ThoughtWorks®

using & building

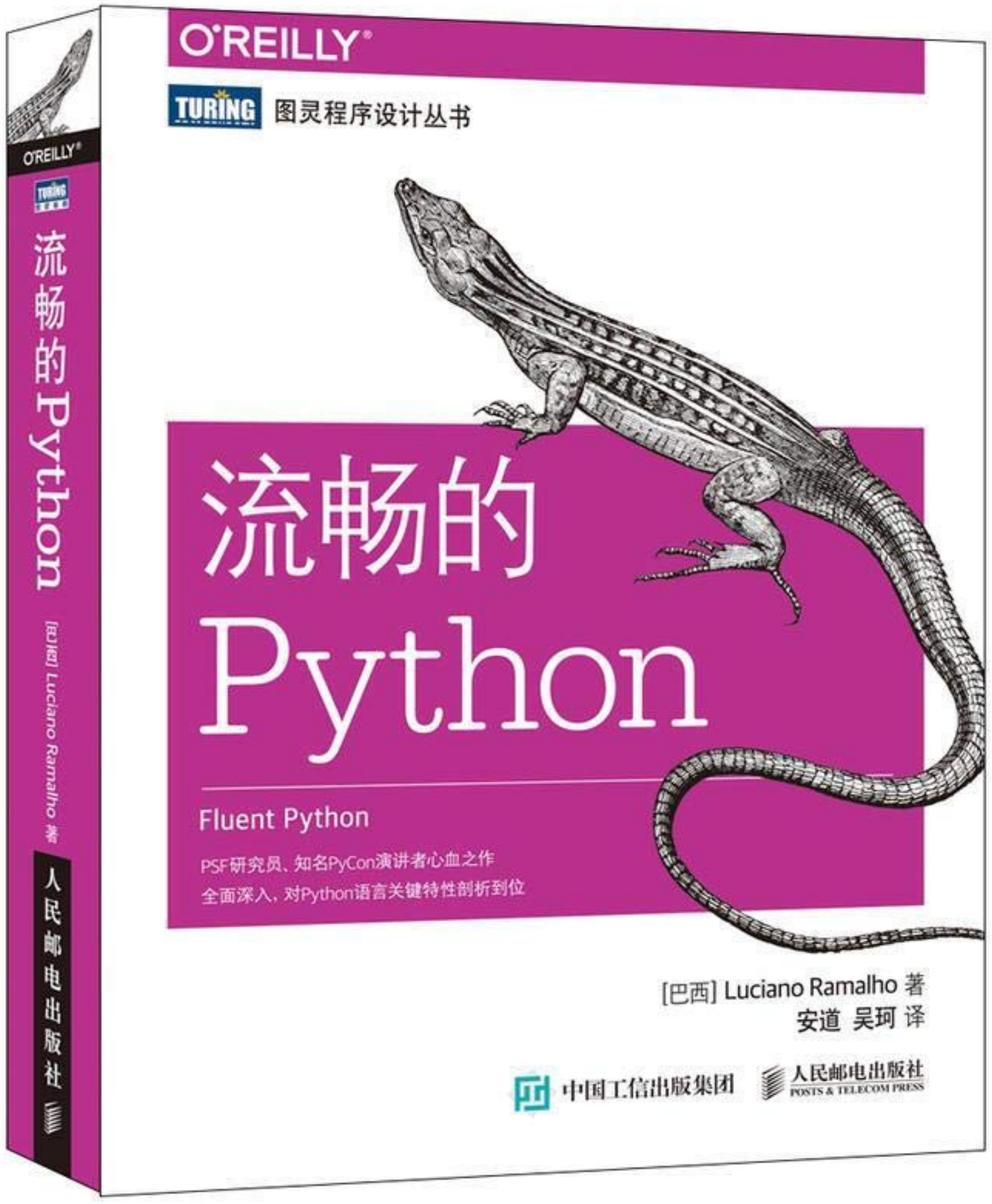
PYTHON SET PRACTICE

Learn great API design ideas from Python's set types.

PYCON CLEVELAND 2019

Luciano Ramalho
@standupdev

FLUENT PYTHON



Available in 9 languages:

- Chinese (simplified)
- Chinese (traditional)
- English
- French
- Russian
- Japanese
- Korean
- Polish
- Portuguese

2nd ed: I'm working on it!

AGENDA

Motivation

Overview of Python Sets

Learning from the set API

The __magic__ behind a set class

ThoughtWorks®

MOTIVATION

Some common use cases for sets

CASE STUDY #1

display product if all words in the query appear in the product description.

HAND-ROLLED SOLUTION #1

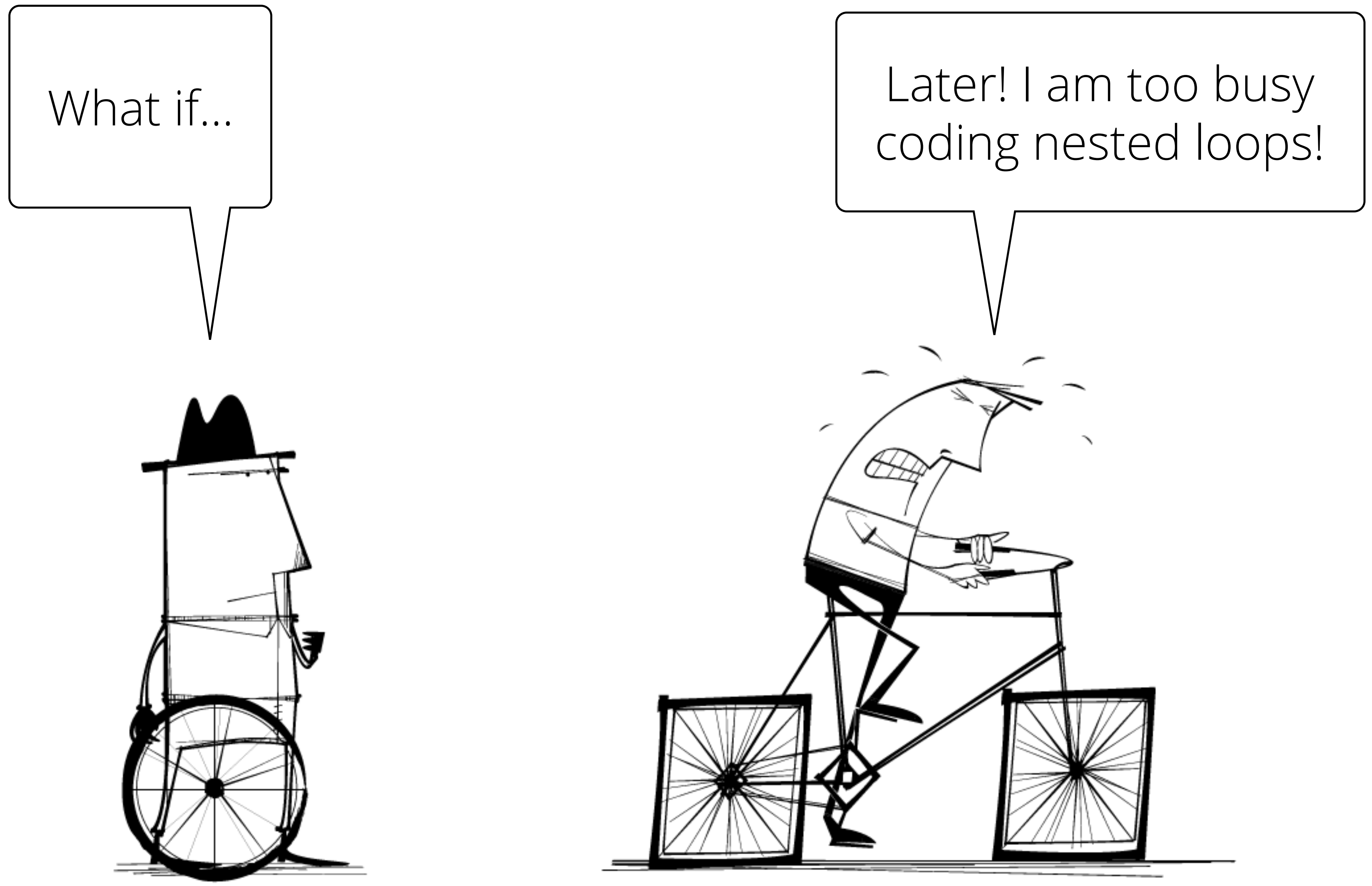
I've written code like this in Go, which lacks built-in sets:

```go
func ContainsAll(slice, subslice []string) bool {
    for _, needle := range subslice {
        found := false
        for _, elem := range slice {
            if needle == elem {
                found = true
                break
            }
        }
        if !found {
            return false
        }
    }
    return true
}
```

HAND-ROLLED SOLUTION #2

More readable, but still inefficient:

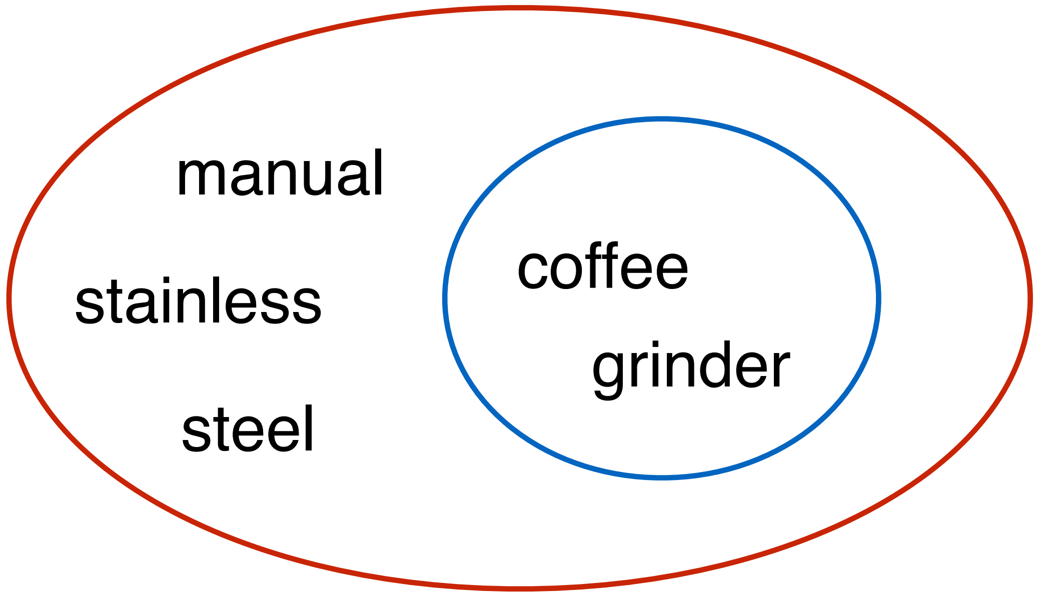
```go
func Contains(slice []string, needle string) bool {
    for _, elem := range slice {
        if needle == elem {
            return true
        }
    }
    return false
}

func ContainsAll(slice, subslice []string) bool {
    for _, needle := range subslice {
        if !Contains(slice, needle) {
            return false
        }
    }
    return true
}
```

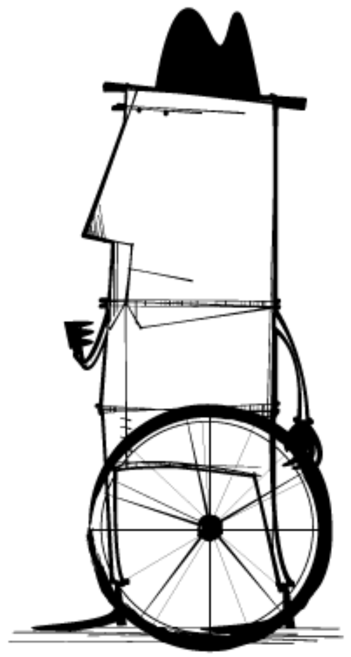

What if...
Later! I am too busy coding nested loops!

CASO DE USO #1

www.workcompass.com/

display product if all words in the query appear in the product description.

