main Functions to Implement:
Seasonal Average: Calculate the average temperature for each season across ALL stations and ALL years. Save the results to "average_temp.txt".

• Use Australian seasons: Summer (Dec-Feb), Autumn (Mar-May), Winter (Jun Aug), Spring (Sep-Nov)
• Output format example: "Summer: 28.5°C"

Temperature Range: Find the station(s) with the largest temperature range (difference between the highest and lowest temperature ever recorded at that station). Save the results to "largest_temp_range_station.txt".

• Output format example: "Station ABC: Range 45.2°C (Max: 48.3°C, Min: 3.1°C)"
• If multiple stations tie, list all of them

Temperature Stability: Find which station(s) have the most stable temperatures (smallest standard deviation) and which have the most variable temperatures (largest standard deviation). Save the results to "temperature_stability_stations.txt".

• Output format example:
o "Most Stable: Station XYZ: StdDev 2.3°C"
o "Most Variable: Station DEF: StdDev 12.8°C"
• If multiple stations tie, list all of them

Code: [5] [6] [7] [8] [9] [10]

```
import os
import pandas as pd
import numpy as np

Folder="temperatures"

#adds in list
months=["January", "February", "March", "April", "May", "June",
```

```
    "July", "August", "September", "October", "November", "December"]

#mapping months to seasons
seasons= {"December": "Summer", "January": "Summer", "February": "Summer",
    "March": "Autumn", "April": "Autumn", "May": "Autumn",
    "June": "Winter", "July": "Winter", "August": "Winter",
    "September": "Spring", "October": "Spring", "November": "Spring"}

data=[]
for file in os.listdir(Folder):
    if file.endswith(".csv"):
        data.append(pd.read_csv(os.path.join(Folder,file)))

df= pd.concat(data, ignore_index=True)# merging all the csv files into one single dataframe

# converting the wide data of the csv to long format. There is column for each
# month which we are changing to month a single column and Temperature with their temperature
value.
temp_df= df.melt(
    id_vars="STATION_NAME",
    value_vars=months,
    var_name="Month",
    value_name="Temperature"
).dropna()

temp_df['Season']=temp_df['Month'].map(seasons)

#season average
season_avg= temp_df.groupby("Season")['Temperature'].mean()

with open("average_temp.txt",'w') as f:  # writing output in a txt file
    for season, avg in season_avg.items():
        f.write(f"{season}:{avg:.1f}C\n")

#largest temperature range

station= temp_df.groupby("STATION_NAME")["Temperature"].agg(
    max_temp='max',
    min_temp='min'
) # using groupby to group station name and its temperature to find the max and the min
temperature of the station

station["range"]=station['max_temp']-station['min_temp'] # calculating the range

max_range=station['range'].max() # max range
```

```
largest_range_station=station[station["range"]==max_range]

with open("largest_temp_rnage_station.txt",'w') as f: # writing in a txt file
    for stations, row in largest_range_station.iterrows():
        f.write(
            f"{stations}: Range {row['range']:.1f}C "
            f"(Max: {row['max_temp']:.1f}C, Min: {row['min_temp']:.1f}C)\n"
        )

#temperature stability

sd= temp_df.groupby("STATION_NAME")["Temperature"].std() #calculating standard deviation

min_sd=sd.min()
max_sd=sd.max()

with open("temperature_stability_stations.txt","w")as f: # writing in a file
    f.write("Most Stable:\n")
    for station in sd[sd==min_sd].index:
        f.write(f"{station}:sd {min_sd:.1f}C\n")

    f.write("Most Variable:\n")
    for station in sd[sd==max_sd].index:
        f.write(f"{station}:sd {max_sd:.1f}C\n")
```

Output:

```
Autumn:27.3C
Spring:27.4C
Summer:32.1C
Winter:21.1C
```

```
BOURKE-AIRPORT-AWS: Range 24.4C (Max: 43.0C, Min: 18.5C)
```

```
Most Stable:
DARWIN-AIRPORT:standard deviation 1.2C
Most Variable:
WAGGA-WAGGA-AMO:standard deviation 6.9C
```

Question 3

Create a program that uses a recursive function to generate a geometric pattern using Python's turtle graphics. The pattern starts with a regular polygon and recursively modifies each edge to create intricate designs.

