Core Python:
Classes and Object-oriented Programming

CLASS ATTRIBUTES, METHODS AND PROPERTIES

Robert Smallshire
COFOUNDER - SIXTY NORTH

@robsmallshire

Austin Bingham
COFOUNDER - SIXTY NORTH

@austin_bingham

Overview

Review:

- Class definition
- Initializer definition
- Instance attributes

Class attributes

Static methods

Class methods

Interactions with inheritance

Overview

Properties

Specializing properties

Designing for extensibility

Naming Special Functions

__feature__

Hard to prounounce!

dunder

Our way of pronouncing special names

A portmanteau of ‘double underscore’

Instead of “underscore underscore name underscore underscore” we'll say “dunder name”

Instances

Instance Attributes

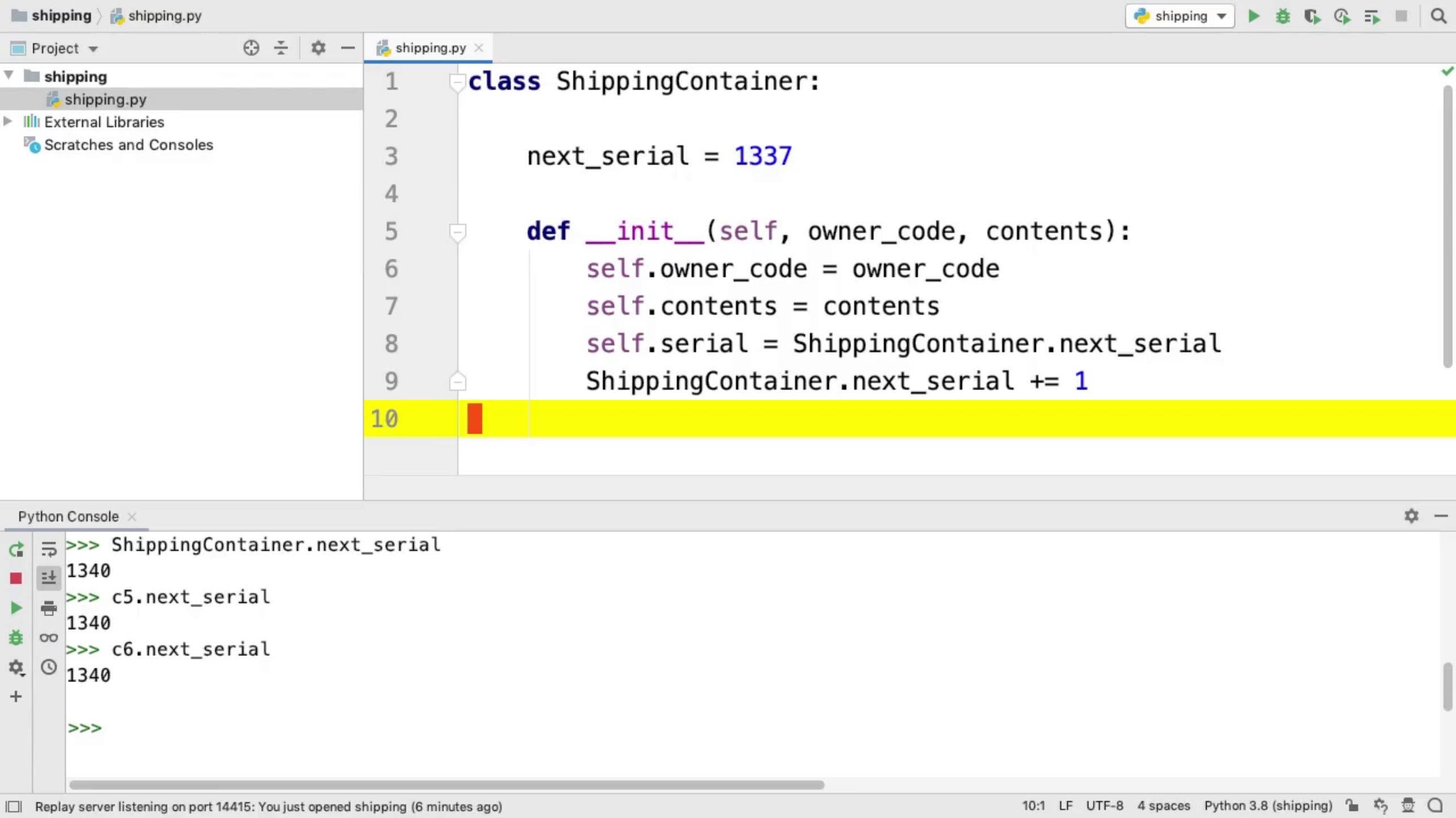
```
class Rectangle:

    def __init__(self, width, height)
        self.width = width
        self.height = height
```


CARGO
CONTAINER
CARGO
CONTAINER
CARGO
CONTAINER
CARGO
CONTAINER
CARGO
CONTAINER
MAX. LOAD
TARE
MAX.GW

Class Attributes

Class Attribute Syntax

```
class MyClass:


    # Define class attributes in the class block

    my_class_attribute = "class attributes go here"

    MY_CONSTANT = "they are often class-specific contants"


    def __init__(self):

        self.my_instance_attribute = "instance attributes here"
```


shipping
shipping.py
Project
shipping
shipping.py
External Libraries
Scratches and Consoles
shipping.py
1 class ShippingContainer:
2
3 next_serial = 1337
4
5 def __init__(self, owner_code, contents):
6 self.owner_code = owner_code
7 self.contents = contents
8 self.serial = ShippingContainer.next_serial
9 ShippingContainer.next_serial += 1
10
Python Console
>>> ShippingContainer.next_serial
1340
>>> c5.next_serial
1340
>>> c6.next_serial
1340
>>>
Replay server listening on port 14415: You just opened shipping (6 minutes ago)
10:1 LF UTF-8 4 spaces Python 3.8 (shipping)

Scopes in Python

Local	Inside the current function
Enclosing	Inside enclosing functions
Global	At the top level of the module
Built-in	In the special `builtins` module

LEGB

Moment of Zen

The Zen of Python

Explicit is better than implicit

Readability counts

```
class MyClass:

    b = "on class"

    def __init__(self):
        self.a = "on instance"
        print(self.a)
        print(MyClass.b)
        print(self.b)
        self.a = "re-bound"
        self.b = "new on instance"
        print(self.b)
        print(MyClass.a)
```

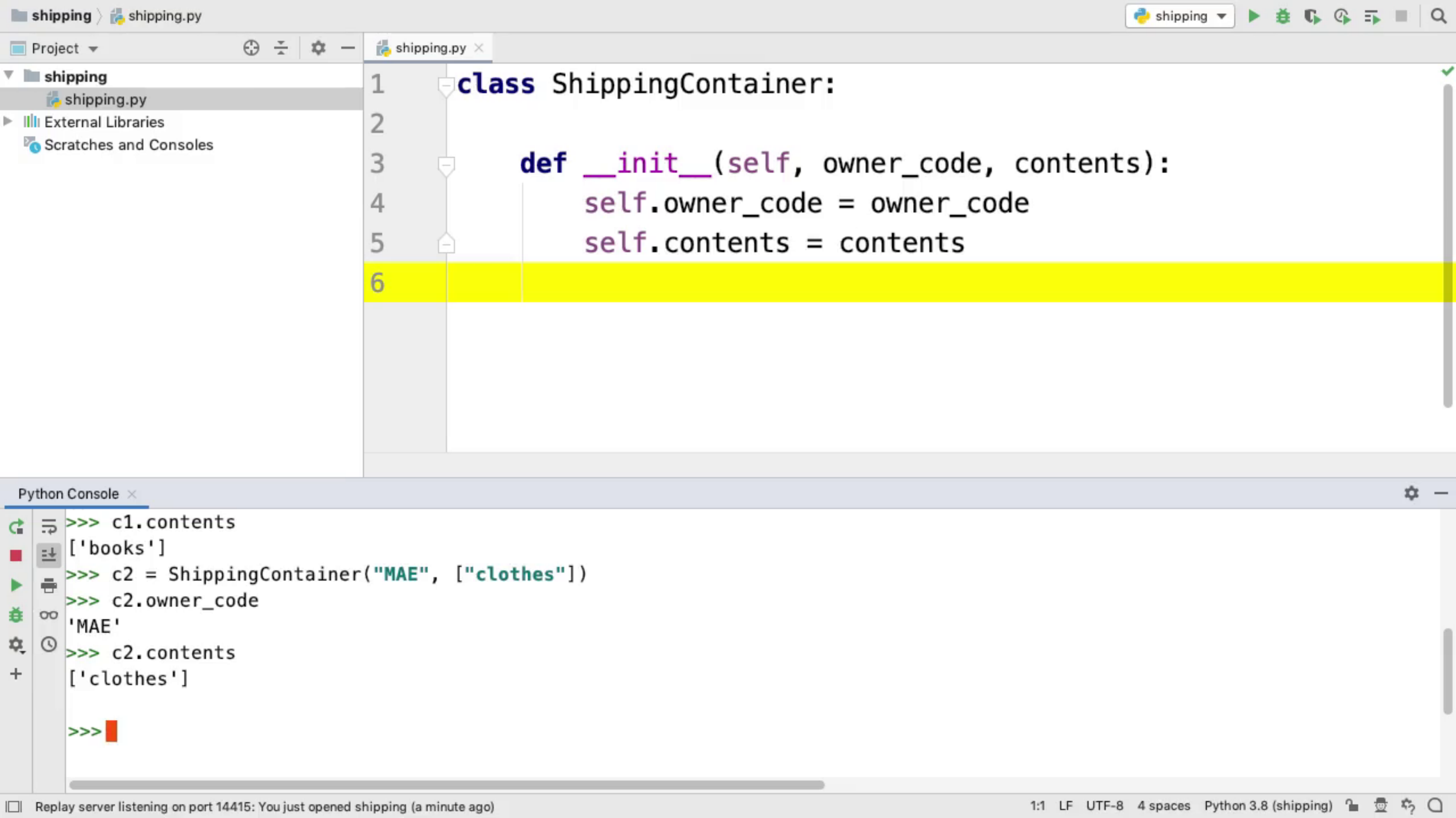
- Accesses the class attribute
- Re-binds the existing instance attribute
- Instance attribute hides the class attribute
- Accesses the instance attribute via `self`
- Access class attribute via the class object

There is
no class scope
in Python.

