

**MSCS-632: Assignment 7: Lab Report: Cross-Paradigm Statistics Calculator Using C, OCaml, and
Python**

Sandesh Pokharel

University of the Cumberland

MSCS-632 Advanced Programming Languages


Dr. Vanessa Cooper

April 23, 2025

Introduction

This project demonstrates how the same statistical problem—calculating the mean, median, and mode from a list of integers—can be approached using three distinct programming paradigms: **Procedural (C)**, **Functional (OCaml)**, and **Object-Oriented (Python)**. The goal was not only to implement the required calculations but also to gain hands-on experience working with different styles of programming. By doing so, we were able to analyze the strengths, limitations, and syntactical differences of each language and paradigm.

GitHub Repository

 All the source code is available in the public GitHub repository:

 <https://github.com/sanspokharel26677/MSCS-632-Assignment7>

Repository contains:

- python-version/
- c-version/
- ocaml-version/
- README.md for instructions

Problem Overview

The core task was simple:

Given a list of integers, compute:

- **Mean** (average of the numbers),
- **Median** (middle number in the sorted list),
- **Mode** (most frequent number(s)).

However, the real learning came from how each programming language and paradigm approached the same problem with different tools and mindsets. The final solution includes fully working programs in C, OCaml, and Python.

Implementation Summaries

Python Version (Object-Oriented Programming)

- Built using a class called `StatisticsCalculator`.
- Each operation (mean, median, mode) was encapsulated as a method.
- Used Python's built-in `list` and `collections.Counter` for simplicity.
- Also included user input for dynamic testing.
- This version was the cleanest and easiest to expand or reuse.

C Version (Procedural Programming)

- Implemented using simple functions and arrays.
- Sorting done manually using bubble sort.

- Memory allocated dynamically with malloc, then freed with free.
- Input handled via scanf, output via printf.
- Required the most boilerplate, but was highly transparent and close to hardware.

OCaml Version (Functional Programming)

- Used immutable lists and pure functions throughout.
- Used higher-order functions like List.fold_left, List.sort, etc.
- Handled mode calculation using custom association list logic.
- Extended version includes user input (split and parsed using functional tools).
- Syntax was quite different and had a learning curve, but logic was very concise once understood.

Comparative Analysis: Paradigm Differences

Feature	Python (OOP)	C (Procedural)	OCaml (Functional)
Ease of use	Very beginner-friendly	Intermediate	Steep learning curve
Memory Handling	Automatic (GC)	Manual (malloc/free)	Automatic (GC)

Input Handling	Simple	Verbose	Verbose and less intuitive
Paradigm Benefit	Modularity & reuse	Control & performance	Immutability & conciseness
Reusability	High	Moderate	Moderate

Each paradigm brings a unique way of thinking. Python's OOP structure felt most familiar and modern. C gave us full control, especially over memory. OCaml introduced a fresh perspective—using recursion and pure functions without side effects.

Challenges and Observations

- OCaml was entirely new to me and required reading documentation just to understand simple operations like reading input or splitting a string.
- C required more care with memory management and sorting logic.
- Python was smooth overall but could hide too much logic behind built-in functions if we weren't careful.
- The biggest mental switch was moving from mutable state (in C & Python) to immutable state in OCaml.
- Installing OCaml and getting opam configured properly on macOS also took a few extra steps.