Pattern Generation Rules:
For each edge of the shape:
1. Divide the edge into three equal segments
2. Replace the middle segment with two sides of an equilateral triangle pointing inward (creating an indentation)
3. This transforms one straight edge into four smaller edges, each 1/3 the length of the original edge
4. Apply this same process recursively to each of the four new edges based on the specified depth

Visual Example:
Depth 0: Draw a straight line (no modification)
Depth 1: Line becomes: ——\/—— (indentation pointing inward)
Depth 2: Each of the 4 segments from depth 1 gets its own indentation
User Input Parameters:

The program should prompt the user for:
Number of sides: Determines the starting shape
Side length: The length of each edge of the initial polygon in pixels
Recursion depth: How many times to apply the pattern rules

Example Execution:
Enter the number of sides: 4
Enter the side length: 300
Enter the recursion depth: 3

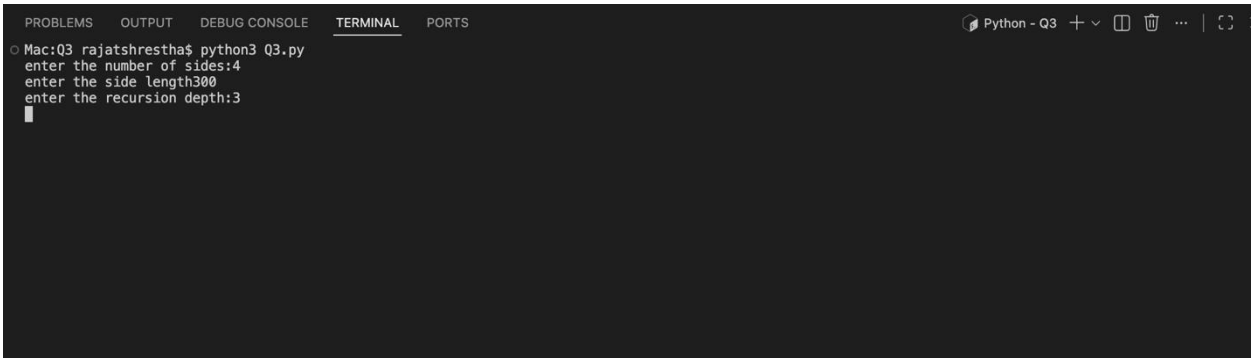
code: [11] [12] [13]

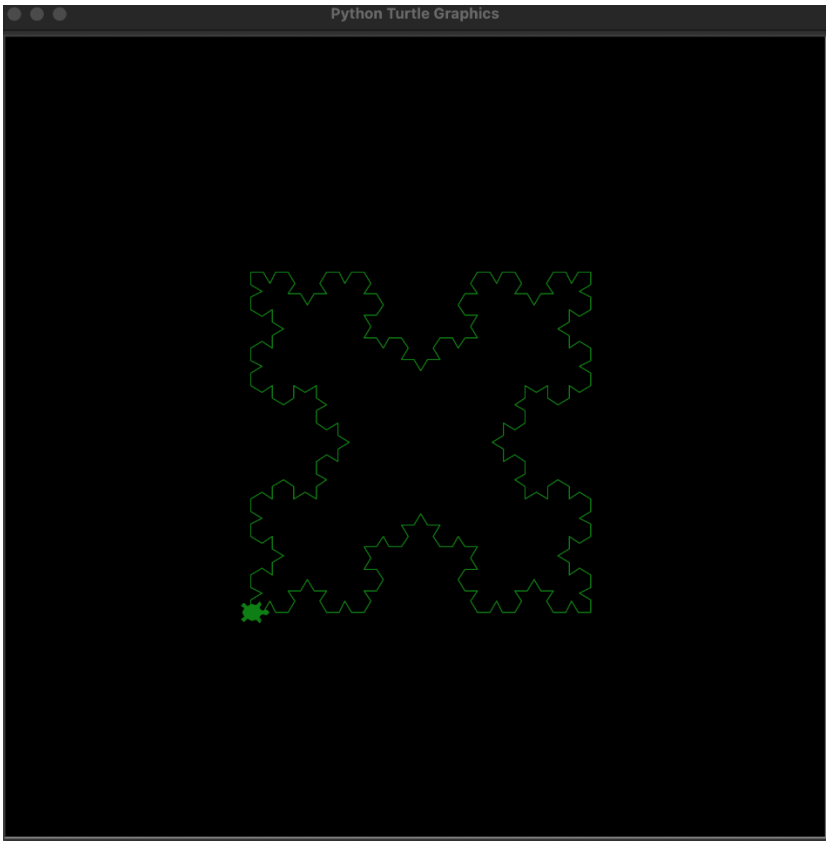
```
import turtle

# Recursive function to draw a single modified edge


def drawEdges(t, length, depth):
    if depth == 0:  # base case drawing a straight line
```

```
        t.forward(length)
    else:  # recursive case divided into 4 segments
        length /= 3
        drawEdges(t, length, depth - 1)
        t.left(60)
        drawEdges(t, length, depth - 1)
        t.right(120)
        drawEdges(t, length, depth - 1)
        t.left(60)
        drawEdges(t, length, depth - 1)

# ------------ Function to draw a polygon ---------


def drawPolygon(t, sides, length, depth):
    angle = 360 / sides
    for _ in range(sides):
        drawEdges(t, length, depth)
        t.left(angle)

# ------ Main Program ----------


def main():

    # --------- User Input ------------
    sides = int(input("enter the number of sides:"))
    length = int(input("enter the side length"))
    depth = int(input("enter the recursion depth:"))

    # ----- Setting Turtle ----------
    screen = turtle.Screen()
    screen.bgcolor("black")
    t=turtle.Turtle()
    t.speed(0)
    t.shape("turtle")
    turtle.bgcolor("black")
    t.color("green")

    # -------- Center the shape -----------
    t.penup()
    t.goto(-length/2, -length/2)
    t.pendown()

    # ------ Generate the pattern ----------
```

```
    drawPolygon(t,sides, length, depth)

    turtle.done()


main()
```