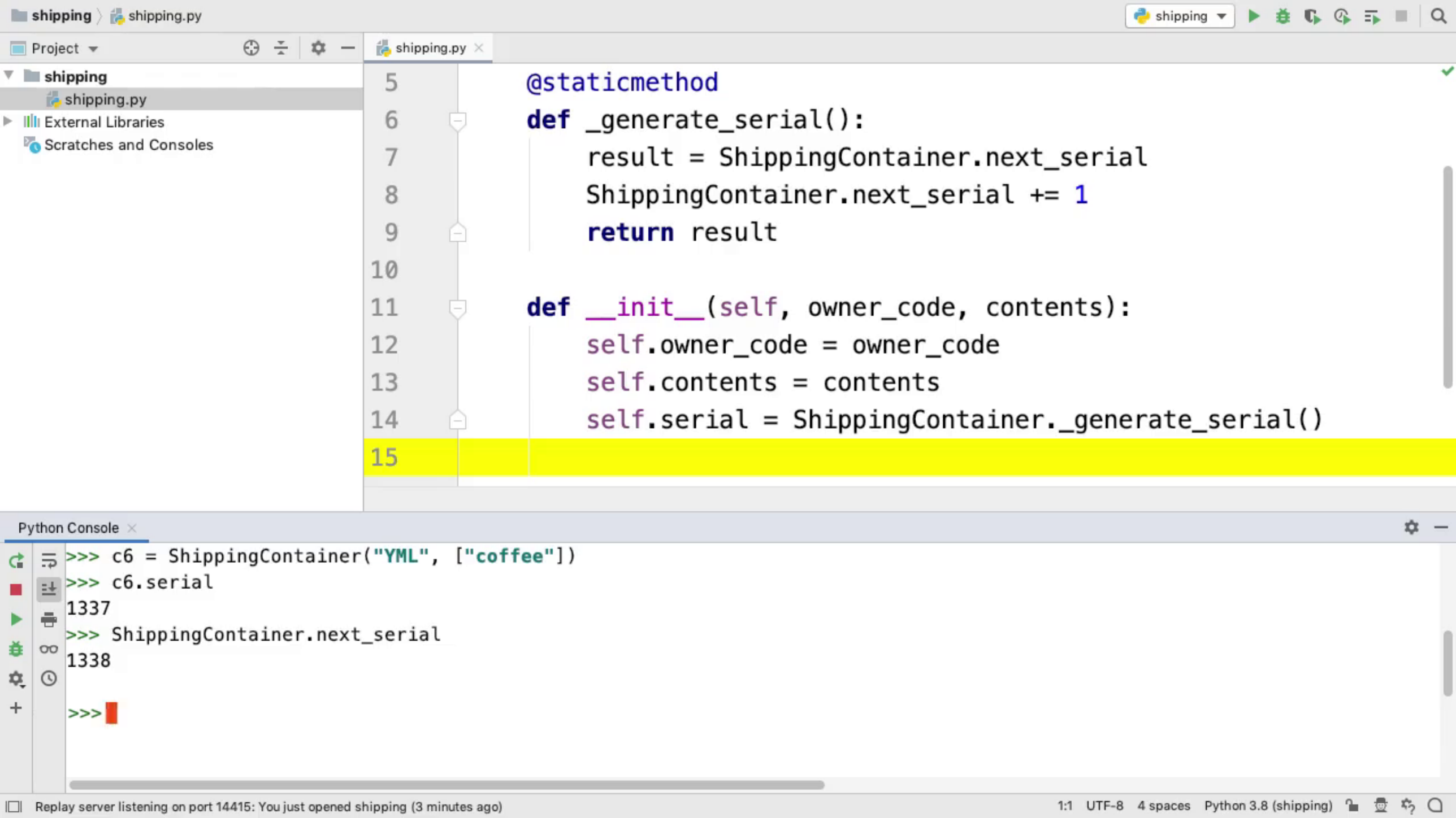
Scopes in Python

Local	Inside the current function
Enclosing	Inside enclosing functions
Global	At the top level of the module
Built-in	In the special `builtins` module

LEGB


shipping
shipping.py
Project
shipping
shipping.py
External Libraries
Scratches and Consoles
shipping.py
1 class ShippingContainer:
2
3 def __init__(self, owner_code, contents):
4 self.owner_code = owner_code
5 self.contents = contents
6
Python Console
>>> c1.contents
['books']
>>> c2 = ShippingContainer("MAE", ["clothes"])
>>> c2.owner_code
'MAE'
>>> c2.contents
['clothes']
>>>
Replay server listening on port 14415: You just opened shipping (a minute ago)
1:1 LF UTF-8 4 spaces Python 3.8 (shipping)

Static Methods


shipping
shipping.py
Project
shipping
shipping.py
External Libraries
Scratches and Consoles
shipping.py
5 @staticmethod
6 def _generate_serial():
7 result = ShippingContainer.next_serial
8 ShippingContainer.next_serial += 1
9 return result
10
11 def __init__(self, owner_code, contents):
12 self.owner_code = owner_code
13 self.contents = contents
14 self.serial = ShippingContainer._generate_serial()
15
Python Console
>>> c6 = ShippingContainer("YML", ["coffee"])
>>> c6.serial
1337
>>> ShippingContainer.next_serial
1338
>>>
Replay server listening on port 14415: You just opened shipping (3 minutes ago)
1:1 UTF-8 4 spaces Python 3.8 (shipping)

The Static Terminology Is a Relic from C and C++

```
#include <stdio.h>

class StaticTest {
private:
    static int x;
public:
    static int count() { return x; }
};

int StaticTest::x = 9;

int main() {
    printf_s("%d\n", StaticTest::count());
}
```

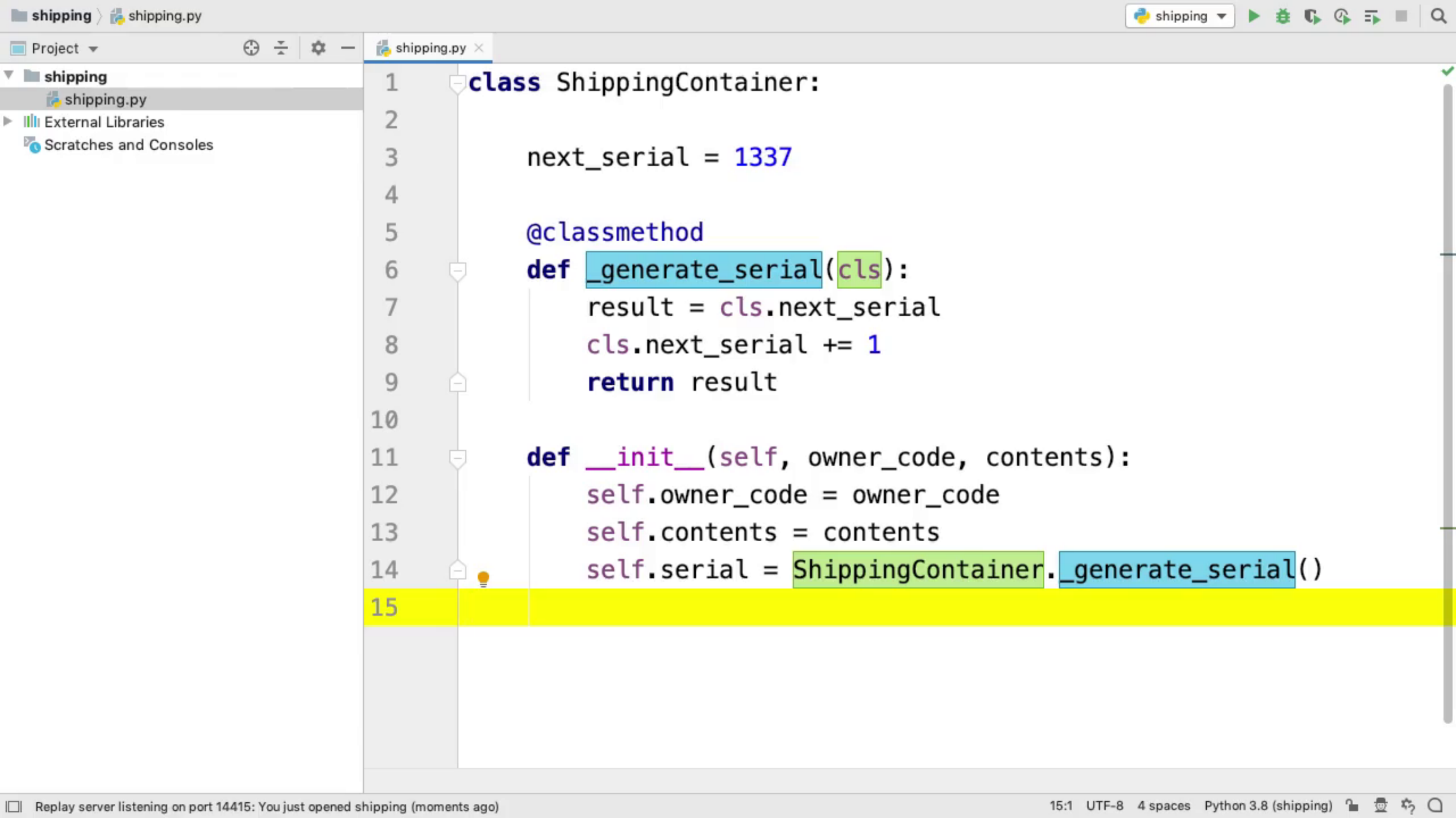
Class Methods

```python
class MyClass:

    attribute = "class attribute"


    @classmethod
    def my_class_method(cls, message):
        cls.attribute = message
```

◀ Decorated by `classmethod`

◀ Accepts `cls` as first argument

◀ Access class attributes via `cls`


shipping
shipping.py
shipping
Project
shipping
shipping.py
External Libraries
Scratches and Consoles
shipping.py
class ShippingContainer:
next_serial = 1337
@classmethod
def _generate_serial(cls):
result = cls.next_serial
cls.next_serial += 1
return result
def __init__(self, owner_code, contents):
self.owner_code = owner_code
self.contents = contents
self.serial = ShippingContainer._generate_serial()
Replay server listening on port 14415: You just opened shipping (moments ago)
15:1
UTF-8
4 spaces
Python 3.8 (shipping)

Choosing

@classmethod

Requires access to the class object to call other class methods or the constructor

@staticmethod

No access needed to either *class* or *instance* objects

Most likely an implementation detail of the class

May be able to be moved outside the class to become a global-scope function in the module

Options for Locating Functions

Global Function

```
class SplineReticulator:

    def reticulate(x, y):
        return _tricky_math(x, y)

# Global function
def _tricky_math(x, y):
    return 2 * x**2 - 4*y
```