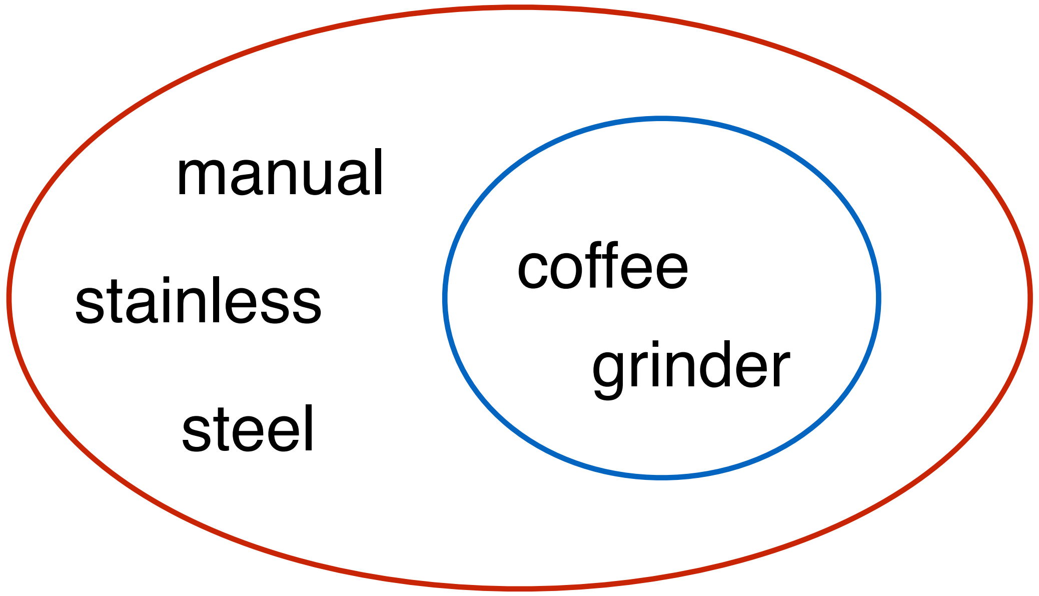


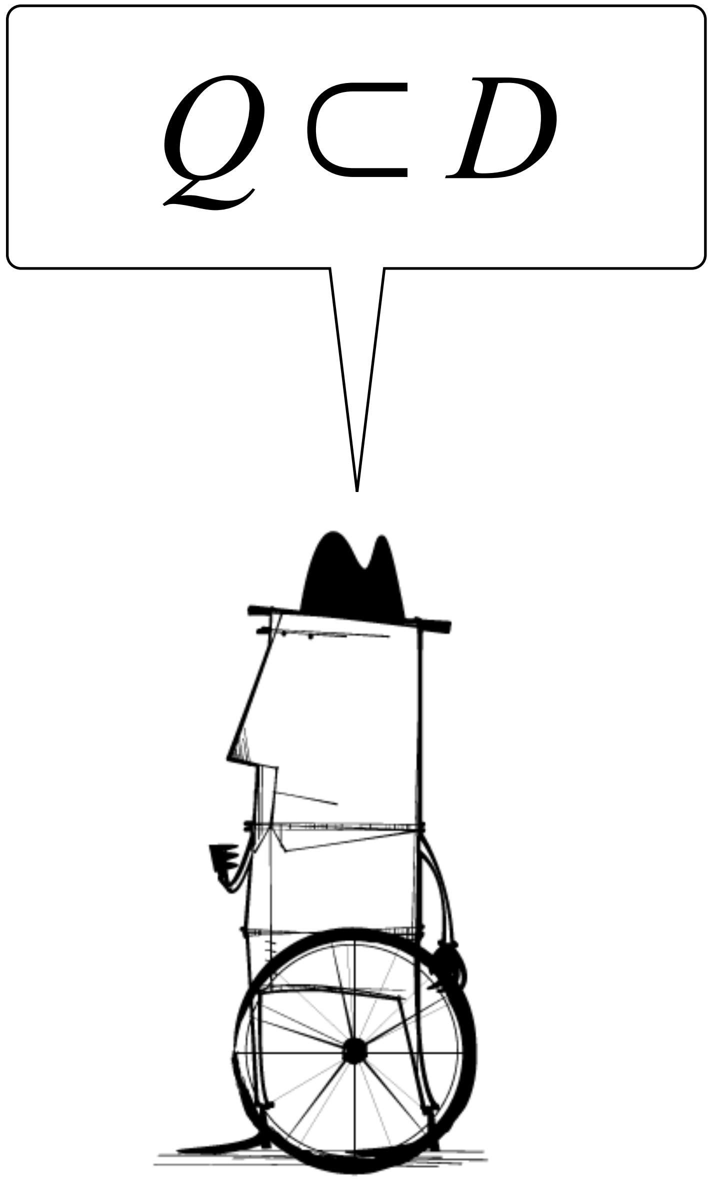


CASO DE USO #1

www.workcompass.com/

display product if all words in the query appear in the product description.



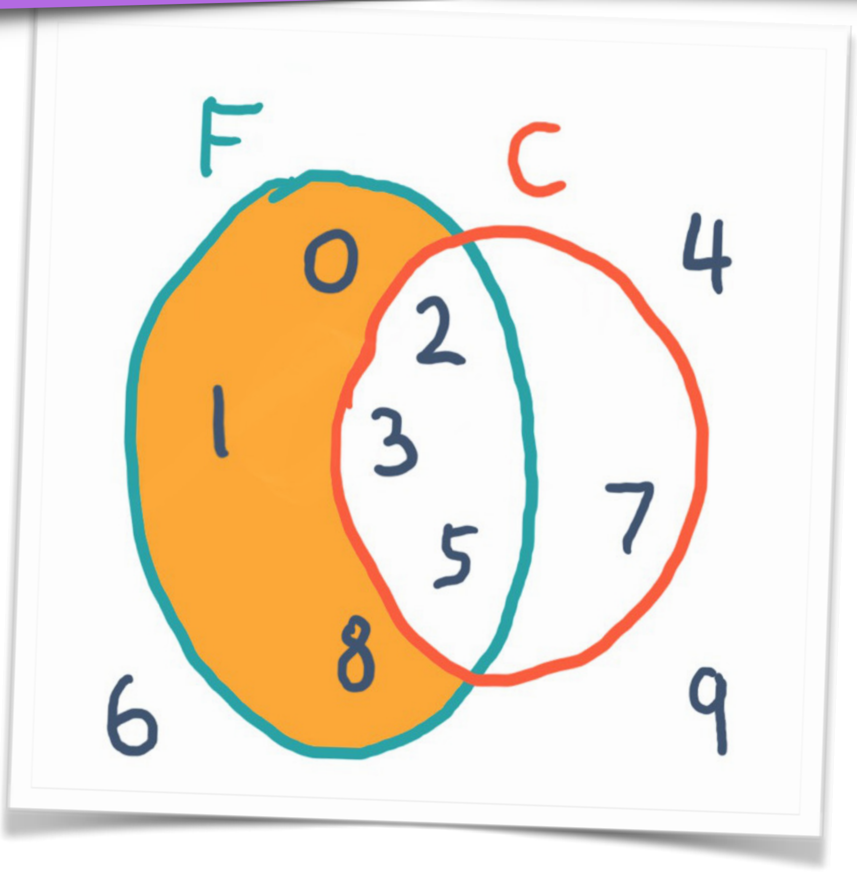


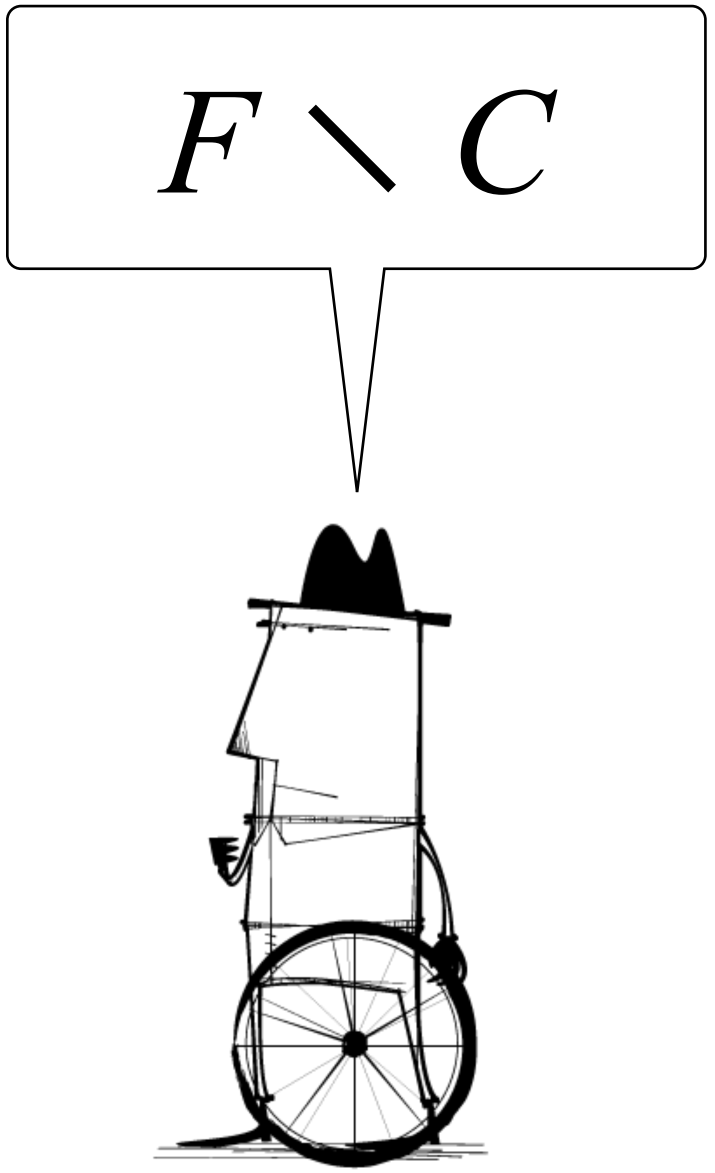
CASO DE USO #2

Mark all products previously favorited, except those already in the shopping cart.

CASO DE USO #2

Mark all products previously favorited, except those already in the shopping cart.





ThoughtWorks®

LOGIC AND SETS

A close relationship

Nobody has yet discovered a branch of mathematics that has successfully resisted formalization into set theory.

Thomas Forster

Logic Induction and Sets, p. 167

LOGIC CONJUNCTION IS INTERSECTION

x belongs to the intersection of A with B.

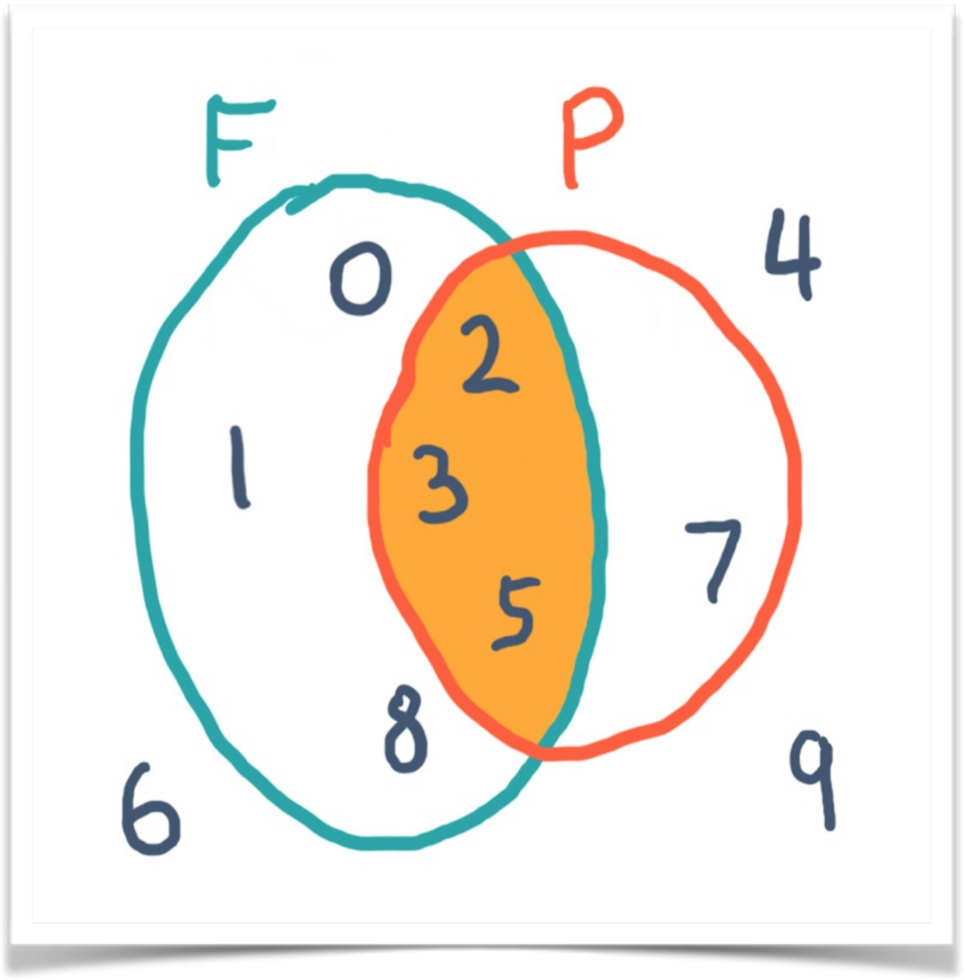
is the same as:

x belongs to A and
x also belongs to B.

Math notation:

$$x \in (A \cap B) \iff (x \in A) \wedge (x \in B)$$

In computing: **AND**



LOGIC DISJUNCTION: UNION

x belongs to the union of A and B.

