Table of Contents

Programming Paradigms: Procedural & Object-Oriented

Introducing the programming paradigms:

Procedural Programming

Gabbrielli & Martini (2010, pg.68) refer to a procedure as nothing more than a name associated with a set of commands. Following from this line of thought a procedure can be regarded as a series of computational steps to be conducted in order to allow one to achieve their intended goal or provide assistance towards this achievement.

Within programming languages procedures are synonymous with the following terms i.e., functions, routines & subroutines. Although within some languages, the term "function", is reserved for subprograms that return a value, while subprograms which interact with the caller via parameters only or the non-local environment are often regarded as procedures (Gabbrielli & Martini, 2019, pg. 167). A program that has not organised its commands into a series of functions would be regarded as aligning itself to an imperative programming paradigm.

Object-Oriented Programming

Romano (2018, pg.185) introduces Object-oriented programming (OOP) as a programming paradigm based on the concept of "*objects*", which are data structures that contain data, in the form of attributes, and code in the form of functions known as methods. As previously expressed, the procedural programming paradigm refers to stored procedures as functions, while OOP refers to procedures as methods i.e., instructions that influence the behaviour or form of specific data. An object is suggested to be an autonomous actor responding to messages, rather than a place to store particular named routines in a procedural program (Lee, 2019, pg. 7).

Objects and classes are of primary concern within OOP. Classes are used to create objects and are often regarded as blueprints for creating an object. A class can define specific sets of characteristics that are shared by all objects of a class (Phillips, 2015); objects are regarded as instances of the classes from which they were created. Objects are models of real-world *things* that have the capacity to do certain things and have certain things done to them (Phillips, 2015). Following from this when objects are initially created by a class, they automatically inherit the class attributes and methods Romano (2018, pg.187). The paradigm of OOP strives to manage

the complexity found within real-world problems through its efforts of abstracting out knowledge and encapsulating it within objects (Lee, 2019). From within a class method, we can refer to an instance by means of a special argument called *self*. This argument is always the first attribute of an instance method. The *self*-argument to a method is simply a reference to the object that the method is being invoked on (Phillips, 2015).

Multi Paradigm Programming Project: GPA Calculator

As requested within the project brief a GPA calculator was to be developed for a fictional university. The specifications addressed within the brief were to be implemented within the programming languages of Python and C. Alongside this the project brief requested that the GPA calculator be developed using a procedural and object-oriented approach.

Project Files:

Procedural paradigm

- GPA_Python_procedural.py
- GPA_C_procedural.c

Object-Oriented paradigm

- GPA_Python_OOP.py

Solutions achieved using a procedural and object-oriented approach

Within both python implementations of the project, the csv file data was read through the usage of a *Pandas dataframe*. In contrast to the version of the project that was written in C, this version utilised the strong tokenizer function (strtok) in order to break a string into a series of tokens using a delimiter.

Python procedural code

```
# Read in the values from the CSV i.e., student records.
# Store this information as a pandas dataframe
df = pd.read_csv("CTASample.csv")
```

```
    results = []
    for m in (moduleNames):
        results.append(df[m][i])
```

Whenever I wished to access a student's result for a particular module, the code above was implemented. This code was called several times throughout the program itself. A similar approach occurred within the C procedural version.

C procedural code

```
{
    // Reading CSV File
    FILE *file = fopen("CTASample.csv", "r");
    char line[256];

    int j = 0;
    while (fgets(line, sizeof(line), file)){
        j++;
        if(j == i)
        {
            // Breaking string into a series of tokens using a delimiter.
            char *name = strtok(line, ",");
            char *IntrotoProgramming = strtok(NULL, ",");
            char *Databases = strtok(NULL, ",");
            char *ComputerArchitecture = strtok(NULL, ",");
            char *EthicsinComputerScience = strtok(NULL, ",");
            char *AdvancedProgramming = strtok(NULL, ",");
            char *PuzzlesandProblemSolving = strtok(NULL, ",");
```

In contrast to the procedural approaches where a block of code was continuously implemented, the OOP approach, only stated such a block of code once i.e., the reading of students results. The reading of students results took place as a **method** i.e., *load_records* within the **class** of *AcademicRecords.*

Python OOP code

```
# Read in values values from a CSV file
# Store this information as a pandas dataframe
def load_records(self):
    df = pd.read_csv("CTASample.csv")
    module_names = list(df.keys()[1:])
    # Store the information within the dataframe as a dictionary
    for student in df.values:
        results = {key: value for (key, value) in zip(module_names,
        list(student[1:]))}
        self.student_registry.register(student[0], results)
```

Within the procedural version, global variables were established, that could be accessed by any functions. The OOP version had similar variables although they were located within the class of *GradeStatistics.*

Python procedural code

```
# Global Variables
grade = ["A+", "A", "A-", "B+", "B", "B-", "C+", "C", "C-", "D+", "D",
"D-", "F"]
gpaVal = [4.2, 4.0, 3.8, 3.2, 3.0, 2.8, 2.2, 2.0, 1.8, 1.2, 1.0, 0.8, 0.
0]
moduleNames = ["Intro to Programming", "Databases", "Computer
Architecture", "Ethics in Computer Science",  "Advanced Programming",
"Puzzles and Problem Solving"]
```

C procedural code

```
// Global Variables:
char grade[][5] = {"A+", "A", "A-", "B+", "B", "B-", "C+", "C", "C-", "D
+", "D", "D-", "F"};
float gpaVal[] = {4.2, 4.0, 3.8, 3.2, 3.0, 2.8, 2.2, 2.0, 1.8, 1.2, 1.0,
0.8, 0.0};
char moduleNames[][50] = {"Intro to Programming", "Databases", "Computer
Architecture", "Ethics in Computer Science",  "Advanced Programming",
"Puzzles and Problem Solving"};
```

Python OOP code

```
# Creation of a new class: Grade Statistics
class GradeStatistics(object):
    # Global Variables
    GRADE = ["A+", "A", "A-", "B+", "B", "B-", "C+", "C", "C-", "D+",
    "D", "D-", "F"]
    GPA_VAL = [4.2, 4.0, 3.8, 3.2, 3.0, 2.8, 2.2, 2.0, 1.8, 1.2, 1.0, 0.
    8, 0.0]
```

References

Fabrizio Romano (2018) Learn Python Programming: The No-nonsense, Beginner's Guide to Programming, Data Science, and Web Development with Python 3.7, 2nd Edition. Birmingham: Packt Publishing. Available at: https://search.ebscohost.com/login.aspx?direct=true&AuthType=ip,sso&db=e000xww&AN=1841873&site=ehost-live&scope=site (Accessed: 19 July 2022).

Gabbrielli, M. & Martini, S., 2010. Programming Languages: Principles and Paradigms 1st ed. 2010., London: Springer London: Imprint: Springer.

Graham Lee (2019), Modern Programming: Object Oriented Programming and Best Practices: Deconstruct Object-oriented Programming and Use It with Other Programming Paradigms to Build Applications, Packt Publishing, Birmingham, viewed 19 July 2022, <https://search.ebscohost.com/login.aspx?direct=true&AuthType=ip,sso&db=e000xww&AN=2181411&site=ehost-live&scope=site>.

Phillips, D. (2015) Python 3 Object-oriented Programming - Second Edition. Birmingham: Packt Publishing (Community Experience Distilled). Available at: https://search.ebscohost.com/login.aspx?direct=true&AuthType=ip,sso&db=e000xww&AN=1055466&site=ehost-live&scope=site (Accessed: 19 July 2022).