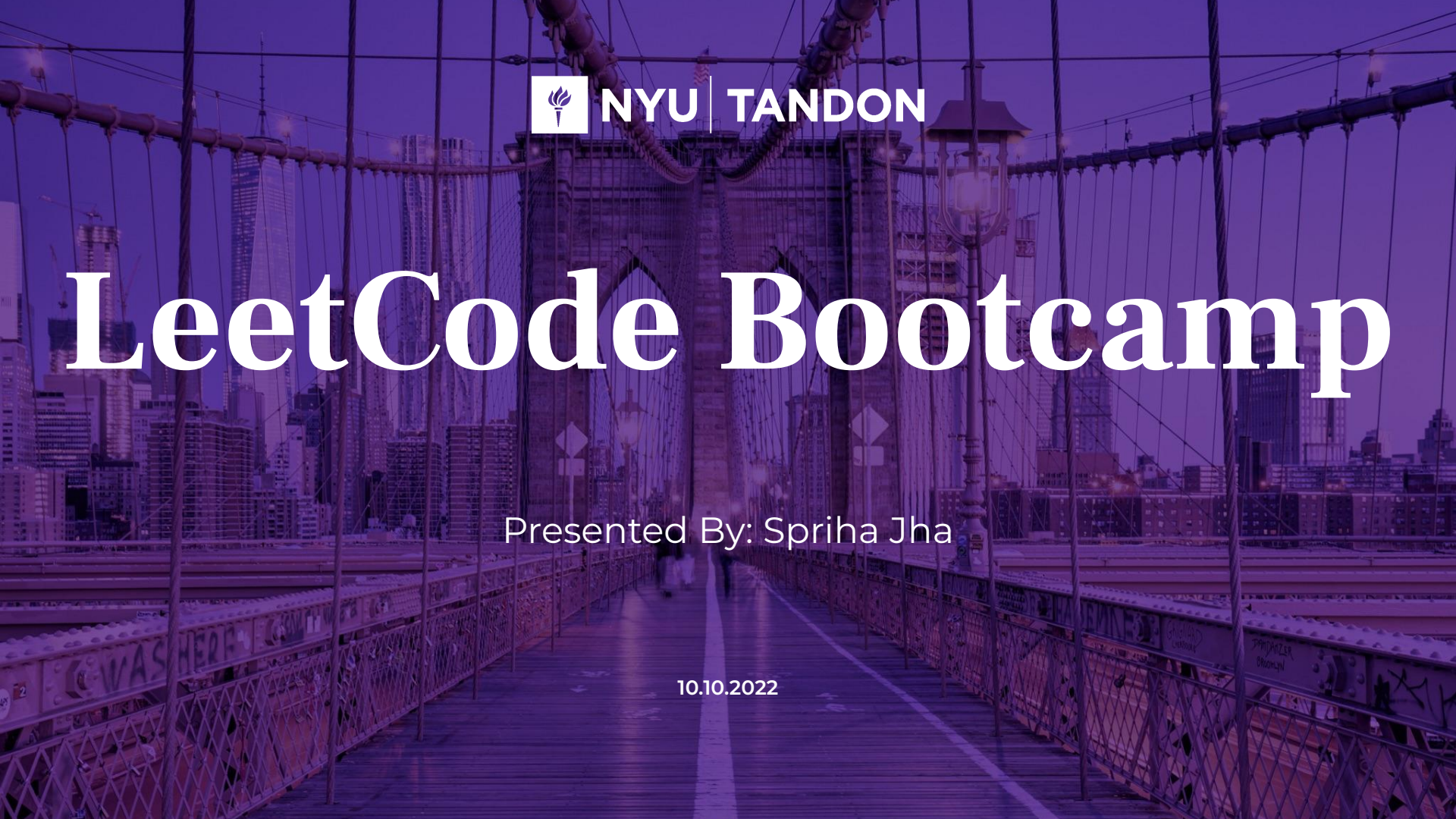


LeetCode Bootcamp

Presented By: Spriha Jha

10.10.2022

Session Outline

01. Introduction to Strings (CheatSheet)

02. Problem Sets

03. Debrief & Q/A

PART 02

Problem Sets

Steps to approach the question:

Understand the problem

Take time to carefully read through the problem from start to finish is critical in finding the correct and complete solution to the problem in hand.

