# 1.1 Count the number of times a word occurs in a paragraph

*def* count\_word\_frequency(*paragraph*, *target*):

    words = *paragraph*.split() # splitting the paragraph

    word\_count = 0 # initialising a variable

    for word in words: # checking each target word in the splitted paragraph

        cleaned\_word = word.strip(".,!?\"'()").lower()

        if cleaned\_word == *target*.lower(): #Comparing  the word with the target word and update the counter

            word\_count += 1  # incrementing the word count

    return word\_count #returning the total count

paragraph = """ A payroll management system is software that enables employers to give salaries to their employees. It is a medium of showing their commitment towards the faculty members and fulfilling their obligations along with keeping financial records in order.

    A payroll management system is an online solution that helps the institutes to pay salaries to the faculty members working there. It is a way how they show their commitment to the faculty in form of benefits, appraisals, and paid leaves. It enables them to fulfill their obligations as stated by the government and keep financial records in order.

    """ # Paragraph in here

target\_word = input("Enter the word to find its frequency in the paragraph: ")  #taking input to find the target word

frequency = count\_word\_frequency(paragraph, target\_word) # Call the function to find the frequency of the target word

print(*f*"The word '{target\_word}' appeared --{frequency}-- time(s) in the given paragraph.")

# 1.2 Lines, words, and characters of a paragraph

print("\n--- Lines, words, and characters of a paragraph ---")

length=len(paragraph)

print(*f*"Number of characters: {length}") # displays the number of characters in a paragraph

para\_words=paragraph.split()

plength=len(para\_words)

print(*f*"Number of words: {plength}") #displays number of words in the paragraph

lines = paragraph.split('\n')

lines\_length=len(lines)

print(*f*"Number of lines: {lines\_length}") # displays number of lines divided by '\n'

# 1.3 arrange words in an alphabetical order

*def* sorted\_word(*word*):

    sorted\_word = ''.join(sorted(*word*))

    return sorted\_word

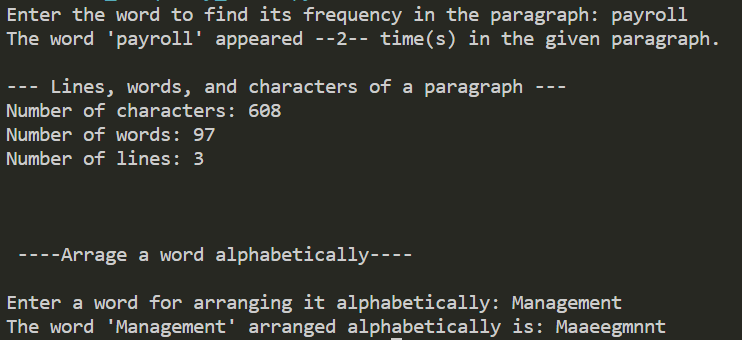
print("\n\n\n ----Arrage a word alphabetically---- \n")

word = input("Enter a word for arranging it alphabetically: ")

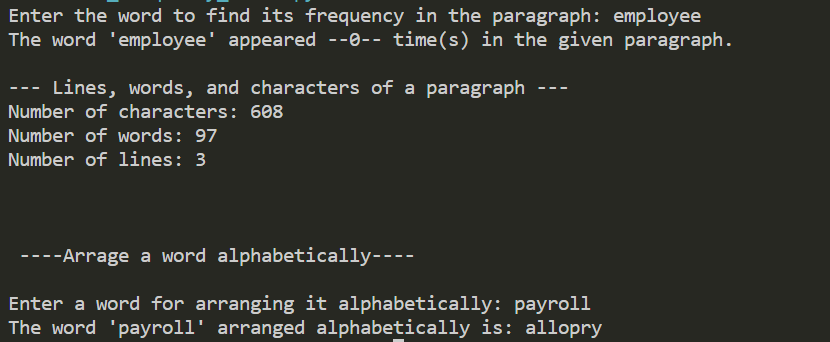
sorted\_word = sorted\_word(word)

print(*f*"The word '{word}' arranged alphabetically is: {sorted\_word}")

OUTPUT-1:



OUTPUT-2:



#2. Encryption of a string with a encryption key

*def* encrypt\_string(*string*, *encryption\_key*):

    encrypted\_string = ""

    for char in *string*:

        if char.isalpha():

            # Check if the character is uppercase or lowercase

            is\_uppercase = char.isupper()

            # Convert the character to its corresponding ASCII value

            ascii\_value = ord(char)

            # Add 'n' to the ASCII value to encrypt the character

            encrypted\_ascii = ascii\_value + *encryption\_key*

            # Adjust the ASCII value to keep it within the alphabet range

            if is\_uppercase:

                encrypted\_ascii = (encrypted\_ascii - 65) % 26 + 65

            else:

                encrypted\_ascii = (encrypted\_ascii - 97) % 26 + 97

            # Convert the encrypted ASCII value back to a character

            encrypted\_char = chr(encrypted\_ascii)

            encrypted\_string += encrypted\_char

        else:

            # If the character is not an alphabet, keep it as it is

            encrypted\_string += char

    return encrypted\_string

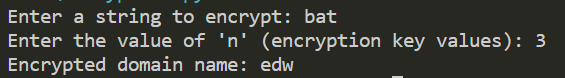
string = input("Enter a string to encrypt: ")

encryption\_key = int(input("Enter the value of 'n' (encryption key values): "))

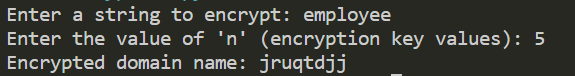
encrypted\_domain = encrypt\_string(string, encryption\_key)

print(*f*"Encrypted domain name: {encrypted\_domain}")

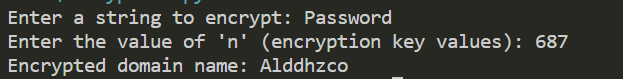
OUTPUT-1:



OUTPUT-2:



OUTPUT-3:



#3.Arranging List of tuples

data = [

('main st.', 4, 4000),

('elm st.', 1, 1200),

('pine st.', 2, 1600)

]

# Sort in ascending order by first numeric value (index 1)

sorted\_by\_first\_numeric = sorted(data, *key*=*lambda* *x*: *x*[1])

print("Sorted in ascending order by first numeric value:")

print(sorted\_by\_first\_numeric)

# Sort in descending order by second numeric value (index 2)

sorted\_by\_second\_numeric = sorted(data, *key*=*lambda* *x*: *x*[2], *reverse*=True)

print("\nSorted in descending order by second numeric value:")

print(sorted\_by\_second\_numeric)

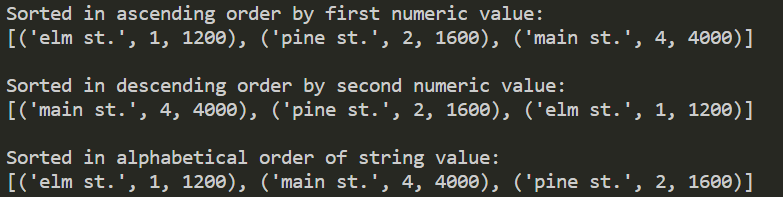
# Sort in alphabetical order of string value (index 0)

sorted\_alphabetically = sorted(data, *key*=*lambda* *x*: *x*[0])

print("\nSorted in alphabetical order of string value:")

print(sorted\_alphabetically)

OUTPUT-1:



# 4. function to calculate the wages pay function

*def* pay(*hourly\_wage*, *hours\_worked*):

    regular\_hours = min(*hours\_worked*, 40)

    overtime\_hours = max(0, *hours\_worked* - 40)

    regular\_pay = regular\_hours \* *hourly\_wage*

    overtime\_pay = overtime\_hours \* (*hourly\_wage* \* 1.5)

    total\_pay = regular\_pay + overtime\_pay

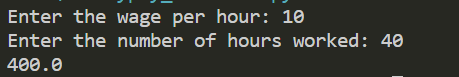
    return total\_pay

wage=int(input("Enter the wage per hour: "))

hours=int(input("Enter the number of hours worked: "))

print(pay(wage,hours))

OUTPUT-1:



OUTPUT-2:

