misc python problems

Python Interview Questions

1. Word/Character Counting

Problem: Count words in a list or characters in a string

Approach 1: Using Dictionary

```
def count_words(word_list):
    """Count frequency of each word in a list"""
    count_dict = {}
    for word in word_list:
        if word in count_dict:
            count_dict[word] += 1
        else:
            count_dict[word] = 1
    return count_dict

# Example usage
words = ["apple", "banana", "apple", "cherry", "banana", "apple"]
print(count_words(words))
# Output: {'apple': 3, 'banana': 2, 'cherry': 1}
```

Approach 2: Using get() method

```
def count_words_v2(word_list):
    """More Pythonic way using get()"""
    count_dict = {}
    for word in word_list:
        count_dict[word] = count_dict.get(word, 0) + 1
    return count_dict
```

Character Counting in String

```
def count_characters(text):
    """Count each character in a string"""
    char_count = {}
    for char in text:
        if char ≠ ' ':  # Skip spaces
            char_count[char] = char_count.get(char, 0) + 1
```

```
    return char_count

# Example
text = "hello world"
print(count_characters(text))
# Output: {'h': 1, 'e': 1, 'l': 3, 'o': 2, 'w': 1, 'r': 1, 'd': 1}
```

2. Finding Missing Numbers

Problem: Find missing numbers in a range

Approach 1: Using Set Difference

```
def find_missing_numbers(numbers, start, end):
    """Find missing numbers in a given range"""
    complete_set = set(range(start, end + 1))
    given_set = set(numbers)
    missing = complete_set - given_set
    return sorted(list(missing))

# Example
numbers = [1, 3, 5, 7, 10]
missing = find_missing_numbers(numbers, 1, 10)
print(missing)  # Output: [2, 4, 6, 8, 9]
```

Approach 2: Using Loop

```
def find_missing_numbers_v2(numbers, start, end):
    """Find missing numbers using loop"""
    numbers_set = set(numbers)
    missing = []

    for i in range(start, end + 1):
        if i not in numbers_set:
            missing.append(i)

    return missing
```

Find Single Missing Number

```
def find_single_missing(numbers, n):
    """Find single missing number from 1 to n"""
    # Using sum formula: n(n+1)/2
    expected_sum = n * (n + 1) // 2
    actual_sum = sum(numbers)
```

```
    return expected_sum - actual_sum

# Example: numbers from 1 to 5, missing 3
numbers = [1, 2, 4, 5]
missing = find_single_missing(numbers, 5)
print(missing)  # Output: 3
```

3. Sorting Without Built-in Methods

Bubble Sort

```
def bubble_sort(arr):
    """Simple bubble sort implementation"""
    n = len(arr)

    for i in range(n):
        # Track if any swaps occurred
        swapped = False

        for j in range(0, n - i - 1):
            if arr[j] > arr[j + 1]:
                # Swap elements
                arr[j], arr[j + 1] = arr[j + 1], arr[j]
                swapped = True

        # If no swaps, array is sorted
        if not swapped:
            break

    return arr

# Example
numbers = [64, 34, 25, 12, 22, 11, 90]
sorted_numbers = bubble_sort(numbers.copy())
print(sorted_numbers)  # Output: [11, 12, 22, 25, 34, 64, 90]
```

Selection Sort

```
def selection_sort(arr):
    """Selection sort implementation"""
    n = len(arr)

    for i in range(n):
        # Find minimum element in remaining array
        min_idx = i
```

```
        for j in range(i + 1, n):
            if arr[j] < arr[min_idx]:
                min_idx = j

        # Swap found minimum with first element
        arr[i], arr[min_idx] = arr[min_idx], arr[i]

    return arr
```

Insertion Sort

```
def insertion_sort(arr):
    """Insertion sort implementation"""
    for i in range(1, len(arr)):
        key = arr[i]
        j = i - 1

        # Move elements greater than key one position ahead
        while j ≥ 0 and arr[j] > key:
            arr[j + 1] = arr[j]
            j -= 1

        arr[j + 1] = key

    return arr
```