Code your solution

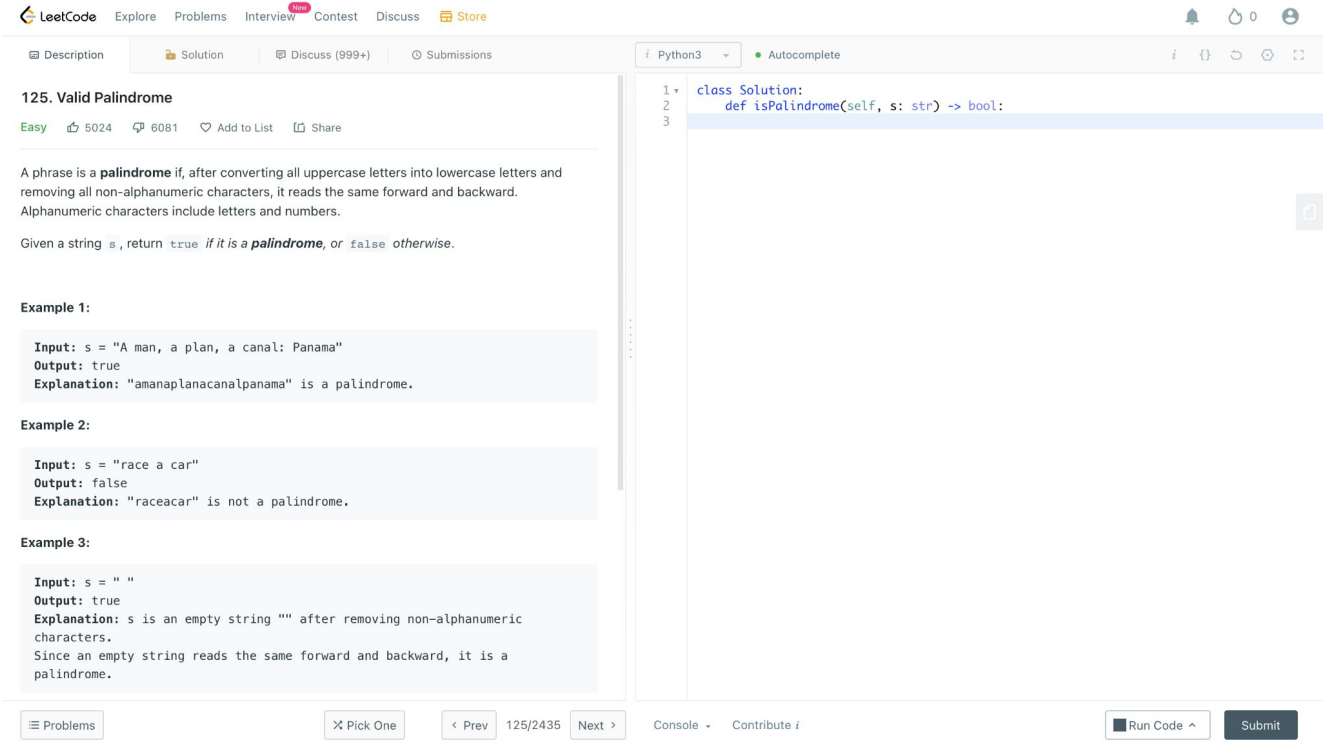
Map out your solution before you write any code. Avoid too much time trying to find the perfect solution. Validate your solution early and often.

Manage your time

Don’t forget, you have multiple questions to complete within a said time. Make sure you allocate enough time to carefully consider all problems.

