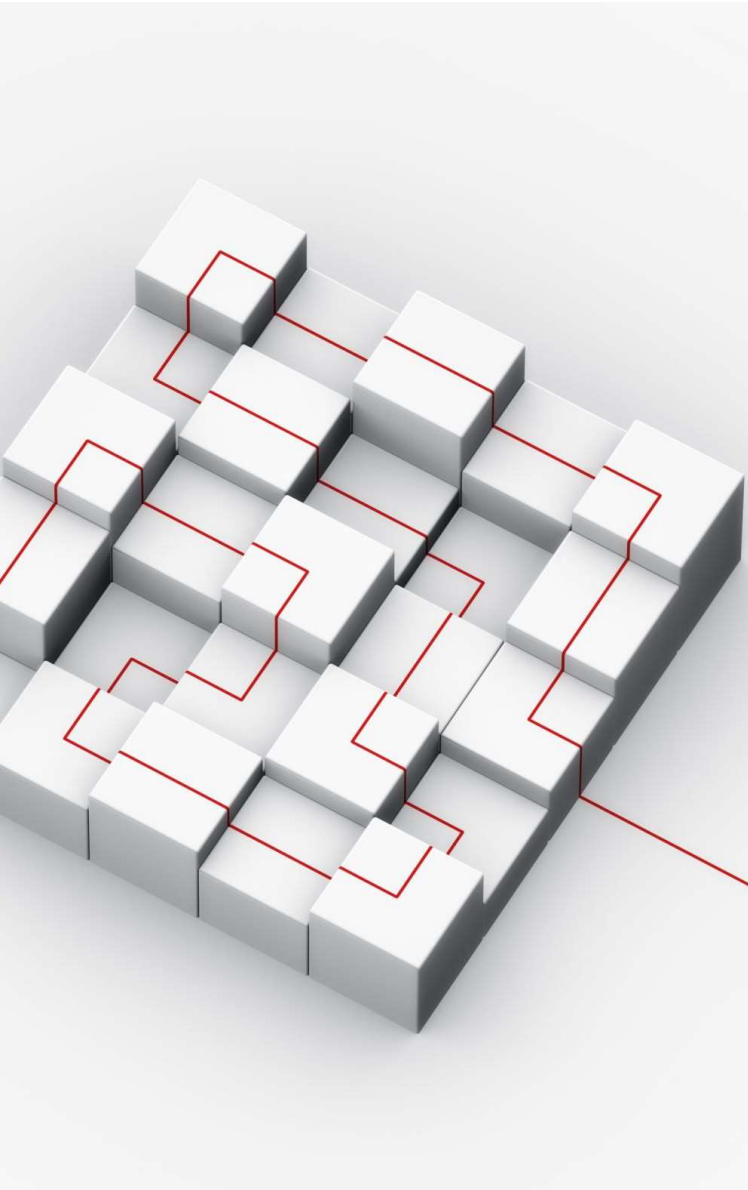


Object-Oriented Programming

Class Coding Basics



This chapter covers

- Classes Generate Multiple Instance Objects
- Classes Are Customized by Inheritance
- Classes and Intercept Python Operators

Classes Generate Multiple Instance Objects

- There are two kinds of objects in Python's OOP model: class objects and instance objects.
- Class objects provide default behavior and serve as factories for instance objects.
- Instance objects are the real objects your programs process - each is a namespace but inherits names in the class from which it was created.
- Class objects come from statements, and instances come from calls; each time you call a class, you get a new instance of that class.

Class Objects Provide Default Behavior

- The class statement creates a class object and assigns it a name.
 - Just like the function def statement, the Python class statement is an executable statement.
 - When reached and run, it generates a new class object and assigns it to the name in the class header.
 - Also, like defs, class statements typically run when the files they are coded in are first imported.

