**Coding Interview (Top 20 Question – without DSA + LeetCode)**

**Q1. Remove Duplicates from the list?**

**Program**

l = [1, 2, 3, 4, 5, 6, 1]  
  
"""  
First Way  
"""  
l = set(l)  
l = list(l)  
print(l)  
  
"""  
Second Way  
"""  
  
k = list()  
*for* i *in* l:  
 *if* i *not in* k:  
 k.append(i)  
print(k)  
  
"""  
Third Way  
"""  
j = list()  
j = [x *for* x *in* l *if* x *not in* j *and not* j.append(x)]  
print(j)

**Q2. Max Profit - [7, 1, 5, 3, 6, 4] = 5, [7, 6, 4, 3, 1] = 0**

**Ans.** *def* max\_profit(input\_list):  
 result = list()  
 *for* ind *in* range(len(input\_list) - 1):  
 *for* j *in* input\_list[ind:]:  
 k = j - input\_list[ind]  
 print(f'{j} and {input\_list[ind]}')  
 result.append(k)  
  
 *if* len(result)==0:  
 *return* 0  
 *return* max(result)  
  
  
  
  
a = max\_profit([])  
print(a)

**Q3.** **Write a function that takes an array and an integer n as input and shifts the array to the right by n steps. Return the shifted array. For eg:- arr =** **[1, 2, 3, 4, 5]**

**n = 2**

**(arr, n) == [****4, 5, 1, 2, 3]**

**Ans.**

*def* array\_shifter(arr, n):  
 n = n % len(arr) *# Handle cases where n is greater than the length of the array  
 if* n == 0:  
 *return* arr *# No need to shift if n is 0  
  
 if* n > 0:  
 *# Right shift  
 for* \_ *in* range(n):  
 k = arr.pop()  
 arr = [k] + arr  
 *else*:  
 *# Left shift  
 for* \_ *in* range(abs(n)):  
 k = arr.pop(0)  
 arr.append(k)  
  
 *return* arr

**Second Way***def* shift\_array(arr: List[int], n: int) -> List[int]:  
 *# Calculate the effective shift value* n = n % len(arr)  
  
 *# Perform the rotation by slicing the array* shifted\_array = arr[-n:] + arr[:-n]  
  
 *return* shifted\_array

**Q4. Detect Duplicate e.g.: - productid’s** **= [1, 2, 3, 4, 5, 6, 7, 8, 9, 10] == False**

def detect\_duplicates(l):

k = []

for i in l:

if i in k:

return True

else:

k.append(i)

return False

a = detect\_duplicates([1,2,3,4,5,6,7,8,9])

print(a)

**Q5. Unique Entry, which is not repeating.**

**assert ([10, 20, 30, 40, 50, 60, 70, 80, 90, 10, 20, 30, 40, 50, 60, 70]) == 80**

**Ans.** *def* unique\_entry(customerIDs):  
 unique\_id = 0  
 *for* customer\_id *in* customerIDs:  
 print(f'Unique Id {unique\_id} and customer\_di is {customer\_id}')  
 unique\_id ^= customer\_id  
 print(f'Result {unique\_id}')  
 *return* unique\_id  
  
  
a = unique\_entry([70, 80, 90, 80, 70])  
print(a)

**Only works for single duplicate element in list..  
interviewers goes crazy if they see u do this.**

**Q6. Intersection of two lists**

**customers1 = [1, 2, 3, 4, 5]**

**customers2 = [4, 5, 6, 7, 8]**

**Ans.** *def* array\_intersection(x1, y1):  
 result = []  
 *for* x *in* x1:  
 *if* x *in* y1:  
 result.append(x)  
  
 *return* result  
  
  
customers1 = [1, 2, 3, 4, 5]  
customers2 = [4, 5, 6, 7, 8]  
  
print(array\_intersection(customers1, customers2))

**Or**

*def* common\_customers(customers1: list[int], customers2: list[int]) -> list[int]:  
 *# Convert the input arrays to sets* set1 = set(customers1)  
 set2 = set(customers2)  
  
 *# Find the common customers using set intersection* common\_customers\_set = set1.intersection(set2)  
  
 *# Sort the common customers in ascending order* sorted\_customers = sorted(common\_customers\_set)  
  
 *# Convert the sorted set back to a list and return it  
 return* list(sorted\_customers)  
  
  
print(array\_intersection(customers1, customers2))

**x**

**Q7. You just got hired at a data storage startup that has decided to store large integers as an array. For example, 256 becomes [2,5,6]. Can you develop a Python function that takes an integer array called "digits" representing a large integer?**