PART 02

Problem 1: Valid Palindrome



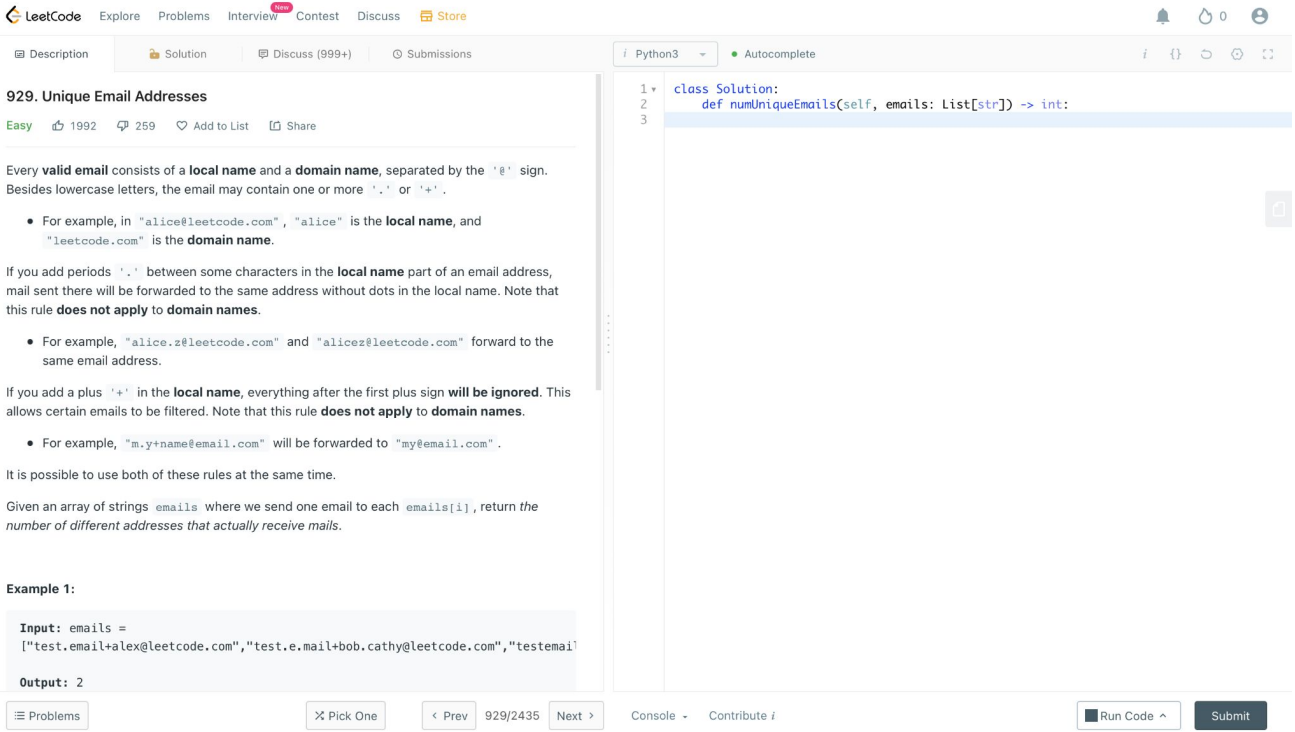
Approach: Two Pointers

```
def isPalindrome(self, s: str) -> bool:

    # Define the two pointers on each end
    i, j = 0, len(s) - 1

    while i < j:
        # Increase the left-end pointer if current character is non-alphanumeric
        while i < j and not s[i].isalnum():
            i += 1

        # Decrease the right-end pointer if current character is non-alphanumeric
        while i < j and not s[j].isalnum():
            j -= 1

        # Compare both pointers should point to equivalent character or else break early
        if s[i].lower() != s[j].lower():
            return False

        #Increase the pointers
        i += 1
        j -= 1

    return True
```

Time complexity: O(n), in length *n* of the string. We traverse over each character at-most once, until the two pointers meet in the middle, or when we break and return early.
Space complexity: O(1), no extra space required, at all.

