Programming with Python

Problem Set

KALINGA INSTITUTE OF INDUSTRIAL TECHNOLOGY (KIIT)

Deemed to be University U/S 3 of UGC Act, 1956

Dr. Parveen Malik
School of Electronics Engineering
KIIT Deemed to be University
Bhubeneswar, Odisha

Contents

1 Problems based upon Print function

1. Print the following lines
   (a) Sammy says, "Hello!"
   (b) *"E:\Chengjun Xie\checkpoints"*
   (c) Sammy's balloon is red.
   (d) *Sammy says,\"The balloon\'s color is red.\"*

2. **Simple Greeting Program** This program prompts the user for their name and then greets them.

```
name = input("What is your name? ")
print("Hello, " + name + "!")
```

3. **Reference a whole list to print function**: Write a python programme to print out a whole list by using format string method.

```
lst = ["Rohan Singh" ,"34","Bhubeneswar"]
print("My name is {} and my age is {}. I live in {}".format(*lst
                                    ))
# Alternatively with indexing
print("My name is {0} and my age is {1}. I live in {2}".format(*
                                    lst))
# With Two Lists
lst1 = ["Rohan Singh" ,"34","Bhubeneswar"]
lst2 = ["Aman" ,"32","Delhi"]
print("My name is {0[0]} and my age is {0[1]}. I live in {0[2]}
                                    wheras my friend's name is {1
                                    [0]} and his age is {1[1]}.
                                    He lives in {1[2]}".format(
                                    lst1,lst2))
```

4. **Sum of Two Numbers:** This program takes two numbers as input and prints their sum.

```
num1 = float(input("Enter first number: "))
num2 = float(input("Enter second number: "))
print("Sum:", num1 + num2)
```

5. **Displaying Multiplication Table:** This program displays the multiplication table of a given number.

```
num = int(input("Enter a number: "))
for i in range(1, 11):
    print(num, "x", i, "=", num * i)
```

6. **Fibonacci Sequence:** This program generates the Fibonacci sequence up to a specified number of terms.

```
n = int(input("Enter the number of terms: "))
a, b = 0, 1
for _ in range(n):
    print(a, end=" ")
    a, b = b, a + b
```

7. **Temperature Converter**: This program converts temperature from Celsius to Fahrenheit.

```
celsius = float(input("Enter temperature in Celsius: "))
fahrenheit = (celsius * 9/5) + 32
print("Temperature in Fahrenheit:", fahrenheit)
```

8. **Character Frequency Counter**

```
string = input("Enter a string: ")
frequency = {}
for char in string:
    if char in frequency:
        frequency[char] += 1
    else:
        frequency[char] = 1
for char, count in frequency.items():
    print(char, ":", count)
```

2 Problems based upon Numeric Data type

1. **Distance Calculator**: Implement a program that calculates the distance between two points in a two-dimensional plane. Prompt the user to input the coordinates of the two points (x1, y1) and (x2, y2), and output the distance between them.

```
x1 = float(input("Enter x-coordinate of first point: "))
y1 = float(input("Enter y-coordinate of first point: "))
x2 = float(input("Enter x-coordinate of second point: "))
y2 = float(input("Enter y-coordinate of second point: "))
distance = ((x2 - x1)**2 + (y2 - y1)**2)**0.5
print("Distance between the points:", distance)
```

2. **Quadratic Equation Solver**: Implement a program that solves a quadratic equation of the form $ax^2+bx+c = 0$. Prompt the user to input the values of coefficients a, b, and c, and output the roots of the equation. Implement a program that calculates the distance between two points in a

two-dimensional plane. Prompt the user to input the coordinates of the two points (x1, y1) and (x2, y2), and output the distance between them.

```
import cmath
a = float(input("Enter coefficient a: "))
b = float(input("Enter coefficient b: "))
c = float(input("Enter coefficient c: "))
# Calculate the discriminant
d = (b**2) - (4*a*c)
# Find two solutions
sol1 = (-b - cmath.sqrt(d)) / (2*a)
sol2 = (-b + cmath.sqrt(d)) / (2*a)
print("The solutions are:", sol1, "and", sol2)
```

3. **BMI Calculator**: Create a program that calculates the Body Mass Index (BMI) using the formula: $BMI = weight(kg)/(height(m))^2$. Prompt the user to input their weight (in kg) and height (in meters), and output their BMI.

```
weight = float(input("Enter weight in kg: "))
height = float(input("Enter height in meters: "))
bmi = weight / (height ** 2)
print("BMI:", bmi)
```

