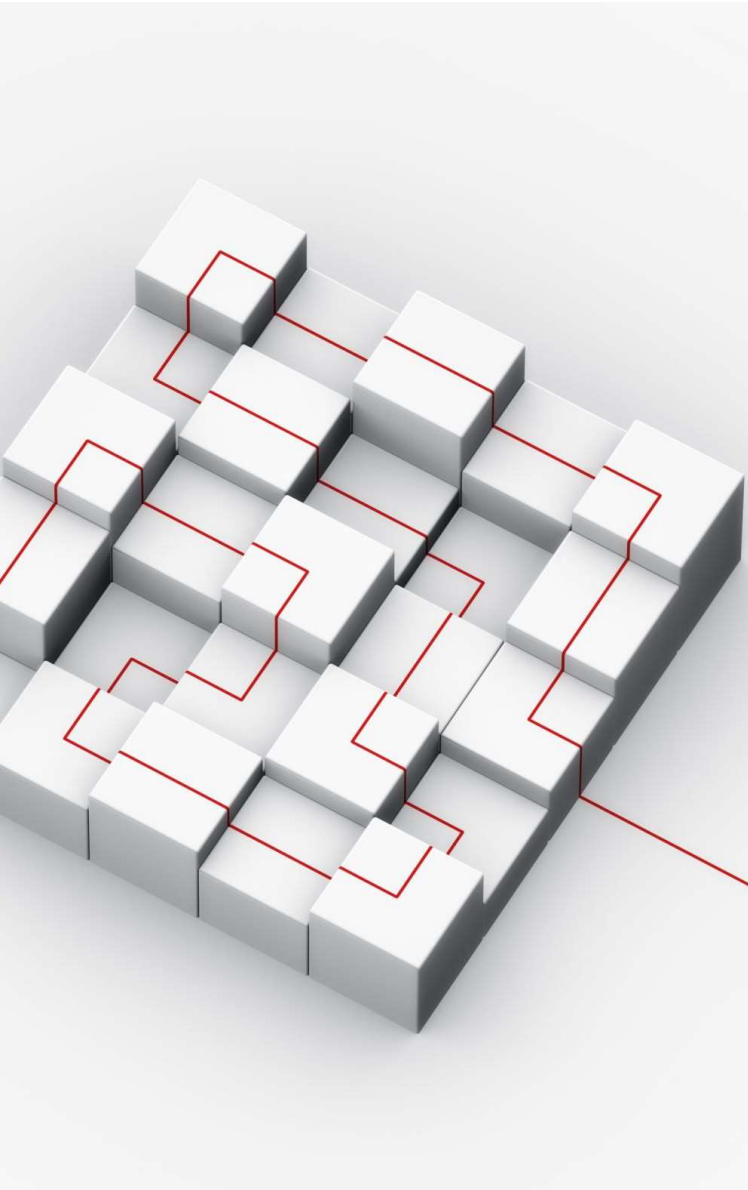


Object-Oriented Programming

Advanced Class Topics



This chapter covers

- Extending Built-In Types
- New-Style Class Changes
- New-Style Class Extensions
- Static and Class Methods
- Decorators and Metaclasses

Extending Built-in Types

- Besides implementing new kinds of objects, classes are sometimes used to extend the functionality of Python's built-in types to support more exotic data structures.
- For instance, to add queue insert and delete methods to lists, you can code classes that wrap (embed) a list object and export insert and delete methods that process the list.

setwrapper.py and typesubclass.py

Example

The “New Style” Class Model

- Besides implementing new kinds of objects, classes are sometimes used to extend the functionality of Python’s built-in types to support more exotic data structures.
- For instance, to add queue insert and delete methods to lists, you can code classes that wrap (embed) a list object and export insert and delete methods that process the list.

setwrapper.py and typesubclass.py

Example

New-Style Class Changes

- Attribute fetch for built-ins: instance skipped.
- Classes and types merged: type testing.
- Automatic object root class: defaults
- Inheritance search order: MRO and diamonds.
- Inheritance algorithm.
- New advanced tools: code impacts.

MRO: Method Resolution Order

- To trace how new-style inheritance works by default, we can also use the new class.__mro__ attribute mentioned in the preceding chapter's class lister examples - technically a new-style extension, but useful here to explore a change.
- This attribute returns a class's MRO - the order in which inheritance searches classes in a new-style class tree.
- This MRO is based on the C3 superclass linearization algorithm initially developed in the Dylan programming language, but later adopted by other languages including Python 2.3 and Perl 6.

tracemro.py

Example

New-Style Class Extensions

- Slots: Attribute Declarations
 - By assigning a sequence of string attribute names to a special `__slots__` class attribute, we can enable a new-style class to both limit the set of legal attributes that instances of the class will have and optimize memory usage and possibly program speed.
 - As we'll find, though, slots should be used only in applications that clearly warrant the added complexity.
 - They will complicate your code, may complicate or break code you may use, and require universal deployment to be effective.

slotbasics.py

Example

" Slots best reserved for rare cases where there are large numbers of instances in a memory-critical application. "

Per Python's own manual, they should not be used except in clearly warranted cases – they are difficult to use correctly.

Static and Class Methods

- static methods work roughly like simple instance-less functions inside a class.
- class methods are passed a class instead of an instance.
- To enable these method modes, you must call special built-in functions named `staticmethod` and `classmethod` within the class or invoke them with the special `@name` decoration syntax.

Static Method Alternatives

- If you just want to call functions that access class members without an instance, perhaps the simplest idea is to use normal functions outside the class, not class methods.
- This way, an instance isn't expected in the call.

bothmethods.py

Example

Decorators and Metaclasses

- Python decorators - like the notion and syntax of annotations in Java - both addressed this specific need and provided a general tool for adding logic that manages both functions and classes, or later calls to them.
 - Function decorators
 - They are useful for adding many types of logic to functions besides the static and class method use cases.
 - Class decorator
 - Like function decorators, but for class. Class decorators may manage classes themselves or later instance creation calls

The End