Here are some logical problems for you to solve in Python:

**1. Prime Number Check**

Write a function that checks whether a given number is prime or not.

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

# Example usage:

print(is\_prime(29)) # Output: True

**2. Unique Elements in a List**

Write a function that takes a list and returns a new list containing only unique elements.

def unique\_elements(lst):

return list(set(lst))

# Example usage:

print(unique\_elements([1, 2, 2, 3, 4, 4, 5])) # Output: [1, 2, 3, 4, 5]

**3. Find the Missing Number in a Sequence**

Given a list of numbers from 1 to N with one missing number, find the missing number.

def find\_missing\_number(arr, n):

total\_sum = n \* (n + 1) // 2

return total\_sum - sum(arr)

# Example usage:

print(find\_missing\_number([1, 2, 4, 5, 6], 6)) # Output: 3

**4. Check if a Number is an Armstrong Number**

An Armstrong number (also known as Narcissistic number) is a number that is equal to the sum of its own digits each raised to the power of the number of digits.

def is\_armstrong(n):

num\_str = str(n)

num\_digits = len(num\_str)

return n == sum(int(digit) \*\* num\_digits for digit in num\_str)

# Example usage:

print(is\_armstrong(153)) # Output: True

**5. Find the First Non-Repeating Character in a String**

Write a function that returns the first non-repeating character in a string.

from collections import Counter

def first\_non\_repeating(s):

freq = Counter(s)

for char in s:

if freq[char] == 1:

return char

return None

# Example usage:

print(first\_non\_repeating("swiss")) # Output: "w"

**6. Reverse a Number**

Write a function that reverses an integer.

def reverse\_number(n):

return int(str(n)[::-1]) if n >= 0 else -int(str(-n)[::-1])

# Example usage:

print(reverse\_number(1234)) # Output: 4321

print(reverse\_number(-567)) # Output: -765

**7. Find the Second Largest Number in a List**

Write a function to find the second largest number in a list.

def second\_largest(lst):

unique\_numbers = list(set(lst))

unique\_numbers.sort(reverse=True)

return unique\_numbers[1] if len(unique\_numbers) > 1 else None

# Example usage:

print(second\_largest([10, 20, 4, 45, 99])) # Output: 45

**8. Count the Frequency of Words in a String**

Given a string, count the occurrence of each word.

from collections import Counter

def word\_frequency(s):

words = s.split()

return dict(Counter(words))

# Example usage:

print(word\_frequency("apple banana apple orange banana apple"))

# Output: {'apple': 3, 'banana': 2, 'orange': 1}

**9. Find Pairs with a Given Sum in a List**

Given a list and a target sum, find all pairs that sum up to the target.

def find\_pairs(lst, target):

pairs = []

seen = set()

for num in lst:

if target - num in seen:

pairs.append((num, target - num))

seen.add(num)

return pairs

# Example usage:

print(find\_pairs([1, 2, 3, 4, 5, 6], 7)) # Output: [(4, 3), (5, 2), (6, 1)]

**10. Check if a String is a Palindrome**

Write a function to check if a string reads the same forward and backward.

def is\_palindrome(s):

return s == s[::-1]

# Example usage:

print(is\_palindrome("madam")) # Output: True

print(is\_palindrome("hello")) # Output: False

Here are some more challenging logical problems in Python:

**1. Find the GCD (Greatest Common Divisor) of Two Numbers**

Write a function to compute the greatest common divisor (GCD) of two numbers using the Euclidean algorithm.

def gcd(a, b):

while b:

a, b = b, a % b

return a

# Example usage:

print(gcd(56, 98)) # Output: 14

**2. Check if a Number is a Perfect Number**

A perfect number is a number that is equal to the sum of its proper divisors (excluding itself).

def is\_perfect(n):

return n == sum(i for i in range(1, n) if n % i == 0)

# Example usage:

print(is\_perfect(28)) # Output: True (1 + 2 + 4 + 7 + 14 = 28)

