**Weekly Pair-Programming Challenges**

**Week 1: Build a Sorted Copy of a List**

You have a list of ten integers, called **original\_list**, arranged in no particular order. Your mission is to write Python code that:

1. **Creates** a new list named **sorted\_list\_copy**.
2. **Populates** **sorted\_list\_copy** with every integer from **original\_list**, arranged in **ascending order** (smallest to largest).
3. **Leaves** **original\_list** unchanged throughout your program.
4. **Avoids** using Python’s built-in sorting capabilities (list.sort() or sorted()). Instead, implement the sorting logic yourself using loops, comparisons, and optional helper functions.
5. **Prints** both lists at the end to verify your work:
   * **Original:** [the original unsorted list]
   * **Sorted Copy:** [your sorted list]

**Starting Data**

original\_list = [7, 2, 9, 4, 1, 8, 5, 10, 3, 6]

**Recommended Function Structure**

def make\_sorted\_copy(source\_list):

# your sorting logic here

return sorted\_list\_copy

* **Input**: source\_list (list of integers)
* **Output**: sorted\_list\_copy (sorted copy of source\_list)
* **Behavior**: Does not modify source\_list.

**Algorithm Constraints**

* Do **not** call .sort() or sorted().
* Use loops (for/while), conditionals (if/else), assignments, and optional helper functions.
* You may use list.append(), list.pop(), and list indexing (source\_list[index]).

**Stretch Goal: Error Handling** Wrap your logic in a try/except to catch invalid inputs and print:

print("Error: Input must be a list of integers.")

**Final Output**

sorted\_list\_copy = make\_sorted\_copy(original\_list)

print("Original:", original\_list)

print("Sorted Copy:", sorted\_list\_copy)

**Submission**

* A single **.py** file with your function and main program.
* Ensure correct output on at least two different ten-integer lists.

**Week 2: Generate All Primes up to N (Sieve of Eratosthenes)**

Implement the Sieve of Eratosthenes to list every prime number up to a given integer **maximum\_value**.

**Function signature**

def sieve\_of\_eratosthenes(maximum\_value):

# your sieve implementation here

return prime\_numbers\_list

* **Input**: maximum\_value (integer ≥ 2)
* **Output**: prime\_numbers\_list (list of all primes ≤ maximum\_value)

**Algorithm Steps**

1. Create a boolean list named **is\_prime\_flag\_list** of length maximum\_value + 1, initialized to all True.
2. Set is\_prime\_flag\_list[0] and is\_prime\_flag\_list[1] to False.
3. For each **candidate\_number** from 2 up to int(maximum\_value\*\*0.5):
   * If is\_prime\_flag\_list[candidate\_number] is True, mark every multiple:
   * multiple = candidate\_number \* candidate\_number
   * while multiple <= maximum\_value:
   * is\_prime\_flag\_list[multiple] = False

multiple += candidate\_number

1. Collect every index number where is\_prime\_flag\_list[number] is True into **prime\_numbers\_list**.
2. Return **prime\_numbers\_list**.

**Constraints**

* Do **not** use external prime libraries or filtering shortcuts.
* Use loops, conditionals, boolean lists, arithmetic, and list operations.
* You may use range() and descriptive variable names.

**Stretch Goal: Error Handling** Wrap your implementation in try/except to catch invalid inputs and print:

print("Error: Input must be an integer ≥ 2.")

**Sample Usage**

upper\_limit = 30

prime\_list = sieve\_of\_eratosthenes(upper\_limit)

print(f"Primes up to {upper\_limit}:", prime\_list)

Expected output:

Primes up to 30: [2, 3, 5, 7, 11, 13, 17, 19, 23, 29]

**Submission**

* A single **.py** file with your function and main program.
* Ensure correct output and error handling.