4. **Complex Number Arithmetic:** Implement a program that performs arithmetic operations on complex numbers. Prompt the user to input two complex numbers and output the results of addition, subtraction, multiplication, and division.

```
num1 = complex(input("Enter first complex number: "))
num2 = complex(input("Enter second complex number: "))
addition = num1 + num2
subtraction = num1 - num2
multiplication = num1 * num2
division = num1 / num2
print("Addition:", addition)
print("Subtraction:", subtraction)
print("Multiplication:", multiplication)
print("Division:", division)
```

5. **Age Classifier** Create a program that classifies a person's age into different categories (e.g., infant, child, teenager, adult) based on predefined age ranges. Prompt the user to input their age and output the corresponding category.

```
 age = int(input("Enter age: "))
if age <= 1:
    category = "infant"
```

```
elif 1 < age <= 12:
    category = "child"
elif 12 < age <= 19:
    category = "teenager"
else:
    category = "adult"
print("Category:", category)
```

3 Problems based upon String data type

1. **Palindrome Checker:** Write a program that checks whether a given string is a palindrome or not. A palindrome is a word, phrase, number, or other sequence of characters that reads the same forwards and backward (ignoring spaces, punctuation, and capitalization).

```
def is_palindrome(s):
    s = s.lower()
    return s == s[::-1]
string = input("Enter a string: ")
if is_palindrome(string):
    print("Palindrome!")
else:
    print("Not a palindrome.")
```

2. **String Reversal:** Implement a program that reverses a given string. For example, if the input string is "hello", the output should be "olleh".

```
string = input("Enter a string: ")
reversed_string = string[::-1]
print("Reversed string:", reversed_string)
```

3. **Vowel Counter:** Create a program that counts the number of vowels (a, e, i, o, u) in a given string. Ignore case sensitivity.

```
def count_vowels(s):
    vowels = 'aeiou'
    count = 0
    for char in s.lower():
        if char in vowels:
            count += 1
    return count

string = input("Enter a string: ")
print("Number of vowels:", count_vowels(string))
```

4. **Word Count:** Write a program that counts the number of words in a given sentence. Assume that words are separated by whitespace.

```python
string = input("Enter a sentence: ")
words = string.split()
print("Number of words:", len(words))
```

5. **Character Frequency Counter:** Develop a program that counts the frequency of each character in a given string and outputs the result as a dictionary.

```python
def count_characters(s):
    freq = {}
    for char in s:
        if char in freq:
            freq[char] += 1
        else:
            freq[char] = 1
    return freq
string = input("Enter a string: ")
print("Character frequency:", count_characters(string))
```

6. **String Concatenation:** Implement a program that concatenates two strings together and outputs the result.

```python
string1 = input("Enter the first string: ")
string2 = input("Enter the second string: ")
concatenated_string = string1 + string2
print("Concatenated string:", concatenated_string)
```

7. **Substring Checker:** Create a program that checks whether a given substring exists within a larger string. Output "Found" if the substring exists, otherwise output "Not Found".

```python
string = input("Enter a string: ")
substring = input("Enter a substring to search for: ")
if substring in string:
    print("Found!")
else:
    print("Not Found.")
```

8. **String Formatting:** Write a program that formats a given string into title case, where the first letter of each word is capitalized.

```python
string = input("Enter a string: ")
formatted_string = string.title()
print("Formatted string:", formatted_string)
```

9. **Anagram Checker:** Implement a program that checks whether two given strings are anagrams of each other. An anagram is a word or phrase formed by rearranging the letters of another word or phrase.

```
def is_anagram(s1, s2):
    return sorted(s1.lower()) == sorted(s2.lower())
string1 = input("Enter the first string: ")
string2 = input("Enter the second string: ")
if is_anagram(string1, string2):
    print("Anagrams!")
else:
    print("Not anagrams.")
```

10. **String Slicing:** Develop a program that takes a given string and extracts a substring based on user-defined start and end indices. Output the extracted substring.

```
string = input("Enter a string: ")
start_index = int(input("Enter the start index: "))
end_index = int(input("Enter the end index: "))
substring = string[start_index:end_index]
print("Substring:", substring)
```

11. **Password Strength Checker:** Write a program that checks the strength of a password entered by the user. The program should check if the password contains at least one uppercase letter, one lowercase letter, one digit, and one special character.

```
import re
def password_strength(password):
    if re.match(r'^(?=.*[a-z])(?=.*[A-Z])(?=.*\d)(?=.*[@\$!%*?&
                                   ])[A-Za-z\d@$!%*?&]{8,}$'
                                   , password):
        return "Strong"
    else:
        return "Weak"
password = input("Enter a password: ")
print("Password strength:", password_strength(password))
```

12. **String Encryption:** Develop a program that encrypts a given string using a simple encryption technique such as Caesar cipher or ROT13. Prompt the user to input a string and specify the encryption key.

```
    def encrypt_string(s, key):
    encrypted = ''
    for char in s:
        encrypted += chr(ord(char) + key)
    return encrypted
```

```
string = input("Enter a string: ")
key = int(input("Enter the encryption key: "))
print("Encrypted string:", encrypt_string(string, key))
```

