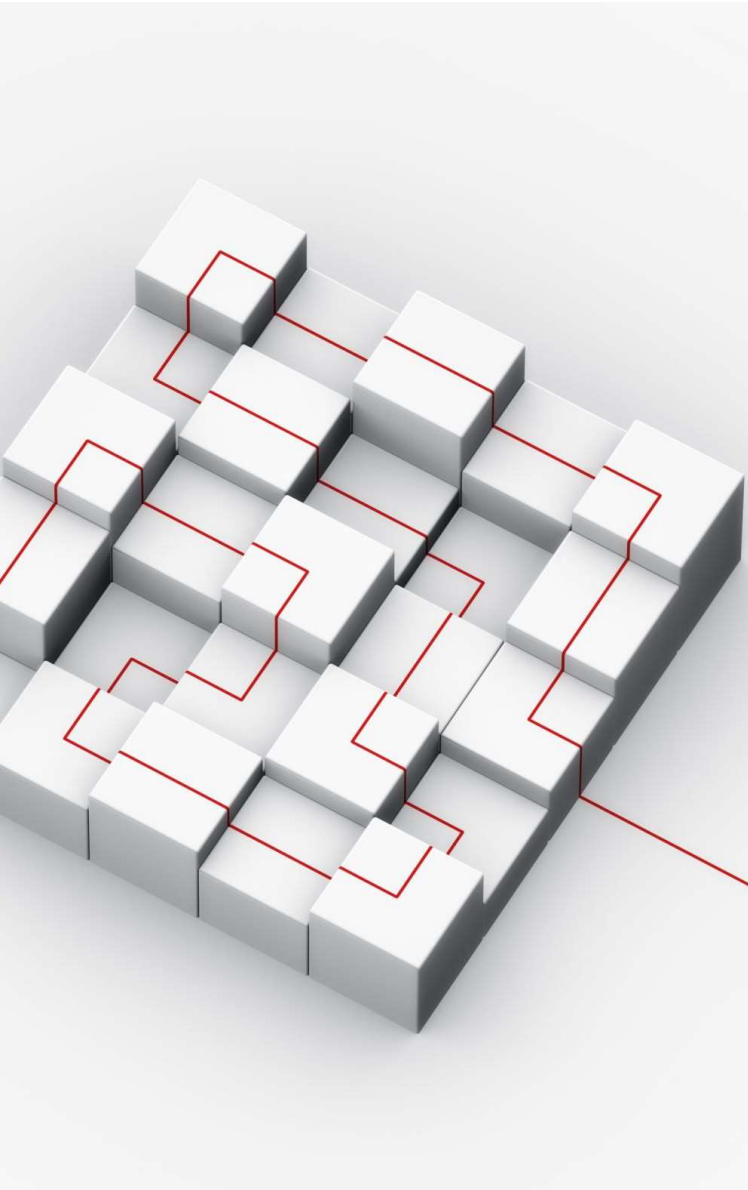


Object-Oriented Programming

# Class Coding Details



# This chapter covers

- The class Statement
- Method
- Inheritance
- Namespaces
- Classes vs Modules

# The class Statement

- Python's class is not a declaration.
- A class statement is an object builder, and an implicit assignment – when run, it generates a class object and stores a reference to it in the name used in the header.
- A class statement is a true executable code.

# General Form

- Class is a compound statement, with a body of statements typically indented appearing under the header.
- Superclasses are listed in parentheses after the class name, separated by commas.
- Within the class statement, any assignments generate class attributes, and specially named methods overload operators e.g. `__init__`

```
class name(superclass,...):           # Assign to name
    attr = value                       # Shared class data
    def method(self,...):             # Methods
        self.attr = value             # Per-instance data
```

mixednames.py

# Example

# Methods

- Methods are just function objects created by `def` statements nested in a class statement's body.
- Methods provide behavior for instance objects to inherit.
- A method's first argument always receives the instance object that called/invoked the method.

