UC Berkeley EECS
Lecturer
Michael Ball

Iterators and Generators

Announcements

- We'll see what happens with the strike.
- **PLEASE Support your GSI's decision!**
  - It is *their* right to strike and not everyone may make the decision.
  - Please do so for CS88 and your other courses. ☺
- CS88 Lectures will still happen.
  - I am actually *not* allowed to cancel class!
  - I'll do my best to keep things humming along.
  - but some things will break.
  - That's OK!
- Our GSIs are amazing, they deserve out support!!

Today:

- Pick up where we left off!
- Iterators – the iter protocol
- Getitem protocol
- Is an object iterable?
- Lazy evaluation with iterators

Generators: turning iteration into an iterable

- *Generator* functions use iteration (for loops, while loops) and the yield keyword
- Generator functions have no return statement, but they don't return None
- They implicitly return a generator object
- Generator objects are just iterators

```
def squares(n):
    for i in range(n):
        yield (i*i)
```

Next element in generator iterable

- Iterables work because they have some "magic methods" on them. We saw magic methods when we learned about classes,
- e.g., __init__, __repr__ and __str__.
- The first one we see for iterables is `__next__`

- `iter()` – transforms a sequence into an iterator
  - `list_iter = iter([1, 2, 3, 4])`
  - `next(list_iter)`

Iterators: The iter protocol

- In order to be *iterable*, a class must implement the iter protocol
- The iterator objects themselves are required to support the following two methods, which together form the iterator protocol:
  - `__iter__()` : Return the iterator object itself. This is required to allow both containers and iterators to be used with the for and in statements.
    - » This method returns an iterator object (which can be `self`)
  - `__next__()` : Return the next item from the container. If there are no further items, raise the `StopIteration` exception.
- Classes get to define how they are iterated over by defining these methods
  - containers (objects like lists, tuples, etc) typically define a Container class and a separate `ContainterIterator` class.

Get Item protocol – Build a Sequene

- Another way an object can behave like a **sequence** is indexing: Using square brackets "[ ]" to access specific items in an object.
- Defined by special method: `__getitem__(self, i)`
  - Method returns the item at a given index

```
class myrange2:
    def __init__(self, n):
        self.n = n

    def __getitem__(self, i):
        if i >= 0 and i < self.n:
            return i
        else:
            raise IndexError

    def __len__(self):
        return self.n
```

Determining if an object is iterable

- from collections.abc import Iterable
- isinstance([1,2,3], Iterable)

- This is more general than checking for any list of particular type, e.g., list, tuple, string...

What can we do now?

- Build our own `for`-loop like functions!

What's the Big Picture?

- We have new tools for building data structures that behave sequences
- We can handle "infinite" streams of data.
- We can build our own for loops, perhaps custom for loops.

If Extra Time – What happens when we yield?