13. **Text Analysis Tool**: Create a program that analyzes a given text and provides various statistics such as the number of words, number of sentences, average word length, most frequent word, etc.

```
 def text_analysis(text):
    words = text.split()
    num_words = len(words)
    num_sentences = text.count('.') + text.count('!') + text.
                                   count('?')
    avg_word_length = sum(len(word) for word in words) /
                                   num_words
    most_frequent_word = max(set(words), key=words.count)

    print("Number of words:", num_words)
    print("Number of sentences:", num_sentences)
    print("Average word length:", avg_word_length)
    print("Most frequent word:", most_frequent_word)

text = input("Enter some text: ")
text_analysis(text)
```

14. **String Compression:** Implement a program that compresses a given string by replacing consecutive repeated characters with a single character followed by the count of repetitions. For example, "aaabbbccc" would be compressed to "a3b3c3".

```
def compress_string(s):
    compressed = ''
    count = 1
    for i in range(len(s)-1):
        if s[i] == s[i+1]:
            count += 1
        else:
            compressed += s[i] + str(count)
            count = 1
    compressed += s[-1] + str(count)
    return compressed

string = input("Enter a string: ")
print("Compressed string:", compress_string(string))
```

15. **String Rotation Checker:** Write a program that checks if one string is a rotation of another. For example, "abcd" is a rotation of "cdab".

```
 def is_rotation(s1, s2):
    if len(s1) != len(s2):
        return False
    return s2 in s1 + s1

string1 = input("Enter the first string: ")
string2 = input("Enter the second string: ")
if is_rotation(string1, string2):
    print("String 2 is a rotation of string 1.")
else:
    print("String 2 is not a rotation of string 1.")
```

16. **Palindrome Substring Finder:** Develop a program that finds all palindromic substrings within a given string. Output the palindromic substrings along with their lengths.

```
def is_palindrome(s):
    return s == s[::-1]

def find_palindromes(s):
    palindromes = []
    for i in range(len(s)):
        for j in range(i+1, len(s)+1):
            if is_palindrome(s[i:j]):
                palindromes.append(s[i:j])
    return palindromes
string = input("Enter a string: ")
print("Palindromic substrings:", find_palindromes(string))
```

17. **Anagram Finder:** Create a program that finds all anagrams of a given word within a list of words. Prompt the user to input a word and a list of words, and output the anagrams found in the list.

```
def is_anagram(word1, word2):
    return sorted(word1.lower()) == sorted(word2.lower())

word = input("Enter a word: ")
word_list = input("Enter a list of words separated by space: ").
                                   split()
anagrams = [w for w in word_list if is_anagram(word, w)]
print("Anagrams of", word, ":", anagrams)
```

18. **String Permutation Generator:** Implement a program that generates all permutations of a given string. Output the permutations in lexicographically sorted order.

```
from itertools import permutations
```

```
string = input("Enter a string: ")
permutations = [''.join(p) for p in permutations(string)]
permutations.sort()
print("Permutations:", permutations)
```

19. **String Matching Algorithm:** Write a program that implements a string matching algorithm such as the Knuth-Morris-Pratt (KMP) algorithm or the Boyer-Moore algorithm. Prompt the user to input a text and a pattern, and output the indices where the pattern is found in the text.

```
 def kmp(text, pattern):
    def build_lps(pattern):
        lps = [0] * len(pattern)
        j = 0
        for i in range(1, len(pattern)):
            while j > 0 and pattern[i] != pattern[j]:
                j = lps[j-1]
            if pattern[i] == pattern[j]:
                j += 1
            lps[i] = j
        return lps

    lps = build_lps(pattern)
    indices = []
    j = 0
    for i in range(len(text)):
        while j > 0 and text[i] != pattern[j]:
            j = lps[j-1]
        if text[i] == pattern[j]:
            j += 1
        if j == len(pattern):
            indices.append(i - len(pattern) + 1)
            j = lps[j-1]
    return indices

text = input("Enter a text: ")
pattern = input("Enter a pattern to search for: ")
matches = kmp(text, pattern)
print("Pattern found at indices:", matches)
```

20. **String Transformation:** Develop a program that transforms a given string into another string by applying a series of transformations. For example, convert all lowercase letters to uppercase, reverse the string, or remove all vowels.

```
def transform_string(s, transformations):
    for transform in transformations:
        if transform == 'uppercase':
            s = s.upper()
        elif transform == 'lowercase':
            s = s.lower()
        elif transform == 'reverse':
            s = s[::-1]
        elif transform == 'remove_vowels':
            s = ''.join(char for char in s if char.lower() not
                                                in 'aeiou')
    return s
string = input("Enter a string: ")
transformations = input("Enter transformations separated by
                                        space: ").split()
transformed_string = transform_string(string, transformations)
print("Transformed string:", transformed_string)
```

