REPORT ON GPA CALCULATOR PROJECT

MODULE: Multi-Paradigm Programming

AUTHOR: Anja Antolkovic

SOURCE CODE

1. INTRODUCTION

This is the report prepared for the module project Multi-Paradigm Programming at ATU. The project was to create a GPA calculator for students at a fictitious university in procedural and object-oriented programming, with the user experience being identical in both implementations. The report includes a user guide for the final code, an overview of the technical implementations, and a final conclusion. The project description and source code can be found by clicking on the tab in the header.

2. USER GUIDE

1. Clone the repository on your machine and run the procedural programming version gpa_calculator.py and the object-oriented programming version gpa_calculator_oop.py.
2. Press Y to import a file from N and insert data manually (Figure 1). The program accepts only csv files with the following format (Figure 2). The column names except A1 can be arbitrary.

```
Would you like to import a file (Y/N):
```

Figure 1: File import

	A	B	C	D	E	F	G
1	Student Name	Intro to Programming	Databases	Computer Architecture	Ethics in Computer Science	Advanced Programming	Puzzles and Problem Solving
2	James Jameson	80	70	55	12	50	80
3	Peter Parker	50	55	50	50	50	50
4	Mary Janeson	40	40	40	23	40	45
5	Felicia Hardiman	70	67	78	70	56	80
6	Lori Grimes	50	30	35	30	30	30
7	Anthony Stones	90	90	65	78	90	80
8							

Figure 2: Format of the CSV file

3. If N, the program prompts for manual data entry (Figure 3). To exit a program, press 0 key.

```
Would you like to import a file (Y/N): N
To quit or insert data for another student press 0
Enter student name: David
Enter module name: Multi-Paradigm Programming
Enter grade (%): 87
Enter module name: Data Representation
Enter grade (%): 80
Enter module name: 0
To quit or insert data for another student press 0
Enter student name: Maria
Enter module name: Multi-Paradigm Programming
Enter grade (%): 60
Enter module name:
```

Figure 3: Manual data import

4. The final results are in the form of dictionary objects (Figure 4).

```
GPA
{'David': 4.1, 'Maria': 3.6}

Highest scoring module
{'David': 'Data Representation', 'Maria': 'Multi-paradigm programming'}

Lowest scoring module
{'David': 'Multi-Paradigm Programming', 'Maria': 'Multi-paradigm programming'}

Standard deviation
{'David': 1.5, 'Maria': 0.0}

Median
{'David': 88.5, 'Maria': 67.0}

Grade point gap to the next highest GPA
{'David': 0.1, 'Maria': 0.5}

Letter grades
{'David': {'Multi-Paradigm Programming': 'A', 'Data Representation': 'A+'}, 'Maria': {'Multi-paradigm programming': 'B+'}}

End
```

Figure 4 Final results

3. TECHNICAL IMPLEMENTATION

3.1. Procedural Python

The create_student() function calls the program. If the user chooses file import, the data is read into Pandas and eventually converted into a dictionary object. If the user chooses manual import, the function uses a while loop to repeatedly prompt the user to enter the required information. At each iteration, the data is appended to a list object and finally converted into a dictionary object, i.e. the dictionary object contains the data used to calculate the required indicators.

Moreover, each task corresponding to each required indicator is divided into individual functions (application of the single responsibility principle). The function names are self-explanatory, e.g. calculate_gpa() (applying the practice of self-documentation). Each of these functions is passed a dictionary object containing student data. The functions return a dictionary object with the results, which is printed on the terminal.

3.2. Object-oriented Python

Most of the program is divided into 3 classes: Grade, Student_app and GPA_calculator. The program is intended as an application for students, used to calculate indicators for students. The Grade class processes grades in percentage points and converts them into letter grades and GPA grades. The Student_app class creates a student object by taking as arguments the name of the student, the name of the module, and the percentage grades in the form of key value arguments. It provides various methods for calculating different indicators. The GPA_calculator class inherits properties from the Student_app class and provides a method for calculating GPA and GPA ranking (example of using inheritance). As with the procedural programming version of the script, the principle of self-documentation and single responsibility is used by giving the class methods self-explanatory names and breaking the class methods down to a single task corresponding to a required indicator. The create_student() function calls the program. When the user selects file import, the data is read into pandas and eventually converted into a dictionary object. A for loop is used to iterate over a dictionary and instantiate GPA_calculator objects (example of using polymorphism). The result is the list of GPA_calculator objects that are subsequently called by the main function. If the user chooses manual

import, the function uses a while loop to repeatedly prompt the user to enter the required information. At each iteration, the data is appended to a list object and eventually passed as an argument to a GPA_calculator class, creating a list of GPA_calculator objects that are subsequently called by the main function.

4. CONCLUSION

The main principle in the procedural program is the use of procedural calls that are executed in a linear fashion. The main object type that contains the data is the dictionary.

On the other hand, the main principle in an object-oriented program is the use of class objects. The main object type that contains the data is the class object. The use of class objects does not require linear execution, for example, we do not need to call the calculate_gpa() method separately to learn how far each student is from the next highest grade point average, although this information is needed. This is because both methods are defined in the same class, since an object 'knows everything about itself'.

Therefore, the object-oriented approach offers more versatility and, although more difficult to understand at first, is the better programming paradigm for this project.

5. REFERENCES

1. College materials (slides and video recordings)
2. Nantais, C. L., & Downey, A. B. (2013). How to think like a computer scientist. In *CreateSpace Independent Publishing Platform eBooks*. http://faculty.salisbury.edu/~dxdefino/COSC%20117/JavaTutorial.pdf
3. *Computing at School (CAS) home page*. (n.d.). https://www.computingatschool.org.uk/
4. *Isaac Computer Science*. (n.d.). Isaac Computer Science. https://isaaccomputerscience.org/concepts/prog_pas_paradigm?examBoard=all&st%20age=all
5. Tech With Tim. (2020, March 29). *Python Object Oriented Programming (OOP) - For Beginners* [Video]. YouTube. https://www.youtube.com/watch?v=JeznW_7DlB0