**3. Find the Longest Consecutive Sequence in an Array**

Given an unsorted array, find the length of the longest sequence of consecutive elements.

def longest\_consecutive(nums):

nums = set(nums)

longest = 0

for num in nums:

if num - 1 not in nums:

length = 1

while num + length in nums:

length += 1

longest = max(longest, length)

return longest

# Example usage:

print(longest\_consecutive([100, 4, 200, 1, 3, 2])) # Output: 4 (sequence: 1, 2, 3, 4)

**4. Generate Pascal’s Triangle**

Write a function to generate Pascal’s Triangle up to n rows.

def pascal\_triangle(n):

triangle = [[1]]

for i in range(1, n):

row = [1] + [triangle[i-1][j] + triangle[i-1][j+1] for j in range(len(triangle[i-1])-1)] + [1]

triangle.append(row)

return triangle

# Example usage:

for row in pascal\_triangle(5):

print(row)

# Output:

# [1]

# [1, 1]

# [1, 2, 1]

# [1, 3, 3, 1]

# [1, 4, 6, 4, 1]

**5. Check if Two Strings are Anagrams**

Write a function to check if two given strings are anagrams (contain the same characters in different order).

from collections import Counter

def is\_anagram(s1, s2):

return Counter(s1) == Counter(s2)

# Example usage:

print(is\_anagram("listen", "silent")) # Output: True

print(is\_anagram("hello", "world")) # Output: False

**6. Find the Majority Element in an Array**

The majority element is the element that appears more than n/2 times in an array.

from collections import Counter

def majority\_element(nums):

count = Counter(nums)

for key, value in count.items():

if value > len(nums) // 2:

return key

return None

# Example usage:

print(majority\_element([3, 3, 4, 2, 3, 3, 3])) # Output: 3

**7. Find All Prime Factors of a Number**

Write a function to return a list of all prime factors of a number.

def prime\_factors(n):

factors = []

i = 2

while i \* i <= n:

while n % i == 0:

factors.append(i)

n //= i

i += 1

if n > 1:

factors.append(n)

return factors

# Example usage:

print(prime\_factors(84)) # Output: [2, 2, 3, 7]

**8. Find All Subsets of a Given Set**

Given a list of unique numbers, generate all possible subsets (power set).

def subsets(nums):

result = [[]]

for num in nums:

result += [curr + [num] for curr in result]

return result

# Example usage:

print(subsets([1, 2, 3]))

# Output:

# [[], [1], [2], [1, 2], [3], [1, 3], [2, 3], [1, 2, 3]]

**9. Find the Longest Palindromic Substring**

Write a function to find the longest palindromic substring in a given string.

def longest\_palindrome(s):

def expand\_around\_center(left, right):

while left >= 0 and right < len(s) and s[left] == s[right]:

left -= 1

right += 1

return s[left + 1:right]

longest = ""

for i in range(len(s)):

odd = expand\_around\_center(i, i)

even = expand\_around\_center(i, i + 1)

longest = max(longest, odd, even, key=len)

return longest

# Example usage:

print(longest\_palindrome("babad")) # Output: "bab" or "aba"

**10. Find the Next Greater Permutation of a Number**

Given a number represented as a list of digits, find the next lexicographically greater permutation.

def next\_permutation(nums):

i = j = len(nums) - 1

while i > 0 and nums[i-1] >= nums[i]:

i -= 1

if i == 0:

return nums[::-1] # No next permutation, return sorted list

k = i - 1 # Find the pivot

while nums[j] <= nums[k]:

j -= 1

nums[k], nums[j] = nums[j], nums[k] # Swap pivot and next greater element

nums[i:] = reversed(nums[i:]) # Reverse the remaining part

return nums

# Example usage:

print(next\_permutation([1, 2, 3])) # Output: [1, 3, 2]

print(next\_permutation([3, 2, 1])) # Output: [1, 2, 3]

Here’s the **solution** to generate the Fibonacci sequence in Python.