4 Problems based upon Lists

1. **Unique Elements:** Write a program that takes a list as input and removes duplicate elements, returning a new list with only unique elements.

```
def unique_elements(lst):
    return list(set(lst))
# Example usage:
my_list = [1, 2, 3, 3, 4, 5, 5]
print(unique_elements(my_list))  # Output: [1, 2, 3, 4, 5]
```

2. **List Reversal:** Implement a program that reverses a given list, without using built-in reverse functions.

```
  def reverse_list(lst):
    return lst[::-1]
# Example usage:
my_list = [1, 2, 3, 4, 5]
print(reverse_list(my_list))  # Output: [5, 4, 3, 2, 1]
```

3. **List Intersection:** Create a program that takes two lists as input and returns a new list containing only the elements common to both lists.

```
def list_intersection(lst1, lst2):
    return list(set(lst1) & set(lst2))
# Example usage:
list1 = [1, 2, 3, 4, 5]
list2 = [4, 5, 6, 7, 8]
print(list_intersection(list1, list2))  # Output: [4, 5]
```

4. **List Sorting:** Write a program that sorts a list of integers in ascending order using the bubble sort algorithm.

```
def bubble_sort(lst):
    n = len(lst)
    for i in range(n):
        for j in range(0, n-i-1):
            if lst[j] > lst[j+1]:
                lst[j], lst[j+1] = lst[j+1], lst[j]
    return lst
# Example usage:
my_list = [64, 34, 25, 12, 22, 11, 90]
print(bubble_sort(my_list))  # Output: [11, 12, 22, 25, 34, 64,
                                   90]
```

5. **List Concatenation:** Develop a program that concatenates two lists together and returns the resulting list.

```
def concatenate_lists(lst1, lst2):
    return lst1 + lst2
# Example usage:
list1 = [1, 2, 3]
list2 = [4, 5, 6]
print(concatenate_lists(list1, list2))  # Output: [1, 2, 3, 4, 5
                                   , 6]
```

6. **List Palindrome Checker:** Implement a program that checks whether a given list is a palindrome (reads the same forwards and backward).

```
def is_palindrome(lst):
    return lst == lst[::-1]
# Example usage:
my_list = [1, 2, 3, 2, 1]
print(is_palindrome(my_list))  # Output: True
```

7. **List Element Removal:** Create a program that removes all occurrences of a specified element from a list and returns the modified list.

```
def remove_element(lst, element):
    return [x for x in lst if x != element]
# Example usage:
my_list = [1, 2, 3, 4, 2, 5]
element_to_remove = 2
print(remove_element(my_list, element_to_remove))  # Output: [1,
                                   3, 4, 5]
```

8. **List Chunking:** Write a program that chunks a given list into smaller lists of a specified size.

```python
def chunk_list(lst, size):
    return [lst[i:i+size] for i in range(0, len(lst), size)]
# Example usage:
my_list = [1, 2, 3, 4, 5, 6, 7, 8, 9]
chunk_size = 3
print(chunk_list(my_list, chunk_size))  # Output: [[1, 2, 3], [4
                                   , 5, 6], [7, 8, 9]]
```

9. **List Rotation:** Develop a program that rotates a given list to the left by a specified number of positions.

```python
    def rotate_list(lst, positions):
    return lst[positions % len(lst):] + lst[:positions % len(lst
                                          )]
# Example usage:
my_list = [1, 2, 3, 4, 5]
num_positions = 2
print(rotate_list(my_list, num_positions))  # Output: [3, 4, 5,
                                   1, 2]
```

10. **List Frequency Counter:** Create a program that counts the frequency of each element in a given list and returns a dictionary with the element counts.

```python
  def count_frequency(lst):
    freq_dict = {}
    for item in lst:
        if item in freq_dict:
            freq_dict[item] += 1
        else:
            freq_dict[item] = 1
```

11. **List Flattening:** Write a program that flattens a nested list structure into a single list. The input list can contain nested lists of arbitrary depth.

```python
def flatten_list(lst):
    flat_list = []
    for item in lst:
        if isinstance(item, list):
            flat_list.extend(flatten_list(item))
        else:
            flat_list.append(item)
    return flat_list

# Example usage:
nested_list = [1, [2, [3, 4], 5], 6]
print(flatten_list(nested_list))  # Output: [1, 2, 3, 4, 5, 6]
```

