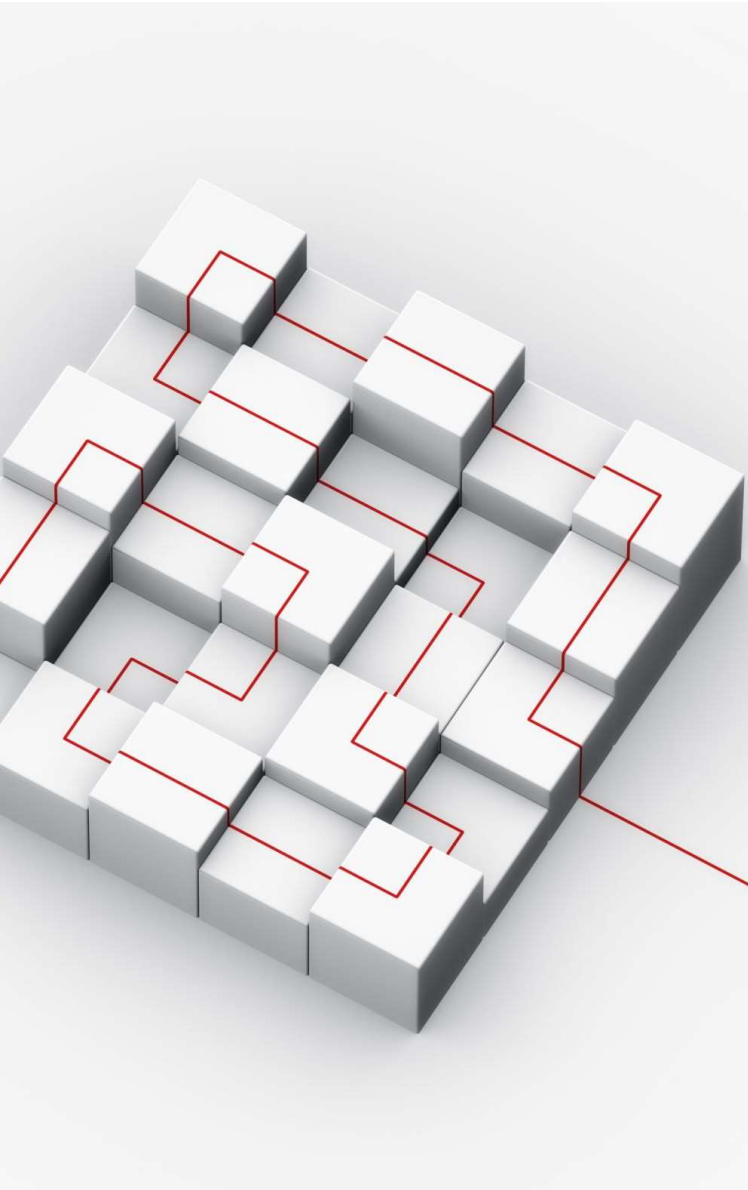


Object-Oriented Programming

The Big Picture



This chapter covers

- Why Use Classes?
- Attribute Inheritance Search
- Classes and Instance
- Method Calls
- Coding Class Trees
- Operator Overloading
- OOP is About Code Reuse

Why Use Classes?

- Inheritance
 - Pizza-making robots are kinds of robots, so they possess the usual robot-y properties.
 - In OOP terms, we say they “inherit” properties from the general category of all robots.
 - These common properties need to be implemented only once for the general case and can be reused in part or in full by all types of robots we may build in the future.

Why Use Classes?

- Composition
 - Pizza-making robots are really collections of components that work together as a team.
 - For instance, for our robot to be successful, it might need arms to roll dough, motors to maneuver to the oven, and so on.
 - In OOP parlance, our robot is an example of composition; it contains other objects that it activates to do its bidding.
 - Each component might be coded as a class, which defines its own behavior and relationships.

Why Use Classes?

- Multiple instances
 - Classes are essentially factories for generating one or more objects.
 - Every time we call a class, we generate a new object with a distinct namespace.
 - Each object generated from a class has access to the class's attributes and gets a namespace of its own for data that varies per object.

Why Use Classes?

- Customization via inheritance
 - Classes also support the OOP notion of inheritance; we can extend a class by redefining its attributes outside the class itself in new software components coded as subclasses.
 - More generally, classes can build up namespace hierarchies, which define names to be used by objects created from classes in the hierarchy.
 - This supports multiple customizable behaviors more directly than other tools.

Why Use Classes?