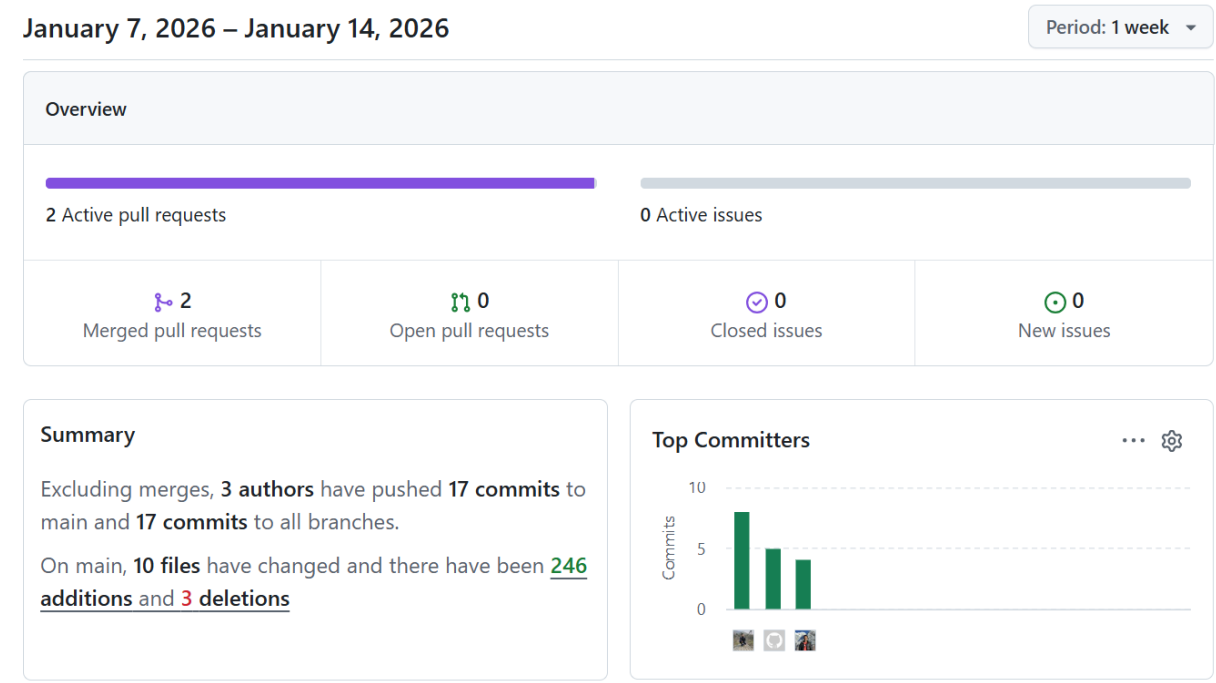
Output:

