**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:**

The solution primarily follows a procedural programming paradigm, where the focus is on executing a series of steps to accomplish a task. Here are the key aspects of procedural programming employed in the solution:

**A. Step-by-step Execution:**

The solution is structured in a sequential manner, with each step being executed in the order they are written. It starts by defining the GPA scale, then reads the CSV file using pandas, calculates letter grades for each module, computes GPAs for each student, and finally calculates the overall GPA.

**B. Control Flow and Conditionals:**

The solution uses conditional statements, specifically the if-else construct, to assign letter grades based on the marks obtained by each student. The conditional statements allow for different outcomes based on the ranges defined in the GPA scale.

**D. Iteration:**

The solution iterates over the module columns to calculate the letter grade for each module. It uses a *for loop* to perform the same set of operations on each module.

The procedural programming paradigm breaks down the solution into manageable steps and leverages control flow and iteration constructs to handle different scenarios efficiently.

**Conclusion:**

In the provided GPA calculator solution, the procedural paradigm is applied to break down the problem into smaller, manageable procedures or functions. This approach enhances code readability, reusability, and ease of testing. However, it is important to consider the limitations of the procedural paradigm, such as limited encapsulation and the potential for code duplication, especially in larger applications.

In this instance 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.

**Object Orientated Programming (OOP) - Python:**

**Introduction:**

The provided code demonstrates the application of Object-Oriented Programming (OOP) in developing a GPA calculator. Object-Oriented Programming is a programming paradigm that focuses on creating objects, which are instances of classes that encapsulate data and behavior. This report will discuss how applying OOP principles worked in the specific solution, highlighting the benefits and limitations of this approach.

**Class Design:**

In the solution, a class named "GPA\_Calculator" is defined, which encapsulates the GPA calculation logic and related functionalities. The class has an initializer method "init" that sets up the necessary data, such as the module columns and the GPA scale. This class design promotes encapsulation, as the data and methods related to GPA calculation are grouped together.

**Encapsulation and Data Hiding:**

The GPA\_Calculator class encapsulates the relevant methods, such as "calculate\_letter\_grade" and "calculate\_gpa," within the class. These methods are accessible only through class instances, ensuring data hiding and preventing direct modification or access to internal data structures. This promotes data integrity and reduces the risk of unintended modifications.

**Method Abstraction and Reusability:**

The GPA\_Calculator class abstracts the GPA calculation logic into the "calculate\_gpa" method, which takes a list of marks as input and returns the calculated GPA. This abstraction promotes reusability, as the method can be used in multiple contexts without rewriting the same logic. The "calculate\_letter\_grade" method is also encapsulated within the class and reused within the GPA calculation process.

**Modularity and Separation of Concerns:**

The GPA\_Calculator class separates the concerns related to GPA calculation, data processing, and live mode functionality. The "process\_data" method processes the data from a CSV file, performs calculations, and returns the result as a pandas DataFrame. The "run\_live\_mode" method handles the interactive mode for calculating GPA based on user inputs. This modular approach enhances code organization and readability.

**Inheritance and Polymorphism:**

In the provided code, the GPA\_Calculator class does not explicitly demonstrate inheritance or polymorphism. However, if the solution were to be extended in the future, additional classes could inherit from the GPA\_Calculator class to provide specialized functionality or different calculation methods. This would facilitate code reuse and support polymorphism, where different objects can be treated interchangeably.

**Benefits of Object-Oriented Programming:**

**Code Organization and Reusability:** OOP promotes modular code design, allowing for better organization and reusability of code. The encapsulation of data and behavior within classes facilitates code reuse and reduces code duplication.

**Abstraction and Data Hiding:** OOP allows for abstraction, where complex operations can be encapsulated within methods, promoting simplicity and ease of use. Data hiding ensures that internal implementation details are hidden, reducing the risk of unintended modifications.

**Encapsulation and Modularity:** OOP encourages encapsulation, grouping related data and behavior together within classes. This promotes modularity, making it easier to understand and maintain the codebase.

