

GitHub Link: https://github.com/vtadigotla35101/MSCS-632_Assignment7

This report evaluates the implementation of a statistics calculator across three distinct programming languages like **C**, **OCaml**, and **Python**. While the mathematical output remains identical, the mental model required for each approach varies significantly.

Execution Screenshot in C Language:

```

1 #include <stdio.h>
2 #include <stdlib.h>
3
4 // Simple bubble sort for the median
5 void sort(int* arr, int n) {
6     for (int i = 0; i < n - 1; i++) {
7         for (int j = 0; j < n - i - 1; j++) {
8             if (arr[j] > arr[j + 1]) {
9                 int temp = arr[j];
10                arr[j] = arr[j + 1];
11                arr[j + 1] = temp;
12            }
13        }
}
Mean: 2.40
Median: 2.00
Mode(s): 2

...Program finished with exit code 0
Press ENTER to exit console.

```

1. C Language

In C, the experience is defined by granularity. We are not only calculating a mean but managing a block of memory.

- **The Experience:** Implementing the solution felt like building a machine from scratch. I had to manually manage the array indices and write a sorting algorithm just to find the median.
- **Challenges:** The lack of built-in high-level functions meant that Mode required nested loops and manual counters. Memory safety is a constant concern; one wrong index could crash the program.
- **Observation:** Procedural code is highly transparent. You see every move the CPU makes, but it requires the most boilerplate code to perform simple tasks.

Execution Screenshot in OCAML Language:

2. OCaml Language

OCaml shifts the focus from "how to do it" to "**what it is.**" The implementation relies on transforming data through expressions rather than changing the state of variables.

- **The Experience:** It felt mathematical. Using List.fold_left to calculate a sum is more abstract than a for loop. The data is **immutable**, meaning once the list is created, it never changes; instead, new lists are created during transformations.
 - **Challenges:** Thinking recursively and avoiding "loops" is the biggest hurdle for those used to C or Python. The syntax is strict, and the type system will not let the code compile if there is even a slight logic error in data types.

Observation: Functional programming is incredibly concise. The code is less prone to bugs because there are no global variables or hidden states to track.

Execution Screenshot in PYTHON Language:

```

1 class StatisticsCalculator:
2     def __init__(self, data):
3         self.data = sorted(data)
4
5     def calculate_mean(self):
6         return sum(self.data) / len(self.data)
7
8     def calculate_median(self):
9         n = len(self.data)
10        mid = n // 2
11        if n % 2 == 0:
12            return (self.data[mid - 1] + self.data[mid]) / 2

```

Mean: 2.4
Median: 2
Mode: [2]

...Program finished with exit code 0
Press ENTER to exit console.

3. Python Language:

Python's OOP approach focuses on organization and encapsulation. The data and the behaviors like mean, median, mode are bundled together into a single Object.

- **The Experience:** Implementing the StatisticsCalculator class felt like creating a reusable tool. Once the object is initialized, all methods have easy access to the data through the `self` keyword.
- **Challenges:** The main challenge is overhead. For a simple math problem, setting up a class might feel like over-engineering. However, it makes the code the easiest to maintain and expand.
- **Observation:** OOP is the most human-readable paradigm. It mimics how we think about objects in the real world, making the logic intuitive and the code highly reusable.

Conclusion

Each language serves a specific purpose. **C** is ideal for performance and low-level control. **OCaml** is superior for mathematical correctness and data transformation. **Python (OOP)** excels in developer productivity and building complex, organized systems.