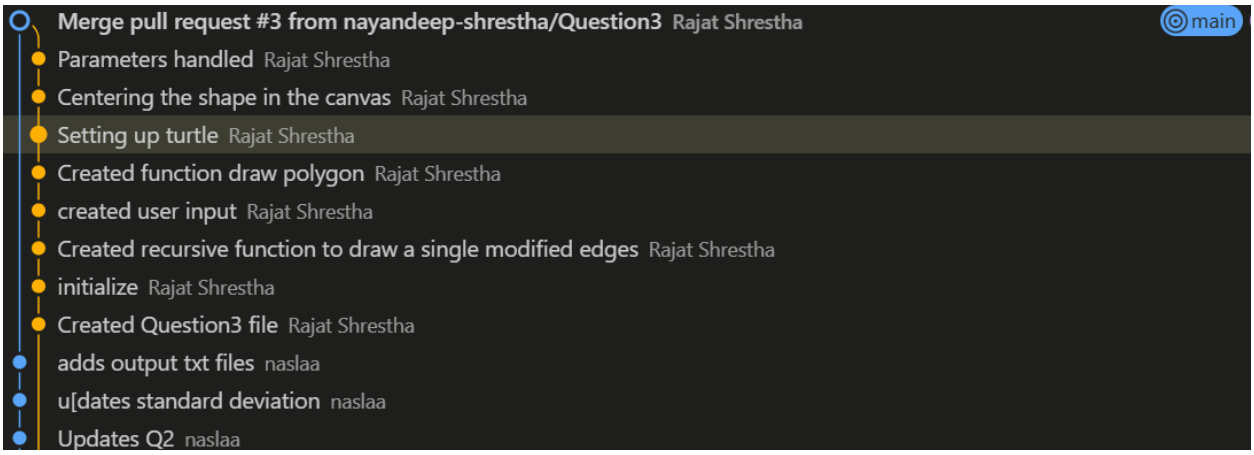


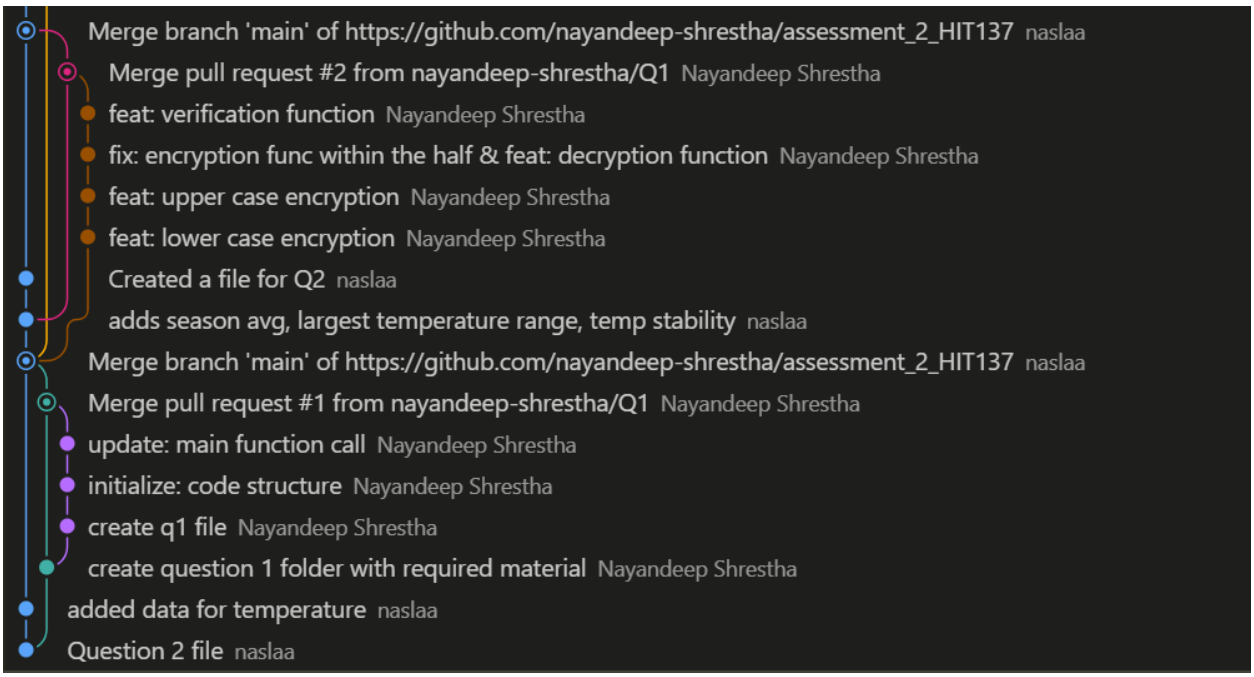


Collaboration Screenshots

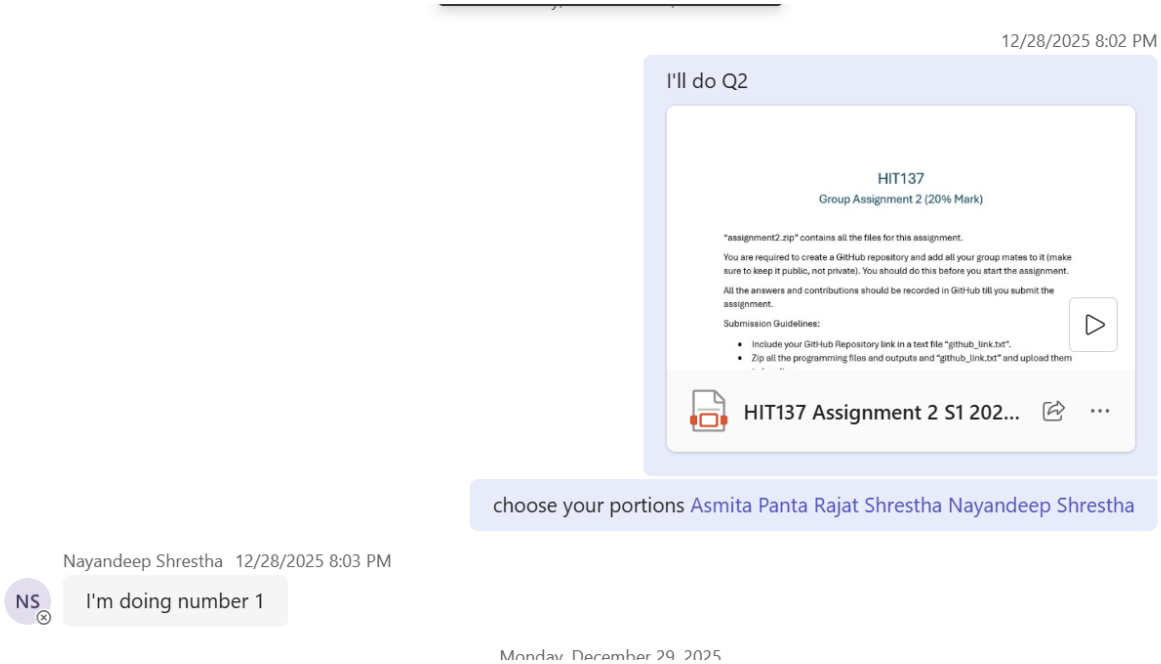
GitHub collaboration:







Teams collaboration:



Rajat Shrestha 12/29/2025 10:47 AM

RS i will be doing number 3 then

Last read

Yesterday

Asmita Panta Yesterday 8:42 PM

AP Hello guys, if you have completed the questions can you send it? I'll finish the documentation part.

Nayandeep Shrestha Yesterday 11:09 PM

NS Will that be okay if i send that tmrw?

Asmita Panta

Today

12:00 PM

I've completed Q2 and its updated in GitHub as well.

Rajat Shrestha 12:00 PM

RS i will push the code to my branch once i have completed Q3

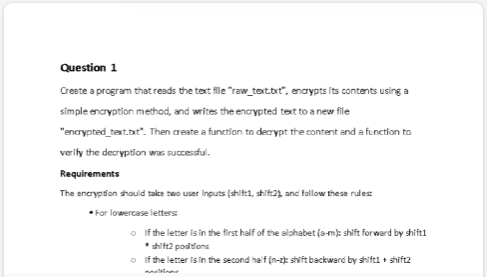
12:01 PM

okay sure thing

Nayandeep Shrestha

NS is this okay **Nasla Maharjan** ?

@



Question 1

Create a program that reads the text file "raw_text.txt", encrypts its contents using a simple encryption method, and writes the encrypted text to a new file "encrypted_text.txt". Then create a function to decrypt the content and a function to verify the decryption was successful.