Screenshots

🔴 Python Version

- 🖼️ Screenshot: Python terminal input/output with user-entered data

```
[mac@Mac ocaml-version % cd ../
[mac@Mac MSCS-632-Assignment7 % ls
c-version      ocaml-version  python-version  README.md      stats
[mac@Mac MSCS-632-Assignment7 % cd python-version
[mac@Mac python-version % ls
statistics_calculator.py
[mac@Mac python-version % python3 statistics_calculator.py
Enter integers separated by space: 1 2 3 4 5

Input Numbers: [1, 2, 3, 4, 5]
Mean: 3.0
Median: 3
Mode: [1, 2, 3, 4, 5]
[mac@Mac python-version % python3 statistics_calculator.py
Enter integers separated by space: 4 2 5 2 3 5 2


Input Numbers: [4, 2, 5, 2, 3, 5, 2]
Mean: 3.2857142857142856
Median: 3
Mode: [2]
[mac@Mac python-version % python3 statistics_calculator.py
Enter integers separated by space: 10 15 10 20 30 15 10 25

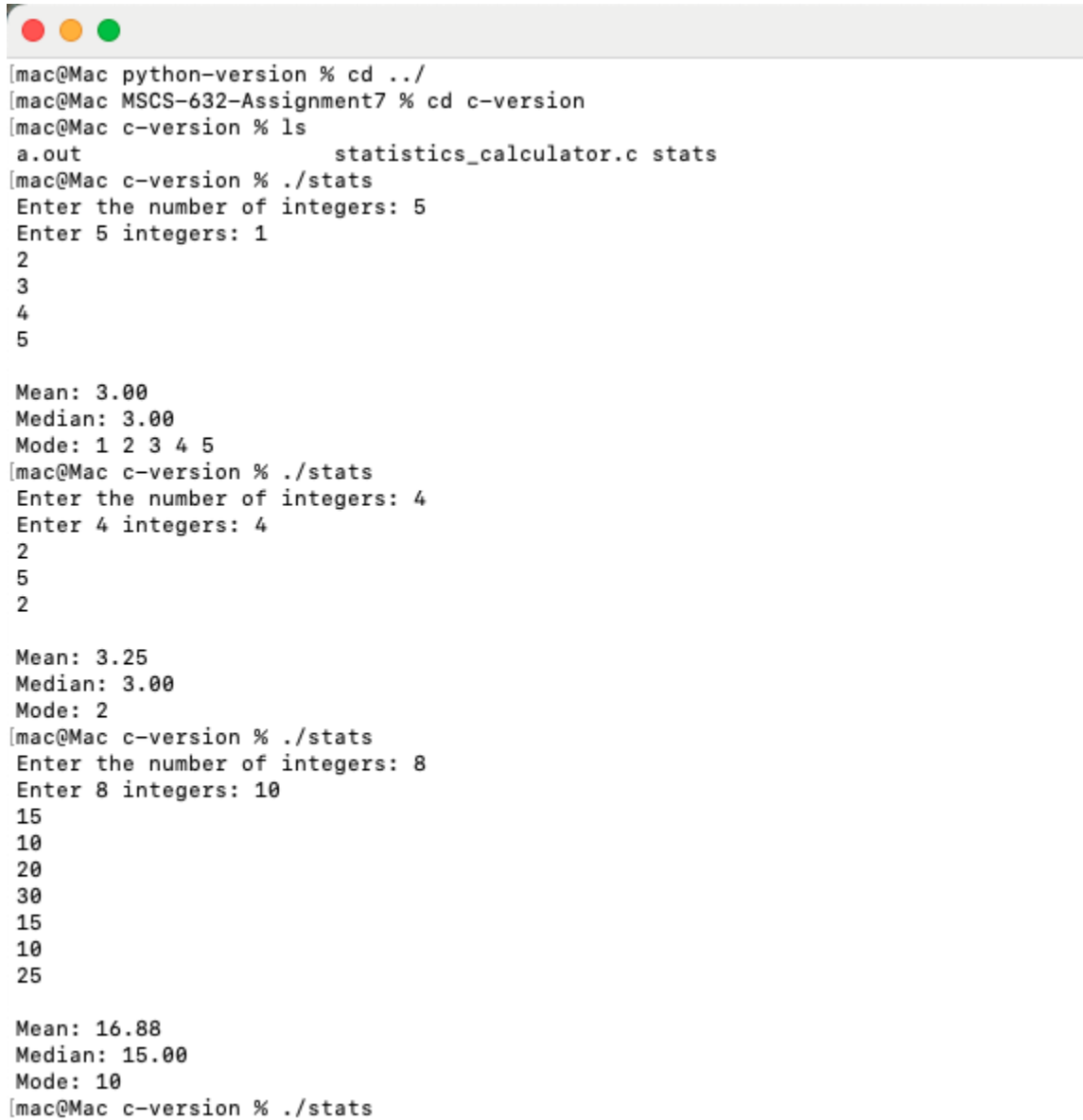
Input Numbers: [10, 15, 10, 20, 30, 15, 10, 25]
Mean: 16.875
Median: 15.0
Mode: [10]
[mac@Mac python-version % python3 statistics_calculator.py
Enter integers separated by space: 7 7 2 2 5 6

Input Numbers: [7, 7, 2, 2, 5, 6]
Mean: 4.833333333333333
Median: 5.5
Mode: [7, 2]
[mac@Mac python-version % python3 statistics_calculator.py
Enter integers separated by space: 8 6 6 8 7 7

Input Numbers: [8, 6, 6, 8, 7, 7]
Mean: 7.0
Median: 7.0
Mode: [8, 6, 7]
[mac@Mac python-version % ]
```


