**Multi Paradigm Programming Report**

**Student Name:** Enda Lynch

**Student Number:** G00398795

**Module Name:** Multii-Paradigm Programming

**Module Number:** 52960

**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 and discuss how applying the paradigms (Object Orientated Programming (OOP) and Procedural Programming) worked in my specific solutions.

**Procedural Programming – Python and C:**

**Introduction:**

The procedural programming solution demonstrates the application of paradigms, specifically procedural programming and data manipulation using pandas, to manage and analyze student performance data. The solution employs 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 is based on a sequence of instructions given to the computer, known as procedures. It follows a top-down approach, where instructions are executed in well-defined sequences.

In this paradigm, the commands are divided into smaller blocks of well-defined code called functions, each created to perform a specific task.

Writing a procedural program involves calling the appropriate function at the correct time within the larger code. These functions typically process data to produce an output. However, unlike object-oriented programming, procedural programming treats data and functions as separate entities.[[1]](#footnote-1)Top of Form

There are a number of features that application written with Procedural Programming have in common such as:

**Pre-Defined Functions:** These functions are identified by name (say, **printf**, **scanf**, **(**in C**)** or **print(), Len(), range() (**in Python**)**, and they each perform a specific task such as taking user input, displaying output on the screen, carrying out mathematical operations.

**Local and Global Variables:** Local variables are declared in the main structure of a function. You will only be able to access the local variable within that function. Global variables on the other hand are declared outside all methods to be accessible from anywhere in the code. Programmers need to be careful they are clear when a variable has been made Local or Global as the difference may drastically change the functionality of an application depending on when/where it is called.

**Programming libraries**: A programming library is a collection of code written previously to utilise whenever a programmer requires it. [[2]](#footnote-2),

**Modularity:** Modular programming is a software design technique that emphasizes separating the functionality of a program into independent, interchangeable modules, such that each contains everything necessary to execute only one aspect of the desired functionality.[[3]](#footnote-3)

**Procedural Programming Paradigm - Python:**

**Below I will look at how I employed some of the features of procedural programming in my python solution when building the GPA Calculator.**

1. **Procedural Execution Flow:** The principal execution flow is organized in the \_\_main\_\_ block at the end of the script. Using the “Top Down” characteristic the program 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.
2. **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.
3. **Data Encapsulation:** Functions are designed to encapsulate specific functionality, promoting code organization and maintainability. For example, 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 that use similar data since they are independent of the specific data used, this enhances the efficiency of the code and highlights why procedural programming is often the best choice for smaller applications.
5. **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.
6. **Separation of Responsibilities:** Each function focuses on a single task, this delivers a separation of responsibilities. 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. **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.

**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 important to remember 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.[[4]](#footnote-4)

**OOP – Python**

**Introduction**

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

Similar to procedural programming, OOP has a number of characteristics that are often found when using that approach such as:

1. **Class:** A user-defined data type with data members and member functions, serving as a blueprint for object creation.
2. **Object:** A basic unit of OOP, representing real-life entities, with identity, state, and behaviour. Objects can interact based on message passing.
3. **Data Abstraction:** Providing essential information about data while hiding implementation details, like a car's mechanism from a driver.
4. **Encapsulation**: Wrapping up data and code within a class to ensure data hiding and access control.
5. **Inheritance:** Capability of a class to derive properties and characteristics from another class, promoting code reuse.
6. **Polymorphism:** The ability of a message to be displayed in multiple forms, allowing different behaviour in different situations.
7. **Dynamic Binding:** Deciding the code to be executed at runtime, enabling flexibility and subtype polymorphism.[[5]](#footnote-5)  
   The code for the example of use of OOP in python implements a GPA (Grade Point Average) calculator using Object-Oriented Programming (OOP) paradigm. Here I will analyse how the OOP paradigm has been applied in this specific solution:
8. **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.
9. **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.
10. **Inheritance**: The code doesn't explicitly use inheritance, but it leverages composition by creating instances of other classes within the **GPACalculator** class.
11. **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.
12. **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.
13. **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 responsibilities 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.

**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 fundamental building blocks of the program. The 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 behaviour (functions) together.

The structs represent entities in the application, 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.

**Overall Summary**  
In conclusion, the multi-paradigm programming report highlights the implementation of different programming paradigms in developing a GPA Calculator using Python and C programming languages. The report explored procedural programming, object-oriented programming (OOP), and a blend of both paradigms in the C programming solution.

The procedural programming approach in Python demonstrated effective code organization, modularity, and parameterization. It allowed for a clear step-by-step execution flow and efficient error handling, making the GPA Calculator solution readable and reusable. Also because of pythons extensive libraries and pre-defined functions it was much easier to apply many of the mathematical functions when creating the procedural method in python.

The OOP approach in Python features the use of classes to encapsulate data and behaviour, promoting abstraction and code reusability. The organization of classes within the GPACalculator class demonstrates the power of OOP principles in promoting maintainability and readability. The division of responsibility allows the programmer to easily debug errors encountered and use those classes (or slightly adjusted versions of those classes to suit the data) in different applications if desired.

The C programming solution primarily followed a procedural paradigm with some elements of OOP using structs for data encapsulation. I found the idea of structs extremely useful in theory, structuring and storing all this data in one object (or objects within those objects), however I will say I found the actual implementation extremely difficult running into multiple issues with memory allocation and formatting of data types. In fact I could never get the standard deviation calculations to come out correctly.