Requirements

The encryption should take two user inputs (shift1, shift2), and follow these rules:

- For lowercase letters:
  - If the letter is in the first half of the alphabet (a-m): shift forward by shift1 * shift2 positions
  - If the letter is in the second half (n-z): shift backward by shift1 + shift2

Q1As2_Nayan.docx

12:10 PM

all good

thanks

Rajat Shrestha 7:01 PM

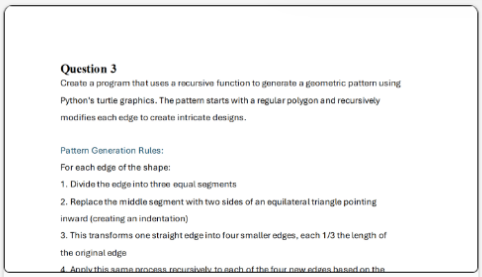
RS i have pushed the code to my branch

i will provide the docs after i complete it

Rajat Shrestha 8:05 PM

RS

Please have a look if there is any changes required **Nasla Maharjan** @



Question 3

Create a program that uses a recursive function to generate a geometric pattern using Python's turtle graphics. The pattern starts with a regular polygon and recursively modifies each edge to create intricate designs.

Pattern Generation Rules:

For each edge of the shape:

1. Divide the edge into three equal segments
2. Replace the middle segment with two sides of an equilateral triangle pointing inward (creating an indentation)
3. This transforms one straight edge into four smaller edges, each 1/3 the length of the original edge
4. Apply this same process recursively to each of the four new edges based on the

Question 3_Rajat.docx

8:05 PM

Okayy

All good thanks

Rajat Shrestha 8:06 PM

RS

okayy

Contributions

1. Asmita Panta - S398448

- Arranged all the citations in the document.
- Added collaboration screenshots and documented contributions.

2. Nasla Maharjan- S398425

- Code for question 2 was done.
- Pushed the updates of the question 2 code in GitHub
- Compiled all the documentation and added references in the documentation.
- Created the final document with all the topics.
- Uploaded the document in the GitHub.
- Submitted the assignment 2.

3. Nayandeep Shrestha - S397732

- Code for question 1 was done.
- Pushed the updates of the question 1 code in GitHub
- Created GitHub collaboration.

4. Rajat Shrestha- S398548

- Code for question 3 was done.
- Pushed the updates of the question 3 code in GitHub

AI Declaration

This assignment was done on our own with the help of the listed references. Any kind of AI tools or LLM has not been used for the content generation of this assignment.

References

[1] "GeeksforGeeks, "Caesar Cipher in Cryptography"," 02 06 2016. [Online]. Available: https://www.geeksforgeeks.org/ethical-hacking/caesar-cipher-in-cryptography/.

[2] "W3Schools, “Python File Open,”," 2019. . [Online]. Available: https://www.w3schools.com/python/python_file_open.asp.

[3] "W3Schools, “Python File Write,”," 2019. [Online]. Available: . https://www.w3schools.com/python/python_file_write.asp.

[4] "W3Schools, “Python File Open"," 2024. [Online]. Available: https://www.w3schools.com/python/python_file_handling.asp.

[5] "Datacamp "Pandas Groupby"," [Online]. Available: https://www.datacamp.com/tutorial/pandas-groupby.

[6] "Geeksforgeeks, "Pandas: melt function"," [Online]. Available: https://www.geeksforgeeks.org/python/python-pandas-melt/.

[7] "Stackover flow "import multiple csv files into pandas and concatenate into one dataframe"," [Online]. Available: https://stackoverflow.com/questions/20906474/import-multiple-csv-files-into-pandas-and-concatenate-into-one-dataframe.

[8] "Stackoverflow " Pandas: find the maximum range across all co;umn"," [Online]. Available: https://stackoverflow.com/questions/24748848/pandas-find-the-maximum-range-in-all-the-columns-of-dataframe.

[9] "W3school "Python File Write"," [Online]. Available: https://www.w3schools.com/python/python_file_write.asp.

[10] "reddit, "read multiple files from csv"," [Online]. Available: www.reddit.com/r/rstats/comments/s1cagl/how_to_read_multiple_csv_files_form_a_bundle_of_a/.

[11] "GeeksforGeeks, “Turtle Programming in Python,” GeeksforGeeks," 24 06 2017. [Online]. Available: https://www.geeksforgeeks.org/python/turtle-programming-python/.

[12] "GeeksforGeeks, “Introduction to Recursion,” GeeksforGeeks,," 11 01 2017.. [Online]. Available: https://www.geeksforgeeks.org/dsa/introduction-to-recursion-2/.

[13] "GeeksforGeeks, “Koch Curve or Koch Snowflake,” GeeksforGeeks," 24 09 2017. [Online]. Available: https://www.geeksforgeeks.org/dsa/koch-curve-koch-snowflake/.