CS 61A SU26

Lecture 10: Iterators and Generators

July 7, 2026

Rebecca Dang

Join pollev.com/rebeccadang831 (or scan QR code) on your phone or laptop

We'll begin at Berkeley time (10 minutes after), as per tradition!

Potpourri

5 min

- Announcements
- Provenance
- Quiz 2 Survey
- Computing in the News

Announcements

- **Midterm exam is next Mon 7/13 from 7-9 pm**: Ed logistics/scope post hopefully tomorrow
- Cats project has been released!
  - **Start now!** You have a shorter time to work on this project

Provenance

- You are required to use the new **Provenance** VS Code extension for Cats
  - See the spec for instructions on how to install and use the extension
  - You can go back to whatever IDE you prefer for all other assignments
- We're piloting this new tool for academic integrity purposes. However, since this is a pilot **we will not use the data collected by the tool for any academic integrity penalties this semester.**
- The design is intentionally as privacy-preserving as possible, e.g.:
  - When the extension is active, it only writes logs to your local computer and makes 0 network requests (**all data stays on your device until you upload the Provenance bundle to Gradescope**)
  - The **logs are in plaintext** and publicly viewable by you
- If you run into issues or have questions about Provenance, please direct them to our academic integrity TA, Sultan (sultan.muratbek@berkeley.edu) (or make an Ed post/come to OH)

Quiz 2 Survey

PollEv

(anonymous, does not affect your grade)

Computing in the News

- You can now customize Siri's pace and expressivity in the latest iOS 27 beta (TechCrunch)
- Discord accidentally banned over 8,000 people for posting grids and other ‘benign’ images (The Verge)
- SpaceX just launched the 1st-ever nuclear-powered commercial satellite (Space.com)

Review

5 min

- A note on dictionary iteration
- Trees

A note on dictionary iteration

While you can iterate through a dictionary using `.keys()`, `.values()`, and `.items()`, you can also iterate through its keys by directly putting it in a `for` loop:

```
>>> roman_numerals = {'I': 1, 'V': 5, 'X': 10}
>>> for key in roman_numerals:
...     print(key)
...
I
V
X
```

When poll is active respond at **PollEv.com/rebeccadang831**

Activate this Poll with the Poll Everywhere Live app.

If video conferencing, you must be sharing your full screen to share the activated poll.

Which variation of the print_tree function would result in the print output 1, 2, 4, 5, 3, 6, 7 (one per line) for the given tree?

v1.3.0

Iterators

20 min

- Definitions
- Examples
- Iterators vs. Iterables
- Why use iterators?
- Built-In Functions for Iteration

Definitions

- An **iterable** is any object that can be processed sequentially. In other words, you can put it in a `for` loop.
  - Ex: Lists, tuples, dictionaries, strings, ranges
- An **iterator** provides access to the iterable's elements in some order.
  - You can create an iterator by calling `iter` on the iterable.
  - You can access the next element by calling `next` on the iterator.
- **Analogy: Iterables are like books, iterators are like bookmarks.**

Examples (1 of 2)

```
>>> lst = [1, 2, 3]  # iterable
>>> tracker = iter(lst)  # iterator
>>> next(tracker)
1
>>> next(tracker)
2
>>> next(tracker)
3
>>> next(tracker)
StopIteration
```

Examples (2 of 2)

```
>>> lst = [1, 3, 2, 7]
>>> iter1 = iter(lst)
>>> next(iter1)
1
>>> lst.append(5)
>>> next(iter1)
3
>>> iter2 = iter(lst)
>>> next(iter2)
1
>>> lst[1] = -10
>>> next(iter2)
-10
```

When poll is active respond at **PollEv.com/rebeccadang831**

Activate this Poll with the Poll Everywhere Live app.

If video conferencing, you must be sharing your full screen to share the activated poll.

Dictionary iterator: What would Python display?

When poll is active respond at **PollEv.com/rebeccadang831**

Activate this Poll with the Poll Everywhere Live app.

If video conferencing, you must be sharing your full screen to share the activated poll.

Range iterator: What would Python display? (one per line)

When poll is active respond at **PollEv.com/rebeccadang831**

Activate this Poll with the Poll Everywhere Live app.