`instance.method(args...)`



`class.method(instance, args...)`

nextclass.py

# Example

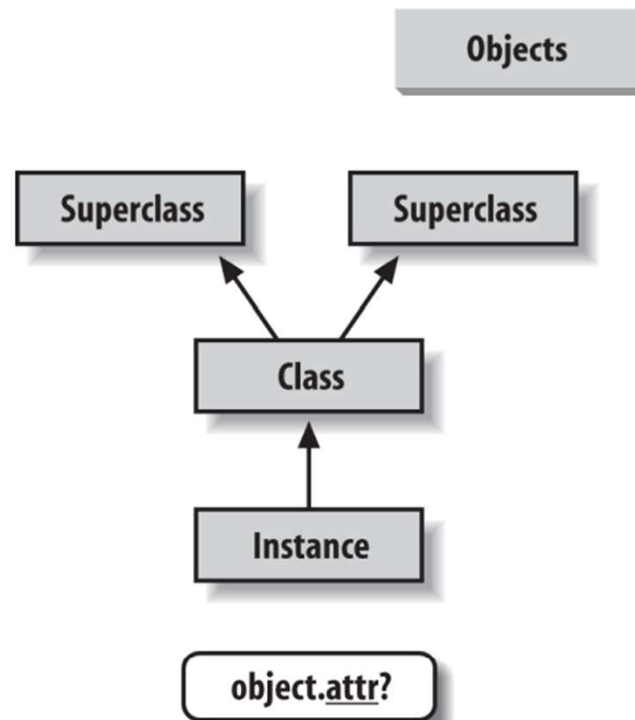
# Inheritance

- In Python, inheritance happens when an object is qualified, and it involves searching an attribute definition tree – one or more namespaces.



# Attribute Tree Construction

- Instance attributes are generated by assignments to self attributes in methods.
- Class attributes are created by statements (assignments) in class statements.
- Superclass links are made by listing classes in parentheses in a class statement header.



**Program**

```
class S1:
```

```
class S2:
```

```
class X(S1, S2):  
    def attr(self,...):  
        self.attr = V
```

```
object = X()
```

# Specializing Inherited Methods

- The tree-searching model of inheritance turns out to be a great way to specialized systems.
- Because inheritance finds names in subclasses before it checks superclasses, subclasses can replace default behavior by redefining their superclasses' attributes.
- In fact, you can build entire system as hierarchies of classes, by adding new subclasses rather than changing existing logic in the superclass.

supersub.py

# Example

# Class Interface Techniques

- Super
  - Defines a method function and a delegate that expects an action in a subclass.
- Inheritor
  - Doesn't provide any new names, so it gets everything defined in Super.
- Replacer
  - Overrides Super's method with a version of its own.
- Extender
  - Customizes Super's method by overriding and calling back to run the default.
- Provider
  - Implements the action method expected by Super's delegate method.

specialized.py

# Example

# Abstract Superclasses

- On the initial `x.delegate` call, Python finds the delegate method in `Super` by searching the `Provider` instance and above. The instance `x` is passed into the method's `self` argument as usual.
- Inside the `Super.delegate` method, `self.action` invokes a new, independent inheritance search of `self` and above. Because `self` references a `Provider` instance, the action method is in the `Provider` subclass.

# Namespaces: The Conclusion

- Unqualified names (e.g., `X`) deal with scopes.
- Qualified names (e.g., `object.X`) use object namespaces.
- Some scopes initialize object namespaces (for modules and classes).
- These concepts sometimes interact—in `object.X`, for example, `object` is looked up per scopes, and then `X` is looked up in the result objects.



manynames.py and otherfile.py

# Example

# Classes vs Modules

Classes	Modules
Implement new full-featured objects	Implement data/logic packages
Are created with class statement	Are created with Python files or other-language extensions
Are used by being called	Are used by being imported
Always live within a module	Form the top-level in Python program structure

Classes also support extra features that modules doesn't, such as operator overloading, multiple instance generation, and inheritance.

**The End**