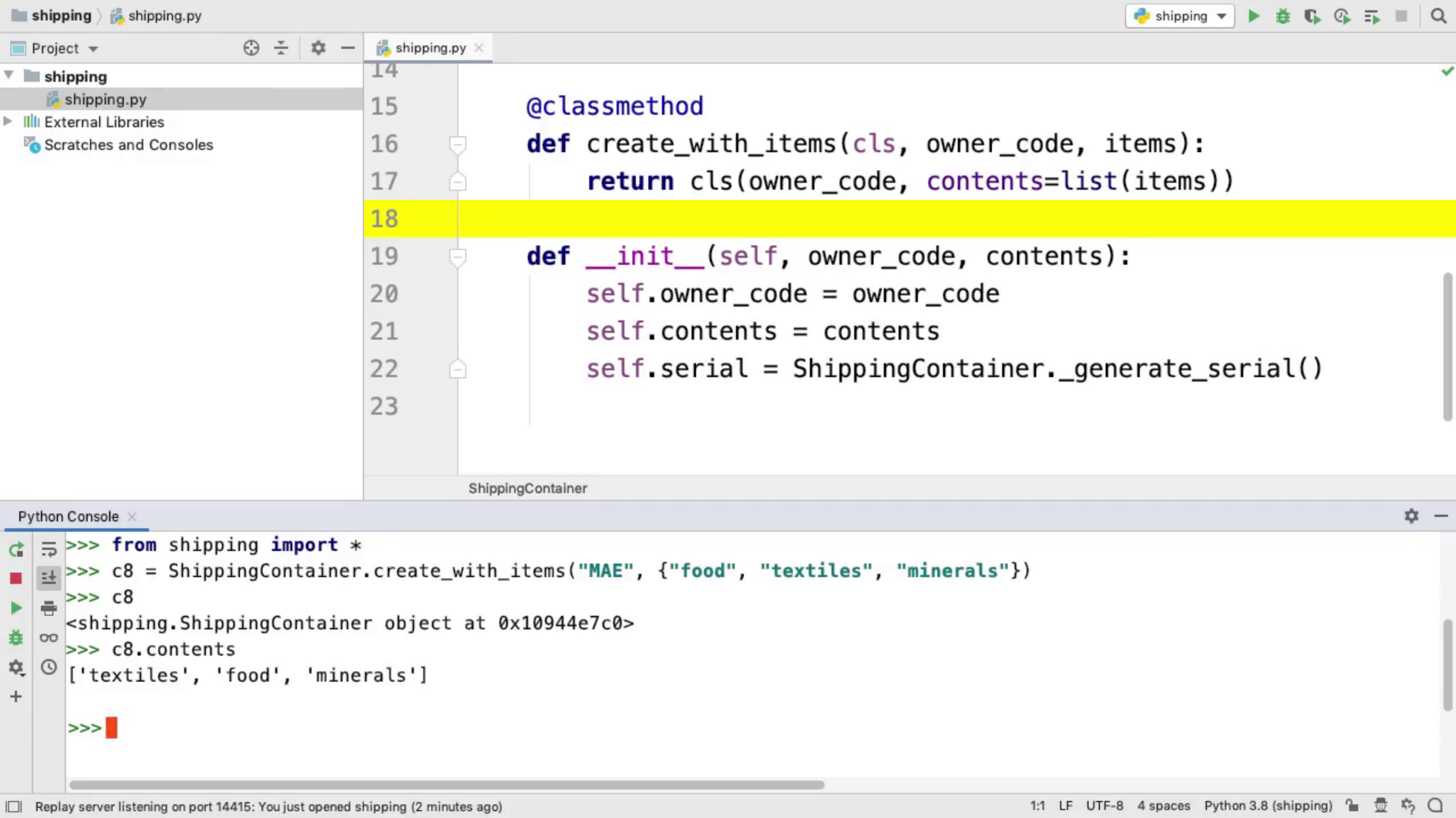
@staticmethod

```
class SplineReticulator:

    def reticulate(x, y):
        return self._tricky_math(x, y)

    @staticmethod
    def _tricky_math(x, y):
        return 2 * x**2 - 4*y
```

The ‘named constructor’ idiom

A factory method which returns an instance of a class. The method name allows callers to express intent, and allows construction to be performed with different combinations of arguments.

Originally a C++ idiom, also applicable in Python


shipping
shipping.py
shipping
Project
shipping
shipping.py
External Libraries
Scratches and Consoles
shipping.py
14
15 @classmethod
16 def create_with_items(cls, owner_code, items):
17 return cls(owner_code, contents=list(items))
18
19 def __init__(self, owner_code, contents):
20 self.owner_code = owner_code
21 self.contents = contents
22 self.serial = ShippingContainer._generate_serial()
23
ShippingContainer
Python Console
>>> from shipping import *
>>> c8 = ShippingContainer.create_with_items("MAE", {"food", "textiles", "minerals"})
>>> c8
<shipping.ShippingContainer object at 0x10944e7c0>
>>> c8.contents
['textiles', 'food', 'minerals']
>>>
Replay server listening on port 14415: You just opened shipping (2 minutes ago)
1:1 LF UTF-8 4 spaces Python 3.8 (shipping)


CAXU
HJCU
8281 98 8
22G1
MAX.GROSS
TARE
PAYLOAD
CUBE
HANJIN
SHIPPING
CO.,LTD.
MSCU
M. G. W.
TARE
NET
CU.CAP.

ISO:6346 BIC Code

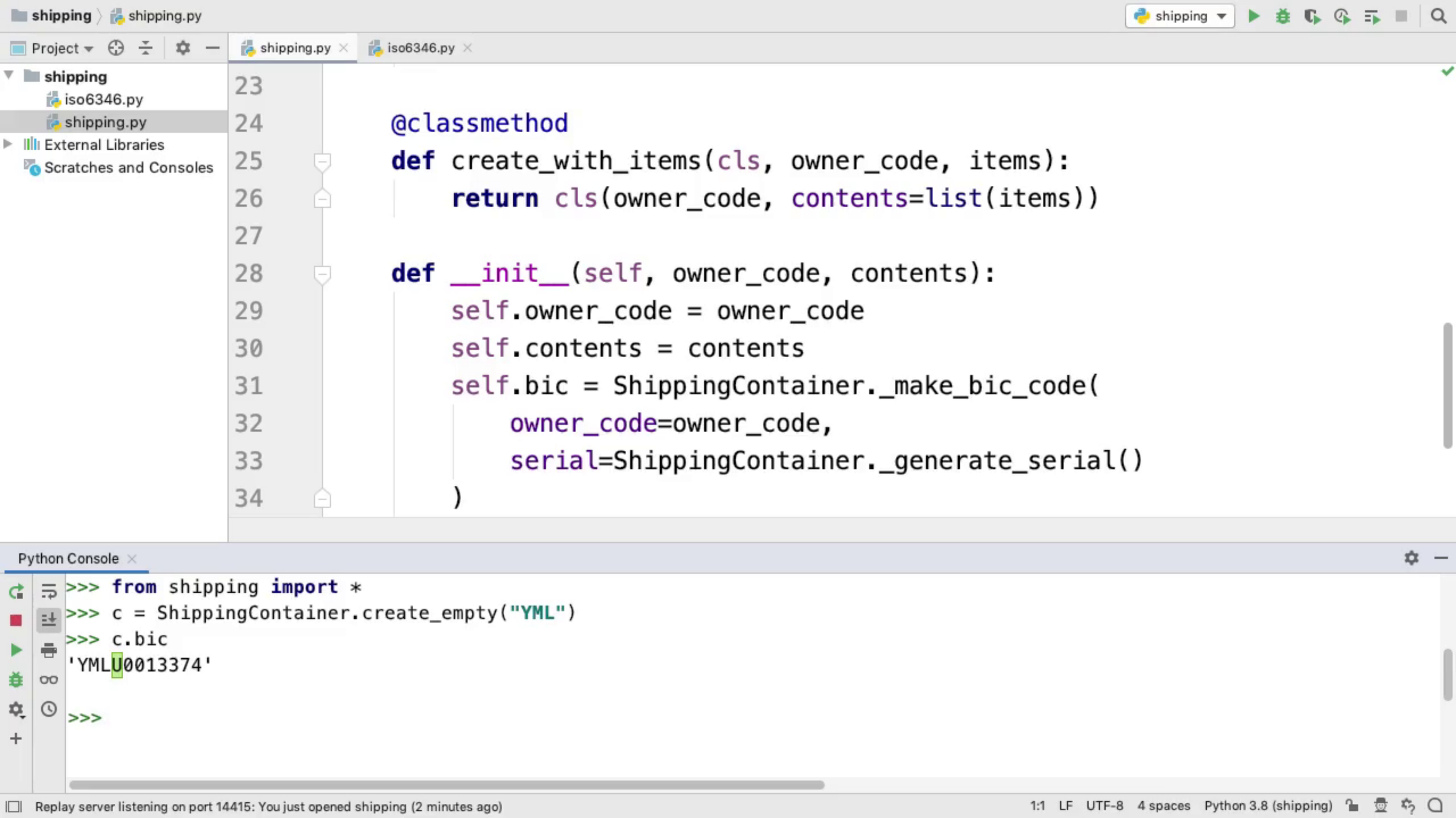
Category identifier

Check digit

CSQU3054383

Owner code

Serial number


shipping
shipping.py
Project
shipping.py
iso6346.py
shipping
iso6346.py
shipping.py
External Libraries
Scratches and Consoles
23
24 @classmethod
25 def create_with_items(cls, owner_code, items):
26 return cls(owner_code, contents=list(items))
27
28 def __init__(self, owner_code, contents):
29 self.owner_code = owner_code
30 self.contents = contents
31 self.bic = ShippingContainer._make_bic_code(
32 owner_code=owner_code,
33 serial=ShippingContainer._generate_serial()
34)
Python Console
>>> from shipping import *
>>> c = ShippingContainer.create_empty("YML")
>>> c.bic
'YMLU0013374'
>>>
Replay server listening on port 14415: You just opened shipping (2 minutes ago)
1:1 LF UTF-8 4 spaces Python 3.8 (shipping)