12. **List Partitioning:** Implement a program that partitions a given list into two sublists such that the sum of elements in each sublist is as close to each other as possible.

```
  def partition_list(lst):
    lst.sort()
    mid = len(lst) // 2
    return lst[:mid], lst[mid:]

# Example usage:
my_list = [1, 2, 3, 4, 5]
print(partition_list(my_list))  # Output: ([1, 2, 3], [4, 5])
```

13. **List Shuffle:** Create a program that shuffles the elements of a given list in-place using the Fisher-Yates shuffle algorithm.

```
import random
def shuffle_list(lst):
    for i in range(len(lst)-1, 0, -1):
        j = random.randint(0, i)
        lst[i], lst[j] = lst[j], lst[i]
# Example usage:
my_list = [1, 2, 3, 4, 5]
shuffle_list(my_list)
print(my_list)  # Output: [4, 2, 1, 3, 5] (randomly shuffled)
```

14. **List Rotation Without Modifying Original List:** Develop a program that rotates a given list to the left by a specified number of positions without modifying the original list.

```
def rotate_list(lst, positions):
    return lst[positions % len(lst):] + lst[:positions % len(lst
                                       )]
# Example usage:
my_list = [1, 2, 3, 4, 5]
num_positions = 2
print(rotate_list(my_list[:], num_positions))  # Output: [3, 4,
                                   5, 1, 2]
```

15. **List Intersection Without Duplicates:** Write a program that finds the intersection of two lists without including duplicate elements in the resulting list.

```
   def intersect_lists(lst1, lst2):
    return list(set(lst1) & set(lst2))
# Example usage:
list1 = [1, 2, 2, 3, 4]
list2 = [2, 3, 3, 4, 5]
print(intersect_lists(list1, list2))  # Output: [2, 3, 4]
```

16. **List Subsequence Checker:** Create a program that checks whether a given list is a subsequence of another list.

```
    def is_subsequence(lst1, lst2):
    i, j = 0, 0
    while i < len(lst1) and j < len(lst2):
        if lst1[i] == lst2[j]:
            i += 1
        j += 1
    return i == len(lst1)

# Example usage:
list1 = [1, 2, 3]
list2 = [1, 2, 3, 4, 5]
print(is_subsequence(list1, list2))  # Output: True
```

17. **List Permutation Checker:** Implement a program that checks whether a given list is a permutation of another list.

```
 from collections import Counter
def is_permutation(lst1, lst2):
    return Counter(lst1) == Counter(lst2)
# Example usage:
list1 = [1, 2, 3]
list2 = [3, 2, 1]
print(is_permutation(list1, list2))  # Output: True
```

18. **List Median Finder:** Write a program that finds the median of a given list of numbers. If the number of elements in the list is even, return the average of the two middle elements.

```
 def find_median(lst):
    sorted_lst = sorted(lst)
    n = len(sorted_lst)
    if n % 2 == 0:
        return (sorted_lst[n//2 - 1] + sorted_lst[n//2]) / 2
    else:
        return sorted_lst[n//2]
# Example usage:
my_list = [1, 2, 3, 4, 5]
print(find_median(my_list))  # Output: 3
```

19. **List Peak Element Finder:** Develop a program that finds a peak element in a given list. A peak element is an element that is greater than or equal to its neighbors.

```
    def find_peak(lst):
    for i in range(1, len(lst)-1):
        if lst[i] >= lst[i-1] and lst[i] >= lst[i+1]:
            return lst[i]
    return None
# Example usage:
my_list = [1, 3, 4, 5, 2]
print(find_peak(my_list))  # Output: 5
```

20. **List Maximum Subarray Sum Finder:** Create a program that finds the contiguous subarray within a one-dimensional list of numbers that has the largest sum.

```
    def max_subarray_sum(lst):
    max_sum = current_sum = lst[0]
    for num in lst[1:]:
        current_sum = max(num, current_sum + num)
        max_sum = max(max_sum, current_sum)
    return max_sum
# Example usage:
my_list = [-2, 1, -3, 4, -1, 2, 1, -5, 4]
print(max_subarray_sum(my_list))  # Output: 6 (from subarray [4,
                                    -1, 2, 1])
```

21. **Splitting and Joining the string** Write a program to do splitting and joining of a string.

```
A = "Rohan went to Delhi"
B = A.split(" ")
C = " ".join(B)
```

5 Problems based upon Tuples

1. **Swapping of two numbers**

```
a,b =1,2
a,b=b,a
```

2. **Tuple Unpacking:** Write a program that unpacks a given tuple into separate variables and prints them.

```
my_tuple = (1, 2, 3)
a, b, c = my_tuple
print(a, b, c)  # Output: 1 2 3
```

