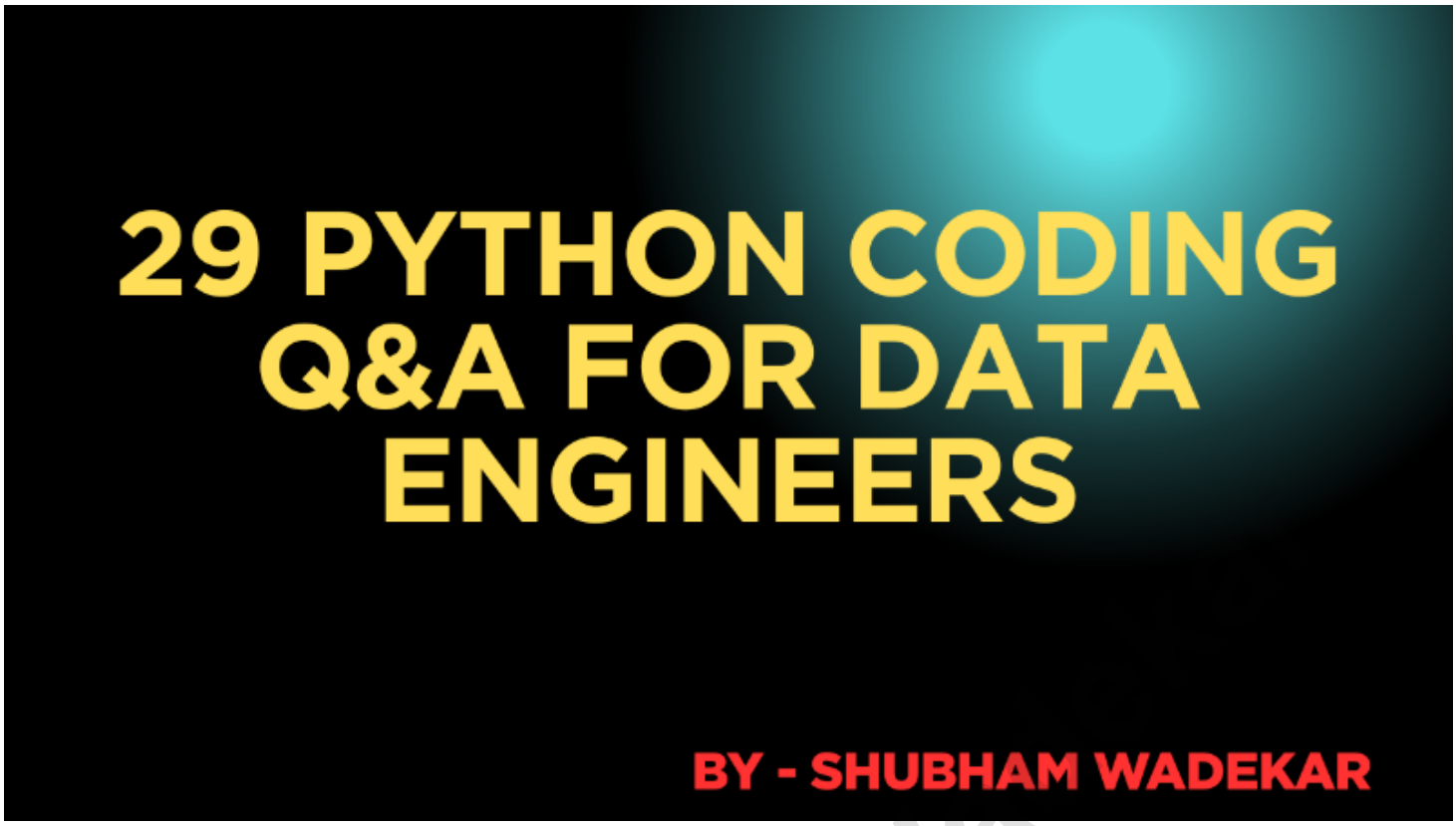

29 PYTHON QUESTIONS ASKED IN DATA ENGINEERING INTERVIEWS

1. Generate an Infinite Fibonacci Series Using a Generator.

Explanation:

This program generates an infinite Fibonacci series using a generator.

Logic:

- Use a generator function (yield) to produce Fibonacci numbers indefinitely.