### ****Problem:****

Generate the first n Fibonacci numbers.

### ****Solution (Iterative Approach)****

def fibonacci(n):

a, b = 0, 1

for \_ in range(n):

print(a, end=" ")

a, b = b, a + b

# Example usage:

fibonacci(5) # Output: 0 1 1 2 3

### ****Explanation:****

1. Start with a = 0 and b = 1 (first two Fibonacci numbers).
2. Print a, then update a and b (b becomes the next term, and a + b becomes the next b).
3. Repeat the process n times.

Here’s the **recursive solution** to generate the Fibonacci sequence.

**Solution (Recursive Approach)**

def fibonacci\_recursive(n):

if n <= 0:

return 0

elif n == 1:

return 1

return fibonacci\_recursive(n-1) + fibonacci\_recursive(n-2)

# Print the first n Fibonacci numbers

def print\_fibonacci(n):

for i in range(n):

print(fibonacci\_recursive(i), end=" ")

# Example usage:

print\_fibonacci(5) # Output: 0 1 1 2 3

**Explanation:**

1. The fibonacci\_recursive(n) function calculates the Fibonacci number for n using recursion.
2. The print\_fibonacci(n) function prints the first n Fibonacci numbers by calling fibonacci\_recursive(i) for each index.
3. The base cases ensure that for n = 0 we return 0, and for n = 1 we return 1.
4. Otherwise, it recursively computes F(n) = F(n-1) + F(n-2).

Here are **practical problems** based on your topics:

### ****1. OOPS Concepts (Classes & Objects)****

**Problem:**  
Create a BankAccount class with methods to deposit, withdraw, and check balance.

**Example:**

class BankAccount:

def \_\_init\_\_(self, account\_holder, balance=0):

self.account\_holder = account\_holder

self.balance = balance

def deposit(self, amount):

self.balance += amount

print(f"Deposited {amount}. New balance: {self.balance}")

def withdraw(self, amount):

if amount > self.balance:

print("Insufficient funds!")

else:

self.balance -= amount

print(f"Withdrew {amount}. New balance: {self.balance}")

# Example usage:

account = BankAccount("John Doe", 500)

account.deposit(200)

account.withdraw(100)

### ****2. Constructor****

**Problem:**  
Create a Student class with a constructor that initializes name and age.

**Example:**

class Student:

def \_\_init\_\_(self, name, age):

self.name = name

self.age = age

def display(self):

print(f"Student Name: {self.name}, Age: {self.age}")

# Example usage:

student = Student("Alice", 21)

student.display()

### ****3. Destructor****

**Problem:**  
Create a class where a destructor message is printed when the object is deleted.

**Example:**

class Demo:

def \_\_init\_\_(self):

print("Object Created")

def \_\_del\_\_(self):

print("Object Destroyed")

# Example usage:

obj = Demo()

del obj # This triggers the destructor

### ****4. Generator****

**Problem:**  
Create a generator that yields the first n even numbers.

**Example:**

def even\_numbers(n):

for i in range(n):

yield i \* 2

# Example usage:

gen = even\_numbers(5)

for num in gen:

print(num) # Output: 0 2 4 6 8

### ****5. Iterator****

**Problem:**  
Create a custom iterator that generates the first n squares.

**Example:**

class Squares:

def \_\_init\_\_(self, n):

self.n = n

self.current = 0

def \_\_iter\_\_(self):

return self

def \_\_next\_\_(self):

if self.current >= self.n:

raise StopIteration

result = self.current \*\* 2

self.current += 1

return result

# Example usage:

squares = Squares(5)

for num in squares:

print(num) # Output: 0 1 4 9 16

### ****6. Data Types (Practical Coding)****

#### **a. Dictionary - Word Frequency Counter**

**Problem:**  
Count the occurrences of words in a sentence.

**Example:**

def word\_count(sentence):

words = sentence.split()