Static Methods with Inheritance


MAERSK
SEALAND
INGER

ISO:6346 BIC Code

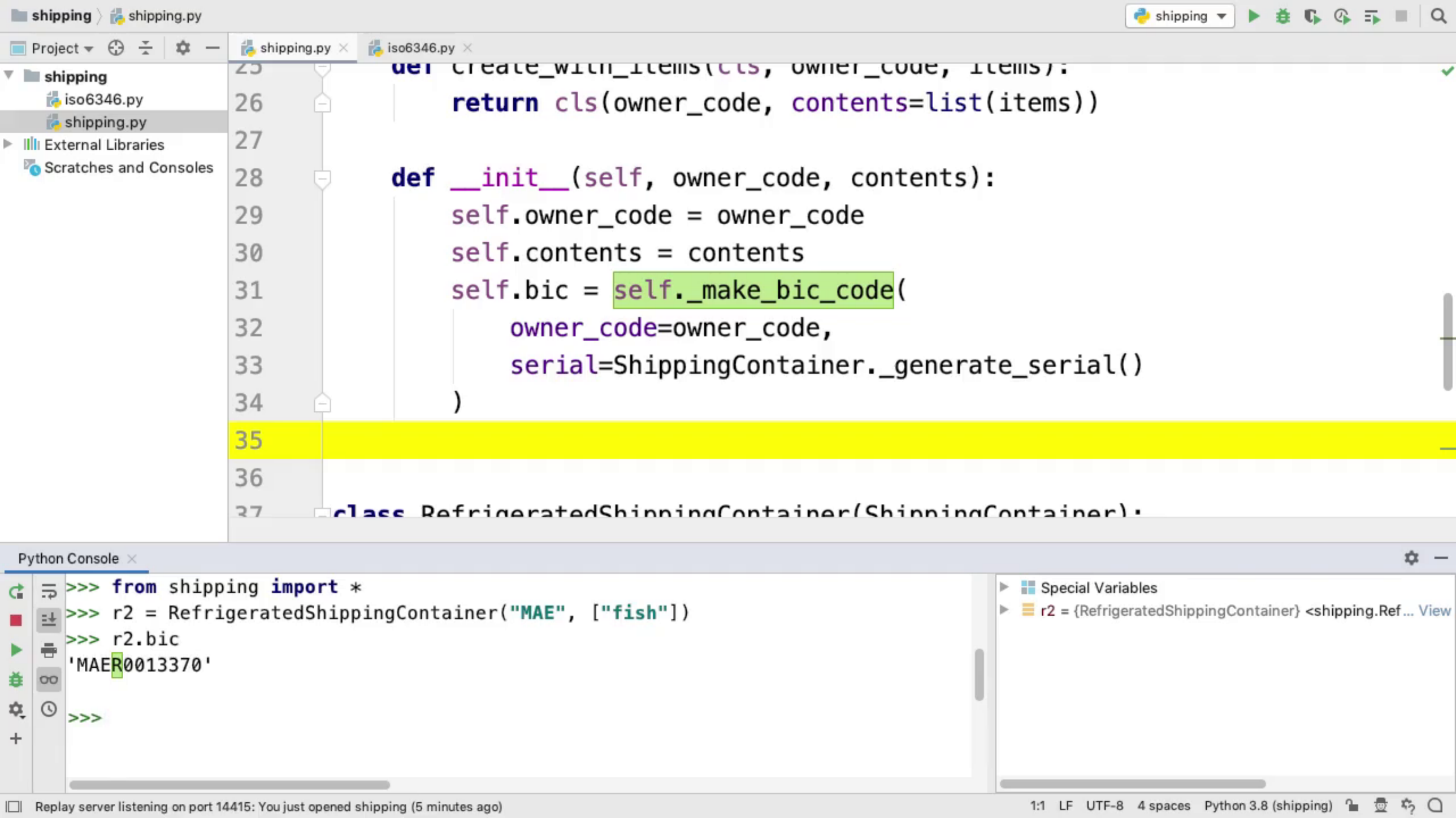
Category identifier

Check digit

CSQR3054381

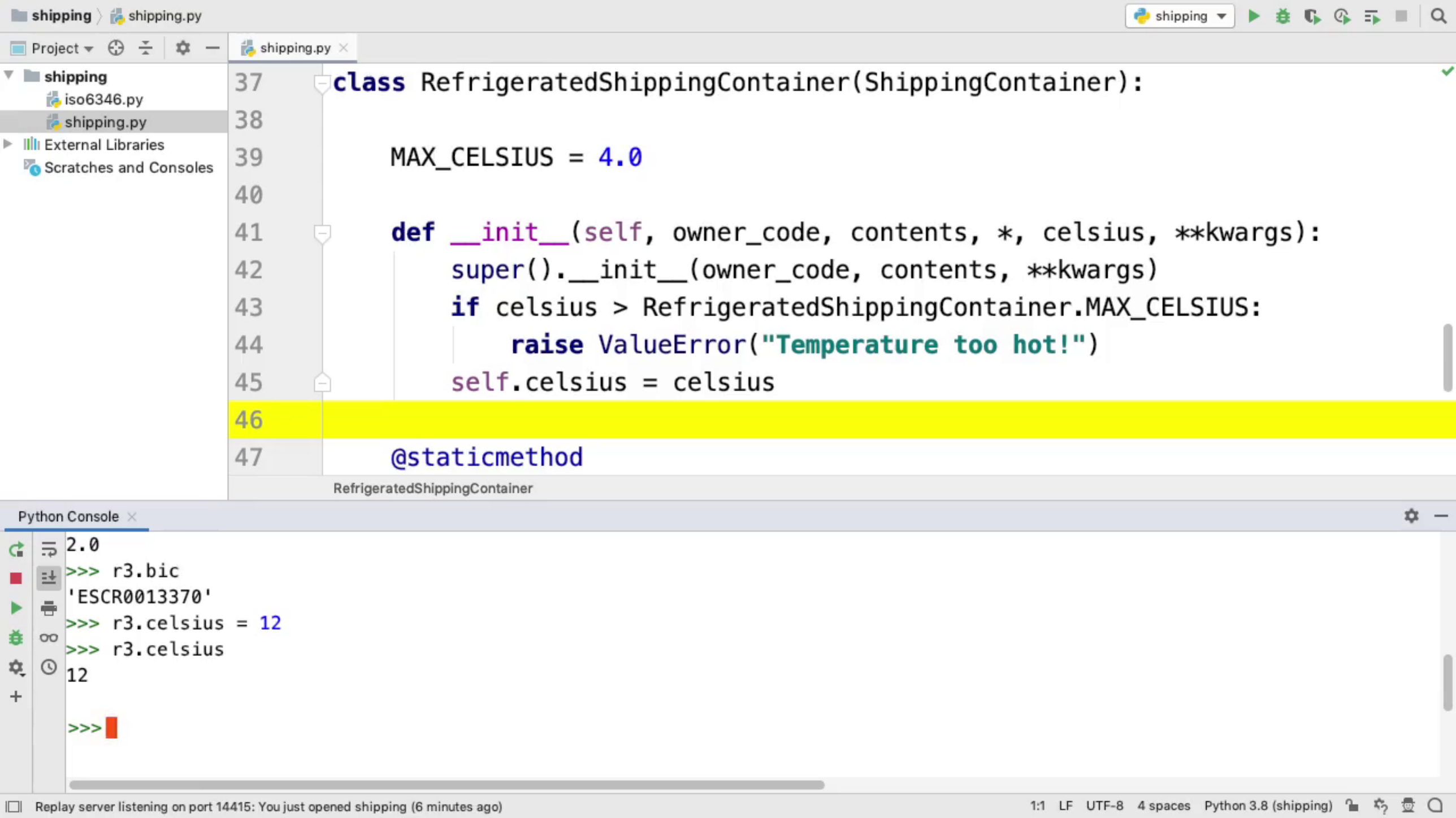
Owner code

Serial number


shipping
shipping.py
Project
shipping.py
iso6346.py
shipping
iso6346.py
shipping.py
External Libraries
Scratches and Consoles
def create_with_items(cls, owner_code, items):
return cls(owner_code, contents=list(items))
def __init__(self, owner_code, contents):
self.owner_code = owner_code
self.contents = contents
self.bic = self._make_bic_code(
owner_code=owner_code,
serial=ShippingContainer._generate_serial()
)
class RefrigeratedShippingContainer(ShippingContainer):
Python Console
>>> from shipping import *
>>> r2 = RefrigeratedShippingContainer("MAE", ["fish"])
>>> r2.bic
'MAER0013370'
>>>
Special Variables
r2 = {RefrigeratedShippingContainer} <shipping.Ref... View
Replay server listening on port 14415: You just opened shipping (5 minutes ago)
1:1
LF
UTF-8
4 spaces
Python 3.8 (shipping)

For polymorphic dispatch invoke static methods through `self`.

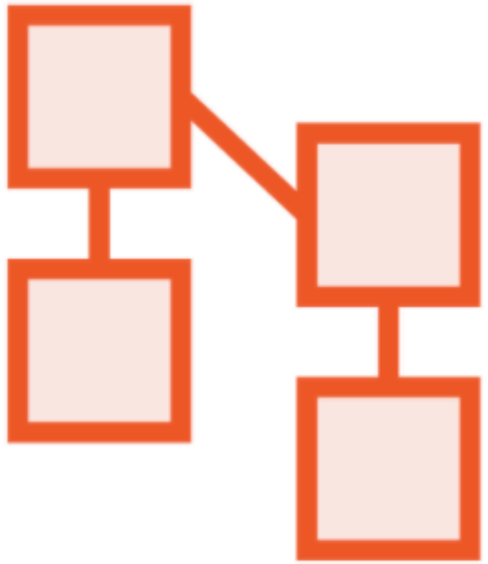
Class Methods with Inheritance


shipping
shipping.py
Project
shipping
iso6346.py
shipping.py
External Libraries
Scratches and Consoles
class RefrigeratedShippingContainer(ShippingContainer):
MAX_CELSIUS = 4.0
def __init__(self, owner_code, contents, *, celsius, **kwargs):
super().__init__(owner_code, contents, **kwargs)
if celsius > RefrigeratedShippingContainer.MAX_CELSIUS:
raise ValueError("Temperature too hot!")
self.celsius = celsius
@staticmethod
RefrigeratedShippingContainer
Python Console
2.0
>>> r3.bic
'ESCR0013370'
>>> r3.celsius = 12
>>> r3.celsius
12
>>>
Replay server listening on port 14415: You just opened shipping (6 minutes ago)
1:1 LF UTF-8 4 spaces Python 3.8 (shipping)

Class Invariant Violation

`celsius` must never exceed `MAX_CELSIUS`

Avoid Circular Dependencies



Base classes should have no knowledge of subclasses

Use **kwargs to thread arguments through named-constructor class-methods to more specialized subclasses.