Class Objects Provide Default Behavior

- Assignments inside class statements make class attributes.
 - Just like in module files, top-level assignments within a class statement (not nested in a def) generate attributes in a class object.
 - Technically, the class statement defines a local scope that morphs into the attribute namespace of the class object, just like a module's global scope.
 - After running a class statement, class attributes are accessed by name qualification: `object.name`.

Class Objects Provide Default Behavior

- Class attributes provide object state and behavior.
 - Attributes of a class object record state information and behavior to be shared by all instances created from the class; function def statements nested inside a class generate methods, which process instances.

Instance Objects Are Concrete Items

- Calling a class object like a function makes a new instance object.
 - Each time a class is called, it creates and returns a new instance object.
 - Instances represent concrete items in your program's domain.

Instance Objects Are Concrete Items

- Each instance object inherits class attributes and gets its own namespace.
 - Instance objects created from classes are new namespaces; they start out empty but inherit attributes that live in the class objects from which they were generated.

Instance Objects Are Concrete Items

- Assignments to attributes of self in methods make per-instance attributes.
 - Inside a class's method functions, the first argument (called self by convention) references the instance object being processed; assignments to attributes of self create or change data in the instance, not the class.

In conclusion...

- The result is that classes define common, shared data and behavior, and generate instances.
- Instances reflect concrete application entities, and record per-instance data that may vary per object.

Example

Classes Are Customized by Inheritance

- Superclasses are listed in parentheses in a class header.
 - To make a class inherit attributes from another class, just list the other class in parentheses in the new class statement's header line.
 - The class that inherits is usually called a subclass, and the class that is inherited from is its superclass.
- Classes inherit attributes from their superclasses.
 - Just as instances inherit the attribute names defined in their classes, classes inherit all the attribute names defined in their superclasses; Python finds them automatically when they're accessed, if they don't exist in the subclasses.

Classes Are Customized by Inheritance

- Instances inherit attributes from all accessible classes.
 - Each instance gets names from the class it's generated from, as well as all that class's superclasses.
 - When looking for a name, Python checks the instance, then its class, then all superclasses.
- Logic changes are made by subclassing, not by changing superclasses.
 - By redefining superclass names in subclasses lower in the hierarchy (class tree), subclasses replace and thus customize inherited behavior.

Classes Are Customized by Inheritance

- Each object.attribute reference invokes a new, independent search.
 - Python performs an independent search of the class tree for each attribute fetch expression.
 - This includes references to instances and classes made outside class statements (e.g., X.attr), as well as references to attributes of the self instance argument in a class's method functions.
 - Each self.attr expression in a method invokes a new search for attr in self and above.

Example

Classes Can Intercept Python Operators

- Methods named with double underscores (`__X__`) are special hooks.
 - In Python classes we implement operator overloading by providing specially named methods to intercept operations.
 - The Python language defines a fixed and unchangeable mapping from each of these operations to a specially named method.
- Such methods are called automatically when instances appear in built-in operations.
 - For instance, if an instance object inherits an `__add__` method, that
 - method is called whenever the object appears in a `+` expression.
 - The method's return value becomes the result of the corresponding expression.

Classes Can Intercept Python Operators

- Classes may override most built-in type operations.
 - There are dozens of special operator overloading method names for intercepting and implementing nearly every operation available for built-in types.
 - This includes expressions, but also basic operations like printing and object creation.
- There are no defaults for operator overloading methods, and none are required.
 - If a class does not define or inherit an operator overloading method, it just means that the corresponding operation is not supported for the class's instances.
 - If there is no `__add__`, for example, `+` expressions raise exceptions.

Classes Can Intercept Python Operators

- New-style classes have some defaults, but not for common operations.
 - In Python 3.X, and so-called “new style” classes in 2.X that we’ll define later, a root class named `object` does provide defaults for some `__X__` methods, but not for many, and not for most used operations.
- Operators allow classes to integrate with Python’s object model.
 - By overloading type operations, the user-defined objects we implement with classes can act just like built-ins, and so provide consistency as well as compatibility with expected interfaces.

Example

The End