**Extensibility:** OOP provides a foundation for building extensible systems. By leveraging concepts like inheritance and polymorphism, new classes can be created to inherit and extend the functionality of existing classes, promoting code scalability and adaptability.

**Limitations of Object-Oriented Programming:**

**Overhead:** OOP can introduce additional overhead due to the need to define classes, create objects, and manage relationships between objects. This can result in slightly slower performance compared to procedural programming in certain scenarios.

**Steeper Learning Curve:** OOP requires a solid understanding of concepts such as classes, objects, inheritance, and polymorphism. Learning these concepts and applying them effectively may take more time and effort compared to procedural programming.

**Potential Complexity:** Poorly designed class hierarchies or excessive use of inheritance can lead to complex and hard-to-maintain code. Careful consideration should be given to the design and structure of classes to prevent unnecessary complexity.

**Conclusion**:

The application of Object-Oriented Programming in the provided GPA calculator solution brings several benefits such as code organization, reusability, encapsulation, and modularity. The use of classes promotes data integrity, abstraction, and separation of concerns. While OOP introduces additional complexity and a steeper learning curve, it provides a solid foundation for extensible and maintainable code. Overall, the solution demonstrates the advantages of OOP in developing a structured and reusable GPA calculator.

**Object Orientated Programming (OOP) – C Programming:**

**Introduction:**

This code includes elements of Object-Oriented Programming (OOP) in C. Although C is not a pure object-oriented language, it is possible to implement certain concepts of OOP using struct and function compositions.

In the given code, the structures GPA\_Scale, Module, and Student represent objects with their respective attributes and behaviors. These structures encapsulate data and related functions.

**The code also demonstrates the following key concepts of OOP:**

**Encapsulation:** The structures (GPA\_Scale, Module, and Student) encapsulate data and related functions together, keeping them organized and self-contained.

**Abstraction:** The structures hide their internal implementation details from the outside world. The user interacts with these structures through well-defined functions (calculate\_letter\_grade, calculate\_gpa, calculate\_highest\_scoring\_module, and process\_data) without needing to know the underlying implementation.

**Data Hiding:** The member variables of the structures (GPA\_Scale, Module, and Student) are declared with the private access modifier by default in C, preventing direct access from outside the structure. Functions like calculate\_letter\_grade and calculate\_gpa provide an interface to modify and access the data in a controlled manner.

**Modularity:** The code is divided into smaller functions, each responsible for a specific task. This promotes modularity, reusability, and easier maintenance.

**NB.:** It's important to note that this code does not utilize concepts such as inheritance or polymorphism, which are commonly associated with OOP.

**Data Manipulation using pandas (Extra Procedural)**:

The solution utilizes the pandas library to manipulate and analyze tabular data. Here are the key instances of data manipulation using pandas in the solution:

**A. Reading the CSV file:**

The pandas read\_csv function is used to read the input CSV file and create a DataFrame. This allows for easy access and manipulation of the data in a tabular format.

**B. Creating New Columns:**

The solution creates new columns in the DataFrame to store the calculated letter grades for each module. It uses the apply method along with a lambda function to apply the grading logic to each mark in the module columns.

C. **Calculating GPAs:**

The solution calculates the GPA for each student by applying the GPA scale to the letter grades for each module. It uses pandas' applymap method to map the letter grades to their corresponding GPA values and then calculates the mean GPA across all modules for each student.

**D. Calculating Overall GPA:**

The solution calculates the overall GPA by taking the mean of all the calculated GPAs for each student.

The use of pandas simplifies data manipulation tasks, allowing for concise and efficient code. The library provides powerful functions and methods that enable operations on tabular data in an intuitive manner.

Data manipulation using pandas allows for easy handling and analysis of tabular data, simplifying complex operations and enhancing code readability. Overall, the solution demonstrates a structured and efficient approach to processing and analyzing student performance data, leveraging the strengths of procedural programming and pandas for effective problem-solving.