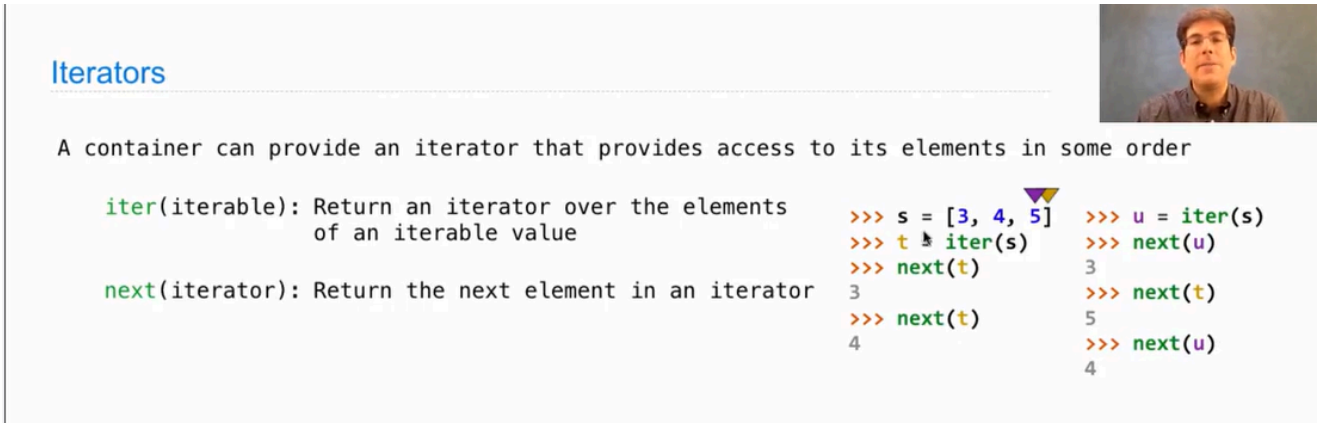
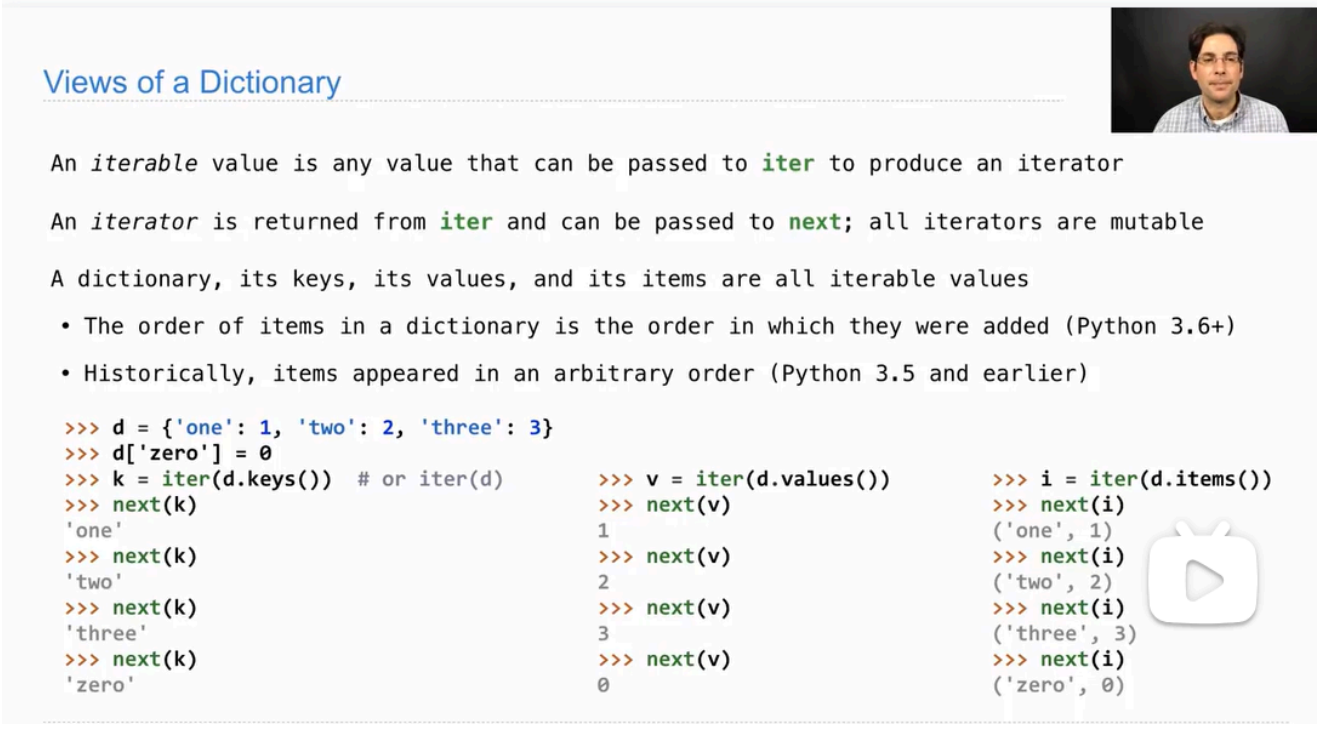
Lecture 17.Iterators