word\_freq = {}

for word in words:

word\_freq[word] = word\_freq.get(word, 0) + 1

return word\_freq

# Example usage:

sentence = "apple banana apple orange banana apple"

print(word\_count(sentence)) # Output: {'apple': 3, 'banana': 2, 'orange': 1}

#### **b. Tuples - Unpacking**

**Problem:**  
Swap two numbers using tuple unpacking.

**Example:**

a, b = 5, 10

a, b = b, a # Swap values

print(a, b) # Output: 10 5

#### **c. List - Remove Duplicates**

**Problem:**  
Write a program to remove duplicates from a list.

**Example:**

def remove\_duplicates(lst):

return list(set(lst))

# Example usage:

nums = [1, 2, 2, 3, 4, 4, 5]

print(remove\_duplicates(nums)) # Output: [1, 2, 3, 4, 5]

Here are some **practical problems** based on different Python **data types**:

## **1. List Problems**

### ****a. Find the Second Largest Number in a List****

🔹 **Problem:** Write a function to find the second largest number in a given list.

**Example:**

def second\_largest(lst):

unique\_numbers = list(set(lst)) # Remove duplicates

unique\_numbers.sort(reverse=True)

return unique\_numbers[1] if len(unique\_numbers) > 1 else None

# Example usage:

nums = [10, 20, 4, 45, 99, 45]

print(second\_largest(nums)) # Output: 45

### ****b. Reverse a List Without Using**** reverse() ****or**** [::-1]

🔹 **Problem:** Reverse a list using a loop.

**Example:**

def reverse\_list(lst):

rev = []

for i in range(len(lst) - 1, -1, -1):

rev.append(lst[i])

return rev

# Example usage:

nums = [1, 2, 3, 4, 5]

print(reverse\_list(nums)) # Output: [5, 4, 3, 2, 1]

### ****c. Find the Most Frequent Element in a List****

🔹 **Problem:** Given a list, find the most occurring element.

**Example:**

def most\_frequent(lst):

return max(set(lst), key=lst.count)

# Example usage:

nums = [1, 3, 2, 3, 4, 3, 5, 3]

print(most\_frequent(nums)) # Output: 3

## **2. Tuple Problems**

### ****a. Swap Two Variables Without Using a Third Variable****

🔹 **Problem:** Swap two numbers using tuple unpacking.

**Example:**

a, b = 10, 20

a, b = b, a

print(a, b) # Output: 20 10

### ****b. Convert a Tuple to a String****

🔹 **Problem:** Convert a tuple of characters into a single string.

**Example:**

def tuple\_to\_string(tpl):

return ''.join(tpl)

# Example usage:

t = ('H', 'e', 'l', 'l', 'o')

print(tuple\_to\_string(t)) # Output: "Hello"

### ****c. Find the Maximum and Minimum in a Tuple****

🔹 **Problem:** Given a tuple of numbers, find the maximum and minimum values.

**Example:**

def max\_min\_tuple(tpl):

return max(tpl), min(tpl)

# Example usage:

t = (3, 1, 8, 6, 9)

print(max\_min\_tuple(t)) # Output: (9, 1)

## **3. Dictionary Problems**

### ****a. Count the Frequency of Characters in a String****

🔹 **Problem:** Given a string, count the frequency of each character.

**Example:**

def char\_frequency(s):

freq = {}

for char in s:

freq[char] = freq.get(char, 0) + 1

return freq

# Example usage:

text = "banana"

print(char\_frequency(text)) # Output: {'b': 1, 'a': 3, 'n': 2}

### ****b. Merge Two Dictionaries****

🔹 **Problem:** Merge two dictionaries.

**Example:**

def merge\_dicts(d1, d2):

d1.update(d2)

return d1

# 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}

### ****c. Find the Key with the Maximum Value****

🔹 **Problem:** Given a dictionary, find the key with the highest value.

