1 a,b

```python
import math

def euclidean_distance(point1, point2):
    """
    Calculate the Euclidean distance between two points in a 2D space.
    """
    x1, y1 = point1
    x2, y2 = point2
    distance = math.sqrt((x2 - x1)**2 + (y2 - y1)**2)
    return distance

def calculate_electricity_bill(units_consumed):
    """
    Calculate electricity bill based on the given rates and charges.
    """
    additional_charges = 0
    if units_consumed <= 100:
        additional_charges = 25.00
        rate = 1.5
    elif 101 <= units_consumed <= 200:
        additional_charges = 50.00
        rate = 2.5
    elif 201 <= units_consumed <= 300:
        additional_charges = 5.00
        rate = 4.0
    elif 301 <= units_consumed <= 350:
        additional_charges = 100.00
        rate = 7.0
    else:
        additional_charges = 1500.00
        rate = 0  # Fixed charge for units above 350

    total_charge = (units_consumed * rate) + additional_charges
    return total_charge
```

2 a,b

```
def analyze_email(email):
    vowels = "aeiouAEIOU"
    consonants = "bcdfghjklmnpqrstvwxyzBCDFGHJKLMNPQRSTVWXYZ"
    digits = "0123456789"

    vowel_count = 0
    consonant_count = 0
    digit_count = 0
    space_count = 0

    for char in email:
        if char in vowels:
            vowel_count += 1
        elif char in consonants:
            consonant_count += 1
        elif char in digits:
            digit_count += 1
        elif char.isspace():
            space_count += 1

    print(f"Vowels: {vowel_count}")
    print(f"Consonants: {consonant_count}")
    print(f"Digits: {digit_count}")
    print(f"White Spaces: {space_count}")

# Example usage:
sample_email = "example123@email.com"
analyze_email(sample_email)
# Program to find the sum of all primes below two million
def is_prime(n):
    if n < 2:
        return False
    for i in range(2, int(n**0.5) + 1):
        if n % i == 0:
            return False
    return True

def sum_of_primes_below_limit(limit):
    primes_sum = 0
    for num in range(2, limit):
        if is_prime(num):
            primes_sum += num
    return primes_sum

# Example usage:
limit = 2000000
result = sum_of_primes_below_limit(limit)
print(f"Sum of all primes below {limit}: {result}")
```

3 a,b,c,d

In [3]:
```python
def filter_numbers(numbers, x, y):
    result = [num for num in numbers if num % x == 0 and num % y != 0]
    return result

# Example usage:
numbers = [10, 15, 20, 25, 30, 35, 40,36]
x = 4
y = 5
filtered_numbers = filter_numbers(numbers, x, y)
print("Numbers divisible by", x, "but not by", y, ":", filtered_numbers)
```

```
Numbers divisible by 4 but not by 5 : [36]
```

In [4]:
```python
def sum_odd_and_even(numbers):
    odd_sum = sum(num for num in numbers if num % 2 != 0)
    even_sum = sum(num for num in numbers if num % 2 == 0)
    return odd_sum, even_sum

# Example usage:
numbers = [23, 10, 15, 14, 63]
odd_sum, even_sum = sum_odd_and_even(numbers)
print("Sum of odd numbers:", odd_sum)
print("Sum of even numbers:", even_sum)
```

```
Sum of odd numbers: 101
Sum of even numbers: 24
```

In [5]:
```python
def numbers_at_odd_index(numbers):
    result = [numbers[i] for i in range(len(numbers)) if i % 2 != 0]
    return result

# Example usage:
numbers = [10, 25, 30, 47, 56, 84, 96]
odd_index_numbers = numbers_at_odd_index(numbers)
print("Numbers at odd index positions:", odd_index_numbers)
```

```
Numbers at odd index positions: [25, 47, 84]
```

In [6]:
```python
def remove_duplicates(numbers):
    unique_numbers = list(set(numbers))
    return unique_numbers

# Example usage:
numbers = [10, 20, 40, 10, 50, 30, 20, 10, 80]
unique_numbers = remove_duplicates(numbers)
print("The unique list is:", unique_numbers)
```

4 a,b,c

In [7]:
```python
def filter_tuples_by_divisibility(tuples_list, k):
    result = [tpl for tpl in tuples_list if all(element % k == 0 for element in tpl)]
    return result

# Example usage:
test_list = [(6, 24, 12), (60, 12, 6), (12, 18, 21)]
K = 6
output = filter_tuples_by_divisibility(test_list, K)
print("Output:", output)
```

```
Output: [(6, 24, 12), (60, 12, 6)]
```

In [8]:
```python
def filter_uppercase_tuples(tuples_list):
    result = [tpl for tpl in tuples_list if all(isinstance(element, str) and element.isupper() for element in tpl)]
    return result

# Example usage:
test_list = [('GFG', 'IS', 'BEST'), ('GFg', 'AVERAGE'), ('GfG', ''), ('Gfg', 'CS')]
output = filter_uppercase_tuples(test_list)
print("Output:", output)
```

```
Output: [('GFG', 'IS', 'BEST')]
```