***# Example inputs and outputs***

***# Input: [1, 2, 3]***

***# Expected Output: [1, 2, 4]***

***# Input: [9, 9, 9, 9]***

***# Expected Output: [1, 0, 0, 0, 0]***

***# Input: [5, 6, 7, 8, 9]***

***# Expected Output: [5, 6, 7, 9, 0]***

***# Input: [0]***

***# Expected Output: [1]***

*def* increment\_large\_integer(digits: list[int]) -> list[int]:  
 carry = 1 *# Start with a carry of 1* result = []  
  
 *for* digit *in* reversed(digits):  
 new\_digit = digit + carry  
 print(f'New Digit {new\_digit % 10}')  
  
 carry = new\_digit // 10 *# Determine the carry for the next digit* print(carry)  
 result.insert(0, new\_digit % 10) *# Append the digit to the result  
  
 # If there is still a carry after processing all digits, insert it at the beginning  
 if* carry:  
 result.insert(0, carry)  
  
 *return* result  
  
  
print(increment\_large\_integer([2]))

**Q8. Zero Movement**

**arr = [0, 1, 0, 3, 12]**

**move\_zeros(arr)**

**#arr is set equal to:**

**[1, 3, 12, 0, 0]**

**Ans.**

def zero\_movement(x):  
 n = []  
 for i in x:  
 if i == 0:  
 n.insert(0, 0)  
 else:  
 n.append(i)  
 return n  
  
   
l = [1, 0, 1, 0, 1, 0]  
print(zero\_movement(l))

**Q9. Target**

**prices = [10, 20, 30, 40, 50]**

**target = 60**

**Ans [1, 3]**

**First Way:**

def target\_hitter(x, target):  
 result = []  
 for index, value in enumerate(x):  
 new\_ind = index  
 for j in x[index + 1:]:  
 new\_ind += 1  
 # print(f'{value} + {j}')  
 if value + j == target:  
 print(f'{value} + {j} -> Successful -> {new\_ind}')  
 result.append([index, new\_ind])  
  
 return result  
  
  
prices = [5, 10, 15, 20, 25]  
target = 30  
  
print(target\_hitter(prices, target))

**Second Way**

class Solution:

def twoSum(self, nums: List[int], target: int) -> List[int]:

dict={}

for i,n in enumerate(nums):

if n in dict:

return dict[n],i

else:

dict[target-n]=i

**Third Way**

def target\_hitter(nums, target):  
 for index, item in enumerate(nums):  
 y = target - item  
 print(f'y = {y} and index is {index}')  
 if y in nums[index+1:]:  
 return [index, nums[index+1:].index(y) + index+1]  
 return []  
  
  
prices = [3, 3]  
target = 6  
  
print(target\_hitter(prices, target))

**Q10. Flip Image**

**Input:**

**image = [[1, 2, 3],**

**[4, 5, 6],**

**[7, 8, 9]]**

**Output:**

**[[9, 8, 7],**

**[6, 5, 4],**

**[3, 2, 1]]**

**Q11. First Nonrepeating Character**

**("lovleyo") == 2**

**Explanation: The first non-repeating character in "lovleyo" is 'v' and its index is 2.**

**Ans.**

def non\_repeating(x):  
 for ind, char in enumerate(x):  
 if char not in x[ind+1:]:  
 return ind  
 return ''  
  
s = 'longelgonr'  
print(non\_repeating(s))

**Q12. Anagram Checker**

**s1 = ""listen""**

**s2 = ""silent""**

**assert is\_anagram(s1, s2) == True**

**The resulting word or phrase contains all the original letters exactly once. For example, the word ""listen"" can be rearranged to form the word ""silent"" making ""silent"" an anagram of ""listen"".**

**Ans.**

*"""  
Easy Way  
"""*def anagram(s1, s2):  
 for i in s1:  
 print(i)  
 if str(i) not in s2:  
 return False  
 return True  
  
  
str\_1 = "hello "  
str\_2 = "world"  
  
print(anagram(str\_1, str\_2))  
  
"""  
Medium way - which they can accept  
"""  
   
  
def anagram(s1, s2):  
 return sorted(s1) == sorted(s2)

**Q13. Palindrome Checker**

**assert is\_palindrome(""A man, a plan, a canal: Panama"") == True**

**The input string ""A man, a plan, a canal: Panama"" is a valid palindrome after ignoring all non-alphanumeric characters.**

**Ans.**

*""" With Slicing """*def is\_palindrome(stringular):  
 remove\_alpha\_num = "".join(filter(str.isalnum, stringular.lower()))  
 return remove\_alpha\_num[::] == remove\_alpha\_num[::-1]  
  
  
word\_to\_check = "0A man, a plan, a canal: Panama0"  
print(is\_palindrome(word\_to\_check))  
  
"""  
Without Slicing  
"""  
  
  
def palindrome\_checker(s1):  
 s1 = "".join(filter(str.isalnum, s1.lower()))  
 result = [s1[ind] == val for ind, val in enumerate(reversed(s1))]  
 return not False in result  
  
  
print(palindrome\_checker(word\_to\_check))  
  
