Code Father – Debugging

> This sheet contains **buggy implementations** and their **corrected versions with explanations.**

PART - A

Question 1: Matrix Diagonal Sum

Intended Functionality: Calculate the sum of the main diagonal of a square matrix.

❌ **Buggy Code:**

```
def diagonal_sum(matrix):
    total = 0
    for i in range(len(matrix)):
        for j in range(len(matrix)):
            if i == j:
                total += matrix[i][j]
    return total

print(diagonal_sum([[1,2,3],[4,5,6],[7,8,9]]))  # Expected: 15
```

✅ **Fixed Code:**

```
def diagonal_sum(matrix):
    total = 0
    for i in range(len(matrix)):
        total += matrix[i][i]
    return total

print(diagonal_sum([[1,2,3],[4,5,6],[7,8,9]]))  # Output: 15
```

📌 **Explanation:** Inner loop was redundant. Diagonal is always `matrix[i][i]`.

Question 2: Password Validator

❌ **Buggy Code:**

```
def validate_password(password):
    if len(password) <= 8:   # ❌ Wrong: excludes exactly 8 chars
        return False
    has_upper = has_lower = has_digit = False

    for char in password:
        if char.isupper():
            has_upper = True
        elif char.islower():
            has_lower = True
        elif char.isdigit():
            has_digit = True

    return has_upper and has_lower and has_digit

print(validate_password("Pass123"))       # Expected: False
print(validate_password("Password123"))   # Expected: True
```

✅ **Fixed Code:**

```
def validate_password(password):
    if len(password) < 8:
        return False

    has_upper = any(c.isupper() for c in password)
    has_lower = any(c.islower() for c in password)
    has_digit = any(c.isdigit() for c in password)

    return has_upper and has_lower and has_digit

print(validate_password("Pass123"))       # Output: False
print(validate_password("Password123"))   # Output: True
```

📌 **Explanation:** Changed condition to `< 8`. Used `any()` for clarity.

Question 3: List Intersection with Duplicates

❌ **Buggy Code:**

```
def find_common(lst1, lst2):
    result = []
    for i in range(len(lst1)):
        if lst1[i] in lst2:
            result.append(lst1[i])
            # ❌ Does not remove from lst2, causing duplicate mismatch
    return result

print(find_common([1,2,2,3], [2,2,4]))  # Expected: [2,2]
```

✅ **Fixed Code:**

```
def find_common(lst1, lst2):
    result = []
    temp = lst2.copy()
    for item in lst1:
        if item in temp:
            result.append(item)
            temp.remove(item)
    return result

print(find_common([1,2,2,3], [2,2,4]))  # Output: [2,2]
```

📌 **Explanation:** Must remove matched element from the second list to handle duplicates properly.

Question 4: Word Frequency Counter

❌ **Buggy Code:**

```
def word_frequency(text):
    words = text.lower().split()
    freq = {}

    for word in words:
        word = word.strip('.,!?')
        freq[word] = freq.get(word, 0) + 1

    return freq

print(word_frequency("Hello world, hello Python!"))
# Expected: {'hello':2, 'world':1, 'python':1}
```

✅ **Fixed Code:**

```
import re

def word_frequency(text):
    words = re.findall(r'\b\w+\b', text.lower())
    freq = {}
    for word in words:
        freq[word] = freq.get(word, 0) + 1
    return freq

print(word_frequency("Hello world, hello Python!"))
# Output: {'hello':2, 'world':1, 'python':1}
```

📌 **Explanation:** Regex handles punctuation consistently; `strip()` misses cases.

Question 5: Perfect Number Check

❌ **Buggy Code:**

```python
def is_perfect(n):
    divisors = []
    for i in range(1, n+1):   # ❌ Includes n itself
        if n % i == 0:
            divisors.append(i)
    return sum(divisors) == n

print(is_perfect(6))
print(is_perfect(28))
print(is_perfect(10))
```

✅ **Fixed Code:**

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

print(is_perfect(6))   # Output: True
print(is_perfect(28))  # Output: True
print(is_perfect(10))  # Output: False
```

📌 **Explanation:** Should exclude `n` itself and optimize divisor summation.

PART - B

Question 6: String Compression

❌ **Buggy Code:**

```python
def compress_string(s):
    if not s:
        return ""

    compressed = ""
    count = 1

    for i in range(len(s)):
        if i+1 < len(s) and s[i] == s[i+1]:
            count += 1
        else:
            compressed += s[i] + str(count)
            count = 1

    return compressed

print(compress_string("aabcccccaaa"))  # Expected: "a2b1c5a3"
```

✅ **Fixed Code:**