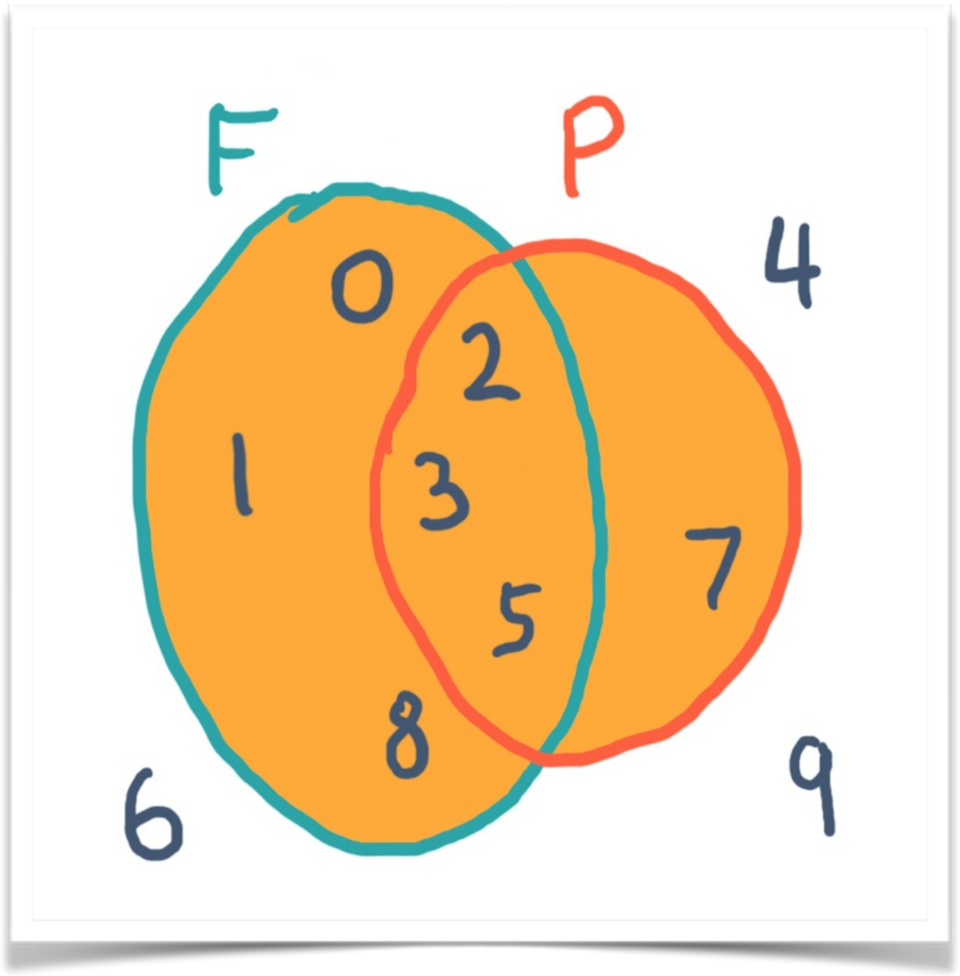
is the same as:

x belongs to A or
x belongs to B.

Math notation:

$$x \in (A \cup B) \iff (x \in A) \vee (x \in B)$$

In computing: **OR**

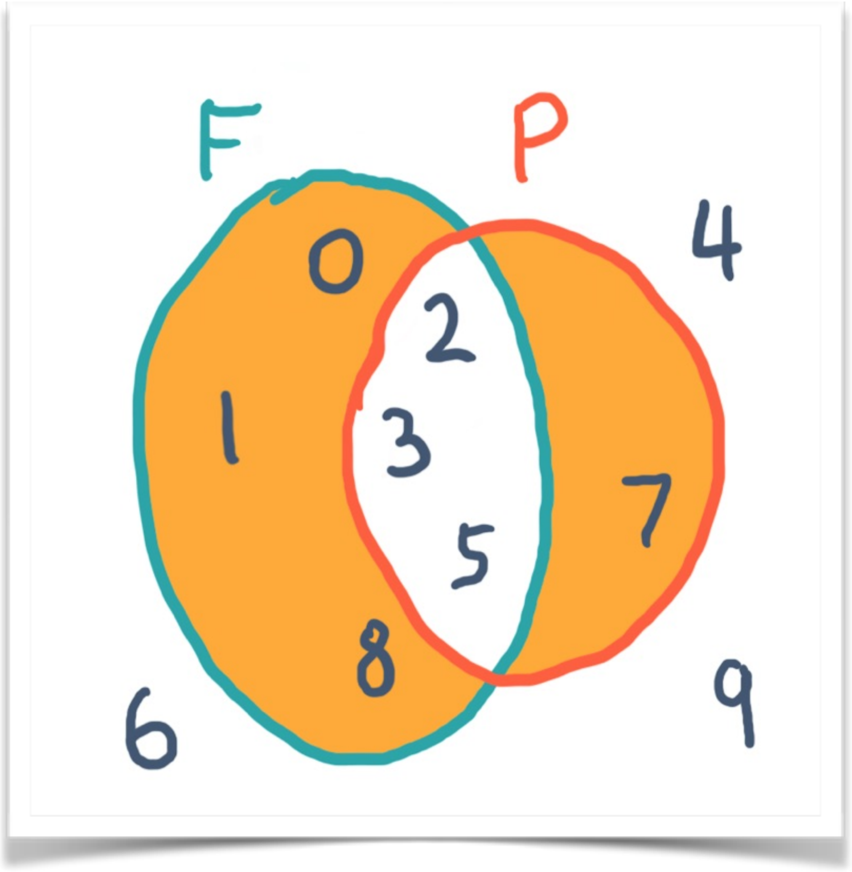


SYMMETRIC DIFFERENCE

x belongs to A or
x belongs to B but
does not belong to both

Is the same as:

x belongs to the union of A with B
less the intersection of A with B.



Math notation:

$$x \in (A \, \Delta \, B) \iff (x \in A) \veebar (x \in B)$$

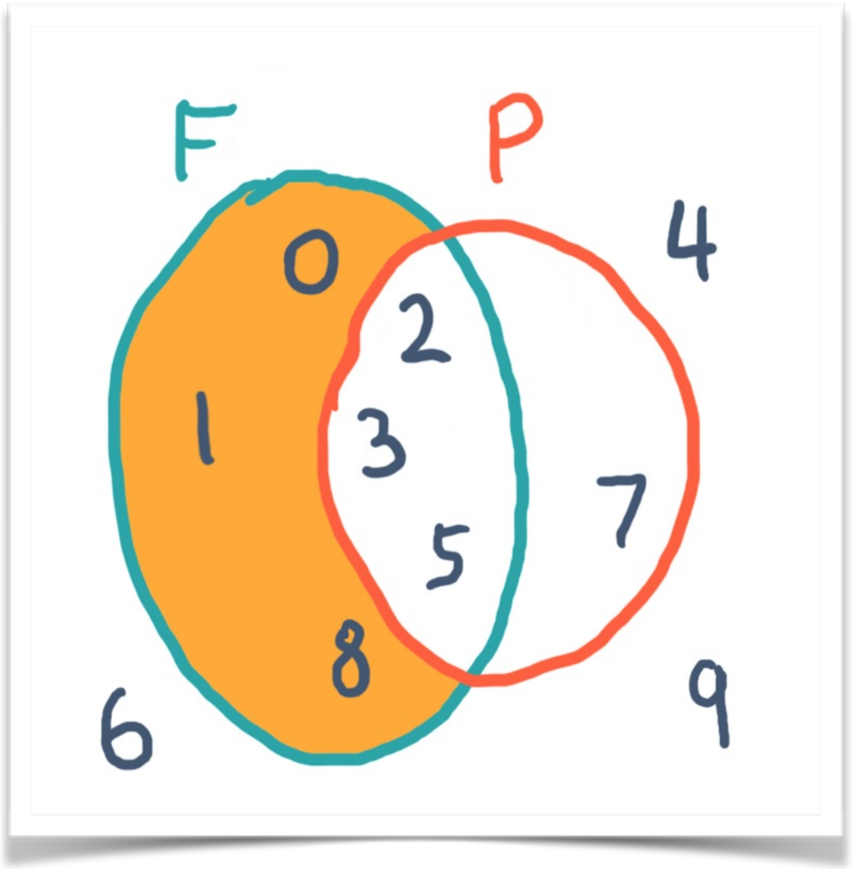
In computing: **XOR**

DIFFERENCE

x belongs to A but
does not belong to B.

is the same as:

elements of A minus elements of B



Math notation:

$$x \in (A \setminus B) \iff (x \in A) \land (x \notin B)$$

ThoughtWorks®

SETS IN SEVERAL LANGUAGES

SETS IN SEVERAL STANDARD LIBRARIES

Some languages/platform APIs that implement sets in their standard libraries

Java	Set interface: < 10 methods; 8 implementations
Python	set, frozenset: > 10 methods and operators
.Net (C# etc.)	ISet interface: > 10 methods; 2 implementations
JavaScript (ES6)	Set: < 10 methods
Ruby	Set: > 10 methods and operators

Python, .Net and Ruby offer rich set APIs

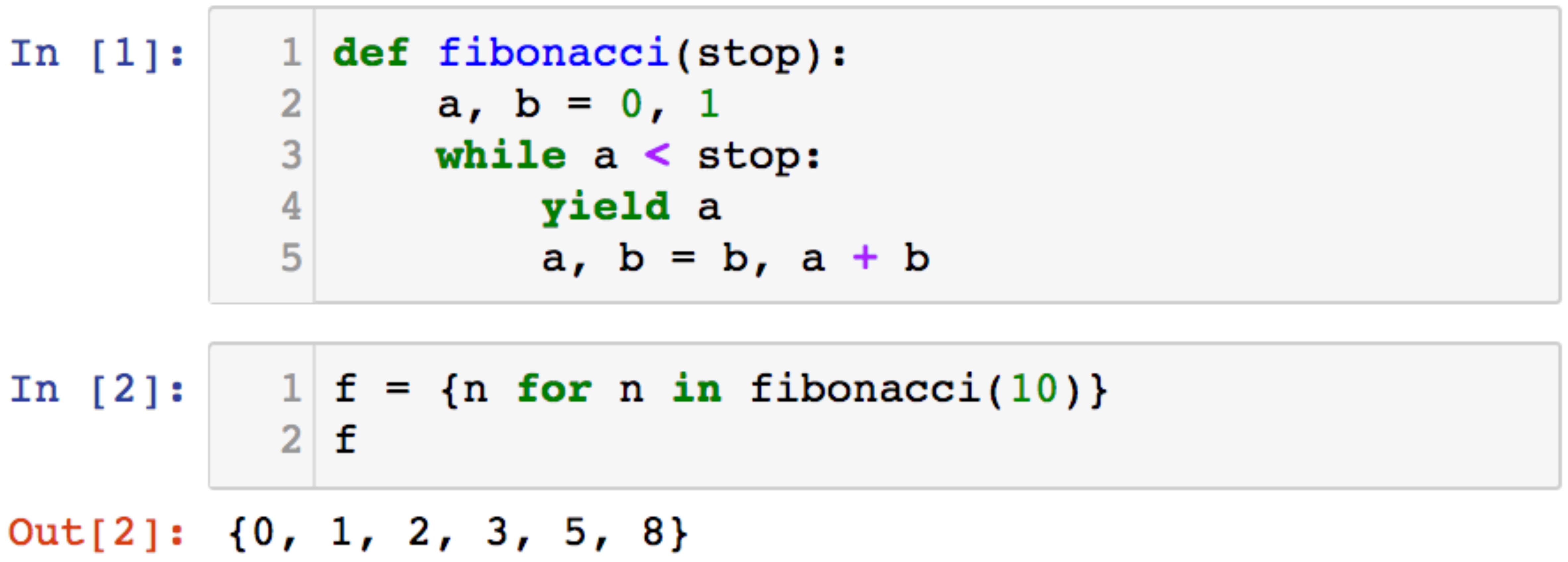
ThoughtWorks®

SETS IN PYTHON

The built-in types

BUILDING A SET FROM A SERIES OF NUMBERS

Using a set comprehension:

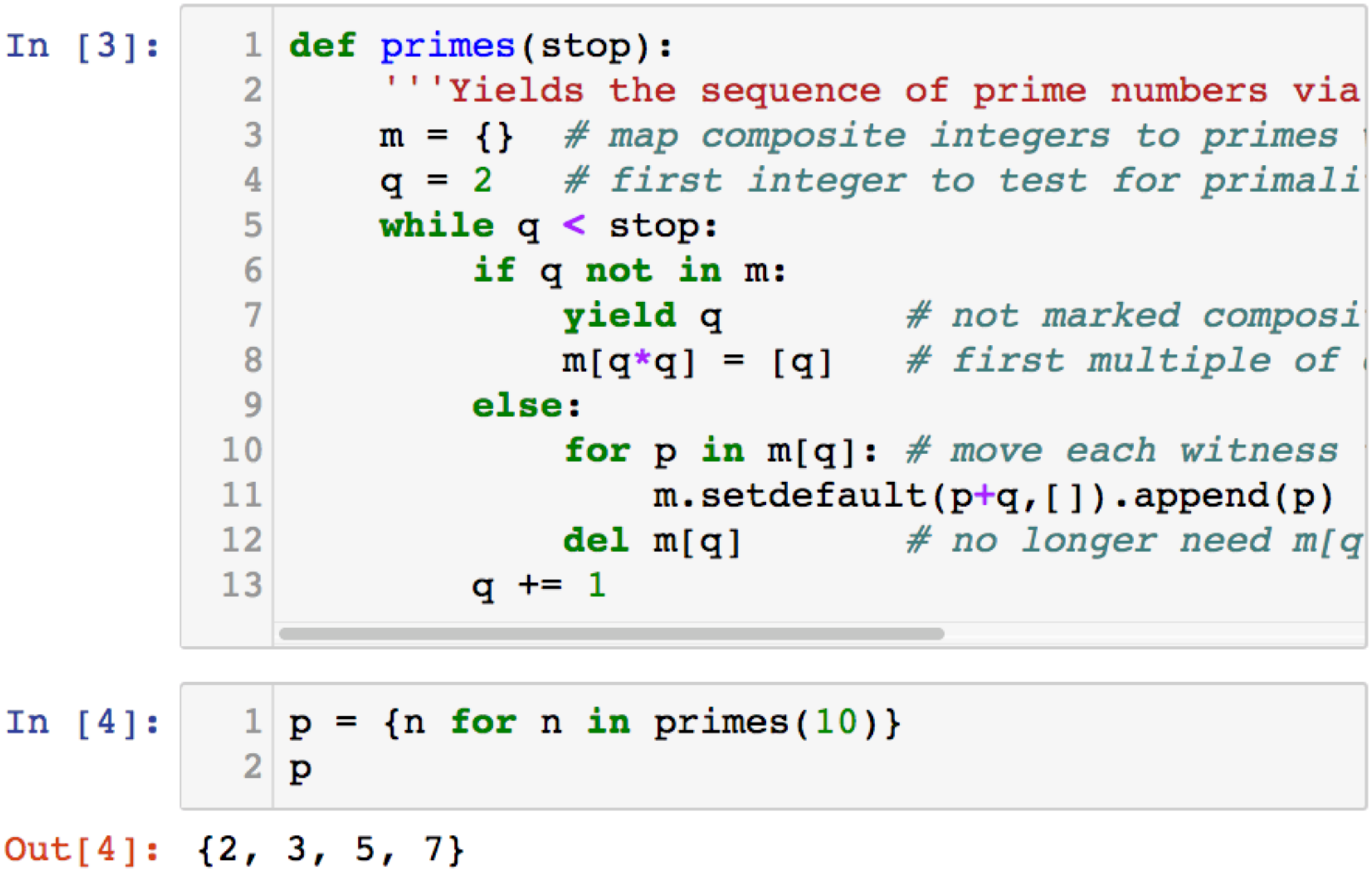


```
In [1]:
def fibonacci(stop):
    a, b = 0, 1
    while a < stop:
        yield a
        a, b = b, a + b
```

```
In [2]:
f = {n for n in fibonacci(10)}
f
```

```
Out[2]: {0, 1, 2, 3, 5, 8}
```

ANOTHER SET, FOR THE EXAMPLES



```
In [3]:
def primes(stop):
    '''Yields the sequence of prime numbers via
    m = {}   # map composite integers to primes
    q = 2    # first integer to test for primali
    while q < stop:
        if q not in m:
            yield q           # not marked composi
            m[q*q] = [q]      # first multiple of
        else:
            for p in m[q]: # move each witness
                m.setdefault(p+q,[]).append(p)
            del m[q]          # no longer need m[q
        q += 1
```

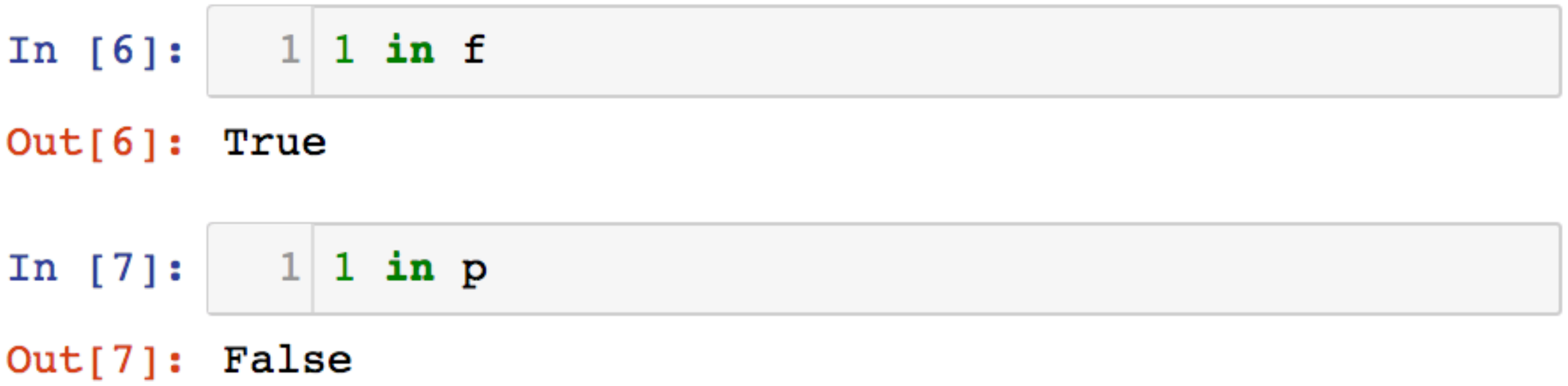
```
In [4]:
p = {n for n in primes(10)}
p
```

```
Out[4]: {2, 3, 5, 7}
```

ELEMENT CONTAINMENT: THE IN OPERATOR

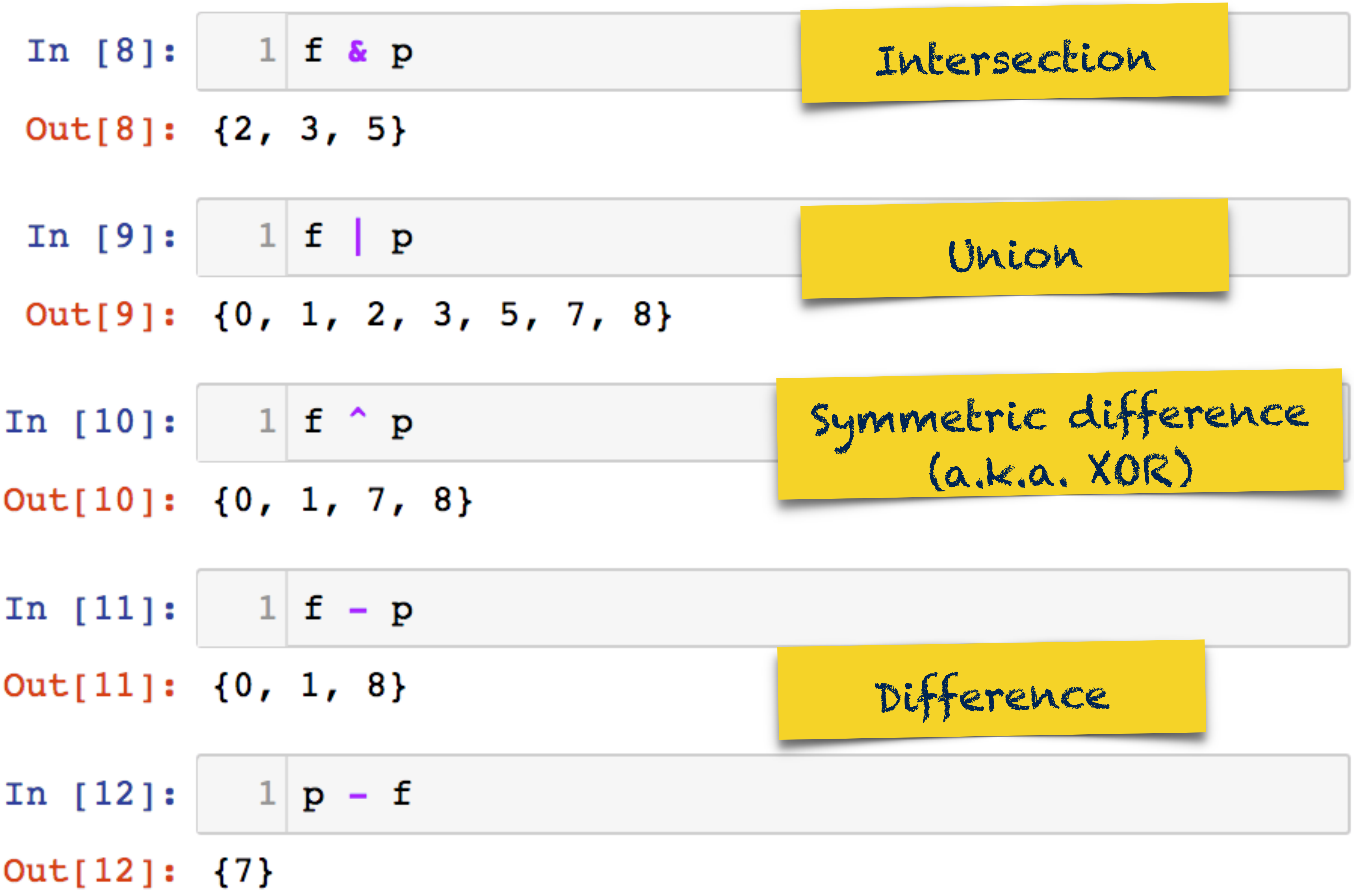
O(1) in sets, because they use a hash table to hold elements.

Implemented by the __contains__ special method:



```
In [6]: 1 in f
Out[6]: True

In [7]: 1 in p
Out[7]: False
```

FUNDAMENTAL SET OPERATIONS



```
In [8]:  f & p
Out[8]:  {2, 3, 5}
```

Intersection

```
In [9]:  f | p
Out[9]:  {0, 1, 2, 3, 5, 7, 8}
```

Union

```
In [10]:  f ^ p
Out[10]:  {0, 1, 7, 8}
```

Symmetric difference
(a.k.a. XOR)

```
In [11]:  f - p
Out[11]:  {0, 1, 8}
```

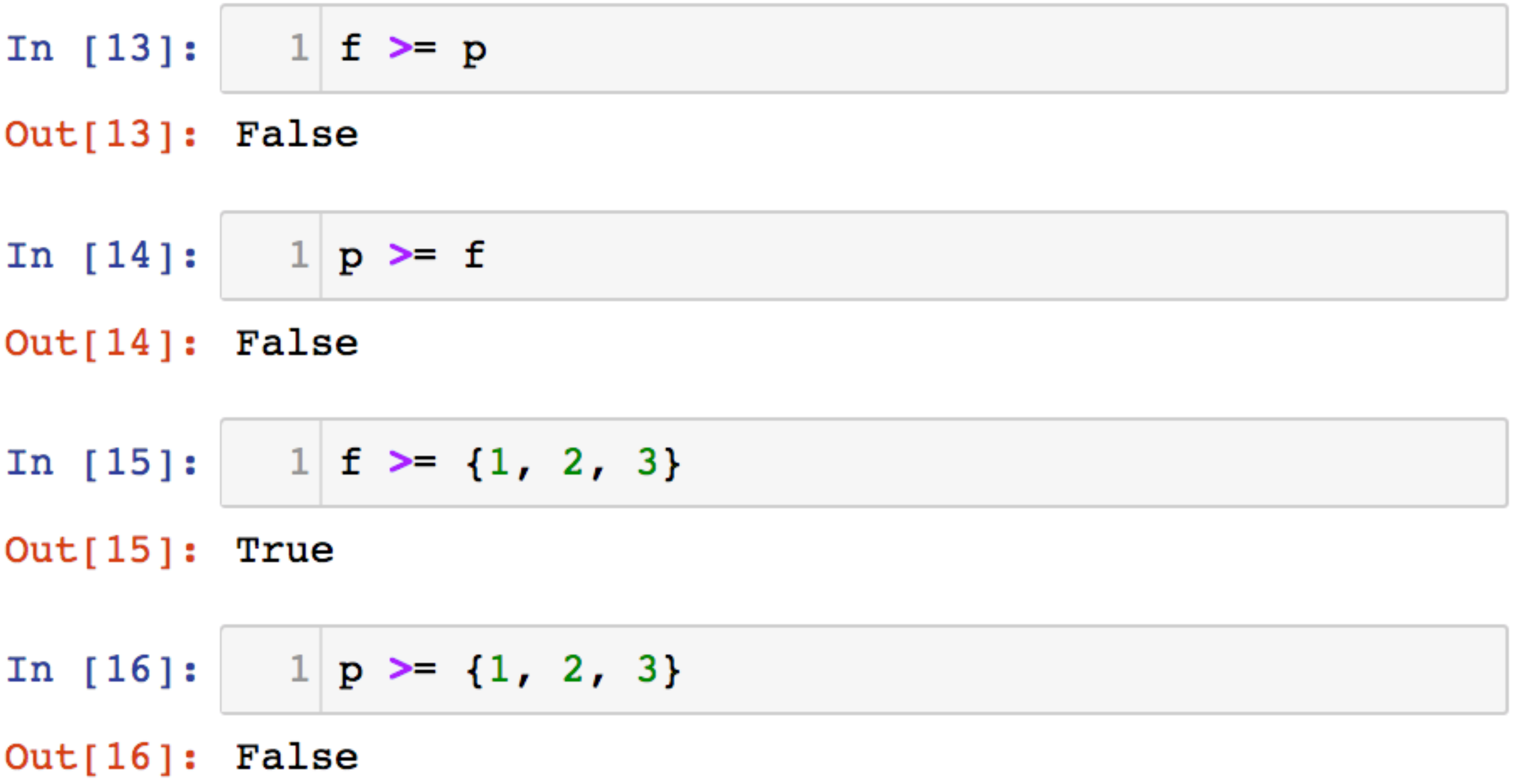
Difference

```
In [12]:  p - f
Out[12]:  {7}
```

