**Multi Paradigm Programming Report**

**Introduction**

This is my report on my experience creating a GPA Calculator using different programming paradigms in python and C programming. The report will analyze how applying the paradigms worked in my specific solutions.

**Procedural Programming - Python:**

**Introduction:**

The procedural programming solution demonstrates the application of paradigms, specifically procedural programming and data manipulation using pandas, to process and analyze student performance data. The solution utilizes the Python programming language and the pandas library to perform calculations and generate insights from the data provided. This report discusses how these paradigms were applied in the solution and evaluates their effectiveness.

**Procedural Programming Paradigm:**

1. **Modularity:** The code is organized into several functions, each responsible for a specific task. Functions like calculate\_letter\_grade, calculate\_gpa, process\_data, and run\_live\_mode handle distinct parts of the GPA calculation and data processing.
2. **Procedural Execution Flow:** Themain execution flow is organized in the \_\_main\_\_ block at the end of the script. It starts by defining the module\_columns and csv\_file, then calls the process\_data function to read and process data from the CSV file. After that, it runs the run\_live\_mode function to enter live mode for GPA calculation and data input.
3. **Data Encapsulation:** Functions are designed to encapsulate specific functionality, promoting code organization and maintainability. For example, calculate\_letter\_grade takes a numerical mark as input and returns the corresponding letter grade, while calculate\_gpa takes a list of marks and returns the GPA.
4. **Reusability:** Functions like calculate\_letter\_grade and calculate\_gpa can be reused in different parts of the code or in other projects since they are independent of the specific data used, enhancing code efficiency.
5. **Error Handling:** The code includes try-except blocks to handle potential errors, improving the program's robustness and providing informative error messages to the user.
6. **Separation of Concerns:** Each function focuses on a single task, promoting a clean separation of concerns. For example, the process\_data function reads the CSV, calculates GPA-related data, and returns a processed DataFrame, while the run\_live\_mode function handles the interactive live mode for adding new student data and displaying processed data.
7. **Parameterization:** The functions are parameterized, making them more flexible and adaptable. For example, the process\_data function takes module\_columns and csv\_file as inputs, allowing it to process data for different sets of modules and CSV files.

**Procedural Programming Approach:**

The solution primarily follows a procedural programming paradigm, executing a series of steps to accomplish the GPA calculation and data processing. It utilizes control flow and conditional statements to assign letter grades based on the marks obtained by each student and iterates over module columns to calculate the letter grades.

**Conclusion:**

In the provided GPA calculator solution, the procedural paradigm is applied effectively to break down the problem into smaller, manageable procedures or functions. This approach enhances code readability, reusability, and ease of testing. However, it is essential to consider the limitations of the procedural paradigm, such as limited encapsulation and the potential for code duplication in larger applications. In this case, procedural programming provides a clear and organized approach to solving the problem, breaking it down into sequential steps and utilizing control flow constructs for decision-making.

**OOP – Python**

**Introduction**

. Object-oriented programming (OOP) is a method of structuring a program by bundling related properties and behaviours into individual objects.

The code for the example of use of OOP in python implements a GPA (Grade Point Average) calculator using Object-Oriented Programming (OOP) paradigm. Let's analyze how the OOP paradigm has been applied in this specific solution:

1. **Classes**: The code defines several classes, each with its own responsibilities:
   * **GPAScale**: Represents the scale of letter grades and their corresponding GPA values. It has methods to calculate letter grades and GPA based on the marks.
   * **Student**: Represents a student with a name and their marks in different modules. It has methods to add marks and calculate GPA.
   * **CSVManager**: Handles reading and saving data to/from a CSV file.
   * **GPACalculator**: Orchestrates the GPA calculation process, processing data from the CSV file, running live mode to interactively add new student data, and displaying the results.
2. **Encapsulation**: Each class encapsulates its data and functionality. For example, the **GPAScale** class hides the GPA scale dictionary and provides methods to interact with it. Similarly, the **Student** class hides the internal **marks** dictionary and provides methods to manipulate it.
3. **Inheritance**: The code doesn't explicitly use inheritance, but it leverages composition by creating instances of other classes within the **GPACalculator** class.
4. **Abstraction**: The classes are designed to abstract specific functionality, making it easier to understand and maintain the code. For instance, the **CSVManager** class abstracts away the complexity of reading and writing data to/from a CSV file.
5. **Polymorphism**: The code doesn't explicitly use polymorphism, but it demonstrates a form of polymorphism through the **calculate\_gpa()** method. Both the **GPAScale** and **Student** classes have this method, but they have different implementations, allowing them to calculate GPA based on different data.
6. **Class Instantiation and Composition**: In the **GPACalculator** constructor, instances of the **CSVManager** and **GPAScale** classes are created and assigned as attributes of the **GPACalculator** object, showing the use of composition.

**OOP Programming Approach:**

I adopted a structured and organized approach to implement a GPA (Grade Point Average) calculator using the Object-Oriented Programming (OOP) paradigm. The code demonstrates a clear separation of responabilities by defining several classes, each responsible for specific tasks. The **GPAScale** class manages the GPA scale, letter grades, and their corresponding values. The **Student** class represents individual students and their marks in different modules, providing methods to calculate their GPA based on the given scale. The **CSVManager** class handles reading and saving data to/from a CSV file, promoting data persistence. The main orchestrator of the GPA calculation process is the **GPACalculator** class, which composes instances of other classes to manage the overall functionality. I effectively utilized composition to leverage the functionalities of other classes, avoiding unnecessary inheritance and promoting code reusability. As with the procedural paradigm the code features an interactive live mode, allowing users to add new student data, display updated data, or process the existing data with ease

**Conclusion:**

Overall, the code applies the OOP paradigm effectively by using classes to model relevant entities, encapsulating data and behaviour within each class, and utilizing composition to combine different classes and their functionalities. This approach promotes code reusability, maintainability, and readability.

**Procedural Programming (with OOP Elements– C Programming:**

Elements of both Object-Oriented Programming (OOP) paradigm and Procedural paradigm have been applied, although the primary paradigm used is Procedural. Here I will analyse how each paradigm is implemented:

1. Procedural Paradigm: The code is primarily structured in a procedural manner, with functions being the central building blocks of the program. Functions are used to perform specific tasks, such as calculating GPA, variance, standard deviation, and processing data. The procedural paradigm focuses on using functions to break down the program's logic into smaller, reusable components.

In the main function, the procedural approach is evident as it sequentially calls the functions to process data and run the live mode GPA calculator. It follows a step-by-step execution flow, starting with processing data and then moving on to live mode. Procedural programming organizes the code into a set of procedures that can be executed sequentially.

1. Object-Oriented Programming (OOP) Paradigm: Although the code doesn't exhibit a full-fledged OOP design, there is an element of OOP in the struct definitions. The code uses structs to create custom data types, "struct GPAData" and "struct GPAScale," to encapsulate related data. These structs define a blueprint for objects, and they group data and behavior (functions) together.

The structs represent entities in the problem domain, such as student data and GPA scales. While the code doesn't utilize inheritance, polymorphism, or explicit object-oriented design principles, the use of structs showcases a basic level of encapsulation, which is one of the key concepts in OOP.

**Conclusion:**

In summary, the code primarily follows the Procedural paradigm with its focus on sequential execution and the use of functions for code organization. However, it also incorporates a basic element of OOP through the definition of structs for data encapsulation and abstraction.

To fully embrace the Object-Oriented Programming paradigm, the code could be further expanded to utilize classes, inheritance, polymorphism, and other OOP principles. Additionally, the code could be organized into classes with member functions and member variables, allowing for better code reuse and maintenance.