In [9]:
```python
def count_occurrences_in_tuple(input_tuple, input_list):
    result = {item: input_tuple.count(item) for item in input_list}
    return result

# Example usage:
input_tuple = ('a', 'a', 'c', 'b', 'd')
input_list = ['a', 'b']
output = count_occurrences_in_tuple(input_tuple, input_list)
print("Output:", output)
```

```
Output: {'a': 2, 'b': 1}
```

```
In [10]:
# 1. Create an empty dictionary with dict() method
my_dict = dict()

# 2. Add elements one at a time
my_dict['key1'] = 'value1'
my_dict['key2'] = 'value2'
my_dict['key3'] = 'value3'

# 3. Update existing key's value
my_dict['key2'] = 'new_value2'

# 4. Access an element using a key
print("Value of key1:", my_dict['key1'])

# 5. Access an element using get() method
print("Value of key2 using get():", my_dict.get('key2'))

# 6. Deleting a key-value using del() method
del my_dict['key3']

# Print the updated dictionary
print("Updated Dictionary:", my_dict)
```

```
Value of key1: value1
Value of key2 using get(): new_value2
Updated Dictionary: {'key1': 'value1', 'key2': 'new_value2'}
```

```
In [11]:
# Create a dictionary
my_dict = {'a': 10, 'b': 20, 'c': 30}

# Apply methods
# a. pop() method
value_a = my_dict.pop('a')

# b. popitem() method
key, value = my_dict.popitem()

# c. clear() method
my_dict.clear()

# Print results
print("Value of 'a' using pop():", value_a)
print("Popped item using popitem():", f"Key: {key}, Value: {value}")
print("Cleared Dictionary:", my_dict)
```

```
Value of 'a' using pop(): 10
Popped item using popitem(): Key: c, Value: 30
Cleared Dictionary: {}
```

```
In [12]:
# Given dictionary
my_dict = {'a': 10, 'b': 20, 'c': 30}

# Find the sum of all items in the dictionary
sum_of_items = sum(my_dict.values())

# Print the result
print("Sum of all items in the dictionary:", sum_of_items)
```

```
Sum of all items in the dictionary: 60
```

```
In [13]:
# Two dictionaries to merge
dict1 = {'a': 10, 'b': 20}
dict2 = {'b': 30, 'c': 40}

# Merge dictionaries using update() method
dict1.update(dict2)

# Print the merged dictionary
print("Merged Dictionary:", dict1)
```

```
Merged Dictionary: {'a': 10, 'b': 30, 'c': 40}
```

6 a

```python
class Car:
    def __init__(self, model_name, color, price, top_speed):
        self.model_name = model_name
        self.color = color
        self.price = price
        self.top_speed = top_speed

    def read_details(self):
        # Display details using print statements
        print(f"Model Name: {self.model_name}")
        print(f"Color: {self.color}")
        print(f"Price: {self.price}")
        print(f"Top Speed: {self.top_speed}")

# Example usage with constructor
car1 = Car("Toyota Camry", "Blue", 30000, 120)
car1.read_details()

# Example usage without constructor
car2 = Car(None, None, None, None)
car2.model_name = "Tesla Model S"
car2.color = "Red"
car2.price = 80000
car2.top_speed = 155
car2.read_details()
```

```
Model Name: Toyota Camry
Color: Blue
Price: 30000
Top Speed: 120
Model Name: Tesla Model S
Color: Red
Price: 80000
Top Speed: 155
```

6 b

In [15]:

```python
class Bank:
    def __init__(self, bank_name, number_cust):
        self.bank_name = bank_name
        self.number_cust = number_cust

    def display(self):
        print(f"Bank Name: {self.bank_name}")
        print(f"Number of Customers: {self.number_cust}")


class GovtBank(Bank):
    def __init__(self, bank_name, number_cust, branch_name, ifsc_code):
        super().__init__(bank_name, number_cust)
        self.branch_name = branch_name
        self.ifsc_code = ifsc_code

    def display(self):
        super().display()
        print(f"Branch Name: {self.branch_name}")
        print(f"IFSC Code: {self.ifsc_code}")


class PrivateBank(Bank):
    def __init__(self, bank_name, number_cust, branch_name, ifsc_code):
        super().__init__(bank_name, number_cust)
        self.branch_name = branch_name
        self.ifsc_code = ifsc_code

    def display(self):
        super().display()
        print(f"Branch Name: {self.branch_name}")
        print(f"IFSC Code: {self.ifsc_code}")


# Example usage
govt_bank = GovtBank("Public Bank", 10000, "Central Branch", "GOV123")
private_bank = PrivateBank("Private Bank", 5000, "Downtown Branch", "PVT456")

print("\nGovernment Bank Details:")
govt_bank.display()

print("\nPrivate Bank Details:")
private_bank.display()
```

```
Government Bank Details:
Bank Name: Public Bank
Number of Customers: 10000
Branch Name: Central Branch
IFSC Code: GOV123

Private Bank Details:
Bank Name: Private Bank
Number of Customers: 5000
Branch Name: Downtown Branch
IFSC Code: PVT456
```