SET COMPARISONS

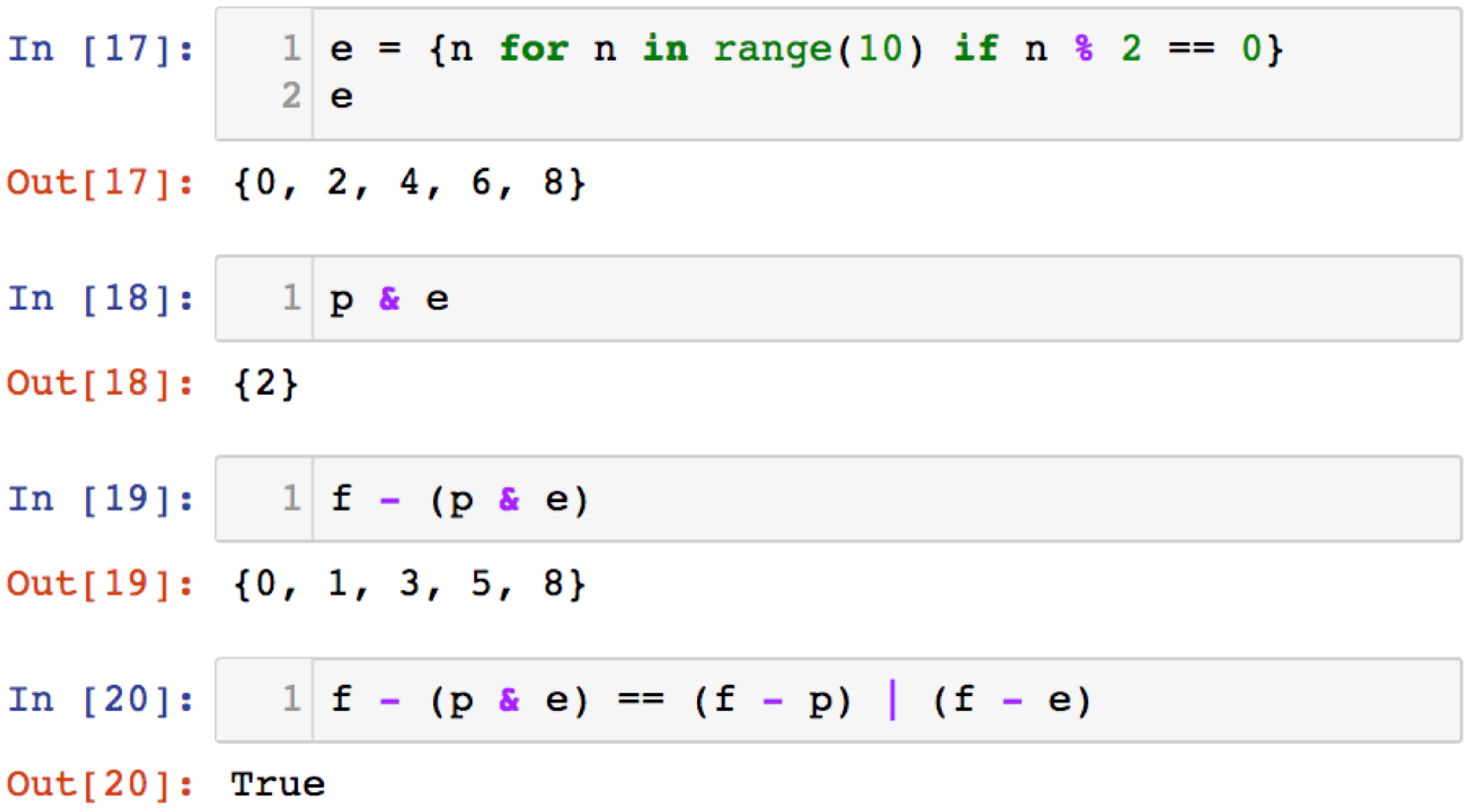
Subset and superset testing.

In math: ⊂, ⊆, ⊃, ⊇.



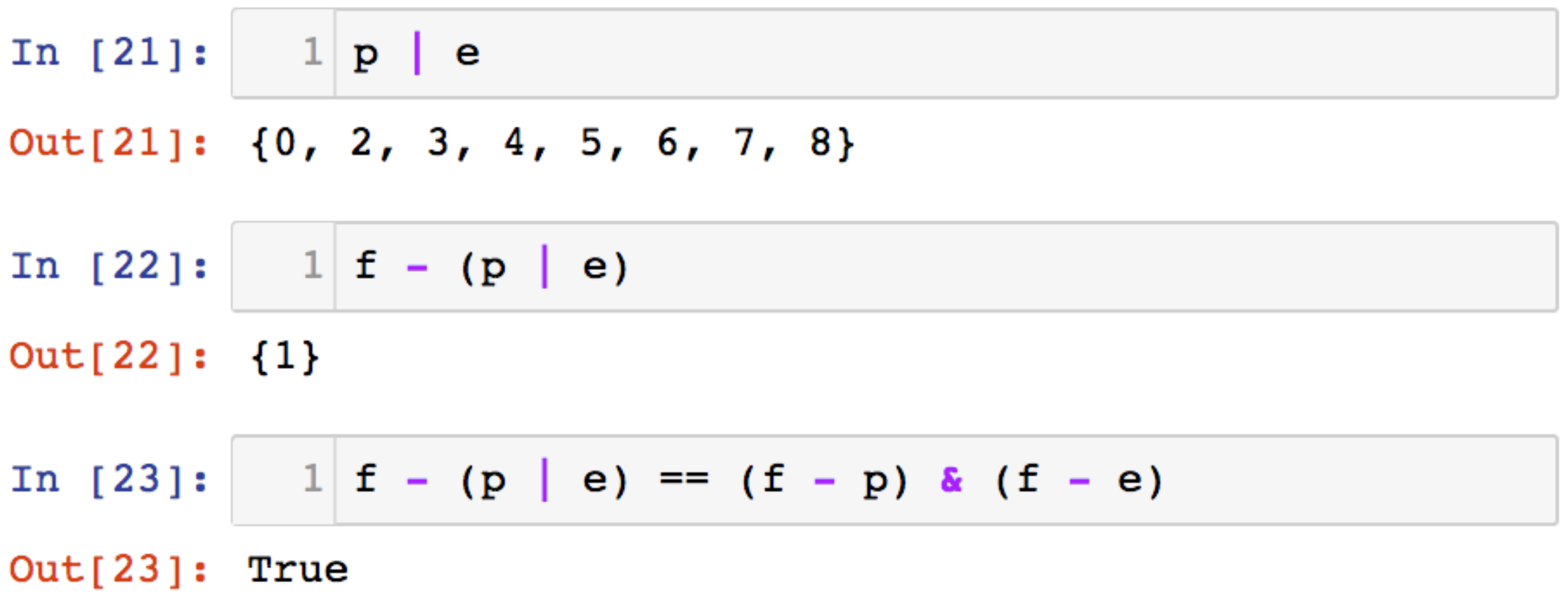
```
In [13]: f >= p
Out[13]: False

In [14]: p >= f
Out[14]: False

In [15]: f >= {1, 2, 3}
Out[15]: True

In [16]: p >= {1, 2, 3}
Out[16]: False
```

DE MORGAN'S LAW: #1



```
In [17]: e = {n for n in range(10) if n % 2 == 0}
         e
Out[17]: {0, 2, 4, 6, 8}

In [18]: p & e
Out[18]: {2}

In [19]: f - (p & e)
Out[19]: {0, 1, 3, 5, 8}

In [20]: f - (p & e) == (f - p) | (f - e)
Out[20]: True
```

DE MORGAN'S LAW: #2



```
In [21]: p | e
Out[21]: {0, 2, 3, 4, 5, 6, 7, 8}

In [22]: f - (p | e)
Out[22]: {1}

In [23]: f - (p | e) == (f - p) & (f - e)
Out[23]: True
```

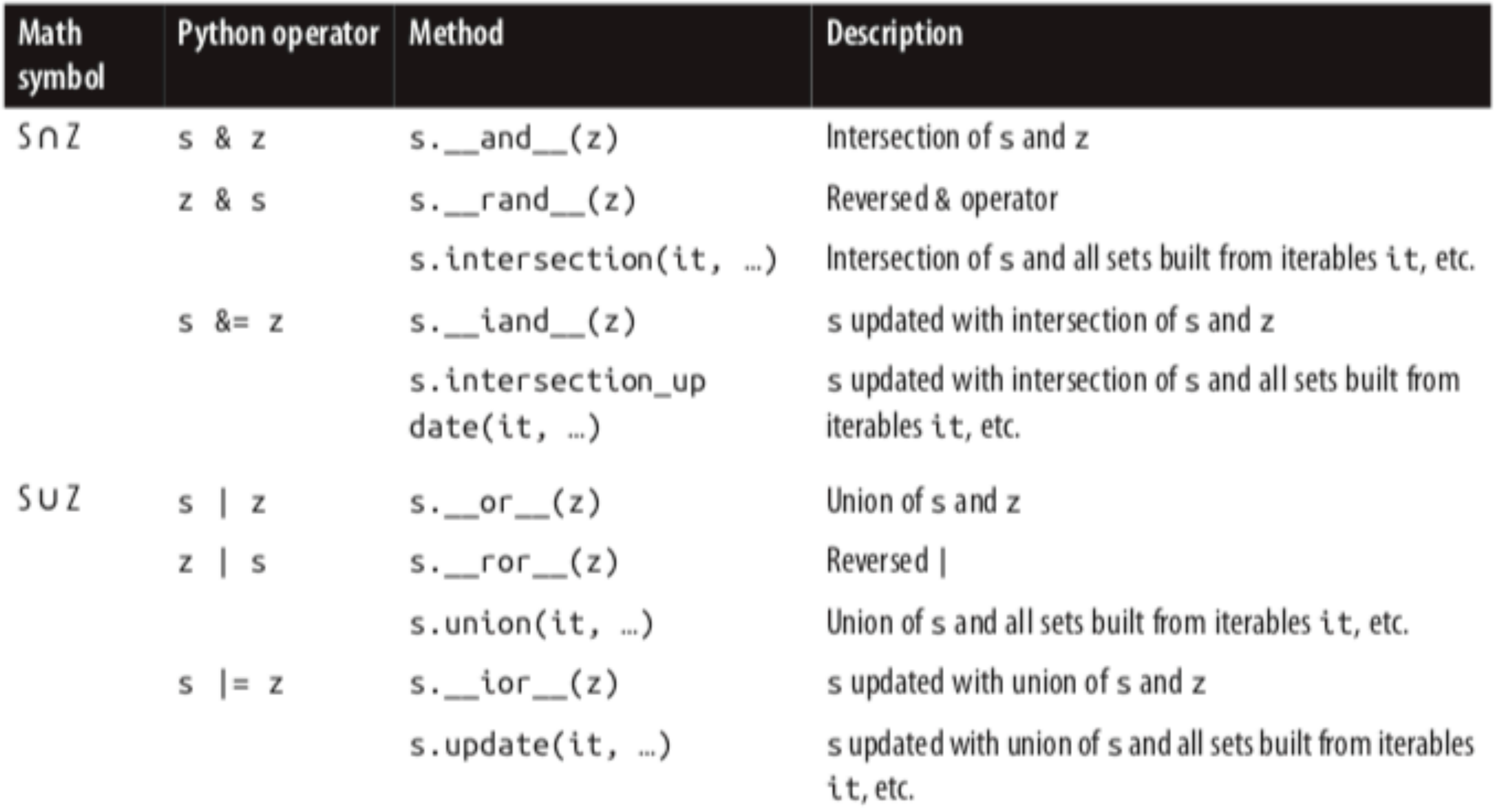
ThoughtWorks®

SET METHODS

Going beyond what operators can do.