3. **Tuple Sorting:** Implement a program that sorts a list of tuples based on a specified key.

```
my_list = [(2, 'b'), (1, 'a'), (3, 'c')]
sorted_list = sorted(my_list, key=lambda x: x[0])
print(sorted_list)  # Output: [(1, 'a'), (2, 'b'), (3, 'c')]
```

4. **Tuple Concatenation:** Create a program that concatenates two tuples together and returns the resulting tuple.

```
tuple1 = (1, 2, 3)
tuple2 = ('a', 'b', 'c')
concatenated_tuple = tuple1 + tuple2
print(concatenated_tuple)  # Output: (1, 2, 3, 'a', 'b', 'c')
```

5. **Tuple Frequency Counter:** Write a program that counts the frequency of each element in a tuple and returns a dictionary with the element counts.

```
my_tuple = (1, 2, 2, 3, 3, 3, 4, 4, 4, 4)
freq_dict = {}
for item in my_tuple:
    if item in freq_dict:
        freq_dict[item] += 1
    else:
        freq_dict[item] = 1
print(freq_dict)  # Output: {1: 1, 2: 2, 3: 3, 4: 4}
```

6. **Tuple Packing and Unpacking:** Develop a program that packs and unpacks tuples, demonstrating the concept of tuple packing and unpacking.

```
# Tuple packing
my_tuple = 1, 2, 3
# Tuple unpacking
a, b, c = my_tuple
print(a, b, c)  # Output: 1 2 3
```

7. **Tuple Index Finder:** Implement a program that finds the index of a specified element in a tuple and returns it.

```
my_tuple = ('a', 'b', 'c', 'd', 'a')
index = my_tuple.index('c')
print(index)  # Output: 2
```

8. **Tuple Slicing:** Create a program that slices a given tuple based on user-defined start and end indices and returns the sliced tuple.

```
my_tuple = (1, 2, 3, 4, 5)
sliced_tuple = my_tuple[1:4]
print(sliced_tuple)  # Output: (2, 3, 4)
```

9. **Tuple Reversal:** Write a program that reverses a given tuple and returns the reversed tuple.

```
my_tuple = (1, 2, 3, 4, 5)
reversed_tuple = my_tuple[::-1]
print(reversed_tuple)  # Output: (5, 4, 3, 2, 1)
```

10. **Tuple Intersection:** Implement a program that finds the intersection of two tuples and returns a tuple containing the common elements.

```
tuple1 = (1, 2, 3, 4)
tuple2 = (3, 4, 5, 6)
intersection = tuple(set(tuple1) & set(tuple2))
print(intersection)  # Output: (3, 4)
```

11. **Tuple Membership Checker:** Create a program that checks whether a specified element exists in a given tuple and returns True or False accordingly.

```
my_tuple = (1, 2, 3, 4, 5)
element = 3
is_member = element in my_tuple
print(is_member)  # Output: True
```

6 Problems based upon Set

1. **Unique Elements Finder:** Write a program that takes a list as input and returns a set containing only the unique elements of the list.

```
def find_unique_elements(lst):
    return set(lst)
# Example usage:
my_list = [1, 2, 2, 3, 3, 4, 5, 5]
print(find_unique_elements(my_list))  # Output: {1, 2, 3, 4, 5}
```

2. **Common Elements Counter:** Implement a program that takes two lists as input and returns the count of elements that are common to both lists.

```
def count_common_elements(lst1, lst2):
    return len(set(lst1) & set(lst2))
# Example usage:
list1 = [1, 2, 3, 4, 5]
list2 = [4, 5, 6, 7, 8]
print(count_common_elements(list1, list2))  # Output: 2
```

3. **Set Membership Checker:** Write a program that checks whether a specified element exists in a given set and returns True or False accordingly.

```
def is_member(element, my_set):
    return element in my_set
# Example usage:
my_set = {1, 2, 3, 4, 5}
print(is_member(3, my_set))  # Output: True
```

4. **Set Power Set Generator:** Write a program that generates the power set of a given set, i.e., a set of all possible subsets, including the empty set and the set itself.

```
def powerset(my_set):
    from itertools import chain, combinations
    return list(chain.from_iterable(combinations(my_set, r) for
                                   r in range(len(my_set)+1)
                                   ))
# Example usage:
my_set = {1, 2, 3}
print(powerset(my_set))
# Output: [(), (1,), (2,), (3,), (1, 2), (1, 3), (2, 3), (1, 2,
                                   3)]
```

7 Problems based upon Dictionary

1. **Duplicate Counter:** Write a function that takes a list as input and returns a dictionary containing the count of each element in the list.

```
def count_duplicates(lst):
    count_dict = {}
    for item in lst:
        count_dict[item] = count_dict.get(item, 0) + 1
    return count_dict
# Example usage:
my_list = [1, 2, 2, 3, 3, 3, 4, 4, 4, 4]
print(count_duplicates(my_list))  # Output: {1: 1, 2: 2, 3: 3, 4
                                   : 4}
```