**Example:**

def max\_key(d):

return max(d, key=d.get)

# Example usage:

scores = {'Alice': 89, 'Bob': 95, 'Charlie': 92}

print(max\_key(scores)) # Output: "Bob"

## **4. Set Problems**

### ****a. Find the Common Elements in Two Sets****

🔹 **Problem:** Find the intersection of two sets.

**Example:**

def common\_elements(set1, set2):

return set1 & set2 # OR use set1.intersection(set2)

# Example usage:

s1 = {1, 2, 3, 4}

s2 = {3, 4, 5, 6}

print(common\_elements(s1, s2)) # Output: {3, 4}

### ****b. Find the Union of Two Sets****

🔹 **Problem:** Find the union of two sets.

**Example:**

def union\_sets(set1, set2):

return set1 | set2 # OR use set1.union(set2)

# Example usage:

s1 = {1, 2, 3}

s2 = {3, 4, 5}

print(union\_sets(s1, s2)) # Output: {1, 2, 3, 4, 5}

### ****c. Find the Difference Between Two Sets****

🔹 **Problem:** Find elements in one set but not the other.

**Example:**

def set\_difference(set1, set2):

return set1 - set2 # OR use set1.difference(set2)

# Example usage:

s1 = {1, 2, 3, 4}

s2 = {3, 4, 5, 6}

print(set\_difference(s1, s2)) # Output: {1, 2}

### ****Bonus Challenge:****

**Given a list with duplicate elements, remove duplicates while maintaining order.**

def remove\_duplicates(lst):

return list(dict.fromkeys(lst))

# Example usage:

nums = [4, 2, 3, 2, 1, 4, 3]

print(remove\_duplicates(nums)) # Output: [4, 2, 3, 1]

Here are some **advanced problems** based on Python **data types (list, tuple, dictionary, set)** with solutions. 🚀

## **1. List Problems**

### ****a. Rotate a List by K Positions****

🔹 **Problem:** Given a list and an integer k, rotate the list to the right by k positions.

💡 **Example:**

def rotate\_list(lst, k):

k %= len(lst) # To handle cases where k > len(lst)

return lst[-k:] + lst[:-k]

# Example usage:

nums = [1, 2, 3, 4, 5]

print(rotate\_list(nums, 2)) # Output: [4, 5, 1, 2, 3]

### ****b. Find All Pairs with a Given Sum****

🔹 **Problem:** Given a list and a target sum, find all unique pairs that sum to the target.

💡 **Example:**

def find\_pairs(lst, target):

seen = set()

pairs = set()

for num in lst:

diff = target - num

if diff in seen:

pairs.add((min(num, diff), max(num, diff)))

seen.add(num)

return list(pairs)

# Example usage:

nums = [1, 2, 3, 4, 5, 6, 7, 8]

print(find\_pairs(nums, 9)) # Output: [(1, 8), (2, 7), (3, 6), (4, 5)]

## **2. Tuple Problems**

### ****a. Find the Tuple with the Maximum Sum****

🔹 **Problem:** Given a list of tuples, find the tuple with the highest sum of elements.

💡 **Example:**

def max\_sum\_tuple(tuples):

return max(tuples, key=sum)

# Example usage:

tup\_list = [(1, 2, 3), (4, 5, 6), (10, 1, 2)]

print(max\_sum\_tuple(tup\_list)) # Output: (4, 5, 6)

### ****b. Convert List of Tuples into a Dictionary****

🔹 **Problem:** Convert a list of tuples into a dictionary where the first element is the key and the second is the value.

💡 **Example:**

def tuple\_to\_dict(tuples):

return dict(tuples)

# Example usage:

tup\_list = [("a", 1), ("b", 2), ("c", 3)]

print(tuple\_to\_dict(tup\_list)) # Output: {'a': 1, 'b': 2, 'c': 3}

## **3. Dictionary Problems**

### ****a. Find the Key with the Maximum Value (Multiple Max Cases)****

🔹 **Problem:** Given a dictionary, return the key(s) with the highest value.