PART 02

Problem 2: Unique email addresses



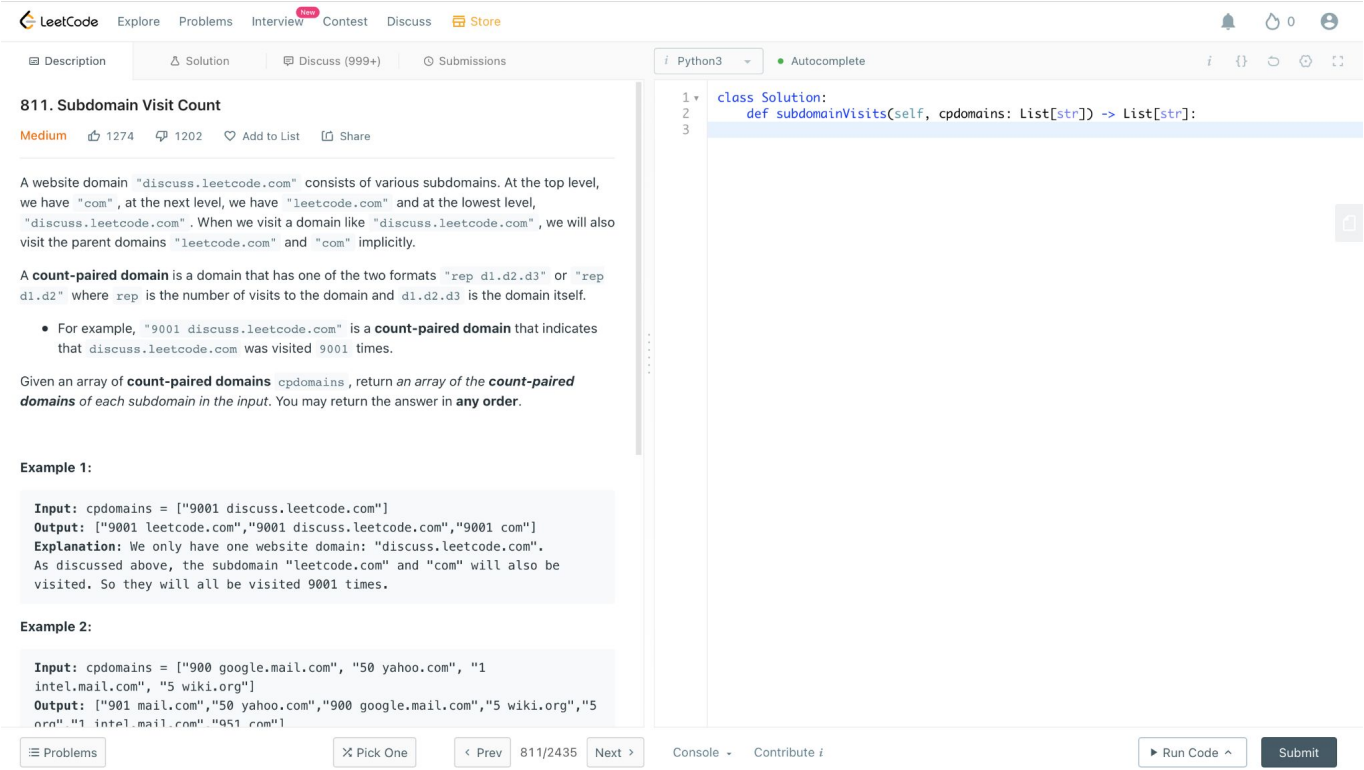
Approach: String Split method

```
def numUniqueEmails(self, emails: List[str]) -> int:
    # Hashset to store all the unique emails.
    uniqueEmails = set()

    for email in emails:
        # Split into two parts: local and domain.
        name, domain = email.split('@')

        # Split local by '+' and replace all '.' with ''.
        local = name.split('+')[0].replace('.', '')

        # Concatenate local, '@', and domain.
        uniqueEmails.add(local + '@' + domain)

    return len(uniqueEmails)
```

Let *N* be the number of the emails and *M* be the average length of an email.

Time complexity: $O(N \cdot M)$, The split method must iterate over all of the characters in each email and the replace method must iterate over all of the characters in each local name.
Space complexity: $O(N \cdot M)$ In the worst case, when all emails are unique, we will store every email address given to us in the hash set.