Encapsulation with Properties

Establishing a Class Invariant

```
class RefrigeratedShippingContainer(ShippingContainer):

    MAX_CELSIUS = 4.0

    def __init__(self, owner_code, contents, *, celsius, **kwargs):
        super().__init__(owner_code, contents, **kwargs)
        if celsius > RefrigeratedShippingContainer.MAX_CELSIUS:
            raise ValueError("Temperature too hot!")
        self.celsius = celsius
```

Class Invariant Violation

`celsius` must never exceed `MAX_CELSIUS`

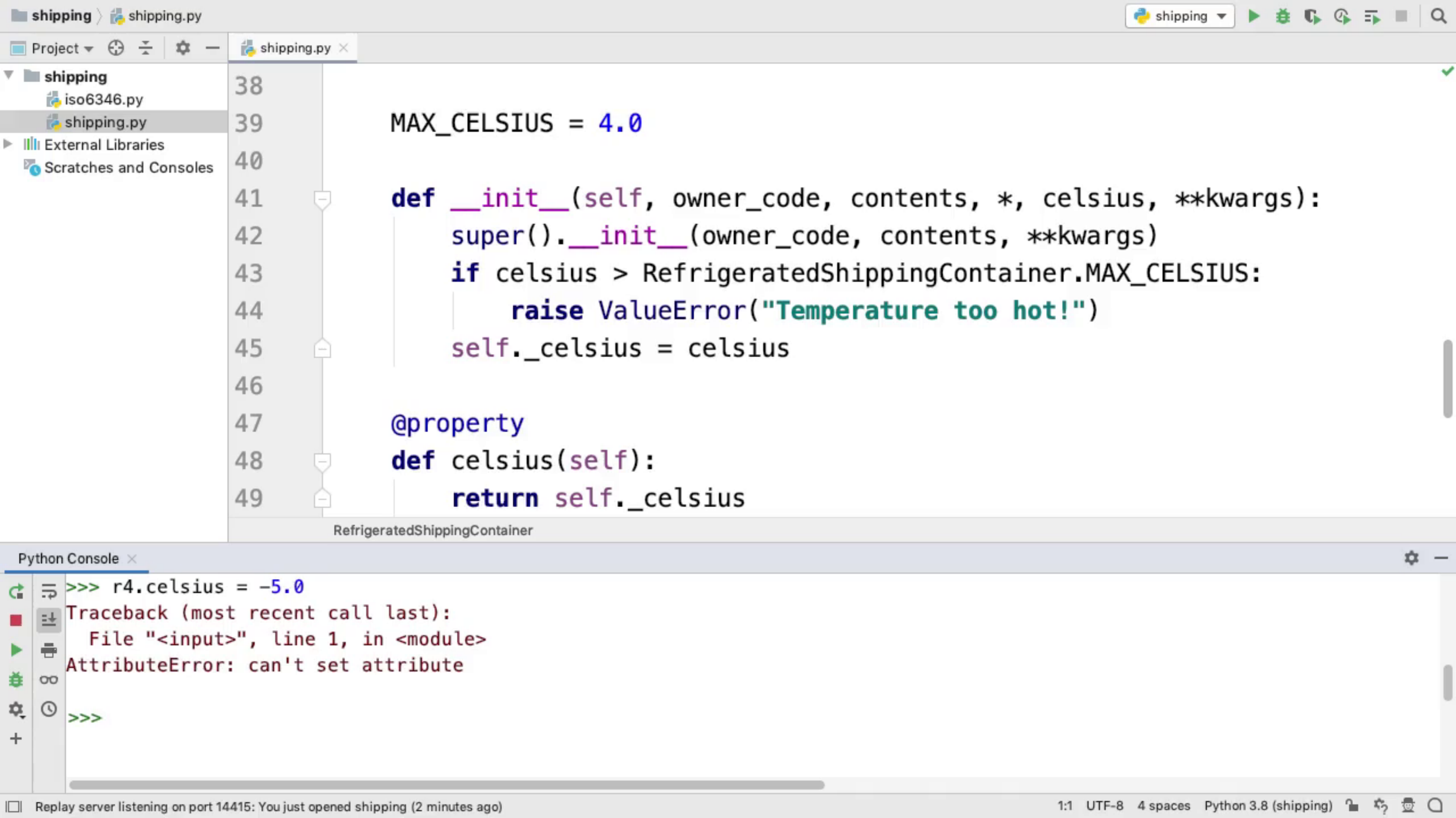
Maintaining the Maximum Temperature Invariant

```
class RefrigeratedShippingContainer(ShippingContainer):

    MAX_CELSIUS = 4.0

    def __init__(self, owner_code, contents, *, celsius, **kwargs):
        super().__init__(owner_code, contents, **kwargs)
        if celsius > RefrigeratedShippingContainer.MAX_CELSIUS:
            raise ValueError("Temperature too hot!")
        self._celsius = celsius

    def get_celsius(self):
        return self._celsius

    def set_celsius(self, value):
        if celsius > RefrigeratedShippingContainer.MAX_CELSIUS:
            raise ValueError("Temperature too hot!")
        self._celsius = value
```

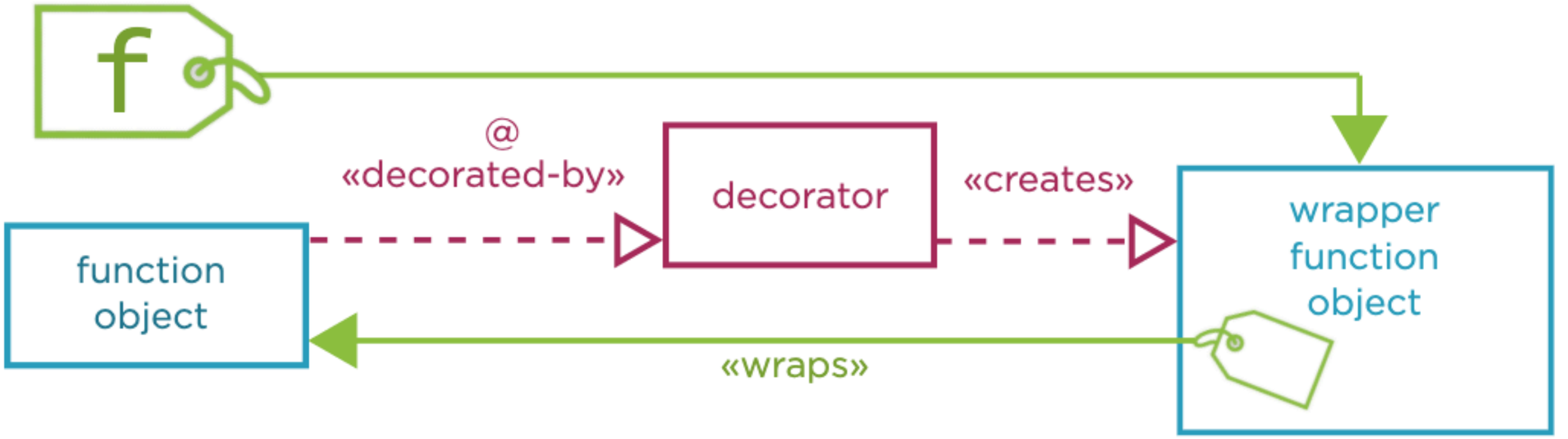
Getters and Setters Are Not Pythonic

≠

Encapsulate getter and setter methods in properties which behave like attributes

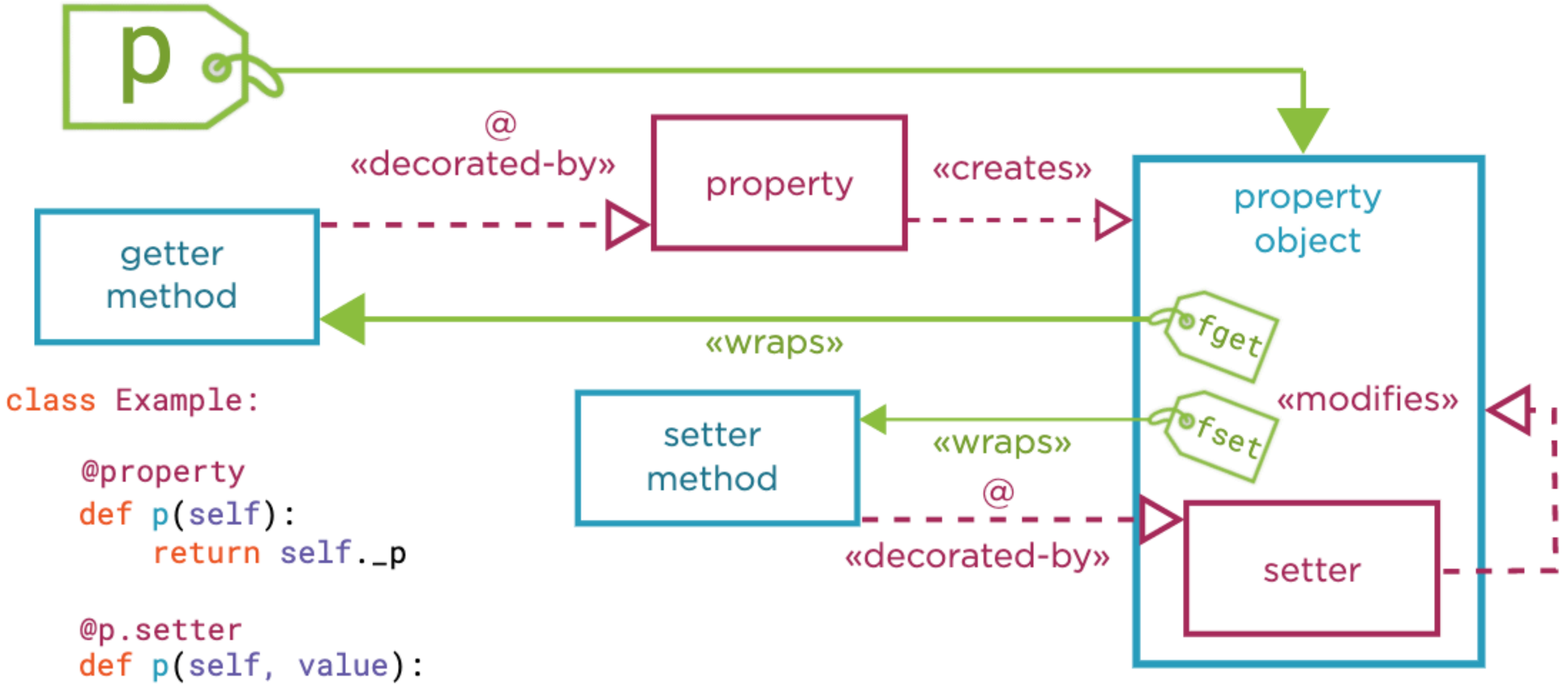

shipping
shipping.py
Project
shipping.py
shipping
iso6346.py
shipping.py
External Libraries
Scratches and Consoles
38
39 MAX_CELSIUS = 4.0
40
41 def __init__(self, owner_code, contents, *, celsius, **kwargs):
42 super().__init__(owner_code, contents, **kwargs)
43 if celsius > RefrigeratedShippingContainer.MAX_CELSIUS:
44 raise ValueError("Temperature too hot!")
45 self._celsius = celsius
46
47 @property
48 def celsius(self):
49 return self._celsius
RefrigeratedShippingContainer
Python Console
>>> r4.celsius = -5.0
Traceback (most recent call last):
File "<input>", line 1, in <module>
AttributeError: can't set attribute
>>>
Replay server listening on port 14415: You just opened shipping (2 minutes ago)
1:1
UTF-8
4 spaces
Python 3.8 (shipping)

