## **1. Longest Common Prefix**

### **Question**

Write a function that finds the longest common prefix among an array of strings. If there is no common prefix, return an empty string ("").

### **Explanation**

* Start by assuming the **first string** is the prefix.
* Iterate through the remaining strings and **reduce the prefix** until it matches the start of all strings.
* If the prefix becomes empty, return "".

### **Code**

def longest\_common\_prefix(strs):

if not strs:

return ""

prefix = strs[0] # Assume the first string is the prefix

for string in strs[1:]:

while not string.startswith(prefix):

prefix = prefix[:-1] # Remove the last character from prefix

if not prefix:

return "" # If prefix becomes empty, return ""

return prefix

# Test cases

print(longest\_common\_prefix(["flower", "flow", "flight"])) # Output: "fl"

print(longest\_common\_prefix(["dog", "racecar", "car"])) # Output: ""

print(longest\_common\_prefix(["interspecies", "interstellar", "interstate"])) # Output: "inter"

print(longest\_common\_prefix([""])) # Output: ""

print(longest\_common\_prefix(["a"])) # Output: "a"

print(longest\_common\_prefix(["abc", "abcde", "abcdef"])) # Output: "abc"

## **2. Largest Odd Number in a String**

### **Question**

Given a numeric string, return the **largest odd number** that can be obtained by removing some digits from the right. If no odd number exists, return an empty string ("").

### **Explanation**

* Iterate the string **from the last character to the first**.
* As soon as an **odd digit** is found, return the substring up to that digit.
* If no odd digit is found, return "".

### **Code**

def largest\_odd\_number(s):

for i in range(len(s) - 1, -1, -1): # Iterate from the last character to the first

if int(s[i]) % 2 == 1: # Check if the digit is odd

return s[:i+1] # Return the substring up to the last odd digit

return "" # Return empty string if no odd digit is found

# Test cases

print(largest\_odd\_number("51")) # "51"

print(largest\_odd\_number("52")) # "5"

print(largest\_odd\_number("35428")) # "354"

print(largest\_odd\_number("2468")) # "" (No odd digit)

print(largest\_odd\_number("13579")) # "13579" (Already odd)

print(largest\_odd\_number("8642")) # "" (No odd digit)

## **3. Remove Outermost Parentheses**

### **Question**

Given a valid parentheses string, remove its **outermost parentheses** and return the new string.

### **Explanation**

* Maintain a **count variable** to track the nesting level.
* Append characters to the result **only if they are not the outermost parentheses**.

### **Code**

def remove\_outermost\_parenthesis(s):

# Check if input is valid

if not s or s[0] != '(' or s[-1] != ')' or s.count('(') != s.count(')'):

return "Invalid input: Unbalanced or incorrectly formatted parentheses"

result = []

count = 0

for char in s:

if char == '(' and count > 0:

result.append(char)

elif char == ')' and count > 1:

result.append(char)

if char == '(':

count += 1

elif char == ')':

count -= 1

return ''.join(result)

# Test cases

print(remove\_outermost\_parenthesis("(()())")) # "()()"

print(remove\_outermost\_parenthesis("(())")) # "()"

## **4. Check if Two Strings Are Anagrams**

### **Question**

Write a function that checks if two strings are **anagrams** of each other.  
(An anagram is formed by rearranging the letters of another string.)

### **Method 1: Using Sorting**

* Sort both strings and compare them.
* If they are identical, return True; otherwise, return False.

#### **Code**

def are\_anagrams(s1, s2):

"""Check if two strings are anagrams using sorting."""

return sorted(s1) == sorted(s2)

# Test cases

print(are\_anagrams("listen", "silent")) # True

print(are\_anagrams("hello", "world")) # False

### **Method 2: Using Counter (Dictionary Method)**

* Count occurrences of each character in both strings using collections.Counter.
* If the counts match, return True; otherwise, return False.

#### **Code**

from collections import Counter

def are\_anagrams(s1, s2):

"""Check if two strings are anagrams using Counter."""

return Counter(s1) == Counter(s2)

# Test cases

print(are\_anagrams("listen", "silent")) # True

print(are\_anagrams("triangle", "integral")) # True

print(are\_anagrams("hello", "world")) # False

### **Method 3: Using a Manual Character Count Dictionary**

* Count characters from the **first string**.
* Subtract character counts using the **second string**.
* If any count is **not zero**, return False.

#### **Code**

def are\_anagrams(s1, s2):

"""Check if two strings are anagrams using a dictionary."""

if len(s1) != len(s2):

return False # Anagrams must have the same length

char\_count = {}

# Count characters in the first string

for char in s1:

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

# Subtract character counts using the second string

for char in s2:

if char not in char\_count or char\_count[char] == 0:

return False

char\_count[char] -= 1

return all(value == 0 for value in char\_count.values())

# Test cases

print(are\_anagrams("listen", "silent")) # True

print(are\_anagrams("triangle", "integral")) # True

print(are\_anagrams("hello", "world")) # False

## **Summary**

| **Function** | **Problem** | **Approach** |
| --- | --- | --- |
| **longest\_common\_prefix** | Find the longest common prefix in a list of strings | Start with first string and shorten until it matches all |
| **largest\_odd\_number** | Find the largest odd number in a string | Iterate from the end to find the last odd digit |
| **remove\_outermost\_parenthesis** | Remove outer parentheses from a valid string | Use a counter to track nesting level |
| **are\_anagrams** | Check if two words are anagrams | Sort strings, use Counter, or manual dictionary |