4. Letter Occurrence in Strings

Basic Letter Count

```
def count_letters(text):
    """Count occurrence of each letter"""
    letter_count = {}

    for char in text.lower():
        if char.isalpha():  # Only count letters
            letter_count[char] = letter_count.get(char, 0) + 1

    return letter_count

# Example
text = "Hello World!"
print(count_letters(text))
# Output: {'h': 1, 'e': 1, 'l': 3, 'o': 2, 'w': 1, 'r': 1, 'd': 1}
```

Find Most Frequent Letter

```
def most_frequent_letter(text):
    """Find the most frequent letter in text"""
    letter_count = count_letters(text)

    if not letter_count:
        return None

    max_count = 0
    most_frequent = None

    for letter, count in letter_count.items():
        if count > max_count:
            max_count = count
            most_frequent = letter

    return most_frequent, max_count

# Example
text = "programming"
result = most_frequent_letter(text)
print(f"Most frequent letter: '{result[0]}' appears {result[1]} times")
```

Check if Two Strings are Anagrams

```
def are_anagrams(str1, str2):
    """Check if two strings are anagrams"""
    # Remove spaces and convert to lowercase
    str1 = str1.replace(" ", "").lower()
    str2 = str2.replace(" ", "").lower()

    # If lengths are different, can't be anagrams
    if len(str1) ≠ len(str2):
        return False

    # Count characters in both strings
    count1 = count_letters(str1)
    count2 = count_letters(str2)

    return count1 == count2

# Example
```

```
print(are_anagrams("listen", "silent"))  # True
print(are_anagrams("hello", "world"))    # False
```

5. Additional Common Questions

Reverse a String Without Built-in Methods

```
def reverse_string(s):
    """Reverse string manually"""
    reversed_str = ""
    for i in range(len(s) - 1, -1, -1):
        reversed_str += s[i]
    return reversed_str

# Alternative using slicing concept manually
def reverse_string_v2(s):
    """Reverse using two pointers"""
    s_list = list(s)  # Convert to list for mutability
    left, right = 0, len(s_list) - 1

    while left < right:
        s_list[left], s_list[right] = s_list[right], s_list[left]
        left += 1
        right -= 1

    return ''.join(s_list)
```

Check if String is Palindrome

```
def is_palindrome(s):
    """Check if string is palindrome"""
    # Clean string: remove spaces, punctuation, make lowercase
    cleaned = ""
    for char in s.lower():
        if char.isalnum():
            cleaned += char

    # Check palindrome
    left, right = 0, len(cleaned) - 1
    while left < right:
        if cleaned[left] ≠ cleaned[right]:
            return False
        left += 1
        right -= 1
```

```
    return True

# Example
print(is_palindrome("A man a plan a canal Panama"))  # True
print(is_palindrome("race a car"))  # False
```

Find Duplicate Elements

```
def find_duplicates(arr):
    """Find all duplicate elements in array"""
    seen = set()
    duplicates = set()

    for item in arr:
        if item in seen:
            duplicates.add(item)
        else:
            seen.add(item)

    return list(duplicates)

# Example
numbers = [1, 2, 3, 4, 2, 5, 6, 3, 7, 8, 1]
print(find_duplicates(numbers))  # Output: [1, 2, 3]
```

Remove Duplicates from List

```
def remove_duplicates(arr):
    """Remove duplicates while preserving order"""
    seen = set()
    result = []

    for item in arr:
        if item not in seen:
            result.append(item)
            seen.add(item)

    return result

# Example
numbers = [1, 2, 3, 4, 2, 5, 6, 3, 7, 8, 1]
print(remove_duplicates(numbers))  # Output: [1, 2, 3, 4, 5, 6, 7, 8]
```

6. Interview Practice Scenarios

Scenario 1: E-commerce Product Analysis

Problem: You have a list of product purchases. Find the most popular product and count total purchases per product.

```
def analyze_purchases(purchases):
    """
    purchases = ['laptop', 'mouse', 'laptop', 'keyboard', 'mouse', 'laptop']
    """
    product_count = {}

    # Count each product
    for product in purchases:
        product_count[product] = product_count.get(product, 0) + 1

    # Find most popular
    most_popular = max(product_count, key=product_count.get)

    return {
        'counts': product_count,
        'most_popular': most_popular,
        'max_count': product_count[most_popular]
    }

# Example usage
purchases = ['laptop', 'mouse', 'laptop', 'keyboard', 'mouse', 'laptop']
result = analyze_purchases(purchases)
print(result)
```