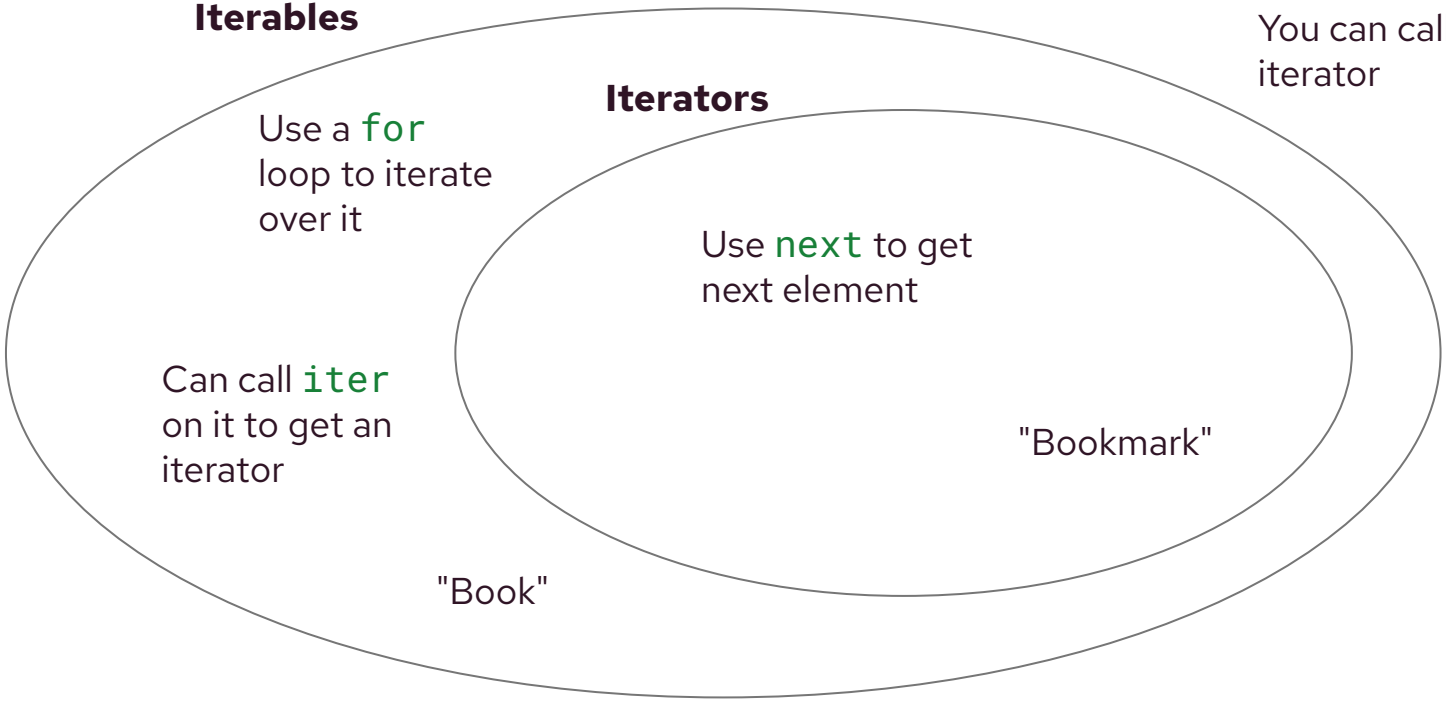
If video conferencing, you must be sharing your full screen to share the activated poll.

iter of an iterator: What would Python display?

Iterators vs. Iterables

All iterators are iterables, but not all iterables are iterators!

You can call `iter` on an existing iterator



Why use iterators?

- Iterators are an additional layer of **abstraction** on top of the iterable
  - Since we make no assumptions about the underlying implementation of the iterable, the programmer can change things internally without breaking downstream code!
- Iterators bundle together the elements of the iterable and the position in the same object
  - Passing the iterator to another function retains the position
  - Useful for **ensuring that each element is only processed once**
  - **Prevents mutation of the original iterable** (you can only call `next` on the iterator)

Built-In Functions for Iteration (1 of 4)

Many built-in Python functions return iterators that compute results **lazily** (e.g. compute values as needed instead of computing everything at once).

`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 (can zip more than 2 iterables, see behavior below)
`reversed(sequence)`	Iterate over `x` in a sequence in reverse order

```
>>> list(zip([1, 2], [3, 4]))
[(1, 3), (2, 4)]
>>> list(zip([1, 2], [3, 4, 5], [6, 7]))
[(1, 3, 6), (2, 4, 7)]
```

Built-In Functions for Iteration (2 of 4)

To view the full 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 (Can optionally take in a `key` function like `min` and `max`)

When poll is active respond at **PollEv.com/rebeccadang831**

Activate this Poll with the Poll Everywhere Live app.

If video conferencing, you must be sharing your full screen to share the activated poll.

Reversed: What would Python display?

Built-In Functions for Iteration (3 of 4)

```
>>> def double(x):
...     print(f'** {x} => {2 * x} **')
...     return 2 * x
...
>>> f = lambda x: x >= 10
>>> t = filter(f, map(double, range(3, 7)))
>>> next(t)
** 3 => 6 **
** 4 => 8 **
** 5 => 10 **
10
```

Built-In Functions for Iteration (4 of 4)

```
>>> next(t)
** 6 => 12 **
12
>>> list(t)
[]
```

Generators

20 min

- Definitions
- Examples
- Infinite Generators
- `yield from`
- Iterators vs. iterables vs. generators
- Yield Partitions

Definitions

A **generator function** is a function that **yields** value(s).

A **generator object** is the return value of a generator function. You can call `next` on a generator object to get the yielded values in order.

Caution: Unfortunately, you might see sometimes in problems that we refer to these things interchangeably even though they aren't. If it's not clear from the instructions, ask for clarification (midterm/final) or report an issue (quizzes).

