The Collection Abstract Base Classes

Robert Smallshire
COFOUNDER - SIXTY NORTH

@robsmallshire

Austin Bingham
COFOUNDER - SIXTY NORTH

@austin_bingham

heapq — Heap queue algorithm

This Page

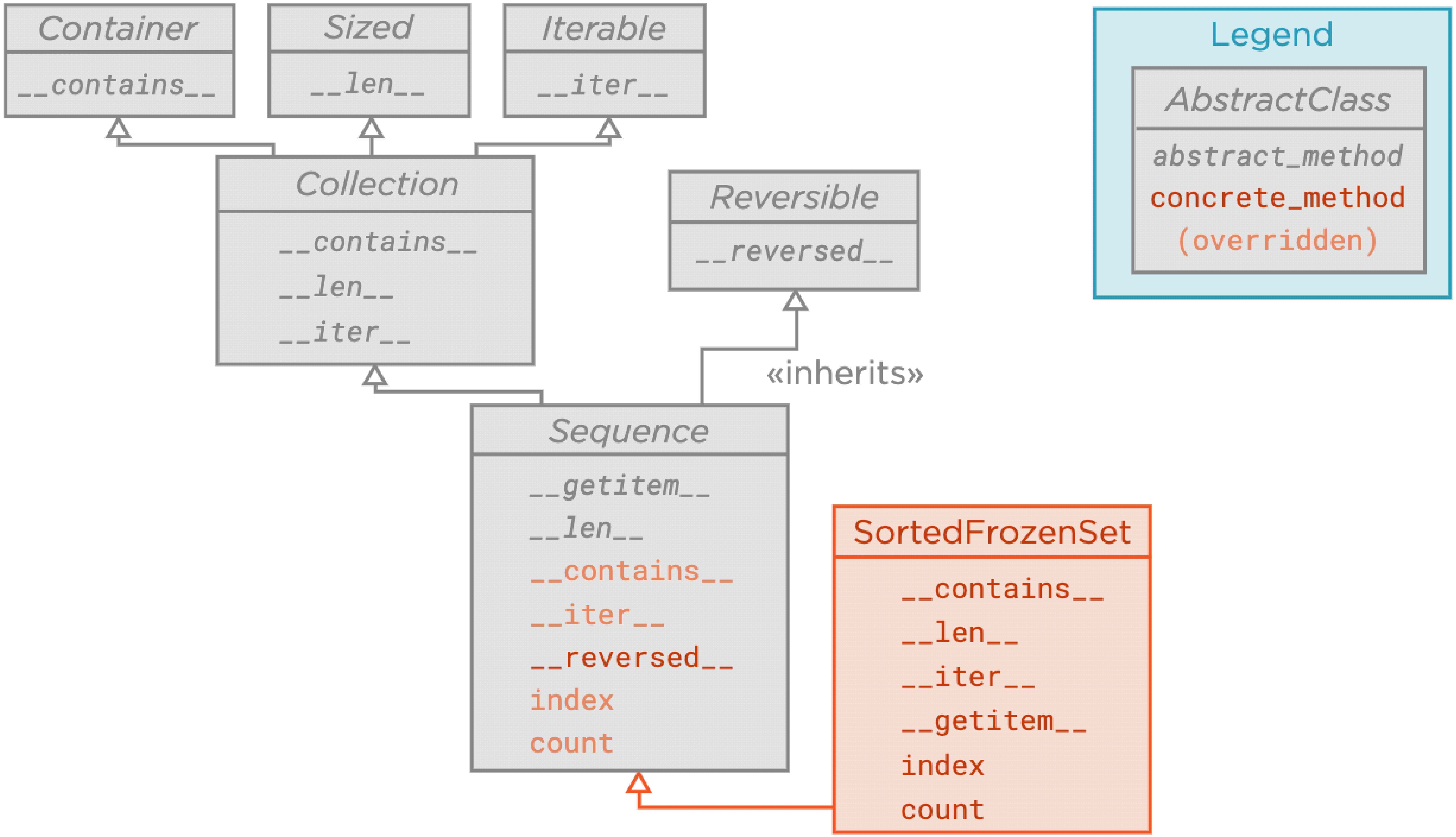
Report a Bug
Show Source

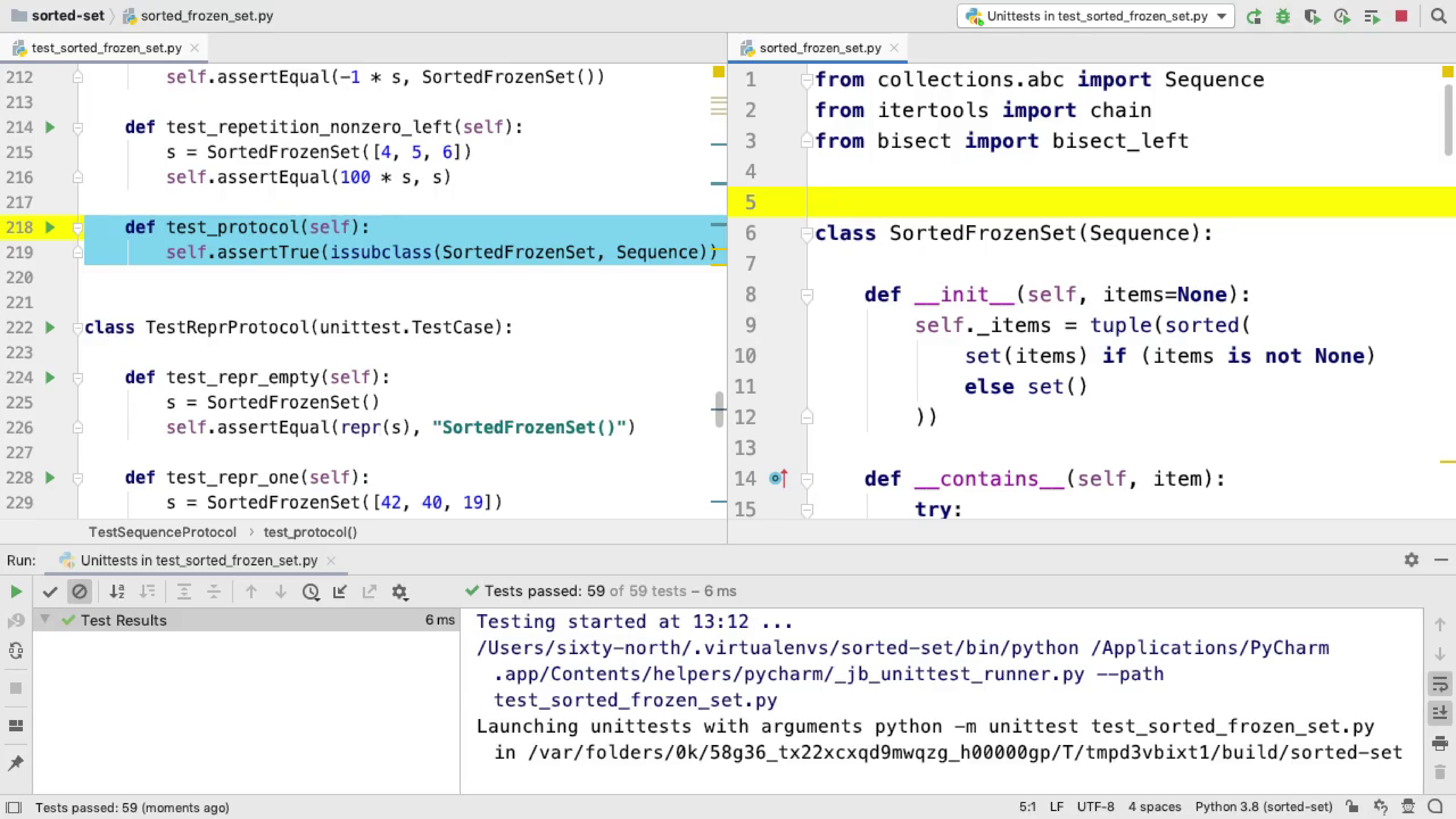
«

Collections Abstract Base Classes

The collections module offers the following ABCs:

ABC	Inherits from	Abstract Methods	Mixin Methods
`Container`		`__contains__`	
`Hashable`		`__hash__`	
`Iterable`		`__iter__`	
`Iterator`	`Iterable`	`__next__`	`__iter__`
`Reversible`	`Iterable`	`__reversed__`	
`Generator`	`Iterator`	`send`, `throw`	`close`, `__iter__`, `__next__`
`Sized`		`__len__`	
`Callable`		`__call__`	
`Collection`	`Sized`, `Iterable`, `Container`	`__contains__`, `__iter__`, `__len__`	
`Sequence`	`Reversible`, `Collection`	`__getitem__`, `__len__`	`__contains__`, `__iter__`, `__reversed__`, `index`, and `count`
`MutableSequence`	`Sequence`	`__getitem__`, `__setitem__`, `__delitem__`, `__len__`, `insert`	Inherited `Sequence` methods and `append`, `reverse`, `extend`, `pop`, `remove`, and `__iadd__`
`ByteString`	`Sequence`	`__getitem__`, `__len__`	Inherited `Sequence` methods
		`__contains__`,	`__le__`, `__lt__`, `__eq__`, `__ne__`,


Container
__contains__
Sized
__len__
Iterable
__iter__
Collection
__contains__
__len__
__iter__
Reversible
__reversed__
«inherits»
Sequence
__getitem__
__len__
__contains__
__iter__
__reversed__
index
count
SortedFrozenSet
__contains__
__len__
__iter__
__getitem__
index
count
Legend
AbstractClass
abstract_method
concrete_method
(overridden)


sorted-set › sorted_frozen_set.py
Unittests in test_sorted_frozen_set.py
test_sorted_frozen_set.py
sorted_frozen_set.py

```python
        self.assertEqual(-1 * s, SortedFrozenSet())

    def test_repetition_nonzero_left(self):
        s = SortedFrozenSet([4, 5, 6])
        self.assertEqual(100 * s, s)

    def test_protocol(self):
        self.assertTrue(issubclass(SortedFrozenSet, Sequence))


class TestReprProtocol(unittest.TestCase):

    def test_repr_empty(self):
        s = SortedFrozenSet()
        self.assertEqual(repr(s), "SortedFrozenSet()")

    def test_repr_one(self):
        s = SortedFrozenSet([42, 40, 19])
```

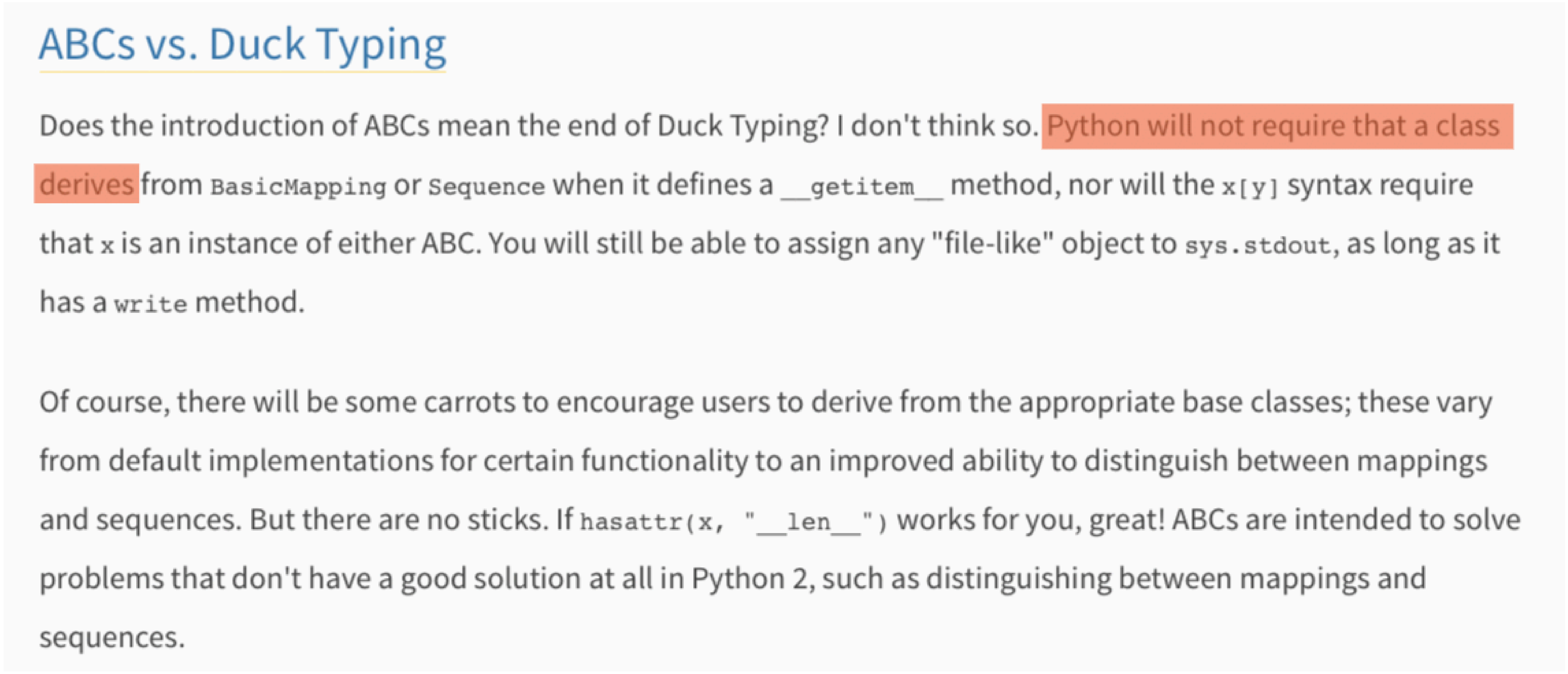
```python
from collections.abc import Sequence
from itertools import chain
from bisect import bisect_left


class SortedFrozenSet(Sequence):

    def __init__(self, items=None):
        self._items = tuple(sorted(
            set(items) if (items is not None)
            else set()
        ))

    def __contains__(self, item):
        try:
```

TestSequenceProtocol › test_protocol()
Run: Unittests in test_sorted_frozen_set.py
Tests passed: 59 of 59 tests – 6 ms
Test Results 6 ms

```
Testing started at 13:12 ...
/Users/sixty-north/.virtualenvs/sorted-set/bin/python /Applications/PyCharm
  .app/Contents/helpers/pycharm/_jb_unittest_runner.py --path
  test_sorted_frozen_set.py
Launching unittests with arguments python -m unittest test_sorted_frozen_set.py
  in /var/folders/0k/58g36_tx22xcxqd9mwqzg_h00000gp/T/tmpd3vbixt1/build/sorted-set
```

Tests passed: 59 (moments ago)
5:1 LF UTF-8 4 spaces Python 3.8 (sorted-set)

Excerpt from PEP 3119



ABCs vs. Duck Typing

Does the introduction of ABCs mean the end of Duck Typing? I don't think so. Python will not require that a class derives from `BasicMapping` or `Sequence` when it defines a `__getitem__` method, nor will the `x[y]` syntax require that `x` is an instance of either ABC. You will still be able to assign any "file-like" object to `sys.stdout`, as long as it has a `write` method.

Of course, there will be some carrots to encourage users to derive from the appropriate base classes; these vary from default implementations for certain functionality to an improved ability to distinguish between mappings and sequences. But there are no sticks. If `hasattr(x, "__len__")` works for you, great! ABCs are intended to solve problems that don't have a good solution at all in Python 2, such as distinguishing between mappings and sequences.

“Does the introduction of ABCs mean the end of Duck Typing?”

“I don’t think so.”

Guido van Rossum, PEP 3119