Building More Python Design Patterns

INTRODUCTION

Gerald Britton
IT SOLUTIONS DESIGNER

@GeraldBritton www.linkedin.com/in/geraldbritton

Overview

Design patterns to be covered

SOLID principles of object-oriented design

Tools you will need

Meta class programming in Python

Other Pluralsight courses on OOP

“Design Patterns: Elements of Reusable Object-Oriented Software”

Gamma, Helm, Johnson and Vlissides

Design Patterns Covered

- Façade Pattern
- Adapter Pattern
- Decorator Pattern
- Template Pattern
- Iterator Pattern
- Composite Pattern
- State Pattern
- Proxy Pattern

SOLID Principles of Object Oriented Design

Single responsibility

Open-closed

Liskov substitution

Interface segregation

Dependency inversion

Tools You Will Need

Python language, either 2.x or 3.x

- https://www.python.org/downloads/

A Development environment

- IDLE (included in Python download)
- PyCharm
- Wing IDE
- PyDev for Eclipse
- Visual Studio
- Many others
- https://wiki.python.org/moin/PythonEditors

Visual Studio Code

- https://code.visualstudio.com/Download

Abstract Base Classes

```
import abc


class MyAbsClass(metaclass=abc.ABCMeta):
    @abc.abstractproperty
    def myproperty(self):
        pass
    @abc.abstractmethod
    def mymethod(self, value):
        pass
class MyAbsClass2_x(object):
    __metaclass__ = abc.ABCMeta
```

- Import the abstract base class module
- Define an abstract class (3.x)
- Example of an abstract property
- Example of an abstract method
- Define an abstract class (2.x)

```
class MyConcreteClass(MyAbsClass):
    @property
    def myproperty(self):
        return 42

    def mymethod(self, value):
        assert 42 == 42

c = MyConcreteClass()
print(c.myproperty)
```

- Implement the abstract class
- Implement the property
- Implement the method
- Instantiate the class
- Print the property

Summary

Design patterns to be covered

Object oriented design principles (SOLID)

Tools you will need

Interfaces in Python

- Abstract Base Classes