Program:

```
def infinite_fibonacci():
    a, b = 0, 1
    while True:
        yield a
        a, b = b, a + b

# Example usage
fib_gen = infinite_fibonacci()
for _ in range(10):  # Print first 10 Fibonacci numbers
    print(next(fib_gen), end=" ")
```

2. Sort a List Without Using the sort Keyword.

Explanation:

This program sorts a list without using the built-in sort method or sorted() function.

Logic:

- Implement a sorting algorithm like Bubble Sort, Selection Sort, or Insertion Sort.

Program (Bubble Sort):

```
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

lst = [64, 34, 25, 12, 22, 11, 90]
print("Sorted list:", bubble_sort(lst))
```

3. Sort a List Using the sort Keyword.

Explanation:

This program sorts a list using the built-in sort method.

Logic:

- Use the sort() method to sort the list in place.

Program:

```
lst = [64, 34, 25, 12, 22, 11, 90]
lst.sort()
print("Sorted list:", lst)
```

4. Check Whether a String is a Palindrome or Not.

Explanation:

This program checks if a string reads the same backward as forward.

Logic:

- Compare the string with its reverse.

Program:

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

input_string = "madam"
if is_palindrome(input_string):
    print(f"'{input_string}' is a palindrome.")
else:
    print(f"'{input_string}' is not a palindrome.")
```

5. Sort a Dictionary by Key.

Explanation:

This program sorts a dictionary by its keys.

Logic:

- Use the sorted() function to sort the dictionary keys and create a new dictionary.

Program:

```
my_dict = {'b': 2, 'a': 1, 'c': 3}
sorted_dict = {k: my_dict[k] for k in sorted(my_dict)}
print("Sorted dictionary by key:", sorted_dict)
```

6. Find the Pair of Sum with a Given Number in the List.

Explanation:

This program finds pairs of numbers in a list that add up to a given sum.

Logic:

- Use nested loops or a set to find pairs.

Program:

```
def find_pairs(lst, target_sum):
    pairs = []
    for i in range(len(lst)):
        for j in range(i + 1, len(lst)):
            if lst[i] + lst[j] == target_sum:
                pairs.append((lst[i], lst[j]))
    return pairs

lst = [1, 2, 3, 4, 5]
target_sum = 5
print("Pairs with sum", target_sum, ":", find_pairs(lst, target_sum))
```

7. Find the Output: "blue is sky the", Input: "the sky is blue".

Explanation:

This program reverses the order of words in a sentence.

Logic:

- Split the sentence into words, reverse the list, and join them back.

Program:

```
def reverse_sentence(sentence):
    return ' '.join(sentence.split()[::-1])

input_sentence = "the sky is blue"
print("Reversed sentence:", reverse_sentence(input_sentence))
```

8. Find the Maximum Repeated Character in a String.

Explanation:

This program finds the character that appears most frequently in a string.

Logic:

- Use a dictionary to count character occurrences and find the maximum.

Program:

```python
from collections import Counter

def max_repeated_char(input_string):
    char_count = Counter(input_string)
    return max(char_count, key=char_count.get)

input_string = "hello world"
print("Most repeated character:", max_repeated_char(input_string))
```

9. Prime Number: Divisible by 1 and Itself.

Explanation:

This program checks if a number is prime.

Logic:

- A prime number is only divisible by 1 and itself.
- Check divisibility from 2 to the square root of the number.

Program:

```python
def is_prime(n):
    if n <= 1:
        return False
    for i in range(2, int(n**0.5) + 1):
        if n % i == 0:
            return False
    return True

num = 17
if is_prime(num):
    print(f"{num} is a prime number.")
else:
    print(f"{num} is not a prime number.")
```

10. Python Program to Find the Factorial of a Number.

Explanation:

This program calculates the factorial of a number.

Logic:

- Use a loop to multiply numbers from 1 to n*n*.

Program:

```python
def factorial(n):
    result = 1
    for i in range(1, n + 1):
        result *= i
    return result

num = 5
print(f"Factorial of {num} is {factorial(num)}")
```

11. Check if a Number is an Armstrong Number.

Explanation:

An Armstrong number is a number that is equal to the sum of its own digits raised to the power of the number of digits.

Logic:

- Calculate the sum of digits raised to the power of the number of digits.
- Compare the sum with the original number.

Program:

```
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)

num = 153
if is_armstrong(num):
    print(f"{num} is an Armstrong number.")
else:
    print(f"{num} is not an Armstrong number.")
```

12. ASCII Value: American Standard Code for Information Interchange.

Explanation:

This program finds the ASCII value of a character.

Logic:

- Use the ord() function to get the ASCII value.

Program:

```
char = 'A'
print(f"ASCII value of '{char}' is {ord(char)}")
```

13. Sum of Squares of First n*n* Natural Numbers.

Explanation:

This program calculates the sum of squares of the first n*n* natural numbers.

Logic:

- Use the formula:

$$\text{Sum} = \frac{n(n+1)(2n+1)}{6}$$

```python
def sum_of_squares(n):
    return n * (n + 1) * (2 * n + 1) // 6

n = 5
print(f"Sum of squares of first {n} natural numbers is {sum_of_squares(n)}")
```

14. Common Letters in Two Strings

Explanation:

This program finds the common letters between two strings.

Logic:

- Convert both strings to sets and find their intersection.

Program:

```python
def common_letters(str1, str2):
    s1 = set(str1)
    s2 = set(str2)
    return list(s1 & s2)

str1 = "reene"
str2 = "naina"
print("Common letters:", common_letters(str1, str2))
```

15. Frequency of Words in a Sentence

Explanation:

This program calculates the frequency of each word in a sentence.

Logic:

- Split the sentence into words and use a dictionary to count occurrences.

Program:

```python
def word_frequency(sentence):
    words = sentence.split()
    frequency = {}
    for word in words:
        if word in frequency:
            frequency[word] += 1
        else:
            frequency[word] = 1
    return frequency

sentence = "the quick brown fox jumps over the lazy dog"
print("Word frequency:", word_frequency(sentence))
```

16. Convert Two Lists to a Dictionary

Explanation:

This program converts two lists (one of keys and one of values) into a dictionary.

Logic:

- Use the zip() function to pair keys and values, then convert to a dictionary.

Program:

```python
def array_to_dict(keys, values):
    return dict(zip(keys, values))

keys = ['one', 'two', 'three']
values = [1, 2, 3]
print("Dictionary:", array_to_dict(keys, values))
```

17. Reverse a String

Explanation:

This program reverses a given string.

Logic:

- Use string slicing to reverse the string.

Program:

```python
def reverse_string(input_string):
    return input_string[::-1]

input_string = "Hello, World!"
print("Reversed string:", reverse_string(input_string))
```

18. Word Count in a Paragraph

Explanation:

This program counts the frequency of each word in a paragraph.

Logic:

- Split the paragraph into words and use a dictionary to count occurrences.

Program:

```
def word_count(paragraph):
    words = paragraph.split()
    frequency = {}
    for word in words:
        if word in frequency:
            frequency[word] += 1
        else:
            frequency[word] = 1
    return frequency

paragraph = "the quick brown fox jumps over the lazy dog and the fox"
print("Word count:", word_count(paragraph))
```

19. Remove Duplicates from a List

Explanation:

This program removes duplicate elements from a list.

Logic:

- Convert the list to a set (which automatically removes duplicates) and back to a list.

Program:

```
def remove_duplicates(lst):
    return list(set(lst))

lst = [3, 6, 7, 9, 2, 3, 7, 1]
print("List without duplicates:", remove_duplicates(lst))
```

20. Find the Smallest and Largest Number in an Array

Explanation:

This program finds the smallest and largest numbers in an array.

Logic:

- Iterate through the array and keep track of the smallest and largest numbers.

Program:

```
def find_min_max(arr):
    mini = float('inf')
    maxi = float('-inf')
    for num in arr:
        if num < mini:
            mini = num
        if num > maxi:
            maxi = num
    return mini, maxi

arr = [12, 45, 10, 11, 33, 44, 90, 2]
print("Smallest and largest numbers:", find_min_max(arr))
```

21. Merge Strings Alternately

Explanation:

This program merges two strings alternately.

Logic:

- Use a loop to alternately add characters from both strings.

Program:

```
def merge_alternately(word1, word2):
    result = []
    i, j = 0, 0
    while i < len(word1) or j < len(word2):
        if i < len(word1):
            result.append(word1[i])
            i += 1
        if j < len(word2):
            result.append(word2[j])
            j += 1
    return ''.join(result)

word1 = "abc"
word2 = "pqr"
print("Merged string:", merge_alternately(word1, word2))
```

22. GCD of Strings

Explanation:

This program finds the greatest common divisor (GCD) of two strings.

Logic:

- If the concatenation of the two strings is not equal in both orders, there is no GCD.
- Otherwise, recursively find the GCD of the shorter string and the remainder of the longer string.

Program:

```
def gcd_of_strings(str1, str2):
    if str1 + str2 != str2 + str1:
        return ""
    if len(str1) == len(str2):
        return str1
    if len(str1) > len(str2):
        return gcd_of_strings(str1[len(str2):], str2)
    return gcd_of_strings(str1, str2[len(str1):])

str1 = "ABCABC"
str2 = "ABC"
print("GCD of strings:", gcd_of_strings(str1, str2))
```

23. Kids with the Greatest Number of Candies

Explanation:

This program determines which kids can have the greatest number of candies after adding extra candies.

Logic:

- Find the maximum number of candies any kid has.
- Check if each kid can have the maximum number after adding extra candies.

Program:

```
def kids_with_candies(candies, extra_candies):
    max_candies = max(candies)
    return [candy + extra_candies >= max_candies for candy in candies]

candies = [2, 3, 5, 1, 3]
extra_candies = 3
print("Kids with max candies:", kids_with_candies(candies, extra_candies))
```

24. Can Place Flowers

Explanation:

This program checks if n new flowers can be planted in a flowerbed without violating the no-adjacent-flowers rule.

Logic:

- Iterate through the flowerbed and check if a flower can be placed at each position.

Program:

```
def can_place_flowers(flowerbed, n):
    flowerbed = [0] + flowerbed + [0]
    for i in range(1, len(flowerbed) - 1):
        if flowerbed[i - 1] == 0 and flowerbed[i] == 0 and flowerbed[i + 1] == 0:
            flowerbed[i] = 1
            n -= 1
    return n <= 0

flowerbed = [1, 0, 0, 0, 1]
n = 1
print("Can place flowers:", can_place_flowers(flowerbed, n))
```

25. Reverse Vowels in a String

Explanation:

This program reverses the vowels in a string.

Logic:

- Use two pointers to swap vowels from the beginning and end of the string.

Program:

```
def reverse_vowels(s):
    vowels = set('aeiouAEIOU')
    s = list(s)
    left, right = 0, len(s) - 1
    while left < right:
        if s[left] in vowels and s[right] in vowels:
            s[left], s[right] = s[right], s[left]
            left += 1
            right -= 1
        elif s[left] not in vowels:
            left += 1
        else:
            right -= 1
    return ''.join(s)

s = "IceCreAm"
print("Reversed vowels:", reverse_vowels(s))
```

26. Move Zeros to the End

Explanation:

This program moves all zeros to the end of the array while maintaining the order of non-zero elements.

Logic:

- Use two pointers to swap non-zero elements to the front.

Program:

```
def move_zeros(nums):
    left = 0
    for right in range(len(nums)):
        if nums[right] != 0:
            nums[left], nums[right] = nums[right], nums[left]
            left += 1
    return nums

nums = [0, 1, 0, 3, 12]
print("Moved zeros:", move_zeros(nums))
```

27. Check if a String is a Subsequence of Another String

Explanation:

This program checks if string s is a subsequence of string t.

Logic:

- Use two pointers to traverse both strings.

Program:

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

s = "abc"
t = "ahbgdc"
print("Is subsequence:", is_subsequence(s, t))
```

28. Unique Number of Occurrences

Explanation:

This program checks if the number of occurrences of each element in the array is unique.

Logic:

- Use a dictionary to count occurrences and check if the counts are unique.

Program:

```
    count = {}
    for num in arr:
        if num in count:
            count[num] += 1
        else:
            count[num] = 1
    return len(count.values()) == len(set(count.values()))

arr = [1, 2, 2, 1, 1, 3]
print("Unique occurrences:", unique_occurrences(arr))
```

29. Remove Element

Explanation:

This program removes all instances of a value from an array and returns the new length.

Logic:

- Use two pointers to overwrite the value to be removed.

Program:

```
def remove_element(nums, val):
    k = 0
    for i in range(len(nums)):
        if nums[i] != val:
            nums[k] = nums[i]
            k += 1
    return k

nums = [3, 2, 2, 3]
val = 3
print("New length after removal:", remove_element(nums, val))
```