**Lab Exercise 2- Code Refactoring & Analysis Using GitHub Copilot in VS Code**

**Objective**

* Analyze existing code with GitHub Copilot.
* Improve efficiency, readability, and structure.
* Use Copilot’s suggestions to refactor code step by step.

**Step 1: Open a Python File with Inefficient Code**

1. Open VS Code and create a new file: complex\_refactor.py.
2. Copy and paste the following inefficient Python script:

import time

# Function to check if a number is prime

def is\_prime(n):

if n < 2:

return False

for i in range(2, n):

if n % i == 0:

return False

return True

# Function to find prime numbers in a range

def find\_primes(start, end):

primes = []

for num in range(start, end):

if is\_prime(num):

primes.append(num)

return primes

# Measure execution time

start\_time = time.time()

primes\_list = find\_primes(1, 50000)

end\_time = time.time()

print(f"Primes Found: {len(primes\_list)}")

print(f"Execution Time: {end\_time - start\_time} seconds")

**Step 2: Use GitHub Copilot to Analyze the Code**

1. Select the is\_prime() function and press Ctrl + I (if using Copilot Chat).
2. Ask Copilot: **How can I improve this function?**
3. Copilot will suggest:
   * Optimizing prime checking using sqrt(n).
   * Using a more efficient algorithm like the **Sieve of Eratosthenes**.

**Step 3: Apply Copilot’s Suggestions for Optimization**

Modify is\_prime() to improve performance:

from math import sqrt

def is\_prime(n):

"""Check if a number is prime using sqrt(n) optimization."""

if n < 2:

return False

if n in (2, 3):

return True

if n % 2 == 0 or n % 3 == 0:

return False

for i in range(5, int(sqrt(n)) + 1, 2):

if n % i == 0:

return False

return True

**Step 4: Use Copilot to Replace find\_primes() with a Faster Algorithm**

1. Select find\_primes() and type a comment above it:  
   # Improve this function for better performance using the Sieve of Eratosthenes
2. Press Tab to accept Copilot’s suggestion.

**Step 5: Optimized Code Using Sieve of Eratosthenes**

import time

def sieve\_of\_eratosthenes(limit):

"""Finds all prime numbers up to a given limit using the Sieve of Eratosthenes."""

primes = [True] \* (limit + 1)

primes[0], primes[1] = False, False

for i in range(2, int(limit\*\*0.5) + 1):

if primes[i]:

for multiple in range(i \* i, limit + 1, i):

primes[multiple] = False

return [num for num, is\_prime in enumerate(primes) if is\_prime]

# Measure execution time

start\_time = time.time()

primes\_list = sieve\_of\_eratosthenes(50000)

end\_time = time.time()

print(f"Primes Found: {len(primes\_list)}")

print(f"Execution Time: {end\_time - start\_time:.6f} seconds")

**Step 6: Validate Performance Improvement**

1. Run the original and optimized scripts.
2. Compare execution times.
3. The optimized version should be significantly faster.

**Lab Summary**

1. Used GitHub Copilot to analyze and suggest improvements.
2. Optimized is\_prime() using sqrt(n).
3. Replaced find\_primes() with the **Sieve of Eratosthenes**.
4. Validated performance improvement by measuring execution time.