Decorator Recap

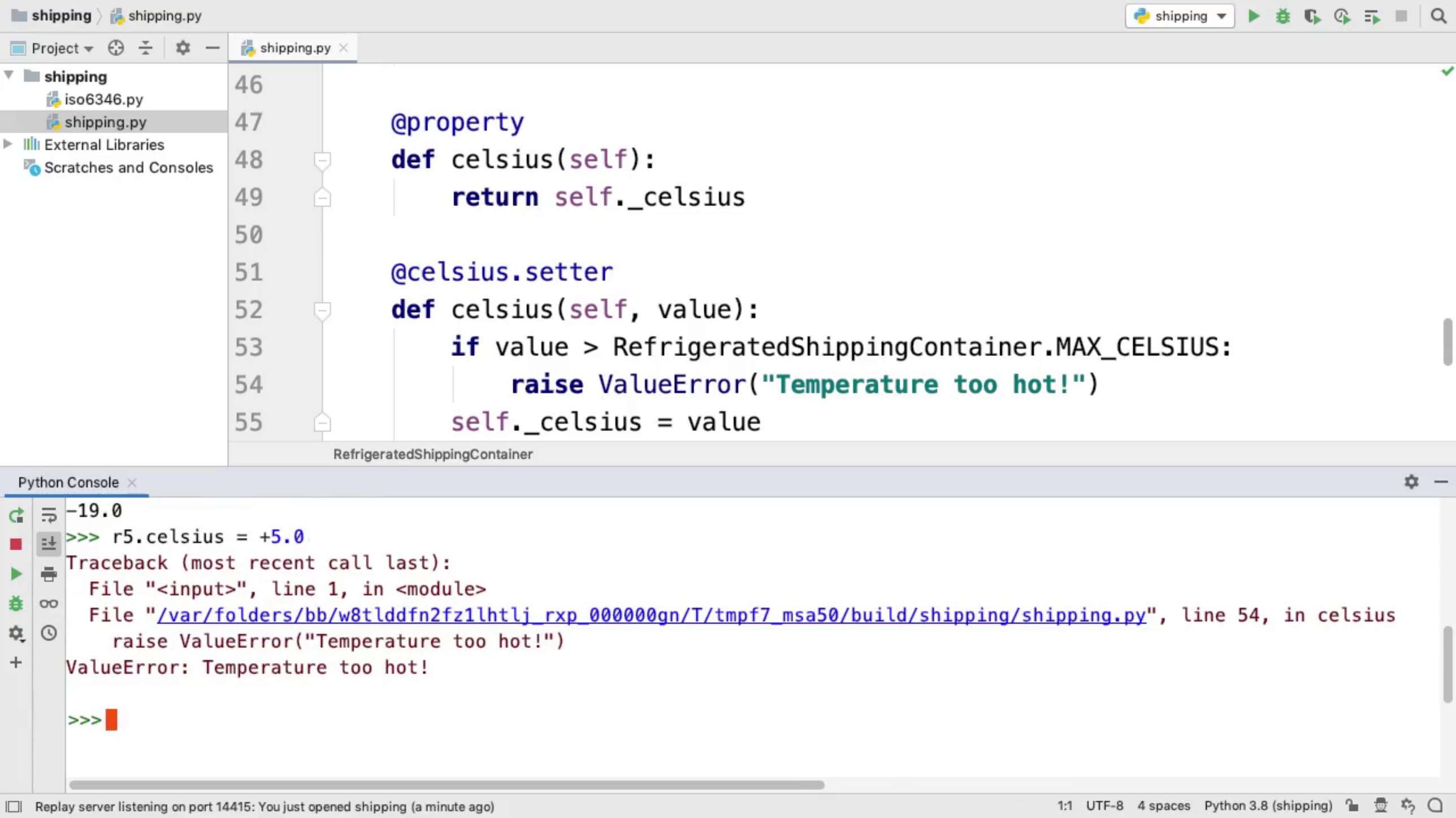



```
@decorator
def f():
    do_something()
```

The @property Decorator




```
class Example:

    @property
    def p(self):
        return self._p

    @p.setter
    def p(self, value):
        self._p = value
```


shipping
shipping.py
Project
shipping.py
shipping
iso6346.py
shipping.py
External Libraries
Scratches and Consoles
46
47 @property
48 def celsius(self):
49 return self._celsius
50
51 @celsius.setter
52 def celsius(self, value):
53 if value > RefrigeratedShippingContainer.MAX_CELSIUS:
54 raise ValueError("Temperature too hot!")
55 self._celsius = value
RefrigeratedShippingContainer
Python Console
-19.0
>>> r5.celsius = +5.0
Traceback (most recent call last):
File "<input>", line 1, in <module>
File "/var/folders/bb/w8tlddfn2fz1lhtlj_rxp_000000gn/T/tmpf7_msa50/build/shipping/shipping.py", line 54, in celsius
raise ValueError("Temperature too hot!")
ValueError: Temperature too hot!
>>>
Replay server listening on port 14415: You just opened shipping (a minute ago)
1:1
UTF-8
4 spaces
Python 3.8 (shipping)


C
F
50
40
30
20
10
0
10
20
30
40
120
100
80
60
40
20
0
20
40


shipping
shipping.py
Project
shipping
iso6346.py
shipping.py
External Libraries
Scratches and Consoles
class RefrigeratedShippingContainer(ShippingContainer):
MAX_CELSIUS = 4.0
def __init__(self, owner_code, contents, *, celsius, **kwargs):
super().__init__(owner_code, contents, **kwargs)
self.celsius = celsius
@staticmethod
def _c_to_f(celsius):
RefrigeratedShippingContainer
Python Console
File "/var/folders/bb/w8tlddfn2fz1lhtlj_rxp_000000gn/T/tmp1zoz7572/build/shipping/shipping.py", line 22, in create_empt
return cls(owner_code, contents=[], **kwargs)
File "/var/folders/bb/w8tlddfn2fz1lhtlj_rxp_000000gn/T/tmp1zoz7572/build/shipping/shipping.py", line 43, in __init__
self.celsius = celsius
File "/var/folders/bb/w8tlddfn2fz1lhtlj_rxp_000000gn/T/tmp1zoz7572/build/shipping/shipping.py", line 60, in celsius
raise ValueError("Temperature too hot!")
ValueError: Temperature too hot!
>>>
Replay server listening on port 14415: You just opened shipping (3 minutes ago)
1:1
UTF-8
4 spaces
Python 3.8 (shipping)

