Design Patterns

What are Design Patterns?
Computer scientists got together and discussed the common types of problems they are usually asked to solve, and realised that a lot ofcreated a set of generic solutions to those problem. These are not full programs, and sometimes not even pseudocode, but represent good ideas or good approaches to solving common problems. Types of Design Patterns include: **Algorithm strategy patterns**, **Computational design patterns**, **Execution patterns**, **Implementation strategy patterns**, and **Structural design patterns**

Some Common Design Patterns

The Iterator Pattern: A design pattern in which an iterator is used to traverse a container and access the container's elements.

The Decorator Pattern: A design pattern which wraps an existing class and can alter the functionality of the methods.

The Observer Pattern: A design pattern which monitors a core class and different observers react different to changes in the core class.

The Strategy Pattern: A design pattern which presents different potential solutions to the same problem, and allows the program to choose the most suitable one.

The State Pattern: A design pattern which represents a system that goes through different states, and records the current state and the transitions between states.

The Singleton Pattern: A design pattern which allows only one object based on a certain class to exist.

The Template Pattern: A design pattern which creates a common base class the can be inherited by multiple class that share common states, these can override the base class methods.

The Adapter Pattern: A design pattern which allows two pre-existing objects to interact with each other, even if their interfaces are not compatible.

The Façade Pattern: A design pattern which presents a simple interface to complex system but encapsulating typical usage into a new object.

The Flyweight Pattern: A design pattern which helps objects that share the same state to use the same memory location.

The Command Pattern: A design pattern which creates an object (an execute object) that can execute another object at a later time.

The Abstract Factory Pattern: A design pattern which returns a different class (or implementation) of the same system, depending on the platform, local settings, or current locale.

The Composite Pattern: A design pattern which allows complex tree-like structures to be built easily from simple components.

These are some of the 23 classic design patterns.

Design Patterns

The Singleton Pattern

The Singleton Pattern is a design pattern which allows only one object based on a certain class to exist. The general Singleton design pattern is as follows:

```
class Singleton:
  IsSingleton = None

  Method NewObject:
    If IsSingleton == None
      Then IsSingleton = super(Singleton).new_object
    EndIf;
    return IsSingleton
  # END NewObject

# END Singleton.
```

In Python the Singleton design pattern can be implented as follows:

```
class OneOnly:
    _singleton = None

    def __new__(cls, *args, **kwargs):
        if not cls._singleton:
        # THEN
            cls._singleton = super(OneOnly, cls
                ).__new__(cls, *args, **kwargs)
        # ENDIF;
        return cls._singleton
    # END __new__

# END OneOnly.
```

In Python we use the _ _new_ _ class to help ensure there's only one instance of a class. When the _ _new_ _ class is called, it typically creates a new instance of that class. When we override it, we first check if our singleton instance has been created; if not, we create it using a super() call.

A sample output would be as follows:

```
>>> o1 = OneOnly()
>>> o2 = OneOnly()
>>> o1 == o2
True
>>> o1
<__main__.OneOnly object at 0xb71c008c>
>>> o2
<__main__.OneOnly object at 0xb71c008c>
```

So even though it looks like two objects are created, in reality they are both at the same address in memory.