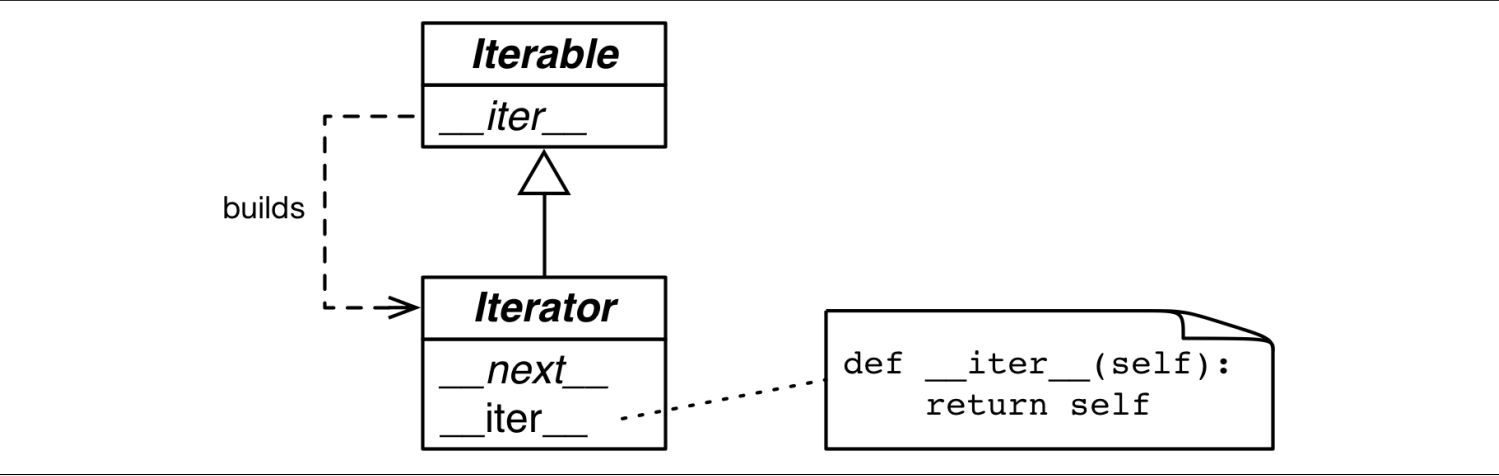
1. If the data doesn’t fit in memory, we need to fetch the items lazily—one at a time and on demand. Every standard collection in Python is iterable. An iterable is an object that provides an iterator, which Python uses to support operations like: • for loops • List, dict, and set comprehensions • Unpacking assignments • Construction of collection instances
2. Whenever Python needs to iterate over an object x, it automatically calls iter(x). The iter built-in function:
   1. Checks whether the object implements \_\_iter\_\_, and calls that to obtain an iterator.
   2. If \_\_iter\_\_ is not implemented, but \_\_getitem\_\_ is, then iter() creates an iter‐ ator that tries to fetch items by index, starting from 0 (zero).
   3. If that fails, Python raises TypeError, usually saying 'C' object is not itera ble, where C is the class of the target object.

That is why all Python sequences are iterable: by definition, they all implement \_\_getitem\_\_. In fact, the standard sequences also implement \_\_iter\_\_, and yours should too, because iteration via \_\_getitem\_\_ exists for backward compatibility and may be gone in the future—although it is not deprecated as of Python 3.10, and I doubt it will ever be removed.

1. If a class provides \_\_getitem\_\_, the iter() built-in accepts an instance of that class as iterable and builds an iterator from the instance. Python’s iteration machinery will call \_\_getitem\_\_ with indexes starting from 0, and will take an IndexError as a signal that there are no more items.
2. In the goose-typing approach, the definition for an iterable is simpler but not as flexi‐ ble: an object is considered iterable if it implements the \_\_iter\_\_ method. No sub‐ classing or registration is required, because abc.Iterable implements the \_\_subclasshook\_\_
3. As of Python 3.10, the most accurate way to check whether an object x is iterable is to call iter(x) and handle a TypeError excep‐ tion if it isn’t. This is more accurate than using isinstance(x, abc.Iterable), because iter(x) also considers the legacy \_\_getitem\_\_ method, while the Iterable ABC does not.
4. Explicitly checking whether an object is iterable may not be worthwhile if right after the check you are going to iterate over the object. After all, when the iteration is attempted on a noniterable, the exception Python raises is clear enough: TypeError: 'C' object is not iterable. If you can do better than just raising TypeError, then do so in a try/except block instead of doing an explicit check. The explicit check may make sense if you are holding on to the object to iterate over it later; in this case, catching the error early makes debugging easier.
5. If an object has an \_\_iter\_\_ the result of evaluation would be: isinstance(object, abs.Iterable) will be true. But if an iterable doesn’t have \_\_iter\_\_ and instead have \_\_getitem\_\_ it will be false although it’s an iterable.
6. How to iterate over a function? Using iter with a Callable. If you want to iterate a function and call it repeatedly this is the answer: d1 = iter(function, sentinel). Sentinel means the place that the iteration must stop at.



1. Because the only methods required of an iterator are \_\_next\_\_ and \_\_iter\_\_, there is no way to check whether there are remaining items, other than to call next() and catch StopIteration. Also, it’s not possible to “reset” an iterator. If you need to start over, you need to call iter() on the iterable that built the iterator in the first place. Calling iter() on the iterator itself won’t help either
2. iterators are supposed to implement both \_\_next\_\_ and \_\_iter\_\_, and doing so makes our iterator pass the issubclass(SentenceIterator, abc.Iterator) test.

11. It may be tempting to implement \_\_next\_\_ in addition to \_\_iter\_\_ in the Sentence class, making each Sentence instance at the same time an iterable and iterator over itself. But this is rarely a good idea. To “support multiple traversals,” it must be possible to obtain multiple independent iterators from the same iterable instance, and each iterator must keep its own internal state, so a proper implementation of the pattern requires each call to iter(my\_itera ble) to create a new, independent, iterator