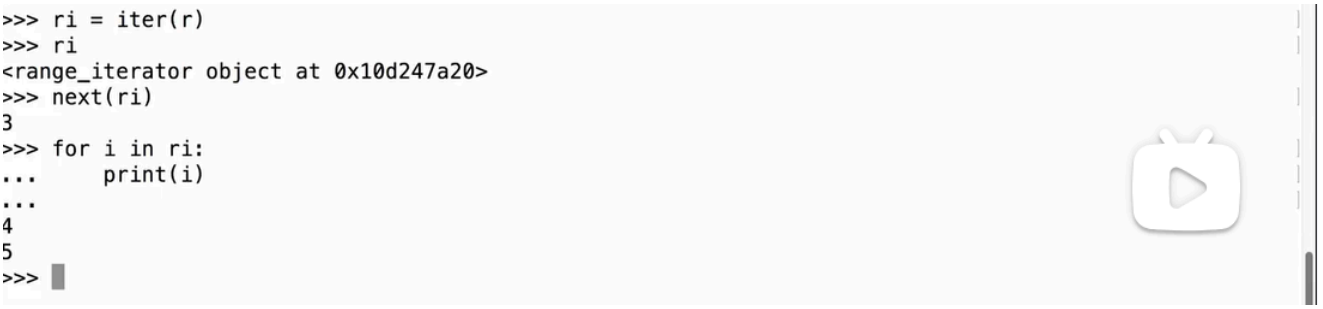
Iterators



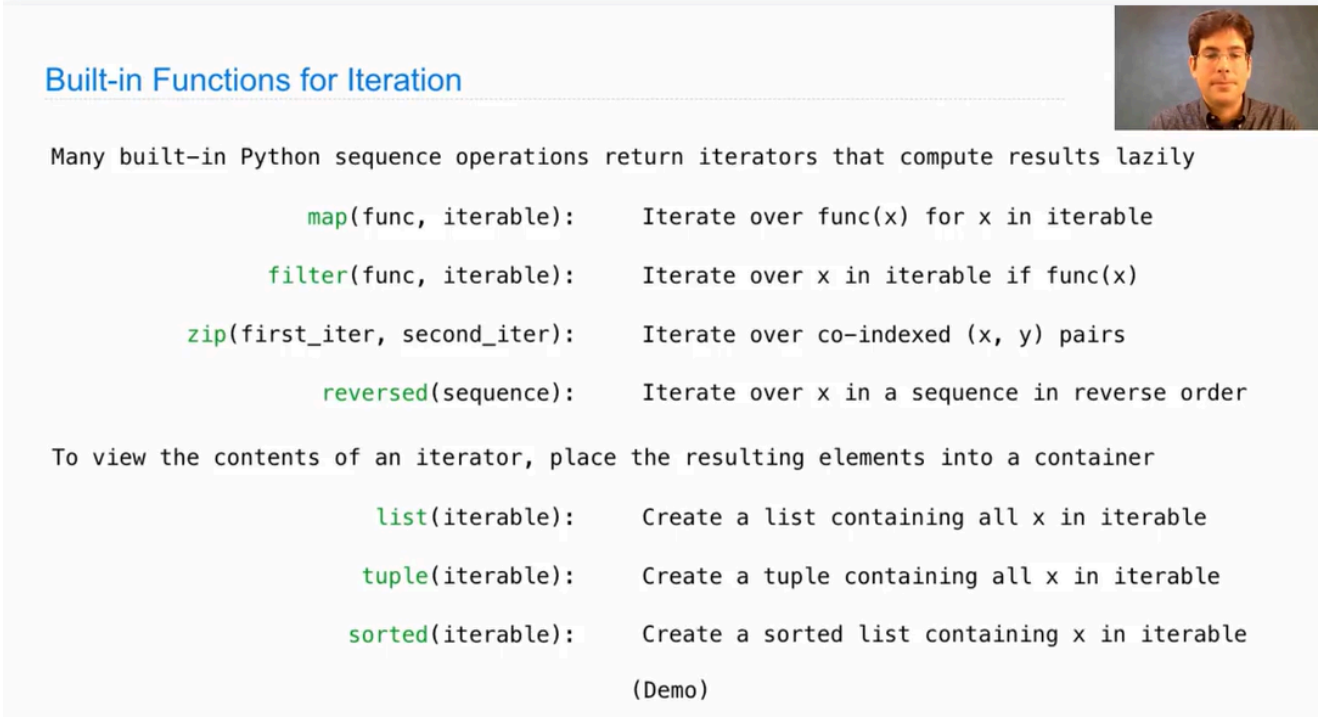
Dictionary Iterators

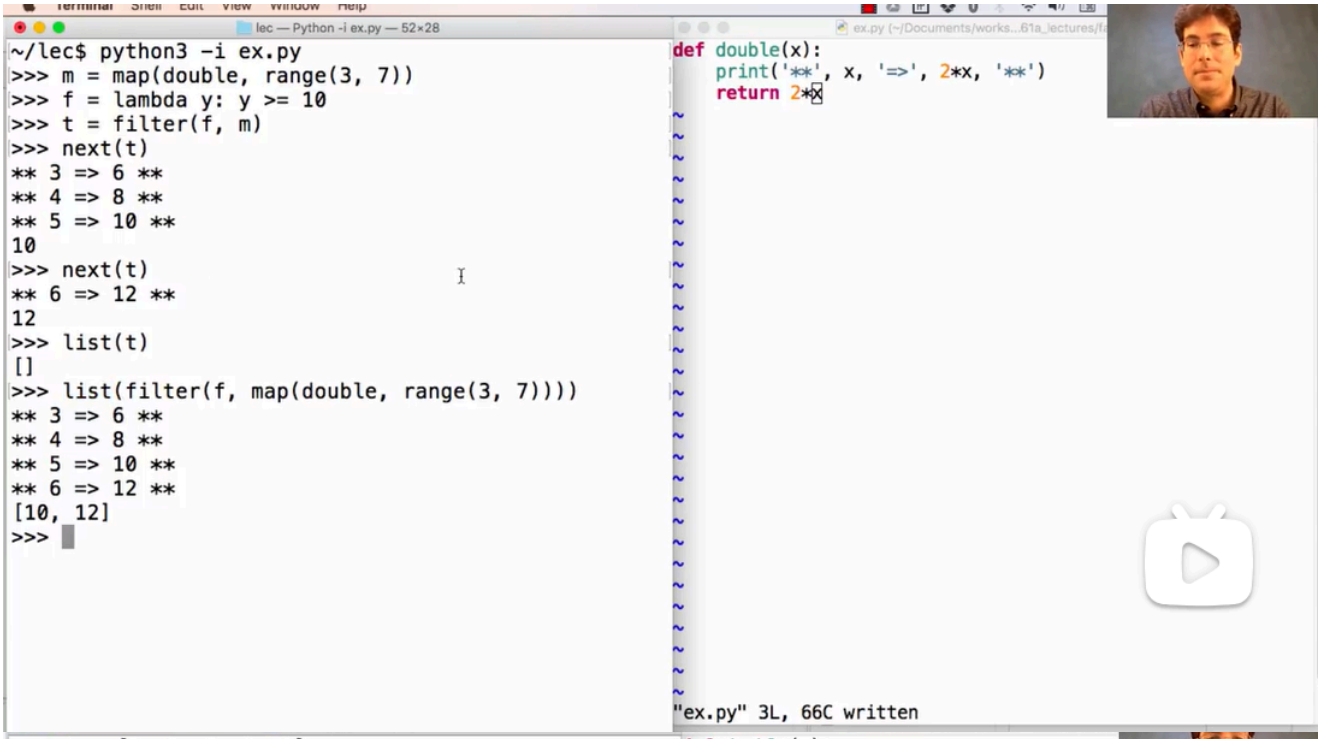


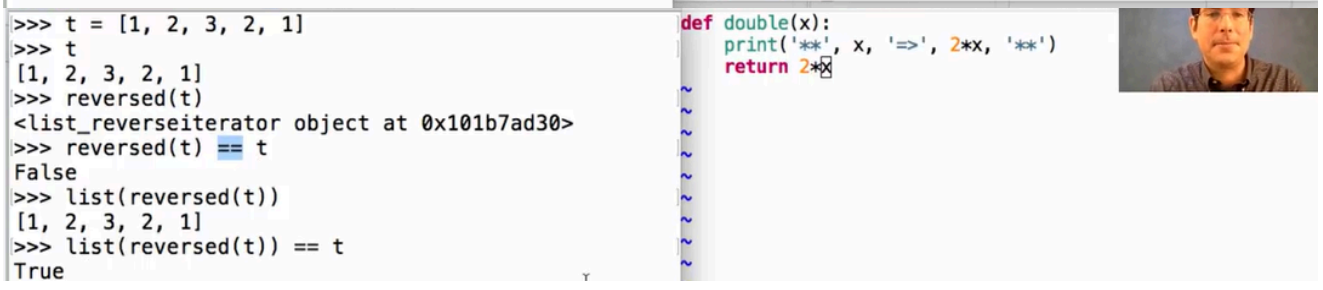
better to see the Iterators as pointers, so record the position and moves, so changing the size of the list/dic is not allowed after the iterators is created, since the iterator will not be allowed to access anymore