2. **Dictionary Merge:** Implement a function that merges two dictionaries into one.

```
def merge_dicts(dict1, dict2):
    return {**dict1, **dict2}
# Example usage:
dict1 = {'a': 1, 'b': 2}
dict2 = {'c': 3, 'd': 4}
print(merge_dicts(dict1, dict2))  # Output: {'a': 1, 'b': 2, 'c
                                   ': 3, 'd': 4}
```

3. **Nested Dictionary Flattener:** Write a function that takes a nested dictionary as input and flattens it into a single-level dictionary.

```
def flatten_dict(nested_dict, parent_key='', sep='_'):
    flat_dict = {}
    for key, value in nested_dict.items():
        new_key = parent_key + sep + key if parent_key else key
        if isinstance(value, dict):
            flat_dict.update(flatten_dict(value, new_key, sep))
        else:
            flat_dict[new_key] = value
    return flat_dict
# Example usage:
nested_dict = {'a': 1, 'b': {'c': 2, 'd': {'e': 3}}}
print(flatten_dict(nested_dict))  # Output: {'a': 1, 'b_c': 2, '
                                   b_d_e': 3}
```

4. **Key Finder:** Implement a function that searches for a key in a dictionary and returns its corresponding value. If the key is not found, return a default value.

```
def find_key(dictionary, key, default=None):
    return dictionary.get(key, default)
# Example usage:
my_dict = {'a': 1, 'b': 2, 'c': 3}
print(find_key(my_dict, 'b'))  # Output: 2
print(find_key(my_dict, 'd', 'Not Found'))  # Output: Not Found
```

5. **Dictionary Inverter:** Write a function that inverts the keys and values of a dictionary.

```
def invert_dict(dictionary):
    return {value: key for key, value in dictionary.items()}
# Example usage:
my_dict = {'a': 1, 'b': 2, 'c': 3}
print(invert_dict(my_dict)) # Output: {1: 'a', 2: 'b', 3: 'c'}
```

6. **Anagram Checker:** Implement a function that takes two strings as input and checks whether they are anagrams of each other.

```
def is_anagram(str1, str2):
    return sorted(str1) == sorted(str2)
# Example usage:
print(is_anagram('listen', 'silent'))  # Output: True
print(is_anagram('hello', 'world'))    # Output: False
```

7. **Letter Frequency Counter:** Write a function that takes a string as input and returns a dictionary containing the count of each letter in the string.

```
def count_letters(string):
    letter_count = {}
    for char in string:
        if char.isalpha():
            char = char.lower()
            letter_count[char] = letter_count.get(char, 0) + 1
    return letter_count
# Example usage:
my_string = 'Hello World'
print(count_letters(my_string))  # Output: {'h': 1, 'e': 1, 'l':
                                   3, 'o': 2, 'w': 1, 'r': 1, '
                                  d': 1}
```

8. **Dictionary Value Checker:** Implement a function that checks whether a specified value exists in any of the values in a dictionary.

```
def value_exists(dictionary, value):
    return value in dictionary.values()
# Example usage:
my_dict = {'a': 1, 'b': 2, 'c': 3}
print(value_exists(my_dict, 2))  # Output: True
print(value_exists(my_dict, 4))  # Output: False
```

9. **Dictionary Key Deleter:** Write a function that removes a specified key from a dictionary.

```
def delete_key(dictionary, key):
    dictionary.pop(key, None)
    return dictionary
# Example usage:
my_dict = {'a': 1, 'b': 2, 'c': 3}
print(delete_key(my_dict, 'b'))  # Output: {'a': 1, 'c': 3}
```

10. **Dictionary Subset Checker:** Implement a function that checks whether one dictionary is a subset of another dictionary.

```
def is_subset(subset, superset):
    return all(item in superset.items() for item in subset.items
                                            ())
# Example usage:
dict1 = {'a': 1, 'b': 2}
dict2 = {'a': 1, 'b': 2, 'c': 3}
print(is_subset(dict1, dict2))  # Output: True
```

11. **Dictionary Comparator:** Write a function that compares two dictionaries and returns True if they have the same key-value pairs.

```
def compare_dicts(dict1, dict2):
    return dict1 == dict2
# Example usage:
dict1 = {'a': 1, 'b': 2}
dict2 = {'b': 2, 'a': 1}
print(compare_dicts(dict1, dict2))  # Output: True
```