Overall, the report emphasizes the importance of choosing the appropriate programming paradigm based on the specific problem and project requirements. Procedural programming offers a straightforward and efficient approach for certain tasks, while OOP provides a more structured and reusable solution for complex projects. Understanding and applying multiple programming paradigms can lead to versatile and robust code designs, enhancing software development capabilities.

**References**

“What Is Procedural Programming Paradigm?” *Programiz pro Resources*, 19 Aug. 2022, programiz.pro/resources/what-is-procedural-programming/.

Waraich, Kushleen . “Procedural Programming: Everything You Need to Know.” *Www.codingninjas.com*, www.codingninjas.com/studio/library/procedural-programming-everything-you-need-to-know.

Kenneth Leroy Busbee, and Dave Braunschweig. “Modular Programming.” *Rebus.community*, Pressbooks, 2019, press.rebus.community/programmingfundamentals/chapter/modular-programming/.

De Las Heras, Javier . “Object Oriented vs Procedural Programming Paradigms.” *Www.computingatschool.org.uk*, www.computingatschool.org.uk/news-and-blogs/2022/may/object-oriented-vs-procedural-programming-paradigms.

sambhav228. “Introduction of Object Oriented Programming.” *GeeksforGeeks*, 5 Sept. 2020, www.geeksforgeeks.org/introduction-of-object-oriented-programming/.

**Bibliography**

Here are some of the websites and resources used in this project – most of the programming was done following the lecture led instructions from the course work provided.

1. *C structures (structs)*. Available at: https://www.w3schools.com/c/c\_structs.php (Accessed: 10 July 2023).
2. *C tutorial*. Available at: https://www.w3schools.com/c/index.php (Accessed: 10 July 2023).
3. Bhatia, S. (no date) *What is procedural programming? [definition] - key features*, *Hackr.io*. Available at: https://hackr.io/blog/procedural-programming (Accessed: Accessed: 05 July 2023).
4. COMS 1202 (2021) *Building a GPA Calculator in python*, *YouTube*. Available at: https://www.youtube.com/watch?v=ctYoxD28Iwo (Accessed: 12 July 2023).
5. Davidson, J. (2021) *A simple guide to using structures in C*, *MUO*. Available at: https://www.makeuseof.com/c-structures-simple-guide/ (Accessed: 10 July 2023).
6. *Format specifiers in C* (2023) *GeeksforGeeks*. Available at: https://www.geeksforgeeks.org/format-specifiers-in-c/ (Accessed: 17 July 2023).
7. freeCodeCamp.org (2018) *How to understand your program’s memory*, *freeCodeCamp.org*. Available at: https://www.freecodecamp.org/news/understand-your-programs-memory-92431fa8c6b/ (Accessed: 17 July 2023).
8. Meghan van DeventerMeghan van Deventer, *How to read a CSV file in C*, *Stack Overflow*. Available at: https://stackoverflow.com/questions/61078280/how-to-read-a-csv-file-in-c (Accessed: 17 July 2023).
9. *Memory layout of C Programs* (2022) *GeeksforGeeks*. Available at: https://www.geeksforgeeks.org/memory-layout-of-c-program/ (Accessed: 20 July 2023).
10. *Programming paradigms in python* (2020) *GeeksforGeeks*. Available at: https://www.geeksforgeeks.org/programming-paradigms-in-python/ (20 July 2023).
11. *Python classes and objects* (2023) *GeeksforGeeks*. Available at: https://www.geeksforgeeks.org/python-classes-and-objects/ (Accessed: 20 July 2023).
12. Real Python (2023) *Object-oriented programming (OOP) in python 3*, *Real Python*. Available at: https://realpython.com/python3-object-oriented-programming/ (Accessed: 20 July 2023).
13. TechWithTim (2020) *Python object oriented programming (OOP) - for Beginners*, *YouTube*. Available at: https://www.youtube.com/watch?v=JeznW\_7DlB0 (Accessed: 22 July 2023).
14. *Unable to read data from a text file in C*, *Stack Overflow*. Available at: https://stackoverflow.com/questions/71972343/unable-to-read-data-from-a-text-file-in-c?rq=2 (Accessed: 01 August 2023).

1. “What Is Procedural Programming Paradigm?” *Programiz pro Resources*, 19 Aug. 2022, programiz.pro/resources/what-is-procedural-programming/. [↑](#footnote-ref-1)
2. Waraich, Kushleen . “Procedural Programming: Everything You Need to Know.” *Www.codingninjas.com*, www.codingninjas.com/studio/library/procedural-programming-everything-you-need-to-know. [↑](#footnote-ref-2)
3. Kenneth Leroy Busbee, and Dave Braunschweig. “Modular Programming.” *Rebus.community*, Pressbooks, 2019, press.rebus.community/programmingfundamentals/chapter/modular-programming/. [↑](#footnote-ref-3)
4. De Las Heras, Javier . “Object Oriented vs Procedural Programming Paradigms.” *Www.computingatschool.org.uk*, www.computingatschool.org.uk/news-and-blogs/2022/may/object-oriented-vs-procedural-programming-paradigms. [↑](#footnote-ref-4)
5. sambhav228. “Introduction of Object Oriented Programming.” *GeeksforGeeks*, 5 Sept. 2020, www.geeksforgeeks.org/introduction-of-object-oriented-programming/. [↑](#footnote-ref-5)