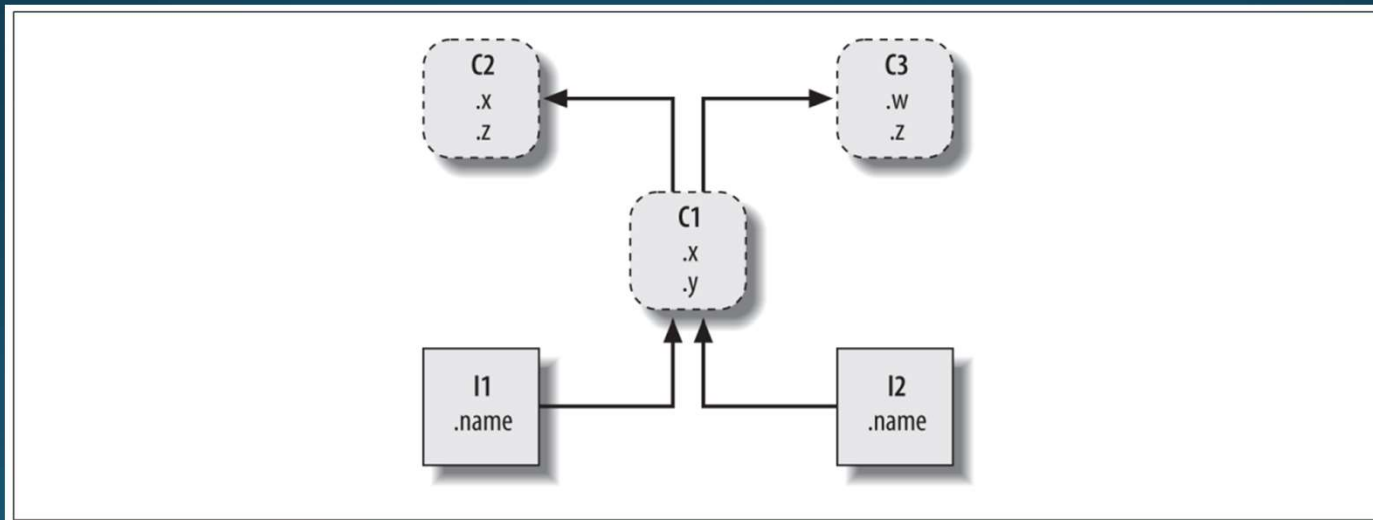
- Operator overloading
 - By providing special protocol methods, classes can define objects that respond to the sorts of operations we saw at work on built-in types.
 - For instance, objects made with classes can be sliced, concatenated, indexed, and so on.
 - Python provides hooks that classes can use to intercept and implement any built-in type operation.

Attribute Inheritance Search

- `object.attribute`
 - Find the first occurrence of attribute by looking in object, then in all classes above it, from bottom to top and left to right.
- Attribute fetches are simply tree searches.
- The term inheritance is applied because objects lower in a tree inherit attributes attached to objects higher in that tree.
- As the search proceeds from the bottom up, in a sense, the objects linked into a tree are the union of all the attributes defined in all their tree parents, all the way up the tree.

Attribute Inheritance Search

- In Python, this is all very literal: we really do build up trees of linked objects with code, and Python really does climb this tree at runtime searching for attributes every time we use the `object.attribute` expression.



Explanations

- In previous figure, there is a tree of five objects labeled with variables, all of which have attached attributes, ready to be searched.
- More specifically, this tree links together three class objects (the ovals C1, C2, and C3) and two instance objects (the rectangles I1 and I2) into an inheritance search tree.
- In terms of search trees, an instance inherits attributes from its class, and a class inherits attributes from all classes above it in the tree.

Classes and Instances

- Classes
 - Serve as instance factories. Their attributes provide behavior - data and functions - that is inherited by all the instances generated from them (e.g., a function to compute an employee's salary from pay and hours).
- Instances
 - Represent the concrete items in a program's domain. Their attributes record data that varies per specific object (e.g., an employee's Social Security number).

Method Calls

- In the prior section, we saw how the attribute reference `I2.w` in our example class tree was translated to `C3.w` by the inheritance search procedure in Python.
- Perhaps just as important to understand as the inheritance of attributes, though, is what happens when we try to call methods - functions attached to classes as attributes.
- If this `I2.w` reference is a function call, what it really means is "call the `C3.w` function to process `I2`."

Coding Class Trees

- Each class statement generates a new class object.
- Each time a class is called, it generates a new instance object.
- Instances are automatically linked to the classes from which they are created.
- Classes are automatically linked to their superclasses according to the way we list them in parentheses in a class header line; the left-to-right order there gives the order in the tree.

Operator Overloading

- If it's coded or inherited, Python automatically calls a method named `__init__` each time an instance is generated from a class.
- The new instance is passed into the self argument of `__init__` as usual, and any values listed in parentheses in the class call go to arguments two and beyond.
- The effect here is to initialize instances when they are made, without requiring extra method calls.
- The `__init__` method is known as the constructor.

Operator Overloading

- It's the most used representative of a larger class of methods called operator overloading methods.
- Such methods are inherited in class trees as usual and have double underscores at the start and end of their names to make them distinct.
- Python runs them automatically when instances that support them appear in the corresponding operations, and they are mostly an alternative to using simple method calls.
- They're also optional: if omitted, the operations are not supported. If no `__init__` is present, class calls return an empty instance, without initializing it.

OOP is About Code Reuse

- Classes support code reuse in ways that other Python program components cannot.
- With classes, we code by customizing existing software, instead of either changing existing code in place or starting from scratch for each new project.
- At a fundamental level, classes are just packages of functions and other names, much like modules.

The End