12. **Dictionary Intersection:** Implement a function that returns a new dictionary containing key-value pairs that are common to both input dictionaries.

```
def dict_intersection(dict1, dict2):
    return {key: dict1[key] for key in dict1 if key in dict2 and
                                             dict1[key] == dict2[key]
                                            }
# Example usage:
dict1 = {'a': 1, 'b': 2, 'c': 3}
dict2 = {'b': 2, 'c': 4, 'd': 5}
print(dict_intersection(dict1, dict2))  # Output: {'b': 2}
```

13. **Dictionary Key Finder:** Write a function that finds a key in a dictionary that has the maximum value.

```
def find_max_value_key(dictionary):
    return max(dictionary, key=dictionary.get)
# Example usage:
my_dict = {'a': 10, 'b': 20, 'c': 30}
print(find_max_value_key(my_dict))  # Output: 'c'
```

14. **Dictionary Key Sorter:** Implement a function that sorts a dictionary based on its keys.

```
def sort_dict_by_keys(dictionary):
    return dict(sorted(dictionary.items()))
# Example usage:
my_dict = {'b': 2, 'a': 1, 'c': 3}
```

```
print(sort_dict_by_keys(my_dict))  # Output: {'a': 1, 'b': 2, 'c
                                   ': 3}
```

15. **Dictionary Key Mapper:** Write a function that maps the keys of a dictionary using a specified function.

```
def map_keys(dictionary, func):
    return {func(key): value for key, value in dictionary.items
                                           ()}
# Example usage:
my_dict = {'a': 1, 'b': 2, 'c': 3}
print(map_keys(my_dict, str.upper))  # Output: {'A': 1, 'B': 2,
                                   'C': 3}
```

16. **Dictionary Key Renamer:** Implement a function that renames a specified key in a dictionary.

```
def rename_key(dictionary, old_key, new_key):
    if old_key in dictionary:
        dictionary[new_key] = dictionary.pop(old_key)
    return dictionary
# Example usage:
my_dict = {'a': 1, 'b': 2, 'c': 3}
print(rename_key(my_dict, 'b', 'x'))  # Output: {'a': 1, 'x': 2,
                                    'c': 3}
```

17. **Dictionary Default Getter:** Write a function that retrieves the value associated with a specified key in a dictionary. If the key is not found, return a default value.

```
 def get_value(dictionary, key, default=None):
    return dictionary.get(key, default)
# Example usage:
my_dict = {'a': 1, 'b': 2, 'c': 3}
print(get_value(my_dict, 'b'))      # Output: 2
print(get_value(my_dict, 'd', 0))   # Output: 0
```

18. **Dictionary Key Length Filter:** Implement a function that filters a dictionary based on the length of its keys.

```
def filter_dict_by_key_length(dictionary, length):
    return {key: value for key, value in dictionary.items() if
                                       len(key) == length}
# Example usage:
my_dict = {'apple': 10, 'banana': 20, 'kiwi': 30, 'orange': 40}
print(filter_dict_by_key_length(my_dict, 5))  # Output: {'apple
                                   ': 10, 'kiwi': 30}
```

19. **Dictionary Key Validator:** Write a function that validates whether all keys in a dictionary are of a specified type.

```
def validate_keys(dictionary, key_type):
    return all(isinstance(key, key_type) for key in dictionary)
# Example usage:
my_dict = {'a': 1, 'b': 2, 'c': 3}
print(validate_keys(my_dict, str))  # Output: True
```

20. **Dictionary Value Transformer:** Implement a function that transforms the values of a dictionary using a specified function.

```
def transform_values(dictionary, func):
    return {key: func(value) for key, value in dictionary.items
                                            ()}
# Example usage:
my_dict = {'a': 1, 'b': 2, 'c': 3}
print(transform_values(my_dict, lambda x: x**2))  # Output: {'a
                                         ': 1, 'b': 4, 'c': 9}
```

21. **Hybrid Example of using dictionaries and lists.** Implement a function that show hybrid usage of dictionaries and lists.

```
# Python list & map structures
customers = [{"name" : "Jeff & Tracy Heaton " , "pets" : ["
                                   Wynton","Cricket","Hickory"]}
                                   ,{"name": "John Smith","pets"
                                   : ["rover"] } ,{"name": "Jane
                                    Doe"}]
print(customers)
for customer in customers:
    print( f"{customer['name']} : {customer.get('pets','no pets
                                       ')}")
```

8 Miscellaneous

1. Write a python program by reading the text and find the maximum count of each word

```
with open('shakespeare.txt','r') as f:
    A = f.read()
    B=A.split()
words_count =dict() # or words_count={}
for word in B:
    word = word.strip('.,!?').lower()
    if word:
        # Update word count dictionary
        words_count[word] = words_count.get(word, 0) + 1

# Find the maximum count and corresponding word(s)
max_count = max(words_count.values())
max_words = [word for word, count in words_count.items() if
                                   count == max_count]
```