SET OPERATORS AND METHODS (1)

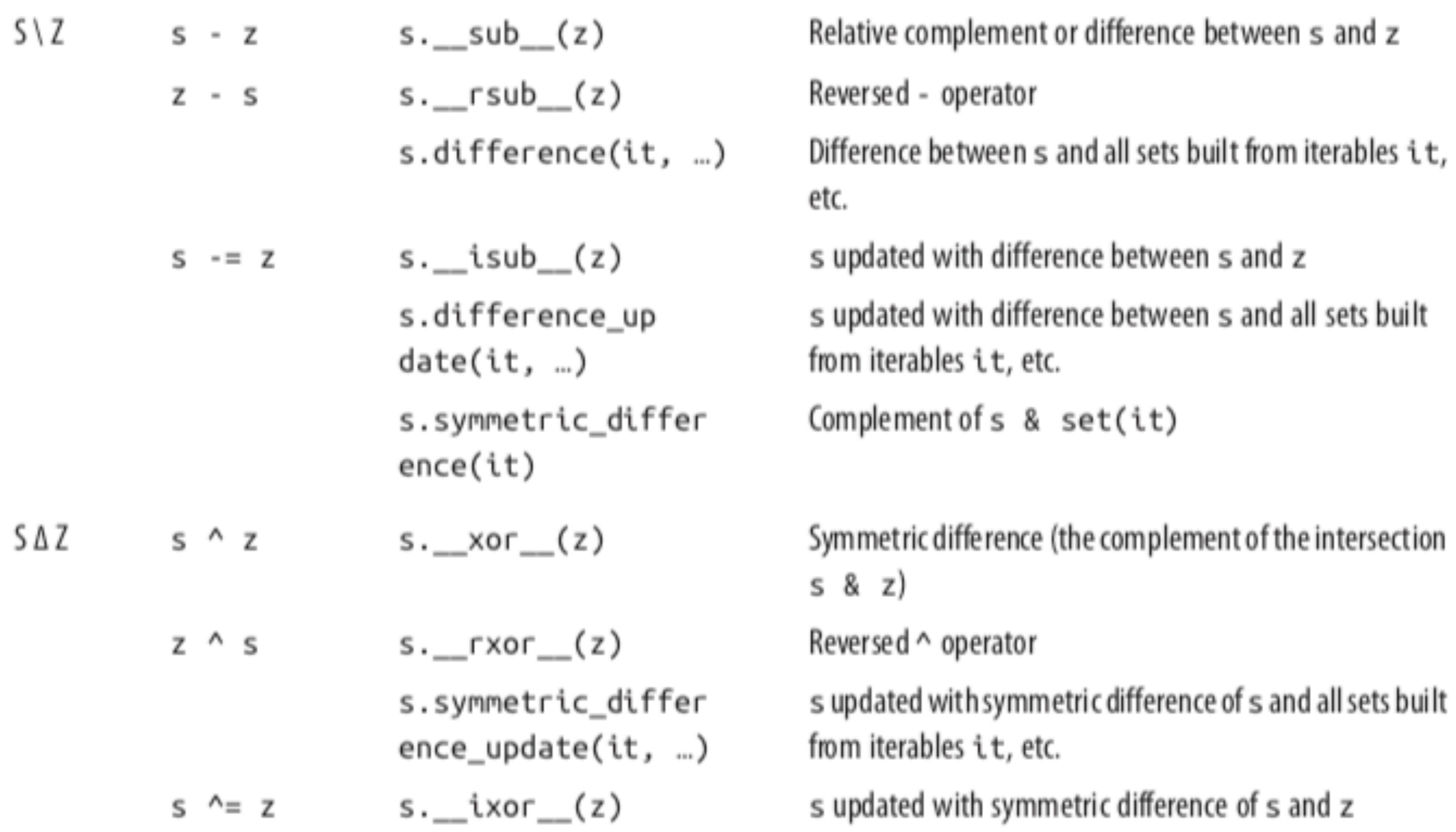
Table 3-2. Mathematical set operations: these methods either produce a new set or update the target set in place, if it's mutable



Math symbol	Python operator	Method	Description
$S \cap Z$	`s & z`	`s.__and__(z)`	Intersection of s and z
	`z & s`	`s.__rand__(z)`	Reversed & operator
		`s.intersection(it, …)`	Intersection of s and all sets built from iterables `it`, etc.
	`s &= z`	`s.__iand__(z)`	s updated with intersection of s and z
		`s.intersection_update(it, …)`	s updated with intersection of s and all sets built from iterables `it`, etc.
$S \cup Z$	`s | z`	`s.__or__(z)`	Union of s and z
	`z | s`	`s.__ror__(z)`	Reversed |
		`s.union(it, …)`	Union of s and all sets built from iterables `it`, etc.
	`s |= z`	`s.__ior__(z)`	s updated with union of s and z
		`s.update(it, …)`	s updated with union of s and all sets built from iterables `it`, etc.

SET OPERATORS AND METHODS (2)

Differences:

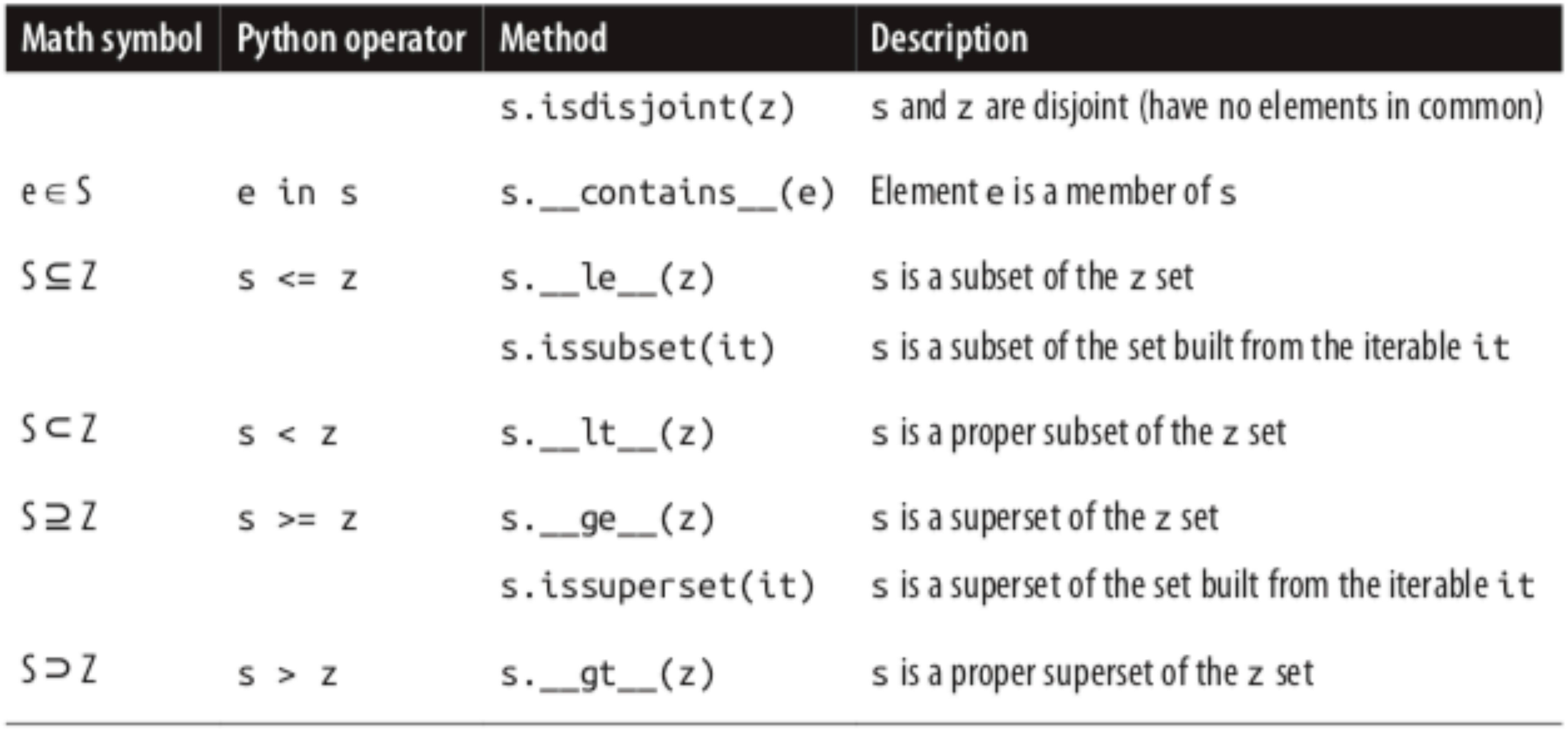


S \ Z	`s - z`	`s.__sub__(z)`	Relative complement or difference between `s` and `z`
	`z - s`	`s.__rsub__(z)`	Reversed `-` operator
		`s.difference(it, …)`	Difference between `s` and all sets built from iterables `it`, etc.
	`s -= z`	`s.__isub__(z)`	`s` updated with difference between `s` and `z`
		`s.difference_update(it, …)`	`s` updated with difference between `s` and all sets built from iterables `it`, etc.
		`s.symmetric_difference(it)`	Complement of `s & set(it)`
S ∆ Z	`s ^ z`	`s.__xor__(z)`	Symmetric difference (the complement of the intersection `s & z`)
	`z ^ s`	`s.__rxor__(z)`	Reversed `^` operator
		`s.symmetric_difference_update(it, …)`	`s` updated with symmetric difference of `s` and all sets built from iterables `it`, etc.
	`s ^= z`	`s.__ixor__(z)`	`s` updated with symmetric difference of `s` and `z`

SET TESTS

All of these return a bool:

Table 3-3. Set comparison operators and methods that return a bool



Math symbol	Python operator	Method	Description
		`s.isdisjoint(z)`	s and z are disjoint (have no elements in common)
$e \in S$	`e in s`	`s.__contains__(e)`	Element e is a member of s
$S \subseteq Z$	`s <= z`	`s.__le__(z)`	s is a subset of the z set
		`s.issubset(it)`	s is a subset of the set built from the iterable `it`
$S \subset Z$	`s < z`	`s.__lt__(z)`	s is a proper subset of the z set
$S \supseteq Z$	`s >= z`	`s.__ge__(z)`	s is a superset of the z set
		`s.issuperset(it)`	s is a superset of the set built from the iterable `it`
$S \supset Z$	`s > z`	`s.__gt__(z)`	s is a proper superset of the z set

ADDITIONAL METHODS

These have nothing to do with math, and all to do with practical computing:

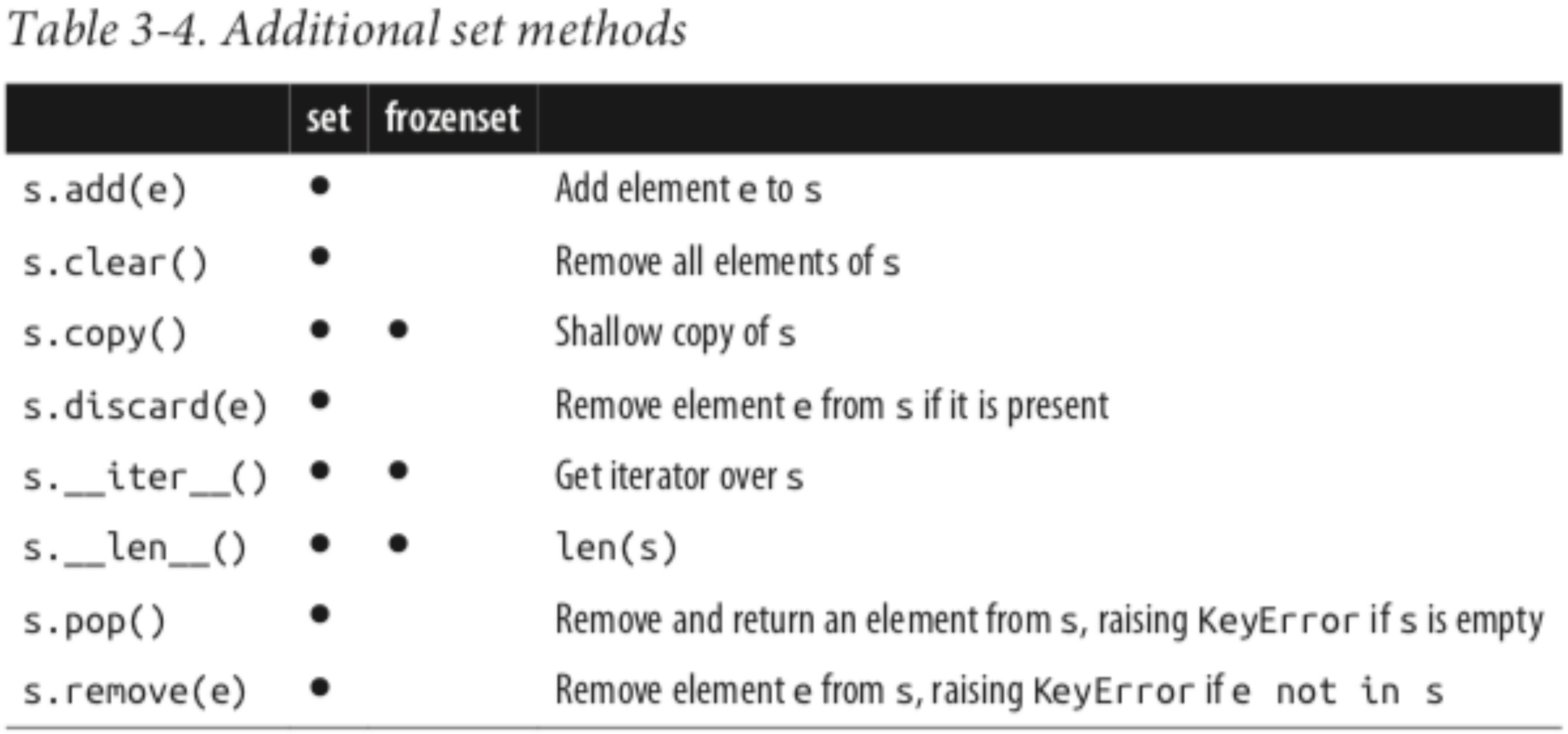


Table 3-4. Additional set methods

	set	frozenset	
`s.add(e)`	●		Add element `e` to `s`
`s.clear()`	●		Remove all elements of `s`
`s.copy()`	●	●	Shallow copy of `s`
`s.discard(e)`	●		Remove element `e` from `s` if it is present
`s.__iter__()`	●	●	Get iterator over `s`
`s.__len__()`	●	●	`len(s)`
`s.pop()`	●		Remove and return an element from `s`, raising `KeyError` if `s` is empty
`s.remove(e)`	●		Remove element `e` from `s`, raising `KeyError` if `e not in s`

ABSTRACT SET INTERFACES

These interfaces are all defined in collections.abc.

set and frozenset both implement Set

set also implements MutableSet

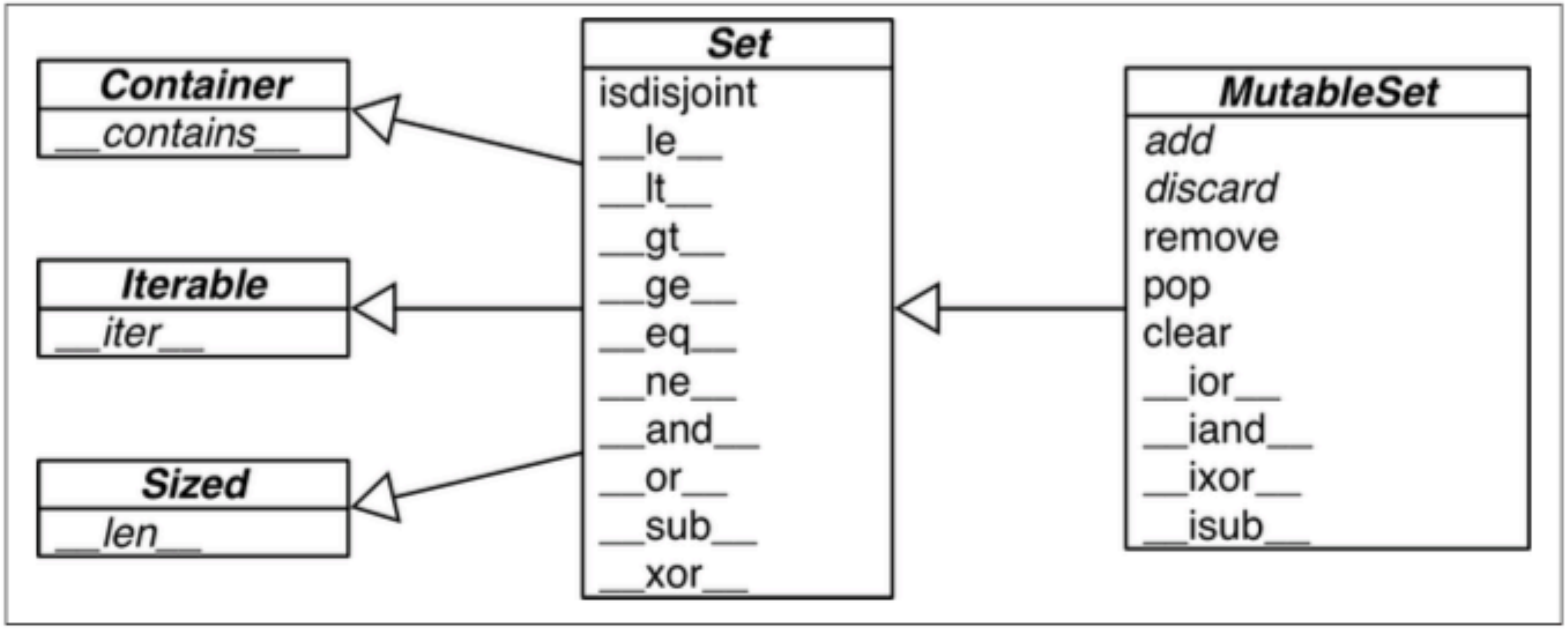


Figure 3-2. UML class diagram for MutableSet and its superclasses from collections.abc (names in italic are abstract classes and abstract methods; reverse operator methods omitted for brevity)

OPERATOR OVERLOADING

Not as bad as they say

COMPARISON OPERATORS

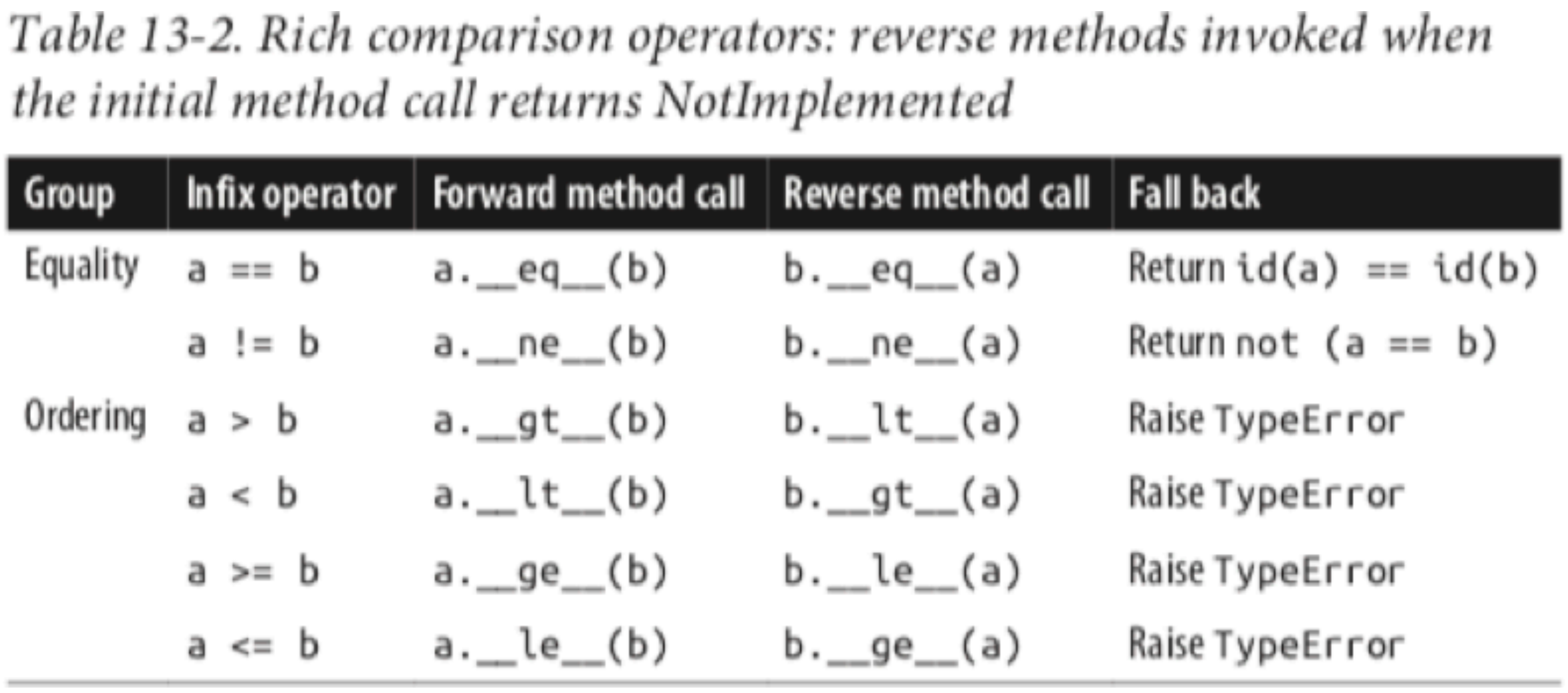


Table 13-2. Rich comparison operators: reverse methods invoked when the initial method call returns NotImplemented

Group	Infix operator	Forward method call	Reverse method call	Fall back
Equality	`a == b`	`a.__eq__(b)`	`b.__eq__(a)`	Return `id(a) == id(b)`
	`a != b`	`a.__ne__(b)`	`b.__ne__(a)`	Return `not (a == b)`
Ordering	`a > b`	`a.__gt__(b)`	`b.__lt__(a)`	Raise `TypeError`
	`a < b`	`a.__lt__(b)`	`b.__gt__(a)`	Raise `TypeError`
	`a >= b`	`a.__ge__(b)`	`b.__le__(a)`	Raise `TypeError`
	`a <= b`	`a.__le__(b)`	`b.__ge__(a)`	Raise `TypeError`

Table 13-1. Infix operator method names (the in-place operators are used for augmented assignment; comparison operators are in Table 13-2)

Operator	Forward	Reverse	In-place	Description
`+`	`__add__`	`__radd__`	`__iadd__`	Addition or concatenation
`-`	`__sub__`	`__rsub__`	`__isub__`	Subtraction
`*`	`__mul__`	`__rmul__`	`__imul__`	Multiplication or repetition
`/`	`__truediv__`	`__rtruediv__`	`__itruediv__`	True division
`//`	`__floordiv__`	`__rfloordiv__`	`__ifloordiv__`	Floor division
`%`	`__mod__`	`__rmod__`	`__imod__`	Modulo
`divmod()`	`__divmod__`	`__rdivmod__`	`__idivmod__`	Returns tuple of floor division quotient and modulo
`**`, `pow()`	`__pow__`	`__rpow__`	`__ipow__`	Exponentiation[a]
`@`	`__matmul__`	`__rmatmul__`	`__imatmul__`	Matrix multiplication[b]
`&`	`__and__`	`__rand__`	`__iand__`	Bitwise and
`|`	`__or__`	`__ror__`	`__ior__`	Bitwise or
`^`	`__xor__`	`__rxor__`	`__ixor__`	Bitwise xor
`<<`	`__lshift__`	`__rlshift__`	`__ilshift__`	Bitwise shift left
`>>`	`__rshift__`	`__rrshift__`	`__irshift__`	Bitwise shift right

[a] `pow` takes an optional third argument, `modulo`: `pow(a, b, modulo)`, also supported by the special methods when invoked directly (e.g., `a.__pow__(b, modulo)`).

[b] New in Python 3.5.

THE BEAUTY OF DOUBLE DISPATCH

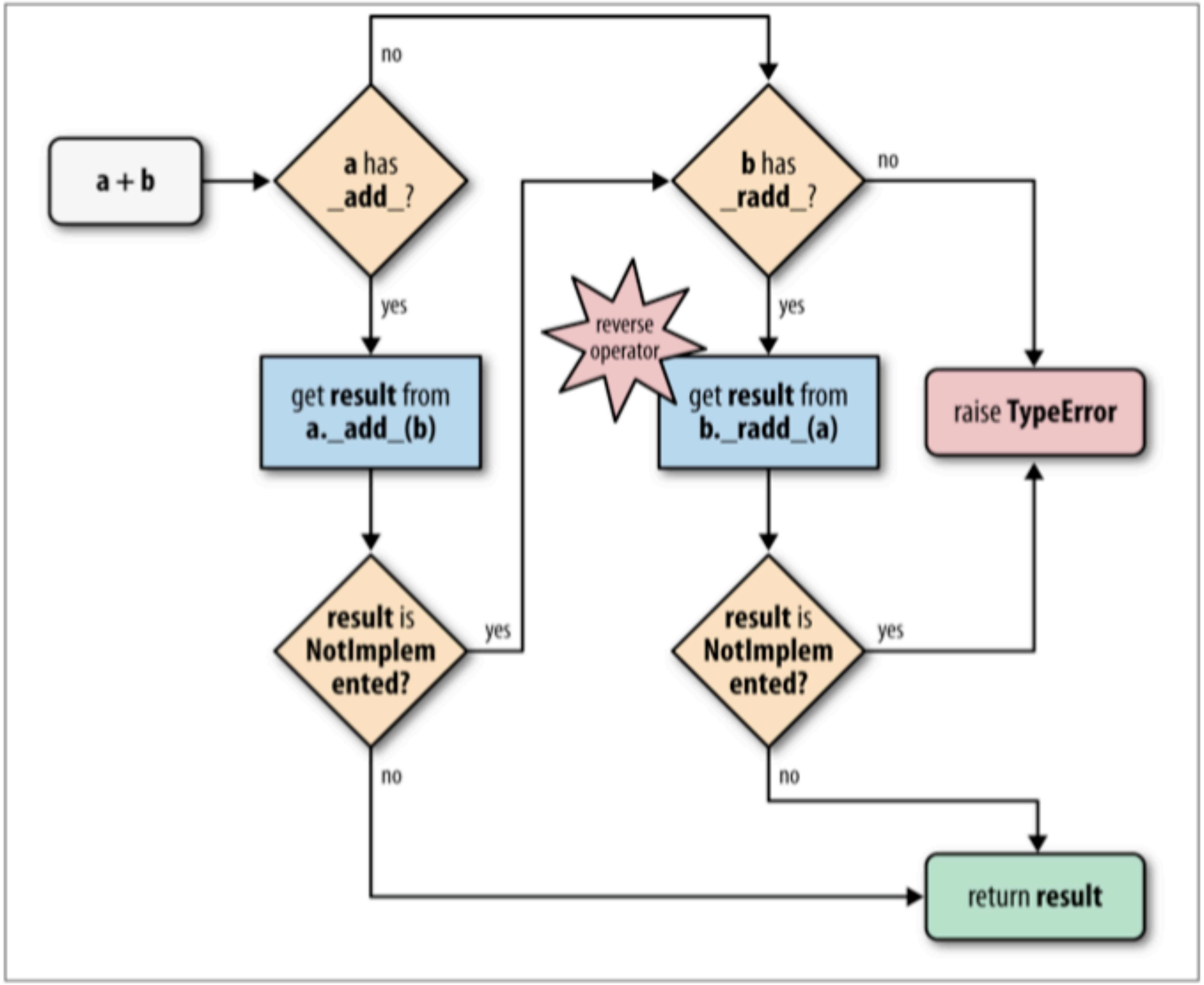

a + b
a has __add__?
no
yes
get result from a.__add__(b)
result is NotImplemented?
yes
no
reverse operator
b has __radd__?
no
yes
get result from b.__radd__(a)
result is NotImplemented?
yes
no
raise TypeError
return result