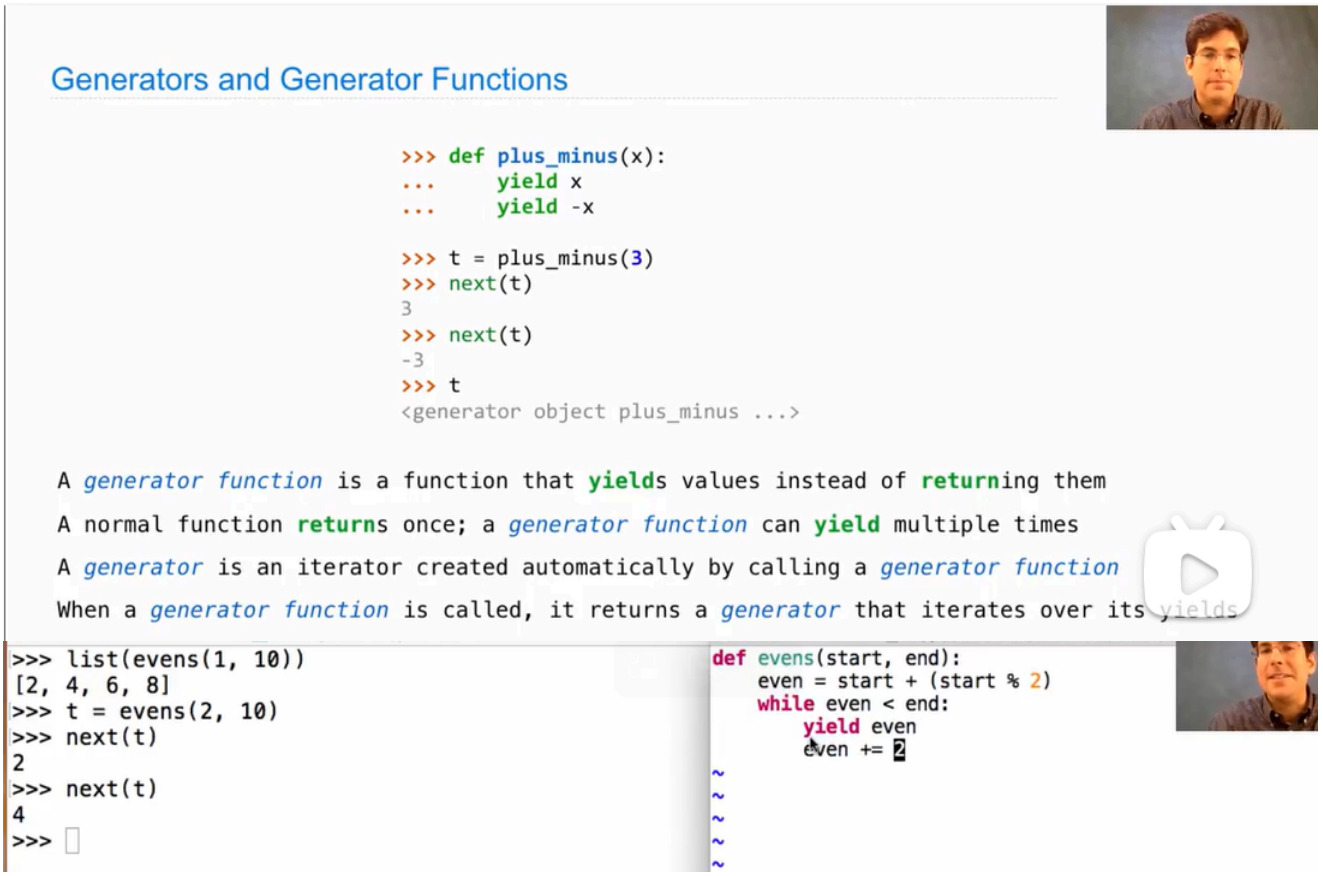
Built-In iterator Functions


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
(Demo)


lec — Python -i ex.py — 52×28
~/lec$ python3 -i ex.py
>>> m = map(double, range(3, 7))
>>> f = lambda y: y >= 10
>>> t = filter(f, m)
>>> next(t)
** 3 => 6 **
** 4 => 8 **
** 5 => 10 **
10
>>> next(t)
** 6 => 12 **
12
>>> list(t)
[]
>>> list(filter(f, map(double, range(3, 7))))
** 3 => 6 **
** 4 => 8 **
** 5 => 10 **
** 6 => 12 **
[10, 12]
>>>
def double(x):
print('**', x, '=>', 2*x, '**')
return 2*x
"ex.py" 3L, 66C written


>>> t = [1, 2, 3, 2, 1]
>>> t
[1, 2, 3, 2, 1]
>>> reversed(t)
<list_reverseiterator object at 0x101b7ad30>
>>> reversed(t) == t
False
>>> list(reversed(t))
[1, 2, 3, 2, 1]
>>> list(reversed(t)) == t
True
def double(x):
print('**', x, '=>', 2*x, '**')
return 2*x

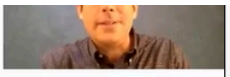
Generators



yield is kinda like a record point here, which return when executed, and will run at the end of the yield sentence next time it was called.

Generators & Iterators

Generators can Yield from Iterators