```
def compress_string(s):
    if not s:
        return ""

    compressed = []
    count = 1
    current_char = s[0]

    for i in range(1, len(s)):
        if s[i] == current_char:
            count += 1
        else:
            compressed.append(current_char + str(count))
            current_char = s[i]
            count = 1
    compressed.append(current_char + str(count))

    return ''.join(compressed)

print(compress_string("aabcccccaaa"))  # Output: "a2b1c5a3"
```

📌 **Explanation:** Original version worked but was inefficient; new version is cleaner and avoids off-by-one confusion.

Question 7: Matrix Rotation

❌ **Buggy Code:**

```python
def rotate_matrix(matrix):
    n = len(matrix)
    rotated = [[0]*n for _ in range(n)]

    for i in range(n):
        for j in range(n):
            rotated[i][j] = matrix[n-j-1][i]  # ❌ Wrong assignment

    return rotated

print(rotate_matrix([[1,2,3],[4,5,6],[7,8,9]]))
# Expected: [[7,4,1],[8,5,2],[9,6,3]]
```

✅ **Fixed Code:**

```python
def rotate_matrix(matrix):
    n = len(matrix)
    rotated = [[0]*n for _ in range(n)]

    for i in range(n):
        for j in range(n):
            rotated[j][n-i-1] = matrix[i][j]

    return rotated

print(rotate_matrix([[1,2,3],[4,5,6],[7,8,9]]))
# Output: [[7,4,1],[8,5,2],[9,6,3]]
```

📌 **Explanation:** The formula for index mapping was incorrect.

Question 8: Valid Parentheses Checker

❌ **Buggy Code:**

```python
def is_valid_parentheses(s):
    stack = []
    mapping = {')': '(', '}': '{', ']': '['}

    for char in s:
        if char in mapping:  # ❌ Only closing brackets checked
            if not stack or stack.pop() != mapping[char]:
                return False
        else:
            stack.append(char)  # ❌ Adds invalid characters too
    return not stack

print(is_valid_parentheses("({[]})"))  # Expected: True
print(is_valid_parentheses("({[}])"))  # Expected: False
```

✅ **Fixed Code:**

```
def is_valid_parentheses(s):
    stack = []
    mapping = {')': '(', '}': '{', ']': '['}

    for char in s:
        if char in mapping.values():  # Opening brackets
            stack.append(char)
        elif char in mapping:         # Closing brackets
            if not stack or stack.pop() != mapping[char]:
                return False
        else:
            return False  # Invalid characters not allowed

    return not stack

print(is_valid_parentheses("({[]})"))  # Output: True
print(is_valid_parentheses("({[}])"))  # Output: False
```

📌 **Explanation:** Must handle both opening and closing properly, and reject invalid chars.

Question 9: List Flattener

❌ **Buggy Code:**

```
def flatten_list(nested_list):
    result = []
    for item in nested_list:
        if isinstance(item, list):
            result.append(flatten_list(item))  # ❌ Appends list, not elements
        else:
            result.append(item)
    return result

print(flatten_list([1, [2, [3, 4], 5]]))  # Expected: [1, 2, 3, 4, 5]
```

✅ **Fixed Code:**

```python
def flatten_list(nested_list):
    result = []
    for item in nested_list:
        if isinstance(item, list):
            result.extend(flatten_list(item))
        else:
            result.append(item)
    return result

print(flatten_list([1, [2, [3, 4], 5]]))  # Output: [1, 2, 3, 4, 5]
```

📌 **Explanation:** Must use `extend` to add flattened elements instead of appending the sublist.

Question 10: Prime Factors

❌ **Buggy Code:**

```python
def prime_factors(n):
    factors = []
    divisor = 2

    while n > 1:
        if n % divisor == 0:
            factors.append(divisor)
            n //= divisor
        # ❌ Missing else → infinite loop if not divisible
    return factors

print(prime_factors(56))  # Expected: [2, 2, 2, 7]
print(prime_factors(30))  # Expected: [2, 3, 5]
```

✅ **Fixed Code:**

```python
def prime_factors(n):
    factors = []
    divisor = 2
    while divisor * divisor <= n:
        while n % divisor == 0:
            factors.append(divisor)
            n //= divisor
        divisor += 1
    if n > 1:
        factors.append(n)
    return factors
```

```
print(prime_factors(56))  # Output: [2, 2, 2, 7]
print(prime_factors(30))  # Output: [2, 3, 5]
```

📌 **Explanation:** Added divisor increment in loop. Optimized by stopping at √n.

Marks Suggestion

Option 1 – Part A Exam (25 Marks Total)

- Q1. Matrix Diagonal Sum → 3 marks
- Q2. Password Validator → 4 marks
- Q3. List Intersection with Duplicates → 6 marks
- Q4. Word Frequency Counter → 5 marks
- Q5. Perfect Number Check → 7 marks

Total = 25 marks

Option 2 – Part B Exam (35 Marks Total)

- Q6. String Compression → 5 marks
- Q7. Matrix Rotation → 7 marks
- Q8. Valid Parentheses Checker → 8 marks
- Q9. List Flattener → 6 marks
- Q10. Prime Factors → 9 marks

Total = 35 marks