BIT2008A/ITEC2000A: Multimedia Data Mangement
Fall 2022

Assignment 3

Python and Django

Question 1. In software engineering, a software design pattern is a general, reusable solution to a commonly occurring problem within a given context in software design. It is not a finished design that can be transformed directly into source or machine code. Rather, it is a description or template for how to solve a problem that can be used in many different situations. Design patterns are formalized best practices that the programmer can use to solve common problems when designing an application or system.[2]

The MVT (Model View Template) is a software design pattern. It is a collection of three important components Model View and Template. The Model helps to handle database. It is a data access layer which handles the data. The Template is a presentation layer which handles User Interface part completely. The View is used to execute the business logic and interact with a model to carry data and renders a template. [1]

Question 2. The difference between an interpreted and a compiled language lies in the result of the process of interpreting or compiling. An interpreter produces a result from a program, while a compiler produces a program written in assembly language. In a compiled language, the compilation process itself can guarantee that our code does not have certain errors. But we cannot be so sure about it when it comes to an interpreted language where the code is executed line by line and we can even see some results before we face an error.

Question 3.

- **List**: It is defined by using square brackets, and can contain a set of elements with any type of value.
- **Tuple**: It is defined by using parentheses. Tuples cannot be changed after initialization meaning that we cannot add or remove elements.
- **Set**: It is defined by using curly brackets, and only its unique elements will be saved after initialization.
- **Dictionary**: It is similar to a set but, it takes key-value pairs separated by colons.

Question 4.

- **Abstraction**: The process by which data and functions are defined in such a way that only essential details can be seen and unnecessary implementations are **hidden** is called Data Abstraction. For example, when we are provided with a class to handle *Person* entity, and we want to develop a class based on it to handle *Student* entity, we do not care about every detail of the implementation of *Person*. Instead, we focus on what is needed to be changed particularly in that class, and we expect the functionalities in common to work just fine.
- **Encapsulation**: Python allows the developers and users to think of a class as a capsule. It means that sometimes, we have to interact with the class to retrieve or manipulate its data instead of doing that directly. For example, if we want to interact with the *Student* class to change the ID, we had better use a *set_id* method (if implemented) as a black box, rather than changing the attribute manually. The attributes also have access modifiers such as **public**, **private**, and **protected**.
- **Inheritance**: Inheritance allows us to define a class that inherits the methods and properties from another class. Parent class is the class being inherited from, also called base class. Child class is the class that inherits from another class, also called derived class.
- **Polymorphism**: In Python, Polymorphism lets us define methods in the child class that have the same name as the methods in the parent class. In inheritance, the child class inherits the methods from the parent class. However, it is possible to modify a method in a child class that it has inherited from the parent class. This is particularly useful in cases where the method inherited from the parent

class does not quite fit the child class. In such cases, we re-implement the method in the child class. This process of re-implementing a method in the child class is known as **Overriding**. For instance, if you override a method *foo* for the *Student* class, your *Student* instances can call *foo* method from both classes making the instance polymorphic.

References

[1] javatpoint. Django MVT - javatpoint.
[2] wikipedia. Design Pattern - Wikipedia.