**Option #2: Using Python Programming language**

Chioma Chance

Colorado State University Global

CSC507: Ethical Leadership in Software Development

Dr. L

01/05/2025

**Summary of Results**

In this assignment, I compared the performance of a Bash script and a Python program to generate 1,000,000 random numbers and write them to a file. Below is a summary of the methods used, observations, and conclusions based on the results.

**Methods Used**

1. **Bash Script (numbers.sh)**:
   * The script used a for loop to generate random numbers using $RANDOM and wrote them to file1.txt.
   * Execution time: **76 seconds**.
2. **Python Program (numbers.py)**:
   * The program generated random numbers using Python's random.randint() function and wrote them to file2.txt.
   * Execution time: **0.84 seconds**.
3. **Optimized Python Program (Multithreading)**:
   * I modified the Python script to use multithreading, dividing the task into smaller chunks handled by multiple threads.
   * Execution time: **1.66 seconds**.

**Results and Observations**

1. The Bash script took significantly longer to complete compared to the Python program. This is likely due to Bash being slower at handling file I/O operations and loops for large-scale tasks. Bash also lacks the efficiency of Python's optimized libraries.
2. The Python program performed exceptionally well, completing the task in under a second. Python's ability to handle large data processing tasks efficiently is apparent here.
3. Interestingly, the multithreaded Python program took slightly longer than the non-threaded version. This is likely due to the overhead of managing multiple threads for a task that is heavily I/O bound (writing to a single file). While multithreading can improve performance in CPU-intensive tasks, it might not always be beneficial for file I/O.

**Conclusion**

From these results, Python is better suited for tasks involving large-scale data processing compared to Bash. While multithreading did not yield a performance gain in this case, it could be useful for more complex or CPU-intensive tasks. If the system had double the processing power, the Python script would likely see even faster results, though the improvement might be less noticeable for I/O-bound tasks like file writing.

This exercise demonstrated the importance of choosing the right tools and techniques for specific tasks, and it gave me a better understanding of how programming languages and methods impact execution times.