"""  
Without .isalnum, without for loop, without slicing  
"""  
  
  
def remove\_alphanumeric(input\_str):  
 result = ""  
 for char in input\_str:  
 # Check if the character is a letter or a number  
 if ('a' <= char <= 'z') or ('A' <= char <= 'Z') or ('0' <= char <= '9'):  
 result += char  
 reversed\_str = "".join(reversed(result))  
 if reversed\_str.lower() == result.lower():  
 return True  
 return False  
  
  
output\_string = remove\_alphanumeric(word\_to\_check)  
print(output\_string)

**Q14. String Search Function**

**This function will take in two strings as parameters, s1 and s2. If s1 is found within s2, the function should return the index of the first occurrence of the substring. Else return -1**

**s1 = ""world"" s2 = ""hello world"" assert search\_substring(s1, s2) == 6**

**The substring ""python"" is not found in the larger string ""java programming"", so the function returns -1.**

**Ans. def search\_substring(s1: str, s2: str) -> int:**

**if s1 in s2:**

**return s2.index(s1)**

**else:**

**return -1**

**Q15. Matching Prefix**

**Input:** **[""charcode"", ""charme"", ""charmes"", ""charlie""]**

**Output: ""char""**

**Explanation: ""char"" is the longest common prefix among all the strings in the database.**

# first way  
  
def longest\_prefix(strs):  
 if not strs:  
 return ""  
 if len(strs) == 1:  
 return strs[0]  
 strs.sort()  
 print(f'Sorted list {strs}')  
 prefix = ""  
 for i in range(len(strs[0])):  
 if strs[0][i] == strs[-1][i]:  
 prefix += strs[0][i]  
 else:  
 break  
 return prefix  
  
  
print(longest\_prefix(["Aharcodess", "Bharm", "Baarmes", "Dharlie"]))  
  
  
# [""charcode"", ""charme"", ""charmes"", ""charlie""]  
  
# Second Way  
  
def longest\_common\_prefix(strings):  
 if not strings:  
 return ""  
  
 # Take the first string as a reference  
 reference\_str = strings[0]  
 common\_prefix = []  
 k = []  
 for i in range(len(reference\_str)):  
 # Check if the current character is common in all strings  
 for s in strings:  
 if i < len(s):  
 if s[i] == reference\_str[i]:  
 k.append(True)  
 else:  
 k.append(False)  
 if all(k):  
 common\_prefix.append(reference\_str[i])  
 else:  
 break  
  
  
 return "".join(common\_prefix)  
  
  
# Example usage:  
input\_strings = ["charcode", "charme", "charmes", "charlie"]  
output = longest\_common\_prefix(input\_strings)  
print(f"Input: {input\_strings}\nOutput: {output}")  
  
def longest\_common\_prefix(strings):  
 if not strings:  
 return ""  
  
 # Take the first string as a reference  
 reference\_str = strings[0]  
 common\_prefix = []  
  
 for i in range(len(reference\_str)):  
 # Check if the current character is common in all strings  
 if all(i < len(s) and s[i] == reference\_str[i] for s in strings):  
 common\_prefix.append(reference\_str[i])  
 else:  
 break  
  
 return "".join(common\_prefix)  
  
# Example usage:  
input\_strings = ["charcode", "charme", "charmes", "charlie"]  
output = longest\_common\_prefix(input\_strings)  
print(f"Input: {input\_strings}\nOutput: {output}")

**Q16.** **Longest common substring**

**[“banana”, “baniyan” , “banker” “Arabanka”, “barabanka”]**

**Output = ban**

def longest\_substring(string: list):  
 print("runned")  
 if not string:  
 return ""  
 if len(string) == 1:  
 return string[0]  
 reference\_string = string[0]  
 common\_longest\_substring = []  
 for i in range(len(reference\_string)):  
 # print(f'i: {i} : {reference\_string[:i + 1]}')  
 # print(list(s in reference\_string[:i + 1] for s in string))  
 if all(reference\_string[:i + 1] in s for s in string):  
 common\_longest\_substring.append(reference\_string[:i + 1])  
  
 for j in range(i + 1, len(reference\_string)):  
 # print(f'range({i},{len(reference\_string)}) and i:j = {i}:{j}')  
 # print(f'j : {reference\_string[i:j+1]}')  
 if all(reference\_string[i:j + 1] in s for s in string):  
 common\_longest\_substring.append(reference\_string[i:j + 1])  
  
 if len(common\_longest\_substring) == 0:  
 return ""  
 return max(common\_longest\_substring, key=len)  
  
  
x = ["raja", "janwar", "magaja", "japper", "jakrta"]  
print(longest\_substring(x))