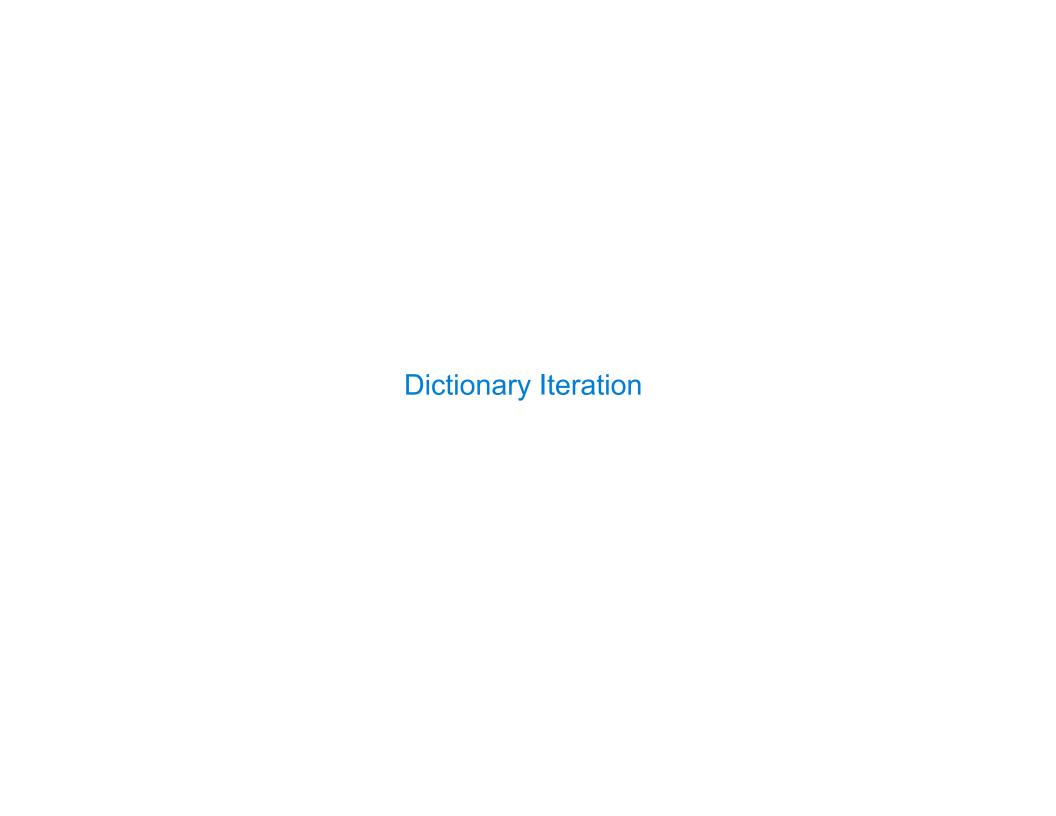


Iterators

A container can provide an iterator that provides access to its elements in order

```
>>> s = [3, 4, 5]
>>> t = iter(s)
>>> next(t)
3
>>> next(t)
4
>>> u = iter(s)
>>> next(u)
3
>>> next(u)
4
>>> next(u)
4
```



Views of a Dictionary

An *iterable* value is any value that can be passed to **iter** to produce an iterator

