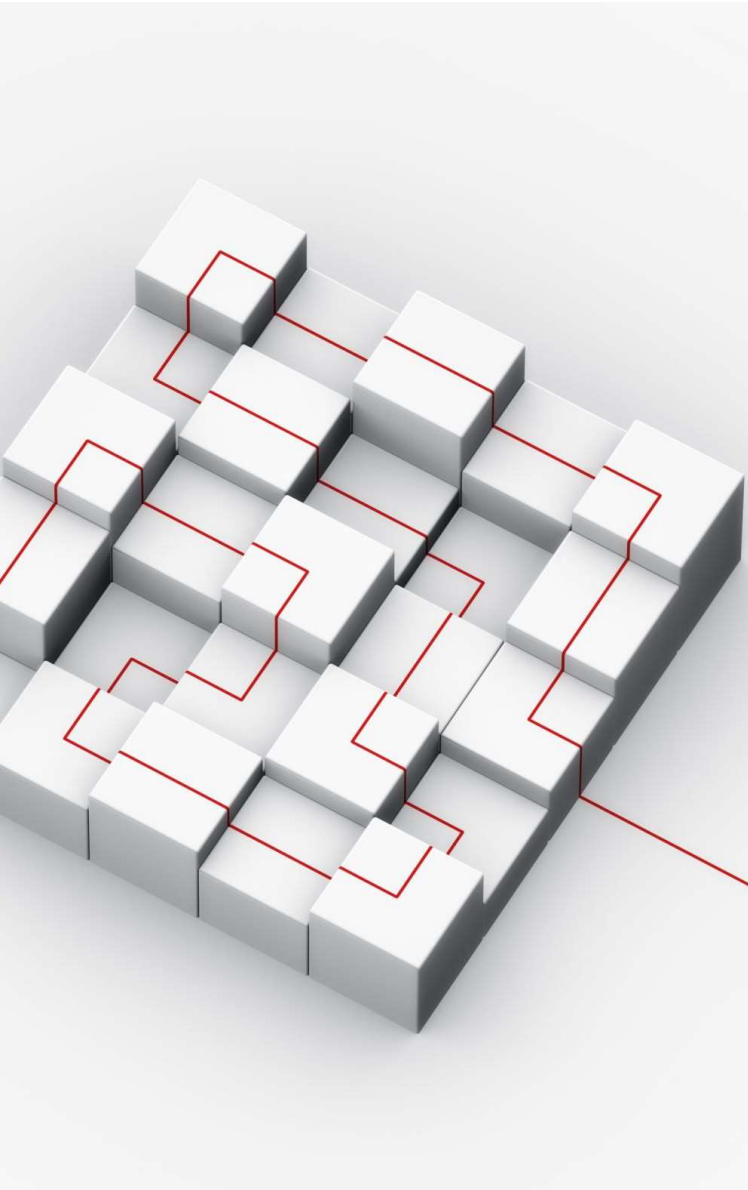


Object-Oriented Programming

Operator Overloading



This chapter covers

- The Basics
- Indexing and Slicing
- Iterable Objects
- Membership
- Attribute Access
- String Representation
- Call Expressions
- Comparisons
- Object Destruction

The Basics

- Operator overloading lets classes intercept normal Python operations.
- Classes can overload all Python expression operators.
- Class can also overload built-in operations such as printing, function calls, attribute access, etc.
- Overloading makes class instances act more like built-in types.
- Overloading is implemented by providing specially named methods in a class.

The Basics

- In other words, when certain specially named methods are provided in a class, Python automatically calls them when instances of the class appear in their associated expressions.
- Your class provides the behavior of the corresponding operation for instance objects created from it.

number.py

Example

Indexing and Slicing: `__getitem__` and `__setitem__`

- The `__getitem__` method is called automatically for instance-indexing operations.
- If used, the `__setitem__` index assignment method intercepts both index and slice assignments.

indexer.py

Example

Iterable Objects: `__iter__` and `__next__`

- Although the `__getitem__` technique of the prior section works, it's really just a fallback for iteration.
- Today, all iteration contexts in Python will try the `__iter__` method first, before trying `__getitem__`.
- Technically, iteration contexts work by passing an iterable object to the `iter` built-in function to invoke an `__iter__` method, which is expected to return an iterator object.
- If it's provided, Python then repeatedly calls this iterator object's `__next__` method to produce items until a `StopIteration` exception is raised.

squares.py

Example

Membership: `__contains__`, `__iter__`, and `__getitem__`

- In the iterations domain, classes can implement the in membership operator as an iteration, using either the `__iter__` or `__getitem__` methods.
- To support more specific membership, though, classes may code a `__contains__` method - when present, this method is preferred over `__iter__`, which is preferred over `__getitem__`.
- The `__contains__` method should define membership as applying to keys for a mapping (and can use quick lookups), and as a search for sequences.

contains.py

Example

Attribute Access: `__getattr__` and `__setattr__`

- The `__getattr__` method intercepts attribute references.
- In the same department, the `__setattr__` intercepts all attribute assignments.

empty.py

Example

String Representation: `__repr__` and `__str__`

- If defined, `__repr__` (or its close relative, `__str__`) is called automatically when class instances are printed or converted to strings.
- These methods allow you to define a better display format for your objects than the default instance display.
- Here, `__repr__` uses basic string formatting to convert the managed `self.data` object to a more human-friendly string for display.

addrepr.py

Example

Call Expressions: `__call__`

- The `__call__` method is called when your instance is called.
- If defined, Python runs a `__call__` method for function call expressions applied to your instances, passing along whatever positional or keyword arguments were sent.

callee.py

Example

Comparisons: `__lt__`, `__gt__`, and Others

```
class C:
    data = 'spam'
    def __gt__(self, other):           # 3.X and 2.X version
        return self.data > other
    def __lt__(self, other):
        return self.data < other

X = C()
print(X > 'ham')                      # True (runs __gt__)
print(X < 'ham')                      # False (runs __lt__)
```

Object Destruction: `__del__`

- We've seen how the `__init__` constructor is called whenever an instance is generated (and noted how `__new__` is run first to make the object).
- Its counterpart, the destructor method `__del__`, is run automatically when an instance's space is being reclaimed.

life.py

Example

The End