🔴 C Version

-  Screenshot: Terminal showing compiled C program and output

A screenshot of a macOS terminal window. The window has a title bar with three colored buttons (red, yellow, green) on the left. The terminal text shows a user navigating through directories and running a C program called 'statistics_calculator.c'. The program prompts for the number of integers and then for the integers themselves. It then calculates and displays the Mean, Median, and Mode for each set of integers.

```
[mac@Mac python-version % cd ../  
[mac@Mac MSCS-632-Assignment7 % cd c-version  
[mac@Mac c-version % ls  
a.out          statistics_calculator.c stats  
[mac@Mac c-version % ./stats  
Enter the number of integers: 5  
Enter 5 integers: 1  
2  
3  
4  
5  
  
Mean: 3.00  
Median: 3.00  
Mode: 1 2 3 4 5  
[mac@Mac c-version % ./stats  
Enter the number of integers: 4  
Enter 4 integers: 4  
2  
5  
2  
  
Mean: 3.25  
Median: 3.00  
Mode: 2  
[mac@Mac c-version % ./stats  
Enter the number of integers: 8  
Enter 8 integers: 10  
15  
10  
20  
30  
15  
10  
25  
  
Mean: 16.88  
Median: 15.00  
Mode: 10  
[mac@Mac c-version % ./stats
```

OCaml Version

-  Screenshot: OCaml input/output from terminal using ./stats

```
mac@Mac c-version % cd ../ocaml-version
mac@Mac ocaml-version % ./stats
Enter integers separated by space:
1 2 3 4 5
Input Numbers: 1 2 3 4 5
Mean: 3.00
Median: 3.00
Mode: 1 2 3 4 5
mac@Mac ocaml-version % ./stats
Enter integers separated by space:
4 2 5 2 3 5 2
Input Numbers: 4 2 5 2 3 5 2
Mean: 3.29
Median: 3.00
Mode: 2
mac@Mac ocaml-version % ./stats
Enter integers separated by space:
10 15 10 20 30 15 10 25
Input Numbers: 10 15 10 20 30 15 10 25
Mean: 16.88
Median: 15.00
Mode: 10
mac@Mac ocaml-version % ./stats
Enter integers separated by space:
7 7 2 2 5 6
Input Numbers: 7 7 2 2 5 6
Mean: 4.83
Median: 5.50
Mode: 2 7
mac@Mac ocaml-version % ./stats
Enter integers separated by space:
8 6 6 8 7 7
Input Numbers: 8 6 6 8 7 7
Mean: 7.00
Median: 7.00
Mode: 6 7 8
mac@Mac ocaml-version % █
```

Conclusion

This assignment was a fun yet insightful exploration of programming paradigms. Writing the same program in three languages helped me truly understand not just syntax differences, but **how differently problems are approached** depending on the paradigm. While OCaml may not be something I use daily, it gave me a whole new appreciation for functional thinking. On the other

hand, C reminded me of the foundations of memory and control, and Python reaffirmed why it's a go-to for clean, readable software.