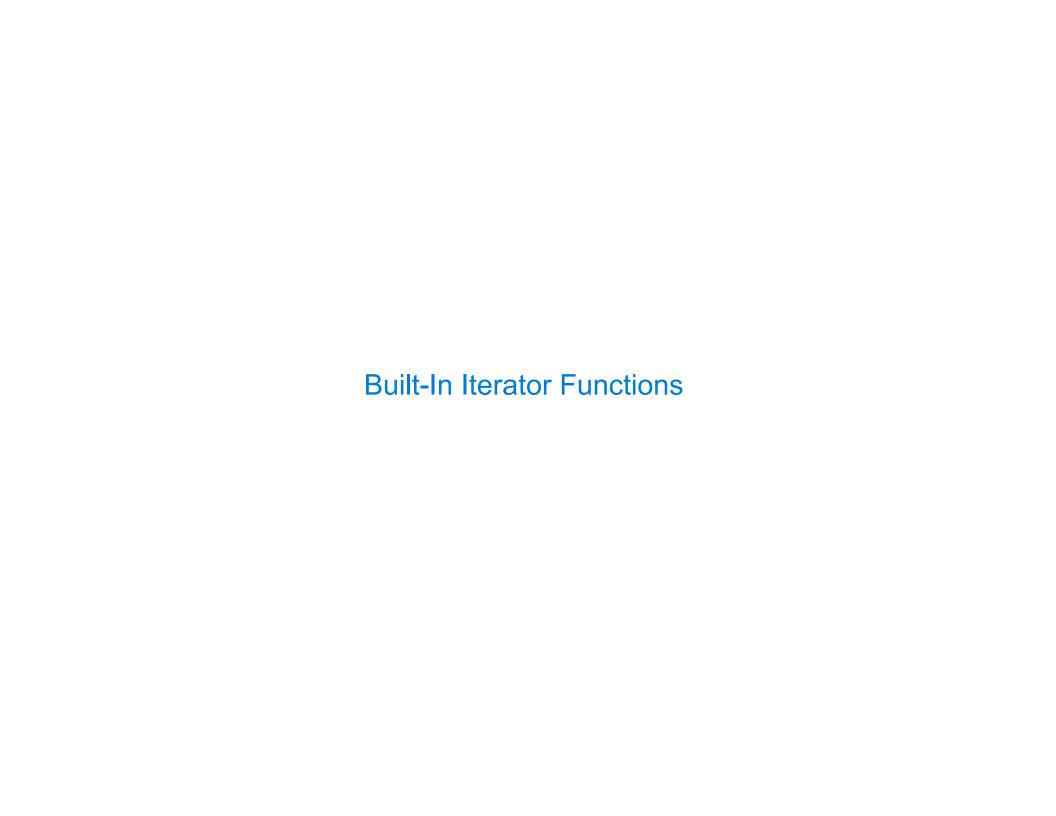
An *iterator* is returned from **iter** and can be passed to **next**; all iterators are mutable

A dictionary, its keys, its values, and its items are all iterable values

- The order of items in a dictionary is the order in which they were added (Python 3.6+)
- Historically, items appeared in an arbitrary order (Python 3.5 and earlier)

```
>>> d = {'one': 1, 'two': 2, 'three': 3}
>>> d['zero'] = 0
>>> k = iter(d.keys()) # or iter(d)
                                             >>> v = iter(d.values())
                                                                              >>> i = iter(d.items())
>>> next(k)
                                             >>> next(v)
                                                                              >>> next(i)
'one'
                                                                              ('one', 1)
                                             1
>>> next(k)
                                             >>> next(v)
                                                                              >>> next(i)
'two'
                                                                              ('two', 2)
>>> next(k)
                                             >>> next(v)
                                                                              >>> next(i)
'three'
                                                                              ('three', 3)
>>> next(k)
                                             >>> next(v)
                                                                              >>> next(i)
'zero'
                                                                              ('zero', 0)
```

For Statements



Built-in Functions for Iteration

Many built-in Python sequence operations return iterators that compute results lazily

map(func, iterable): Iterate over func(x) for x in iterable

filter(func, iterable): Iterate over x in iterable if func(x)

zip(first_iter, second_iter):
Iterate over co-indexed (x, y) pairs

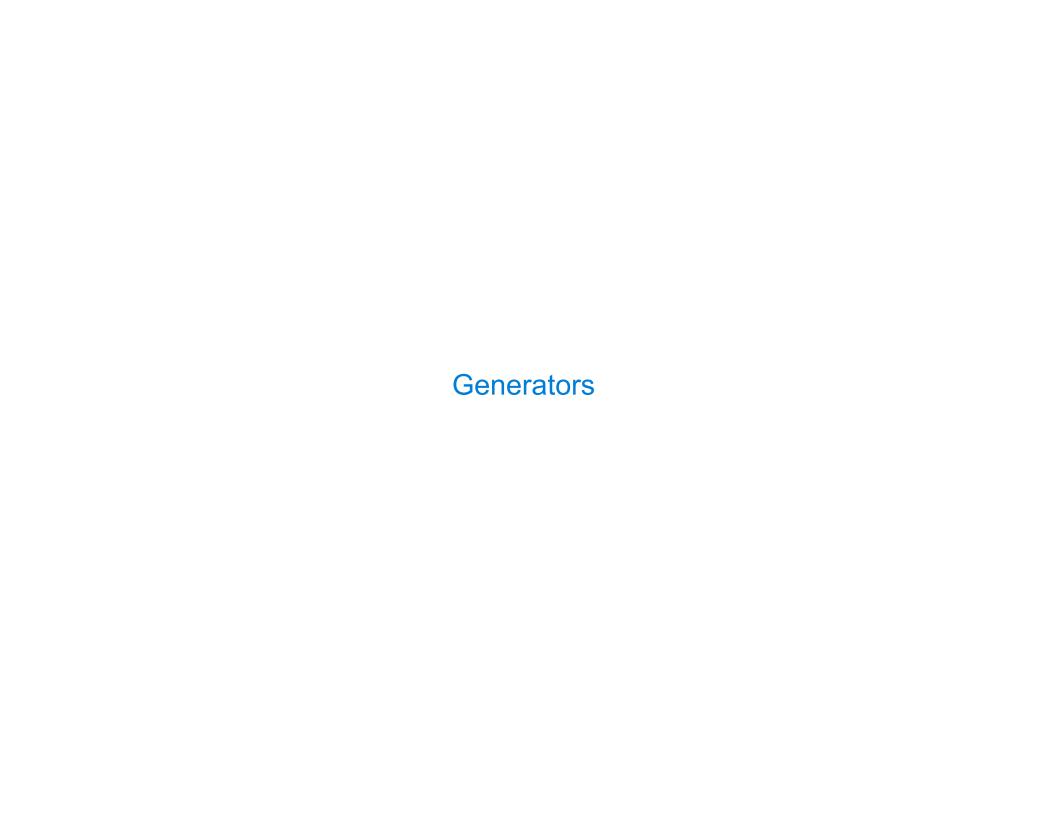
reversed(sequence): Iterate over x in a sequence in reverse order

