Design Patterns with Python

INTRODUCTION TO DESIGN PATTERNS

Gerald Britton
IT SPECIALIST

@GeraldBritton www.linkedin.com/in/geraldbritton

Overview

What are design patterns?

Why do we need them?

Classification of design patterns

Principles of object-oriented design

SOLID

Tools you will need

Defining interfaces in Python

What Are Design Patterns?

A design pattern is a model solution to a common design problem. It describes the problem and a general approach to solving it.

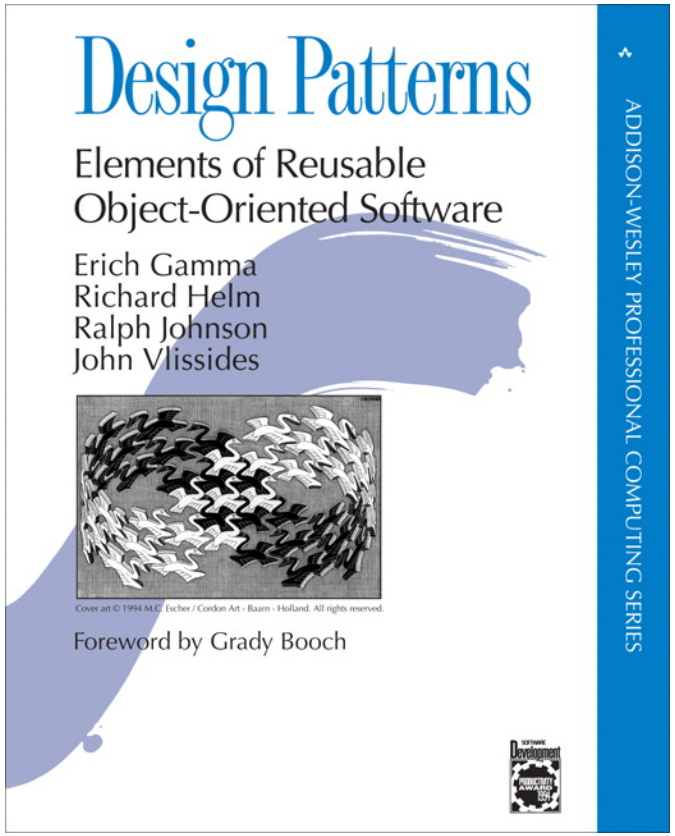
“Each pattern describes a problem which occurs over and over again in our environment and then describes the core of the solution to the problem”

Christopher Alexander, *A Pattern Language,* Oxford University Press, New York, 1977

Examples of Design Patterns

- Building architecture
- Electrical and plumbing codes
- Automobile design
- Mobile phone interfaces

We need design patterns to ensure that our work is consistent, reliable and understandable



First published in 1995

“Gang of Four”

- Gamma, Helm, Johnson and Vlissides

First comprehensive work on the topic

Remains the authoritative reference

This course would look very different without this book.

Classification

Creational

Object creation

Structural

Object composition

Behavioral

Object interaction and responsibility

SOLID Principles of Object Oriented Design

Single responsibility

Open-closed

Liskov substitution

Interface segregation

Dependency inversion

Tools You Will Need

Python language, either 2.7.x or 3.5.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

Interfaces in Python

The "I" in SOLID

Supported in Java, C#, Visual Basic with Interface definitions

Supported in C++ with Abstract Classes

Previously no provision in Python

Introduced by PEP 3119

First appeared in Python versions 2.6 and 3.0

Abstract Base Class Definition

abc module

Make class abstract

Abstract method

Abstract property

```
import abc

class MyABC(object):
"""Abstract Base Class Definition"""
__metaclass__ = abc.ABCMeta

@abc.abstractmethod
def do_something(self, value):
    """Required method"""

@abc.abstractproperty
def some_property(self):
    """Required property"""
```

Concrete Class Implementation

Inherit from ABC

Standard constructor

Implement abstract method

Implement abstract property

```
class MyClass(MyABC):
"""Implementation of MyABC"""

def __init__(self, value=None):
    self._myprop = value

def do_something(self, value):
    """Implementation of abstract method"""
    self._myprop *= 2 + value

@property
def some_property(self):
    """Implementation of abstract property"""
    return self._myprop
```

Python Catches Missing Implementations

```
>>> class BadClass(MyABC):
        pass

>>> bad = BadClass()

Traceback (most recent call last):
  File "<pyshell#37>", line 1, in <module>
    bad = BadClass()
TypeError: Can't instantiate abstract class BadClass with
abstract methods do_something, some_property
```

Summary

- What design patterns are
- Why we need them
- Object oriented design principles (SOLID)
- Tools you will need
- Interfaces in Python
- “Gentlemen’s agreement”