💡 **Example:**

def max\_value\_keys(d):

max\_val = max(d.values())

return [k for k, v in d.items() if v == max\_val]

# Example usage:

scores = {'Alice': 89, 'Bob': 95, 'Charlie': 95, 'David': 90}

print(max\_value\_keys(scores)) # Output: ['Bob', 'Charlie']

### ****b. Merge Multiple Dictionaries****

🔹 **Problem:** Merge multiple dictionaries into one. If a key appears in multiple dictionaries, sum its values.

💡 **Example:**

from collections import Counter

def merge\_dicts(\*dicts):

result = Counter()

for d in dicts:

result.update(d)

return dict(result)

# Example usage:

dict1 = {'a': 2, 'b': 3}

dict2 = {'b': 4, 'c': 5}

dict3 = {'a': 1, 'c': 2, 'd': 3}

print(merge\_dicts(dict1, dict2, dict3))

# Output: {'a': 3, 'b': 7, 'c': 7, 'd': 3}

## **4. Set Problems**

### ****a. Find Missing and Extra Elements in Two Sets****

🔹 **Problem:** Given two sets, find elements present in one set but not the other (both ways).

💡 **Example:**

def find\_diff(set1, set2):

return {"missing\_in\_set1": set2 - set1, "missing\_in\_set2": set1 - set2}

# Example usage:

s1 = {1, 2, 3, 4, 5}

s2 = {4, 5, 6, 7}

print(find\_diff(s1, s2))

# Output: {'missing\_in\_set1': {6, 7}, 'missing\_in\_set2': {1, 2, 3}}

### ****b. Find Elements Present in Either but Not Both (Symmetric Difference)****

🔹 **Problem:** Find elements that are in either of the two sets but not both.

💡 **Example:**

def symmetric\_difference(set1, set2):

return set1 ^ set2 # OR use set1.symmetric\_difference(set2)

# Example usage:

s1 = {1, 2, 3, 4}

s2 = {3, 4, 5, 6}

print(symmetric\_difference(s1, s2)) # Output: {1, 2, 5, 6}

## **5. Advanced Mixed Problems**

### ****a. Flatten a Nested List****

🔹 **Problem:** Convert a nested list into a flat list.

💡 **Example:**

def flatten\_list(nested\_list):

flat\_list = []

for sublist in nested\_list:

if isinstance(sublist, list):

flat\_list.extend(flatten\_list(sublist))

else:

flat\_list.append(sublist)

return flat\_list

# Example usage:

nested = [[1, 2, [3, 4]], [5, 6], 7]

print(flatten\_list(nested)) # Output: [1, 2, 3, 4, 5, 6, 7]

### ****b. Find the Longest Word in a Sentence****

🔹 **Problem:** Given a sentence, find the longest word.

💡 **Example:**

def longest\_word(sentence):

words = sentence.split()

return max(words, key=len)

# Example usage:

text = "Artificial Intelligence is transforming the world"

print(longest\_word(text)) # Output: "Intelligence"

### ****c. Group Anagrams from a List of Words****

🔹 **Problem:** Given a list of words, group them into anagrams.

💡 **Example:**

from collections import defaultdict

def group\_anagrams(words):

anagrams = defaultdict(list)

for word in words:

sorted\_word = "".join(sorted(word))

anagrams[sorted\_word].append(word)

return list(anagrams.values())

# Example usage:

word\_list = ["eat", "tea", "tan", "ate", "nat", "bat"]

print(group\_anagrams(word\_list))

# Output: [['eat', 'tea', 'ate'], ['tan', 'nat'], ['bat']]

### ****🔹 Bonus Challenge:****

Find the **first non-repeating character** in a string.

from collections import Counter

def first\_unique\_char(s):

count = Counter(s)

for char in s:

if count[char] == 1:

return char

return None

# Example usage:

text = "swiss"

print(first\_unique\_char(text)) # Output: "w"