7

In [16]:
```python
class Time:
    def __init__(self, hour=0, minute=0, second=0):
        self.hour = hour
        self.minute = minute
        self.second = second

    def __add__(self, other):
        total_seconds = self.hour * 3600 + self.minute * 60 + self.second
        total_seconds += other.hour * 3600 + other.minute * 60 + other.second

        new_hour, remainder = divmod(total_seconds, 3600)
        new_minute, new_second = divmod(remainder, 60)

        return Time(new_hour, new_minute, new_second)

    def __sub__(self, other):
        total_seconds = self.hour * 3600 + self.minute * 60 + self.second
        total_seconds -= other.hour * 3600 + other.minute * 60 + other.second

        new_hour, remainder = divmod(abs(total_seconds), 3600)
        new_minute, new_second = divmod(remainder, 60)

        return Time(new_hour, new_minute, new_second)

    def display(self):
        print(f"{self.hour:02d}:{self.minute:02d}:{self.second:02d}")


# Example usage
time1 = Time(5, 30, 45)
time2 = Time(2, 15, 20)

# Add two TIME objects
result_addition = time1 + time2
print("Addition Result:")
result_addition.display()

# Subtract two TIME objects
result_subtraction = time1 - time2
print("\nSubtraction Result:")
result_subtraction.display()
```

```
Addition Result:
07:46:05

Subtraction Result:
03:15:25
```

8 a,b,c
8

In [18]:
```python
def analyze_file(filename):
    try:
        with open(filename, 'r') as file:
            content = file.read()
            sentences = content.count('.') + content.count('!') + content.count('?')
            words = len(content.split())
            characters = len(content)

            print(f"Sentences: {sentences}")
            print(f"Words: {words}")
            print(f"Characters: {characters}")

    except FileNotFoundError:
        print(f"File {filename} not found.")

# Example usage
filename = input("Enter the filename: ")
analyze_file(filename)
```

In [19]:
```python
def copy_to_lowercase(file_source, file_target):
    try:
        with open(file_source, 'r') as source, open(file_target, 'w') as target:
            for line in source:
                target.write(line.lower())

        print(f"Content copied to {file_target} with only lowercase alphabets.")
        with open(file_target, 'r') as target:
            num_lines_copied = sum(1 for line in target)
            print(f"Number of lines copied: {num_lines_copied}")

    except FileNotFoundError:
        print(f"File {file_source} not found.")

# Example usage
source_filename = "practice.txt"
target_filename = "target.txt"
copy_to_lowercase(source_filename, target_filename)
```

```
Content copied to target.txt with only lowercase alphabets.
Number of lines copied: 2
```

In [*]:
```python
class Student:
    def __init__(self, name, roll_number, branch):
        self.name = name
        self.roll_number = roll_number
        self.branch = branch

def print_branch_students(students, branch):
    branch_students = [student for student in students if student.branch == branch]
    for student in branch_students:
        print(f"Name: {student.name}, Roll Number: {student.roll_number}, Branch: {student.branch}")

# Example usage
N = int(input("Enter the number of students: "))
students = []

for i in range(N):
    name = input(f"Enter name of student {i+1}: ")
    roll_number = input(f"Enter roll number of student {i+1}: ")
    branch = input(f"Enter branch of student {i+1}: ")
    students.append(Student(name, roll_number, branch))

branch_to_print = input("Enter the branch to print details: ")
print_branch_students(students, branch_to_print)
```

9 a,b,c,d

```
In [*]:  def is_symmetrical_and_palindrome(s):
             # Check if the string is symmetrical
             is_symmetrical = s[:len(s)//2] == s[len(s)//2:][::-1]

             # Check if the string is palindrome
             is_palindrome = s == s[::-1]

             return is_symmetrical, is_palindrome

         # Example usage:
         input_string = input("Enter a string: ")
         symmetrical, palindrome = is_symmetrical_and_palindrome(input_string)
         print(f"Symmetrical: {symmetrical}")
         print(f"Palindrome: {palindrome}")
```

```
In [ ]:  def count_vowels_and_print_except_es(s):
             vowels = "aeiouAEIOU"
             count_vowels = sum(1 for char in s if char in vowels)
             modified_string = "".join(char for char in s if char.lower() not in ['e', 's'])

             print(f"Number of vowels: {count_vowels}")
             print(f"Modified string (excluding 'e' and 's'): {modified_string}")

         # Example usage:
         input_string = input("Enter a string: ")
         count_vowels_and_print_except_es(input_string)
```

```
In [ ]:  def remove_initial_word(s):
             words = s.split(maxsplit=1)
             if len(words) > 1:
                 result = words[1]
             else:
                 result = ""
             return result

         # Example usage:
         input_text = input("Enter a line of text: ")
         new_text = remove_initial_word(input_text)
         print(f"Modified text (removed initial word): {new_text}")
```

```
In [ ]:  def letter_histogram(s):
             histogram = {}
             for char in s:
                 if char.isalpha():
                     histogram[char] = histogram.get(char, 0) + 1
             return histogram

         # Example usage:
         input_string = input("Enter a string: ")
         result_histogram = letter_histogram(input_string)
         print("Letter Histogram:")
         for char, count in result_histogram.items():
             print(f"{char}: {count}")
```