Figure 13-1. Flowchart for computing a + b with __add__ and __radd__

EXAMPLE IMPLEMENTATION

A set for non-negative integers

UINTSET: A SET CLASS FOR NON-NEGATIVE INTEGERS

Inspired by the **intset** example in chapter 6 of *The Go Programming Language* by A. Donovan and B. Kernighan

An empty set is represented by zero.

A set of integers {**a**, **b**, **c**} is represented by **on** bits in an integer at offsets **a**, **b**, and **c**.

Source code:

`https://github.com/standupdev/uintset`

REPRESENTING SETS OF INTEGERS AS BIT PATTERNS

This set:

```
UintSet({13, 14, 22, 28, 38, 53, 64, 76, 94, 102, 107, 121,
136, 143, 150, 157, 169, 173, 187, 201, 213, 216, 234, 247,
257, 268, 283, 288, 290})
```

REPRESENTING SETS OF INTEGERS AS BIT PATTERNS

This set:

```
UintSet({13, 14, 22, 28, 38, 53, 64, 76, 94, 102, 107, 121,
136, 143, 150, 157, 169, 173, 187, 201, 213, 216, 234, 247,
257, 268, 283, 288, 290})
```

Is represented by this integer

```
250215800770294692189743128168123011668092585423464438593870З
3633964549718976522837278 72
```

REPRESENTING SETS OF INTEGERS AS BIT PATTERNS

This set:

```
UintSet({13, 14, 22, 28, 38, 53, 64, 76, 94, 102, 107, 121,
136, 143, 150, 157, 169, 173, 187, 201, 213, 216, 234, 247,
257, 268, 283, 288, 290})
```

Is represented by this integer

```
2502158007702946921897431281681230116680925854234644385938703
36339645497189765228372787 2
```

Which has this bit pattern:

```
1010000100000000000000100000000001000000000100000000000010000
0000000000000100100000000000100000000000001000000000000010001
0000000000010000001000000100000010000000000000010000000000000
1000010000000100000000000000000100000000000100000000001000000
00000000100000000010000010000000110000000000000
```

REPRESENTING SETS OF INTEGERS AS BIT PATTERNS

This set:

```
UintSet({290})
```

Is represented by this integer

```
1989292945639146568621528992587283360401824603189390869761855
907572637988050133502132224
```

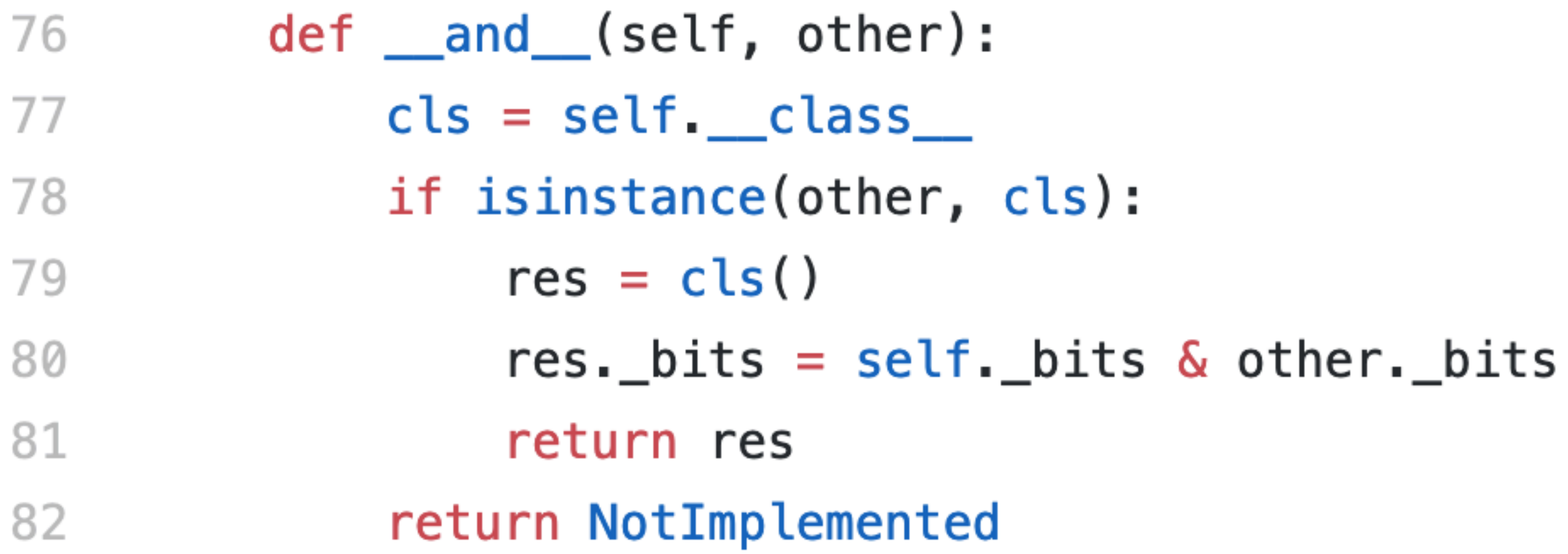
Which has this bit pattern:

```
1000000000000000000000000000000000000000000000000000000000000
0000000000000000000000000000000000000000000000000000000000000
0000000000000000000000000000000000000000000000000000000000000
0000000000000000000000000000000000000000000000000000000000000
00000000000000000000000000000000000000000000000
```

REPRESENTING SETS OF INTEGERS AS BIT PATTERNS (2)

```
UintSet() → 0                                                   |0|

UintSet({0}) → 1                                                |1|

UintSet({1}) → 2                                              |1|0|

UintSet({0, 1, 2, 4, 8}) → 279                  |1|0|0|0|1|0|1|1|1|

UintSet({0, 1, 2, 3, 4, 5, 6, 7, 8, 9}) → 1023  |1|1|1|1|1|1|1|1|1|1|

UintSet({10}) → 1024                           |1|0|0|0|0|0|0|0|0|0|0|

UintSet({0, 2, 4, 6, 8, 10, 12, 14, 16, 18}) → 349525

                                   |1|0|1|0|1|0|1|0|1|0|1|0|1|0|1|0|1|0|1|

UintSet({1, 3, 5, 7, 9, 11, 13, 15, 17, 19}) → 699050

                                 |1|0|1|0|1|0|1|0|1|0|1|0|1|0|1|0|1|0|1|0|
```

SAMPLE METHOD: INTERSECTION OPERATOR &



```python
    def __and__(self, other):
        cls = self.__class__
        if isinstance(other, cls):
            res = cls()
            res._bits = self._bits & other._bits
            return res
        return NotImplemented
```

SAMPLE METHOD: INTERSECTION WITH ITERABLES

```python
    def intersection(self, *others):
        cls = self.__class__
        res = cls()
        res._bits = self._bits
        for other in others:
            if isinstance(other, cls):
                res._bits &= other._bits
            try:
                second = cls(other)
            except TypeError:
                raise TypeError(INVALID_ITER_ARG_MSG)
            else:
                res._bits &= second._bits
        return res
```

DIVE INTO THE CODE

`https://github.com/standupdev/uintset`

ThoughtWorks®

CONCLUSION

KEY TAKEAWAYS

1. Set operations allow simpler, faster solutions for many tasks.
2. Python’s set classes are lessons in idiomatic API design.
3. A set class provides good context for operator overloading.

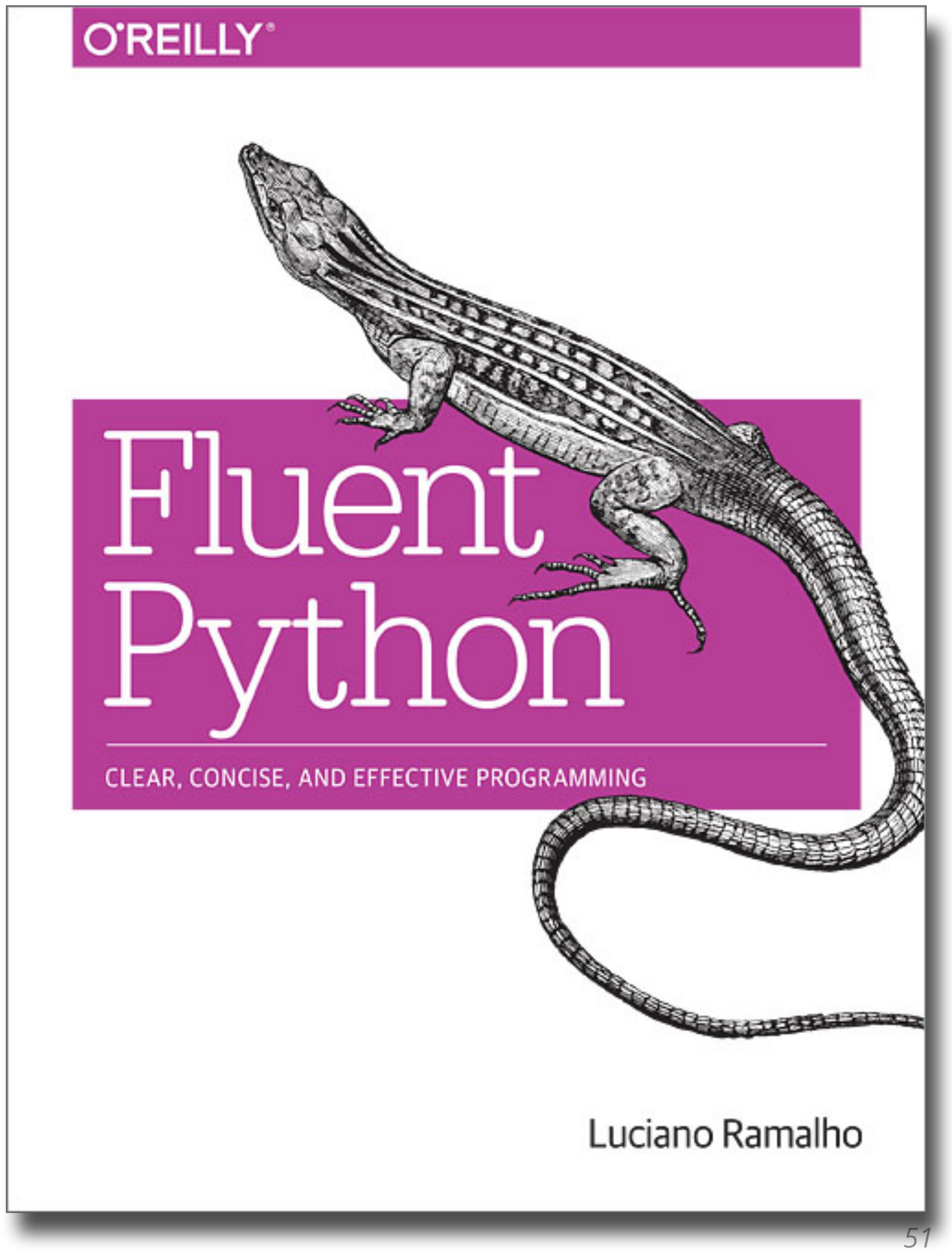
THANK YOU! COME SEE ME AT THE EXPO ALL…

A deeper look at the code for **UintSet**

- Today, 11:45 at the JetBrains/PyCharm booth

Fluent Python book signing
—*handing out free copies!*

- Today, 4:00 at the O'Reilly booth



ThoughtWorks®

THANK YOU!

Luciano Ramalho

@ramalhoorg | @standupdev

luciano.ramalho@thoughtworks.com

ThoughtWorks®