## **PRACTICE SHEET SOLUTION**

Q1)

[0]

[0, 1]

[0, 1, 2]

Q2) b)

Concept: Tuple is immutable, but its **mutable elements like lists inside** can be changed.

Q3) Expected Output: <class 'generator'>  
Tuple comprehension creates a **generator**, not a tuple!

Q4)

from typing import Tuple

def signal\_path(n: int, signal: float = 100, r: float = 0.9) -> Tuple[float, int]:

if n == 0:

return (round(signal, 2), 0)

# Reduce signal

new\_signal = signal \* r

# Recursive call

final\_strength, count\_below\_40 = signal\_path(n - 1, new\_signal, r)

# Check if this tower receives < 40% signal

if new\_signal < 40:

count\_below\_40 += 1

return (final\_strength, count\_below\_40)

# Example

print(signal\_path(5)) # Output: (59.05, 0)

Q5)

def drawBox(size):

# Special case: Draw nothing if size is less than 1:

    if size < 1:

        return

# Draw back line on top surface:

    print(' ' \* (size + 1) + '+' + '-' \* (size \* 2) + '+')

# Draw top surface:

    for i in range(size):

        print(' ' \* (size - i) + '/' + ' ' \* (size \* 2) + '/' + ' ' \* i + '|')

# Draw top line on top surface:

    print('+' + '-' \* (size \* 2) + '+' + ' ' \* size + '+')

# Draw front surface:

    for i in range(size - 1, -1, -1):

        print('|' + ' ' \* (size \* 2) + '|' + ' ' \* i + '/')

# Draw bottom lie on front surface:

    print('+' + '-' \* (size \* 2) + '+')

# In a loop, call drawBox() with arguments 1 to 5:

for i in range(1, 6):

    drawBox(i)

Q6)

def longest\_unique\_subarray(arr):

seen = set()

left = 0

max\_len = 0

start\_idx = 0

for right in range(len(arr)):

while arr[right] in seen:

seen.remove(arr[left])

left += 1

seen.add(arr[right])

if right - left + 1 > max\_len:

max\_len = right - left + 1

start\_idx = left

return max\_len, arr[start\_idx:start\_idx + max\_len]

# Example

arr = [5, 1, 3, 5, 2, 3, 4, 1]

length, subarray = longest\_unique\_subarray(arr)

print(f"Length: {length}")

print(f"Subarray: {subarray}")

Q7)

def smart\_sum(\*args):

total = 0

for item in args:

if isinstance(item, (list, tuple)):

total += sum(item)

else:

total += item

    return total

Q8)

def max\_depth(lst):

if not isinstance(lst, list):

return 0

if not lst: # empty list has depth 1

return 1

return 1 + max(max\_depth(item) for item in lst)

Q9)

def median(numbers):

# Special case: If the numbers list is empty, return None:

if len(numbers) == 0:

return None

# Sort the numbers list:

numbers.sort()

# Get the index of the middle number:

middleIndex = len(numbers) // 2

# If the numbers list has an even length, return the average of the

# middle two numbers:

if len(numbers) % 2 == 0:

return (numbers[middleIndex] + numbers[middleIndex - 1]) / 2

# If the numbers list has an odd length, return the middlemost number:

else:

return numbers[middleIndex]

Q10)

def multiply\_hex\_arrays(lst1, lst2):

# Step 1: Convert digit arrays to hex strings

hex\_str1 = ''.join(lst1)

hex\_str2 = ''.join(lst2)

# Step 2: Convert hex strings to integers

num1 = int(hex\_str1, 16)

num2 = int(hex\_str2, 16)

# Step 3: Multiply the numbers

product = num1 \* num2

# Step 4: Convert result to hex string (without '0x') and pad to 20 digits

product\_hex = hex(product)[2:].zfill(20).upper()

# Step 5: Convert to array of hex digits

result\_lst = list(product\_hex)

return result\_lst

# Example input lists (10-digit hexadecimal numbers)

lst1 = ['1', 'A', '3', 'F', '0', '4', '2', 'C', '9', 'B']

lst2 = ['0', 'F', '2', '3', 'D', '1', 'A', 'E', 'B', '7']

# Multiply and print result

result = multiply\_hex\_arrays(lst1, lst2)

print("Result List:", result)

Q11)

def rotate\_in\_place(arr, k):

    n = len(arr)

    k %= n  # Handle k > n

    # Helper to reverse part of array in-place

    def reverse(start, end):

        while start < end:

            arr[start], arr[end] = arr[end], arr[start]

            start += 1

            end -= 1

    # Step 1: Reverse entire array

    reverse(0, n - 1)

    # Step 2: Reverse first k elements

    reverse(0, k - 1)

    # Step 3: Reverse remaining n-k elements

    reverse(k, n - 1)

# Example

arr = [1, 2, 3, 4, 5, 6]

rotate\_in\_place(arr, 2)

print(arr)

Q12)

from typing import List

def diagonal\_sum(matrix: List[List[int]]) -> int:

    n = len(matrix)

    total = 0

    for i in range(n):

        total += matrix[i][i]             # Primary diagonal

        total += matrix[i][n - 1 - i]     # Secondary diagonal

    if n % 2 == 1:

        total -= matrix[n // 2][n // 2]   # Subtract center element once

    return total

# Example input

matrix = [

    [1, 2, 3],

    [4, 5, 6],

    [7, 8, 9]

]

print(diagonal\_sum(matrix))  # Output: 25

Q13)

# Loop over am and pm:

for meridiem in ['am', 'pm']:

# Loop over every hour:

for hour in ['12', '1', '2', '3', '4', '5', '6', '7', '8', '9', '10', '11']:

# Loop over every 15 minutes:

for minutes in ['00', '15', '30', '45']:

# Print the time:

print(hour + ':' + minutes + ' ' + meridiem)

Q14)

def product\_except\_self\_prefix\_suffix(nums):

    n = len(nums)

    pre = [1] \* n

    suf = [1] \* n

    ans = [1] \* n

    # Prefix product

    for i in range(1, n):

        pre[i] = pre[i - 1] \* nums[i - 1]

    # Suffix product

    for i in range(n - 2, -1, -1):

        suf[i] = suf[i + 1] \* nums[i + 1]

    # Final result

    for i in range(n):

        ans[i] = pre[i] \* suf[i]

    return ans

print(product\_except\_self\_prefix\_suffix([1, 2, 3, 4]))     # [24, 12, 8, 6]

print(product\_except\_self\_prefix\_suffix([0, 1, 2, 3]))     # [6, 0, 0, 0]

print(product\_except\_self\_prefix\_suffix([1, 0, 3, 0]))     # [0, 0, 0, 0]

print(product\_except\_self\_prefix\_suffix([5]))               # [1]

Q15)

def rob(nums):

    def helper(i):

        if i >= len(nums):

            return 0

        # Rob current house and skip next, or skip current house

        return max(nums[i] + helper(i + 2), helper(i + 1))

    return helper(0)

print(rob([1, 2, 3, 1]))  # Output: 4 (Rob house 1 and 3)

print(rob([2, 7, 9, 3, 1]))  # Output: 12 (Rob house 1, 3, and 5)

print(rob([2,1,1,9])) # Output: 11 (Rob house 1 and 3)