Problem 3: Subdomain visit count



NYU TANDON

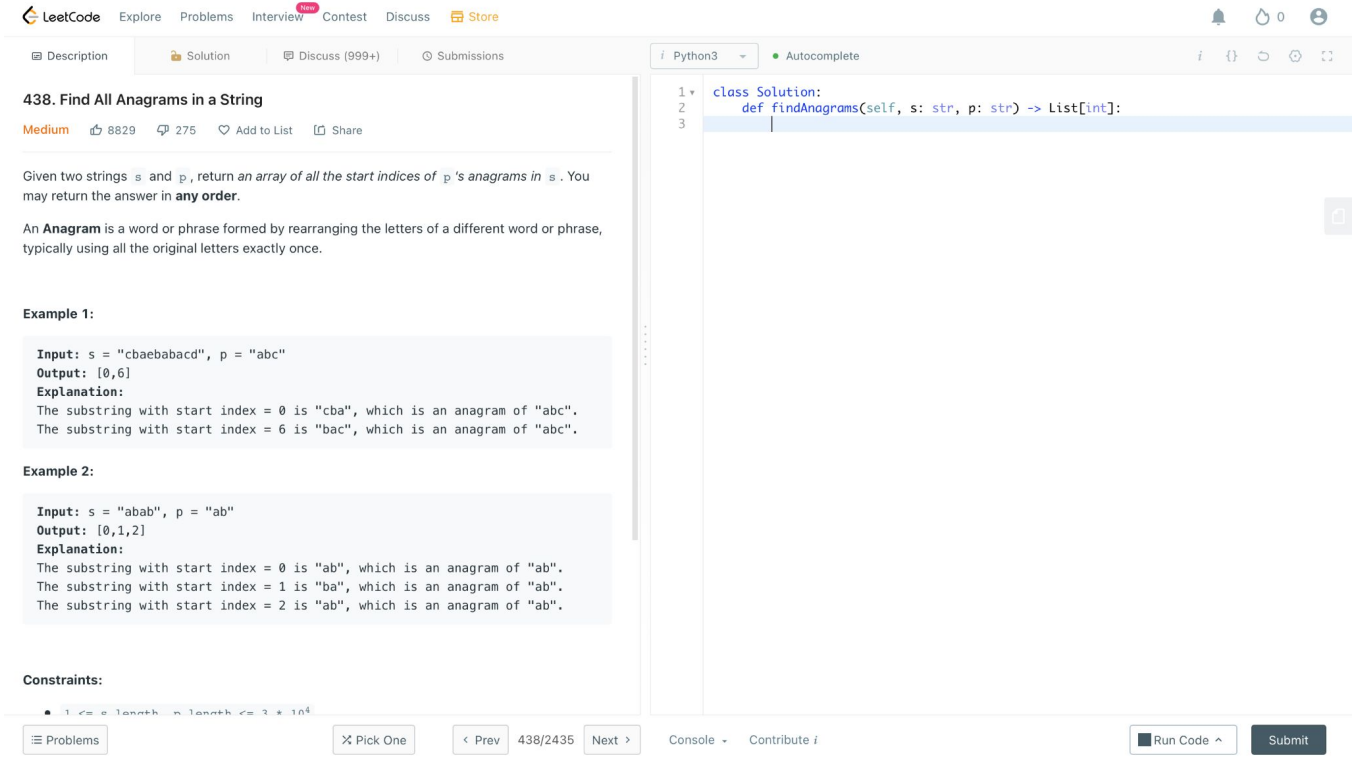
Approach: Hash Map

```
def subdomainVisits(self, cpdomains):
    ans = collections.Counter()

    for domain in cpdomains:
        count, domain = domain.split() # Separate the domain and its count
        count = int(count) # Converts count from String to Int
        fragments = domain.split('.') # Separates the subdomain fragments

        # Combines the count of subdomains
        for i in range(len(fragments)):
            ans[".".join(fragments[i:])] += count

    return ["{} {}".format(ct, dom) for dom, ct in ans.items()]
```

Time complexity: $O(N)$, where N is the length of `cpdomains`, and assuming the length of `cpdomains[i]` is fixed.
Space complexity: O(N), the space used in our count.

PART 02

Problem 4: Find all Anagrams in a String



NYU TANDON

Approach: Sliding window with Array

```
def findAnagrams(self, s: str, p: str) -> List[int]:
    s_length, p_length = len(s), len(p)
    if s_length < p_length:
        return []

    p_count, s_count = [0] * 26, [0] * 26
    # build reference array using string p
    for char in p:
        p_count[ord(char) - ord('a')] += 1

    output = []
    # sliding window on the string s
    for i in range(s_length):
        # add one more letter on the right side of the window
        s_count[ord(s[i]) - ord('a')] += 1
        # remove one letter from the left side of the window
        if i >= p_length:
            s_count[ord(s[i - p_length]) - ord('a')] -= 1
        # compare array in the sliding window with the reference array
        if p_count == s_count:
            output.append(i - p_length + 1)

    return output
```

PART 06

Q/A

Slack Hours

Join Slack Workspace!

Office Hours: Tuesday (10AM - 1PM)

Problem Assignments

01. Longest Palindrome (Easy)

02. Valid Parentheses (Easy)

03. Isomorphic Strings (Easy)

04. Zigzag Conversion (Medium)

05. Longest Palindromic Substring (Medium)

06. Longest Substring without repeating characters (Medium)

07. Minimum window substring (Hard)


NYU TANDON
Thank you!
Upcoming: Linked Lists (17/10)