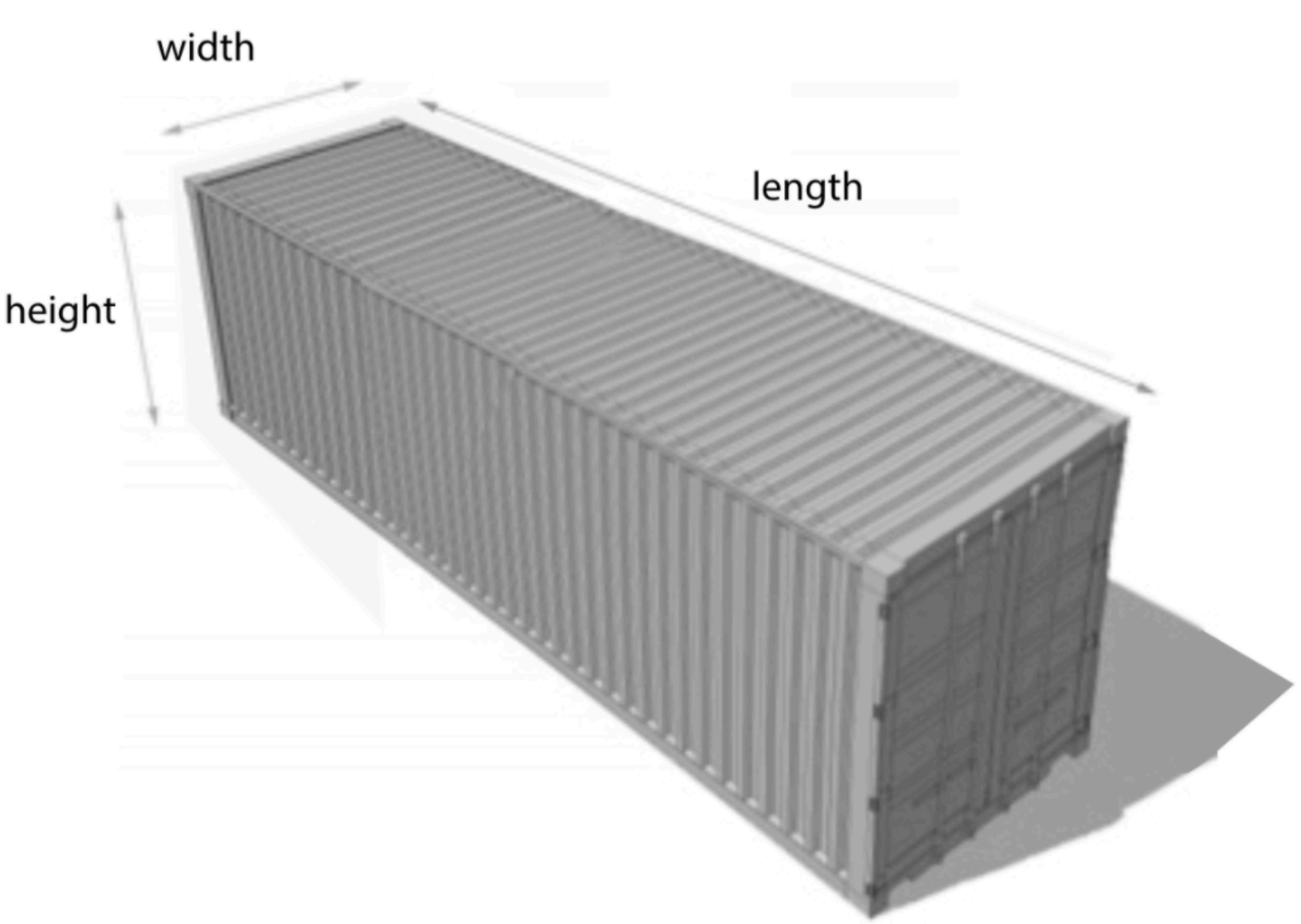
Weak Encapsulation

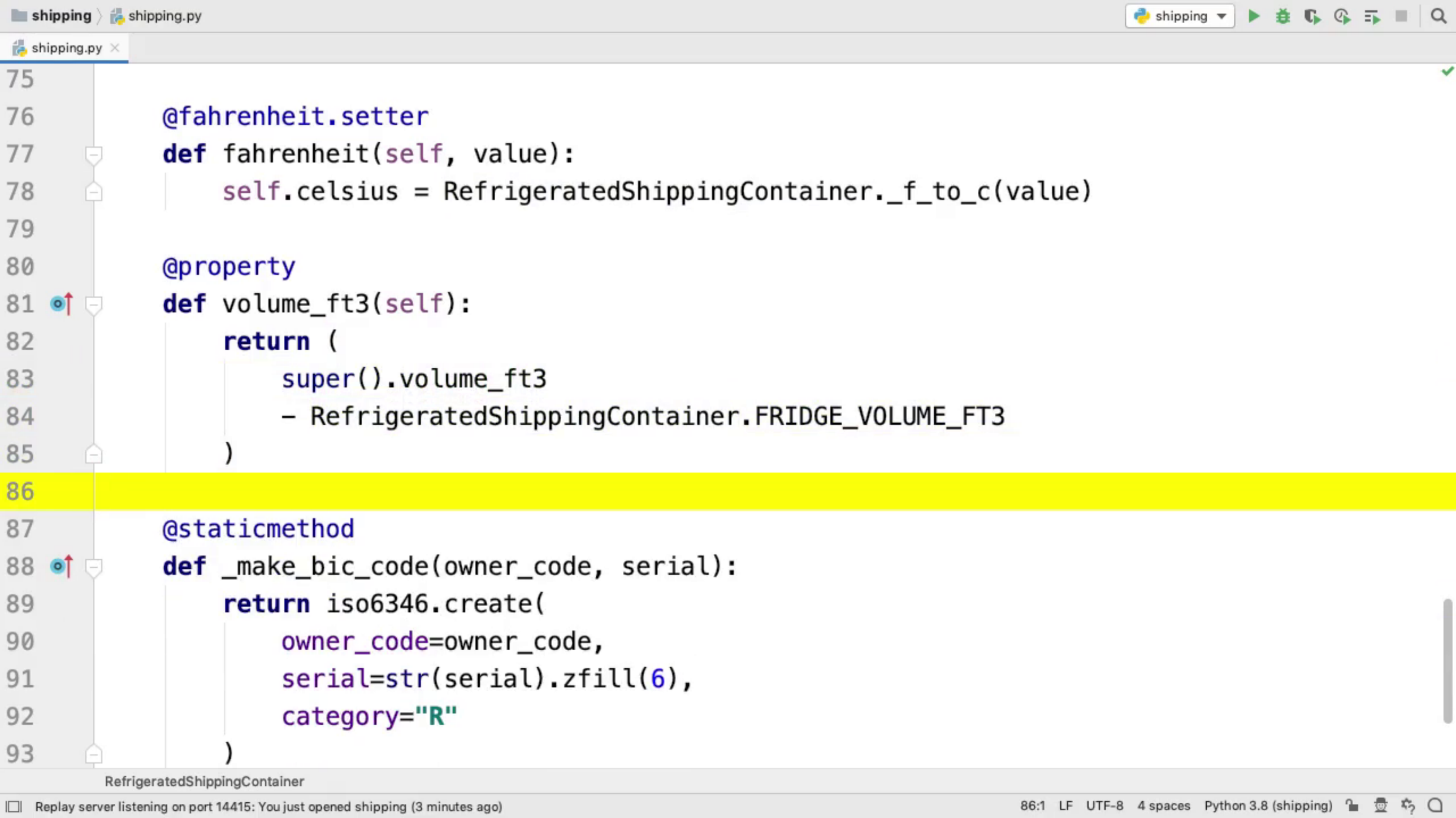
Too many properties can lead to
excessive coupling

Tell! Don't ask.

Tell other objects what to do instead of asking them their state and responding to it.

Properties and Inheritance


width
length
height



```
75
76      @fahrenheit.setter
77      def fahrenheit(self, value):
78          self.celsius = RefrigeratedShippingContainer._f_to_c(value)
79
80      @property
81      def volume_ft3(self):
82          return (
83              super().volume_ft3
84              - RefrigeratedShippingContainer.FRIDGE_VOLUME_FT3
85          )
86
87      @staticmethod
88      def _make_bic_code(owner_code, serial):
89          return iso6346.create(
90              owner_code=owner_code,
91              serial=str(serial).zfill(6),
92              category="R"
93          )
```

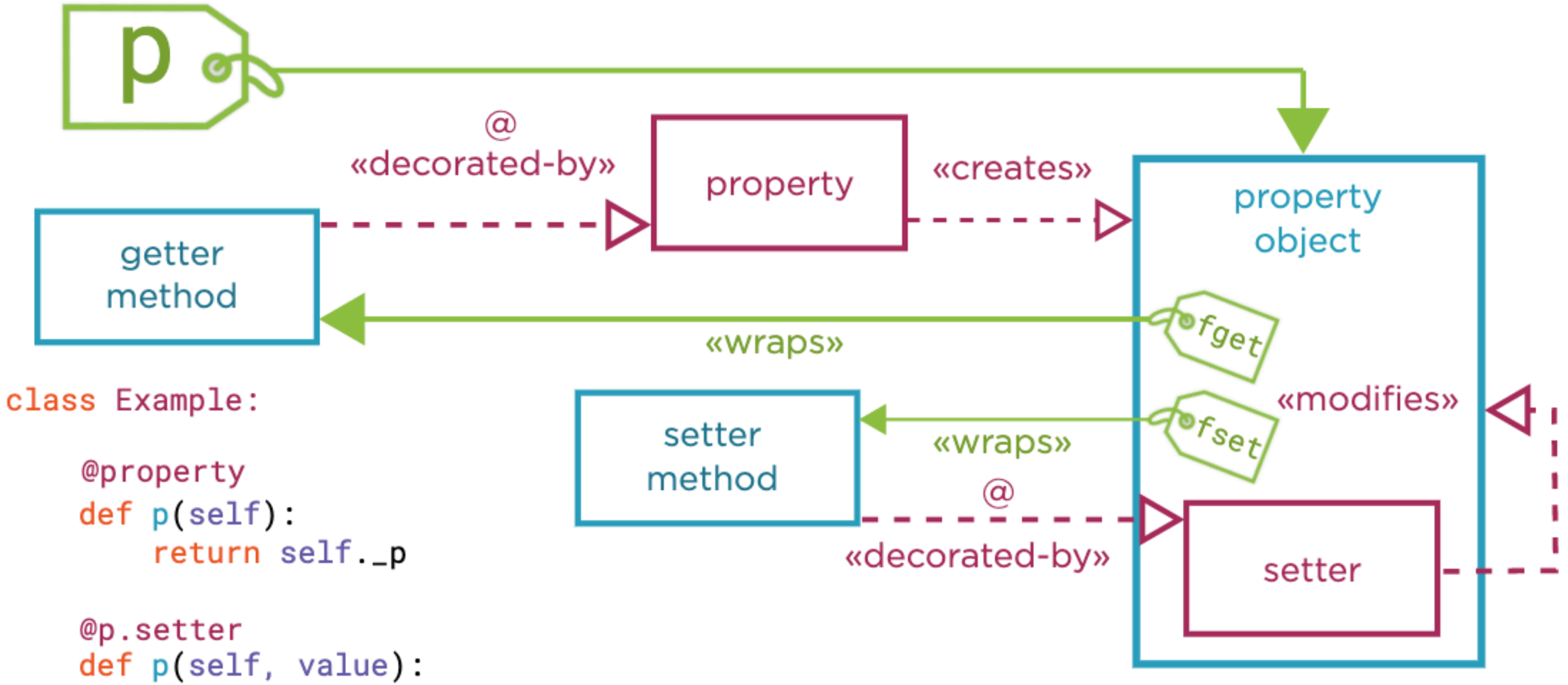
To override a property getter, redefine in a derived class.


Carrier
THINLINE

shipping.py

```
92              category="R"
93          )
94
95
96  class HeatedRefrigeratedShippingContainer(RefrigeratedShippingContainer):
97
98      MIN_CELSIUS = -20
99
100     @RefrigeratedShippingContainer.celsius.setter
101     def celsius(self, value):
102         if value < HeatedRefrigeratedShippingContainer.MIN_CELSIUS:
103             raise ValueError("Temperature too cold!")
104         RefrigeratedShippingContainer.celsius.fset(self, value)
105
```

The @property Decorator



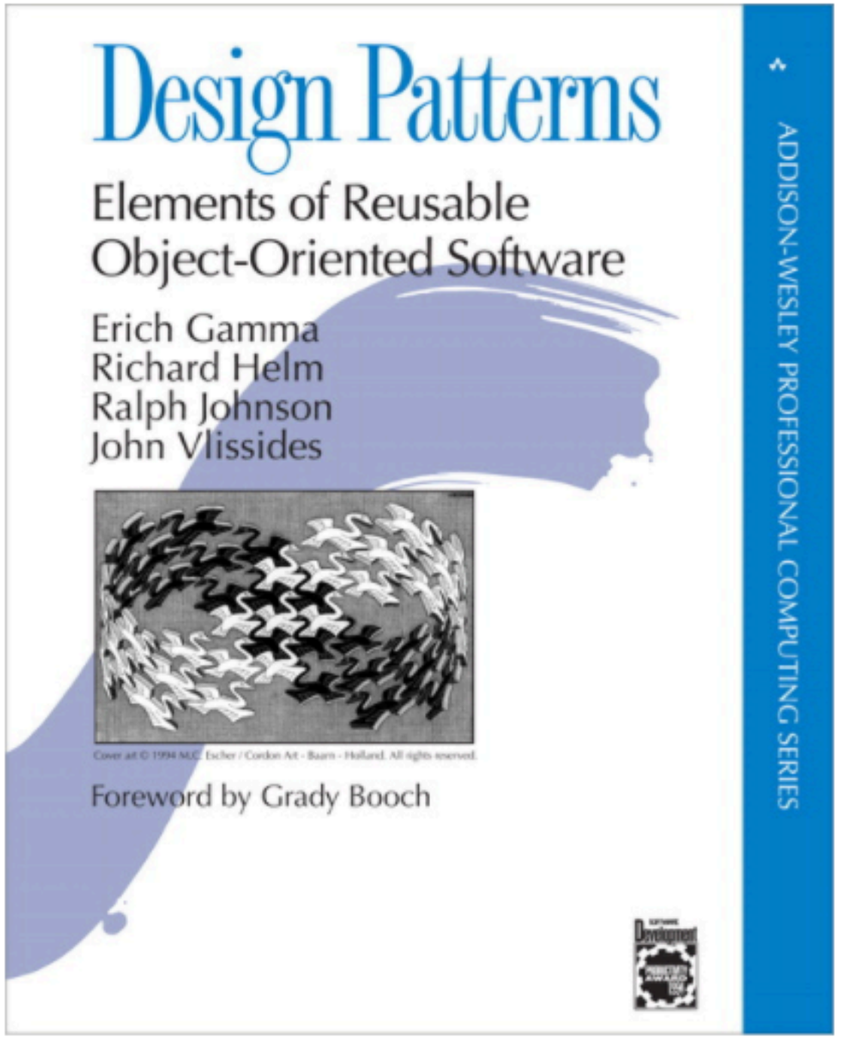

```
class Example:

    @property
    def p(self):
        return self._p

    @p.setter
    def p(self, value):
        self._p = value
```

