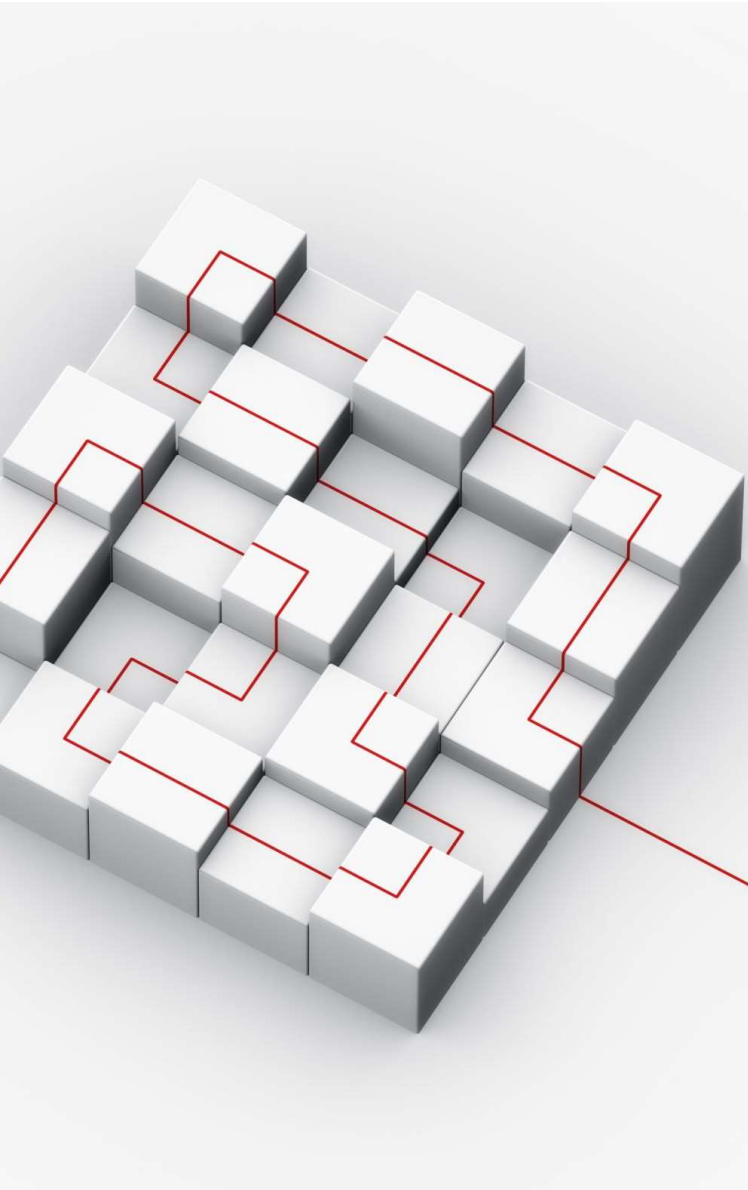


Object-Oriented Programming

# Designing with Classes



# This chapter covers

- Python and OOP
- Inheritance
- Composition
- Delegation
- Methods Are Objects
- Classes Are Objects
- Multiple Inheritance

# Python and OOP

- Inheritance
  - Inheritance is based on attribute lookup in Python (in `X.name` expressions).
- Polymorphism
  - In `X.method`, the meaning of method depends on the type (class) of subject object `X`.
- Encapsulation
  - Methods and operators implement behavior, though data hiding is a convention by default.

# OOP and Inheritance: “Is-a” Relationships

- A class defines a set of properties that may be inherited and customized by more specific sets.
- From a programmer’s point of view, inheritance is kicked off by attribute qualifications, which trigger searches for names in instances, their classes, and then any superclasses.
- From a designer’s point of view, inheritance is a way to specify set membership.

employees.py

# Example

# OOP and Composition: “Has-a” Relationships

- Composition has to do with components—parts of a whole.
- From a programmer’s point of view, composition involves embedding other objects in a container object, and activating them to implement container methods.
- To a designer, composition is another way to represent relationships in a problem domain.

pizzashop.py

# Example

# OOP and Delegation: “Wrapper” Proxy Objects

- Beside inheritance and composition, object-oriented programmers often speak of delegation, which usually implies controller objects that embed other objects to which they pass off operation requests.
- In a sense, delegation is a special form of composition, with a single embedded object managed by a wrapper (sometimes called a proxy) class that retains most or all the embedded object's interface.



trace.py

# Example

# Methods Are Objects: Bound or Unbound

- Bound (instance) method objects: self + function pairs
  - If a function is an attribute of class and it is accessed via the instances, they are called bound methods.
  - A bound method is one that has 'self' as its first argument.
  - Since these are dependent on the instance of classes, these are also known as instance methods.

# Methods Are Objects: Bound or Unbound

- Unbound (class) method objects: no self
  - Methods that do not have an instance of the class as the first argument are known as unbound methods.
  - As of Python 3.0, the unbound methods have been removed from the language.
  - They are not bounded with any specific object of the class.
  - To make the method `myFunc2()` work in the above class it should be made into a static method.
  - Static methods are similar to class methods but are bound completely to class instead of particular objects. They are accessed using class names.

sample.py

# Example

# Classes Are Objects: Generic Object Factories

- Because classes are also “first class” objects, it’s easy to pass them around a program, store them in data structures, and so on.
- You can also pass classes to functions that generate arbitrary kinds of objects; such functions are sometimes called factories in OOP design circles.

factory.py

# Example

# Multiple Inheritance: “Mix-in” Classes

- As we’ve seen, in a class statement, more than one superclass can be listed in parentheses in the header line.
- When you do this, you leverage multiple inheritance—the class and its instances inherit names from all the listed superclasses.

multiple.py

# Example



**The End**