To view the contents of an iterator, place the resulting elements into a container

list(iterable): Create a list containing all x in iterable

tuple(iterable): Create a tuple containing all x in iterable

sorted(iterable): Create a sorted list containing x in iterable



Generators and Generator Functions

```
>>> def plus_minus(x):
...     yield x
...     yield -x

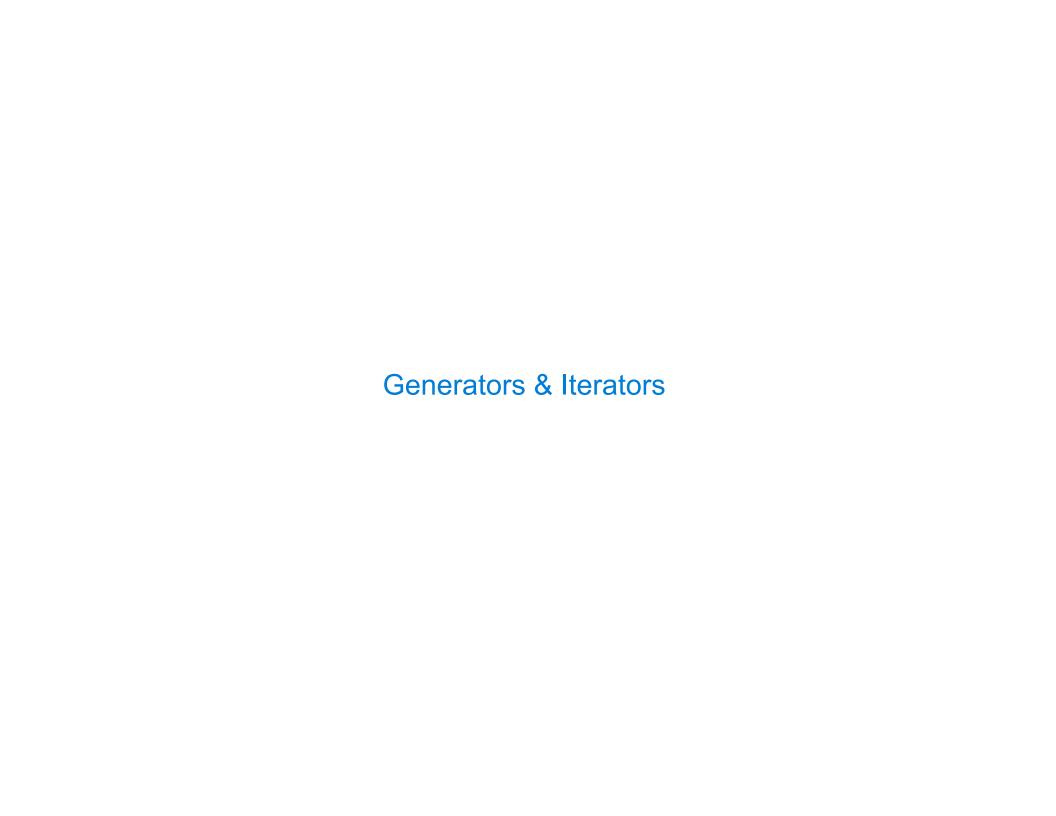
>>> t = plus_minus(3)
>>> next(t)
3
>>> next(t)
-3
>>> t
<generator object plus_minus ...>
```

A generator function is a function that yields values instead of returning them

A normal function returns once; a generator function can yield multiple times

A generator is an iterator created automatically by calling a generator function

When a generator function is called, it returns a generator that iterates over its yields



Generators can Yield from Iterators

yield k

yield from countdown(k-1)