Scenario 2: Student Grade System

Problem: You have student IDs from 1001 to 1010, but some students didn't submit assignments. Find missing student IDs.

```
def find_missing_students(submitted_ids, start_id=1001, end_id=1010):
    """Find students who didn't submit assignments"""
    all_students = set(range(start_id, end_id + 1))
    submitted = set(submitted_ids)
    missing = all_students - submitted
    return sorted(list(missing))

# Example
submitted = [1001, 1003, 1004, 1007, 1009, 1010]
missing_students = find_missing_students(submitted)
print(f"Students who didn't submit: {missing_students}")
```

Scenario 3: Text Message Analysis

Problem: Analyze a text message to find the most used words and character frequency.

```
def analyze_message(message):
    """Comprehensive text analysis"""
    # Word analysis
    words = message.lower().split()
    clean_words = []

    # Clean words (remove punctuation)
    for word in words:
        clean_word = ""
        for char in word:
            if char.isalpha():
                clean_word += char
        if clean_word:
            clean_words.append(clean_word)

    # Count words
    word_count = {}
    for word in clean_words:
        word_count[word] = word_count.get(word, 0) + 1

    # Count characters
    char_count = {}
    for char in message.lower():
        if char.isalpha():
            char_count[char] = char_count.get(char, 0) + 1

    return {
        'word_count': word_count,
        'char_count': char_count,
        'total_words': len(clean_words),
        'total_chars': sum(char_count.values())
    }

# Example
message = "Hello world! Hello Python programming world."
analysis = analyze_message(message)
print(analysis)
```

Scenario 4: Survey Response Validation

Problem: Sort survey responses by score without using built-in sort, and find duplicate responses.

```
def process_survey_responses(responses):
    """
    responses = [{'id': 1, 'score': 85}, {'id': 2, 'score': 92}, ...]
    """
    # Extract scores for sorting
    scores = [response['score'] for response in responses]

    # Bubble sort the responses based on scores
    n = len(responses)
    sorted_responses = responses.copy()

    for i in range(n):
        for j in range(0, n - i - 1):
            if sorted_responses[j]['score'] > sorted_responses[j + 1]['score']:
                sorted_responses[j], sorted_responses[j + 1] = sorted_responses[j
+ 1], sorted_responses[j]

    # Find duplicate scores
    score_count = {}
    for response in responses:
        score = response['score']
        score_count[score] = score_count.get(score, 0) + 1

    duplicate_scores = [score for score, count in score_count.items() if count >
1]

    return {
        'sorted_responses': sorted_responses,
        'duplicate_scores': duplicate_scores
    }

# Example
responses = [
    {'id': 1, 'score': 85},
    {'id': 2, 'score': 92},
    {'id': 3, 'score': 78},
    {'id': 4, 'score': 85},
    {'id': 5, 'score': 91}
]

result = process_survey_responses(responses)
print("Sorted by score:", result['sorted_responses'])
print("Duplicate scores:", result['duplicate_scores'])
```

Key Interview Tips

1. Always Ask Clarifying Questions

- "Should I consider case sensitivity?"
- "How should I handle empty inputs?"
- "Are there memory or time constraints?"

2. Think Out Loud

- Explain your approach before coding
- Discuss trade-offs between different solutions
- Mention time and space complexity

3. Start Simple, Then Optimize

- Get a working solution first
- Then discuss optimizations
- Consider edge cases

4. Common Time Complexities to Know

- **O(n)**: Single loop through data
- **O(n²)**: Nested loops (like bubble sort)
- **O(n log n)**: Efficient sorting algorithms
- **O(1)**: Constant time (hash table lookups)

5. Test Your Code

- Walk through with example inputs
- Consider edge cases (empty lists, single elements)
- Think about invalid inputs