Examples

```
>>> def plus_minus(x):
...     yield x
...     yield -x
...
>>> plus_minus(5)
<generator object ...>
>>> gen = plus_minus(5)
>>> next(gen)
5
>>> next(gen)
-5
>>> next(gen)
StopIteration
```

Infinite Generators

```
>>> def evens():
...     i = 0
...     while True:
...         yield i
...         i += 2
...
>>> gen = evens()
>>> next(gen)
0
>>> next(gen)
2
>>> list(gen)
<infinite loop occurs>
```

When poll is active respond at **PollEv.com/rebeccadang831**

Activate this Poll with the Poll Everywhere Live app.

If video conferencing, you must be sharing your full screen to share the activated poll.

True or false: A function must either yield or return, but not both.

yield from (1 of 3)

```
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(a_then_b([1, 2], [3, 4]))
[1, 2, 3, 4]
```

`yield from` (2 of 3)

A common reason for using `yield from` is to call generator functions recursively.

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


>>> list(countdown(3))
[3, 2, 1, 'Blastoff!']
```

yield from (3 of 3)

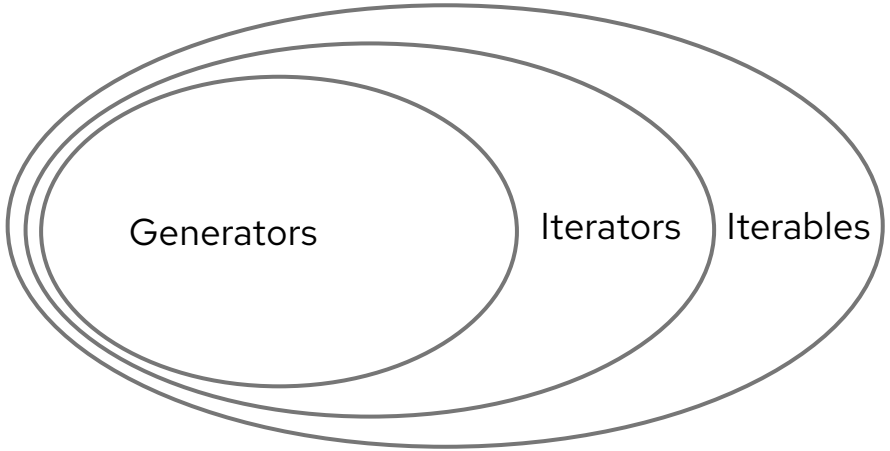
What if we didn't use `yield from` and just used `yield`?

```
def countdown(k):
    if k > 0:
        yield k
        yield countdown(k - 1)
    else:
        yield 'Blastoff!'


>>> list(countdown(3))
[3, <generator object ...>]
```

Iterables vs. iterators vs. generators

- **`iter(iterable)` → iterator**
- **`iter(iterator)` → iterator** (same object as the one passed in)
- Both **iterators** and **generators** are a type of iterable because we can iterate over them.



Yield Partitions (1 of 2)

Recall: The number of partitions of a positive integer `n`, using parts up to size `m`, is the number of ways in which `n` can be expressed as the sum of positive integer parts up to `m` in increasing order. For example, there are 9 partitions of 6 using parts up to 4.

1. 6 = 2 + 4
2. 6 = 1 + 1 + 4
3. 6 = 3 + 3
4. 6 = 1 + 2 + 3
5. 6 = 1 + 1 + 1 + 3
6. 6 = 2 + 2 + 2
7. 6 = 1 + 1 + 2 + 2
8. 6 = 1 + 1 + 1 + 1 + 2
9. 6 = 1 + 1 + 1 + 1 + 1 + 1

Yield Partitions (2 of 2)

Switch to VS Code demo

You can follow along by downloading the `10-partitions.py` file!

5 min break

Practice

(reminder to self to use high contrast theme)

25 min

- Palindrome
- `min_abs_indices`
- Yield Paths

Practice: Palindrome

- Implement `palindrome`. Challenge: Implement this 2 ways:
  - (1) Using `reversed`
  - (2) Using `zip` and `reversed`
- Download starter code `10.py` from the course website under Lecture 10
- Run doctests with `python3 -m doctest 10.py`
- For time, let's do this together!

Practice: `min_abs_indices`

- Implement `min_abs_indices`. Challenge: Implement this 2 ways:
  - (1) With a list comprehension
  - (2) Without a list comprehension (e.g. only use built-in functions for iteration)
- Download starter code `10.py` from the course website under Lecture 10
- Run doctests with `python3 -m doctest 10.py`
- 5 min: Try implementing it yourself
- 5 min: Discuss with someone next to you and compare your implementations
  - What worked?
  - What didn't?
  - Why?

Practice: Yield Paths

- Implement `yield_paths`
  - Hint 1: Read the doctests carefully!
  - Hint 2: The tree `t` in the doctests is pictured to the right
  - Hint 3: How can you use the solution from `count_paths` in `09-sol.py`?
- Download starter code `10.py` from the course website under Lecture 10
- Run doctests with `python3 -m doctest 10.py`
- 5 min: Try implementing it yourself
- 5 min: Discuss with someone next to you and compare your implementations
  - What worked?
  - What didn't?
  - Why?