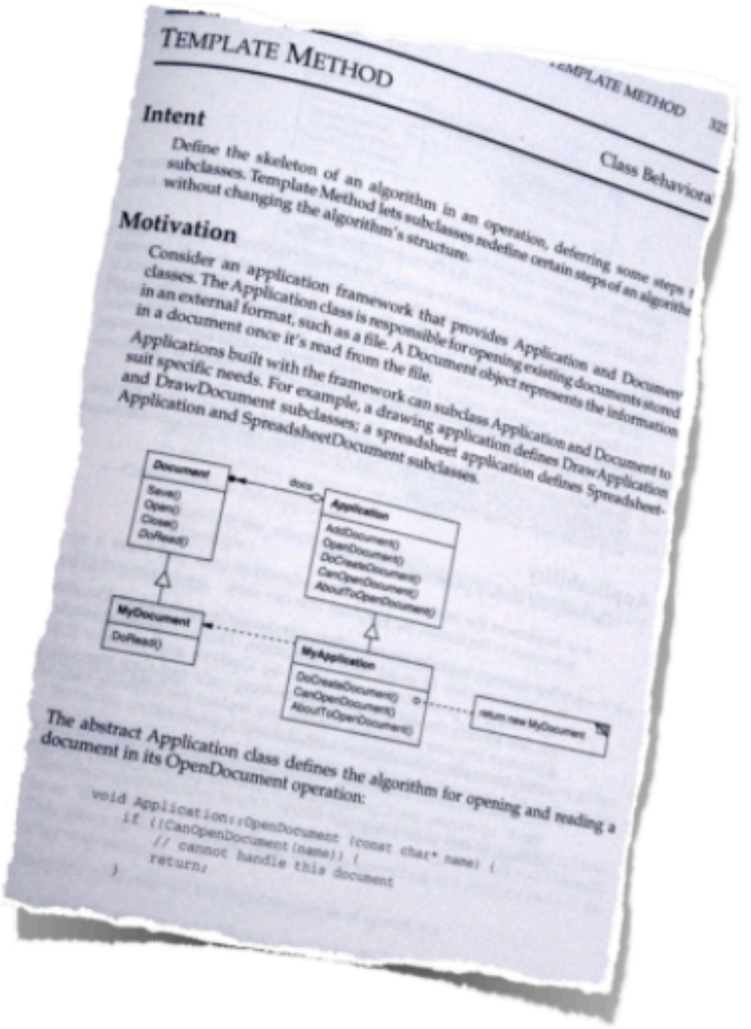
Overriding Properties with the Template Method

Overriding @property Setters Gets Messy

```
class HeatedRefrigeratedShippingContainer(RefrigeratedShippingContainer):

    MIN_CELSIUS = -20

    @RefrigeratedShippingContainer.celsius.setter
    def celsius(self, value):
        if value < HeatedRefrigeratedShippingContainer.MIN_CELSIUS:
            raise ValueError("Temperature too cold!")
        RefrigeratedShippingContainer.celsius.fset(self, value)
```

The Template Method Design Pattern



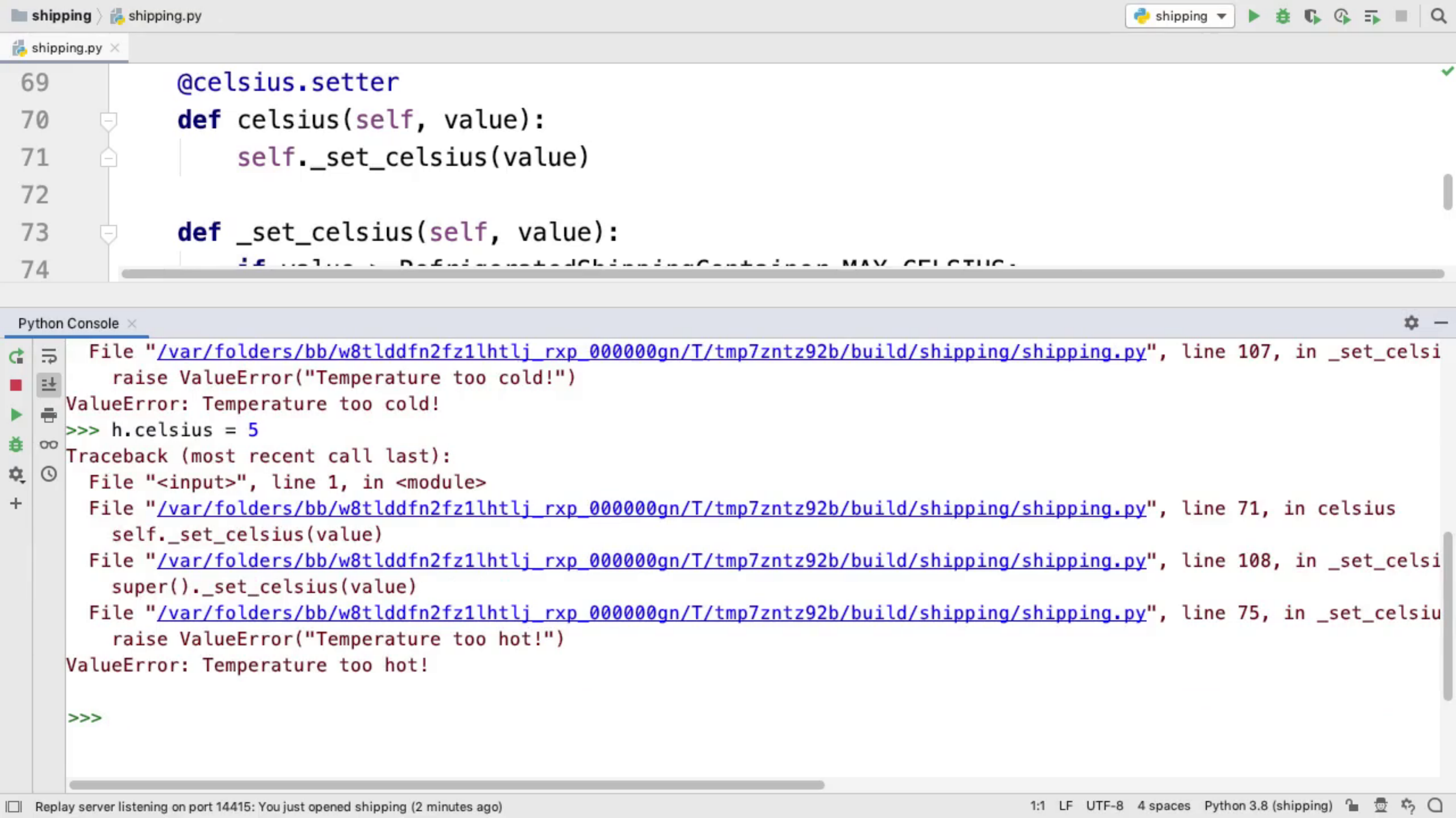


```python
class AbstractClass:

    def template_method(self):
        self._part1()  # No implementation
        self._part2()  # Abstract implementation
        self._part3()  # Default implementation

    def _part2(self):
        raise NotImplementedError("Override this method")

    def _part3(self):
        print("Done!") # Optionally override


class ConcreteClass(AbstractClass):

    def _part1(self):
        print("About to perform action")

    def _part2(self):
        perform_action()

    def _part3(self):
        print("Action performed!")
```

- Operation defined in terms of as yet undefined steps.
- Abstract sub-operations must be overridden
- Or may have default implementations, which may be overridden
- The concrete class fills in the details by overriding methods from the abstract class.

"All problems in computer science can be solved by another level of indirection"

David Wheeler


shipping
shipping.py
shipping
shipping.py
69 @celsius.setter
70 def celsius(self, value):
71 self._set_celsius(value)
72
73 def _set_celsius(self, value):
74
Python Console
File "/var/folders/bb/w8tlddfn2fz1lhtlj_rxp_000000gn/T/tmp7zntz92b/build/shipping/shipping.py", line 107, in _set_celsi
raise ValueError("Temperature too cold!")
ValueError: Temperature too cold!
>>> h.celsius = 5
Traceback (most recent call last):
File "<input>", line 1, in <module>
File "/var/folders/bb/w8tlddfn2fz1lhtlj_rxp_000000gn/T/tmp7zntz92b/build/shipping/shipping.py", line 71, in celsius
self._set_celsius(value)
File "/var/folders/bb/w8tlddfn2fz1lhtlj_rxp_000000gn/T/tmp7zntz92b/build/shipping/shipping.py", line 108, in _set_celsi
super()._set_celsius(value)
File "/var/folders/bb/w8tlddfn2fz1lhtlj_rxp_000000gn/T/tmp7zntz92b/build/shipping/shipping.py", line 75, in _set_celsiu
raise ValueError("Temperature too hot!")
ValueError: Temperature too hot!
>>>
Replay server listening on port 14415: You just opened shipping (2 minutes ago)
1:1 LF UTF-8 4 spaces Python 3.8 (shipping)

Don't override properties directly. Delegate to regular methods and override those instead.

"All problems in computer science can be solved by another level of indirection ... except for the problem of too many levels of indirection"

Kevlin Henney

Summary

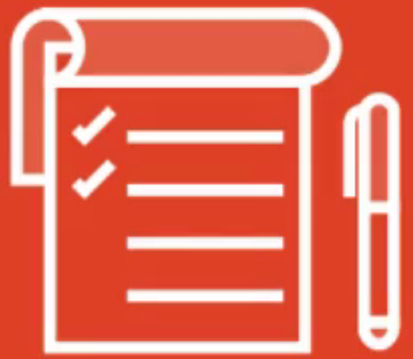
Class attributes versus instance attributes

Class attributes are shared between instances

Navigate to class attributes via the class

Assigning to self always creates an instance attribute

Summary



Use `@staticmethod` for methods which need neither the class nor the instance

Use `@classmethod` for methods which needs the class but not the instance

Use `@classmethod` for the named-constructor idiom

Static- and class-methods can be overridden

Summary

Static- and class-methods are polymorphic when invoked through `self`

Use the `@property` decorator instead of getters and setters

Easily override properties by delegating to regular methods