A `yield from` statement yields all values from an iterator or iterable (Python 3.3)

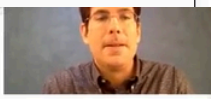
```
>>> list(a_then_b([3, 4], [5, 6]))
[3, 4, 5, 6]
```

```
def a_then_b(a, b):
    for x in a:
        yield x
    for x in b:
        yield x
```

```
def a_then_b(a, b):
    yield from a
    yield from b
```

```
>>> list(countdown(5))
[5, 4, 3, 2, 1]
```

```
def countdown(k):
    if k > 0:
        yield k
        yield from countdown(k-1)
```



```
~/lec$ python3 -i ex.py
>>> prefixes('both')
<generator object prefixes at 0x102379f30>
>>> list(prefixes('both'))
['b', 'bo', 'bot', 'both']
>>> ^D
~/lec$ python3 -i ex.py
>>> substrings('tops')
<generator object substrings at 0x101379f30>
>>> list(substrings('tops'))
['t', 'to', 'top', 'tops', 'o', 'op', 'ops', 'p', 'ps', 's']
>>>
```

```
def countdown(k):
    if k > 0:
        yield k
        yield from countdown(k-1)
    else:
        yield 'Blast off'

def prefixes(s):
    if s:
        yield from prefixes(s[:-1])
        yield s

def substrings(s):
    if s:
        yield from prefixes(s)
        yield from substrings(s[1:])
~
~
```