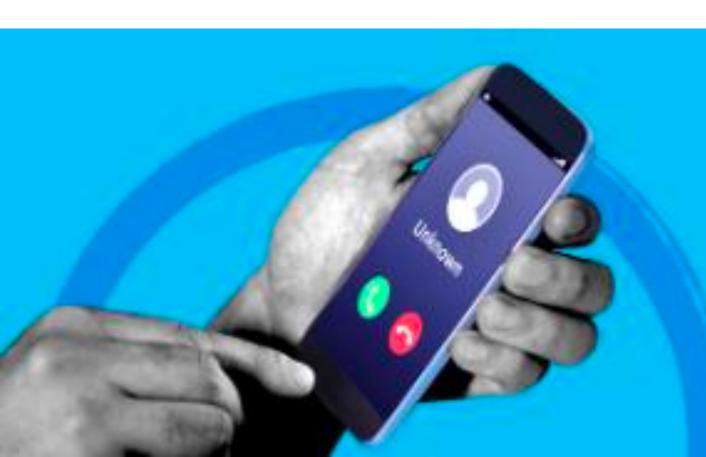
UC Berkeley's CS61A – Lecture 31 – Iterators

The frightening future of robocalls: Numbers and voices you know

PHOTO ILLUSTRATION: SHUTTERSTOCK / CHN

www.cnn.com/2019/04/06/tech/robocalls-scam-voice



"...My iPhone's caller ID flashed my own phone number, along with a picture of my face. It was a robocall using spoofing technology to pretend it was calling from my own number. ... Nearly 30% of all calls made each day are robocalls — but a call from my own number was a new one for me. Yet some experts warn this is just a mild taste of bigger dangers to come: a world where you receive robocalls calls from numbers you recognize and the person on the other end sounds like someone you know.

Spoofing, a form of robo-calling, is increasingly common. It's when someone makes a call from a voice-over-IP service, such as Skype, and are able to enter a host number. While a carrier must provide a number when a call is made from a cell phone or landline, any number sequence can be entered via a VoIP service, whether it's a made up number, a number in your address book, or one from the White House. It's so easy, anyone could do it. Because a scammer knows you're more likely to pick up if you recognize the caller, they might enter a number they think is in your address book. They could even one day use voice manipulation technology to impersonate that person. (Think deepfakes for robocalls)."





Iterators

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

```
iter(iterable): Return an iterator over the elements
    of an iterable value

next(iterator): Return the next element in an iterator
```

```
>>> 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
```

(Demo)

Dictionary Iteration

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

(Demo)

6

For Statements

(Demo)

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)

9



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

(Demo)

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 from a
        yield from b

    for x in b:
        yield x
```