THE `iter()` FUNCTION

What happens when Python performs an iterationon over an iterable?

The very first thing Python does is call the `iter()` function on the object we want to iterate

If the object implements the `__iter__` method, that method is called

and Python uses the returned iterator

What happens if the object does not implement the `__iter__` method?

Is an exception raised immediately?

Sequence Types

So how does iterating over a sequence type – that maybe only implemented `__getitem__` work?

I just said that Python always calls `iter()` first

You'll notice I did not say Python always calls the `__iter__` method

I said it calls the `iter()` function!!

In fact, if `obj` is an object that only implements `__getitem__`

`iter(obj)` → returns an iterator type object!

Some form of magic at work?

Not really!

Let's think about sequence types and how we can iterate over them

Suppose `seq` is some sequence type that implements `__getitem__` (but not `__iter__`)

Remember what happens when we request an index that is out of bounds from the `__getitem__` method? → `IndexError`

```
index = 0

while True:
    try:
        print(seq[index])
        index += 1
    except IndexError:
        break
```

Making an Iterator to iterate over any Sequence

This is basically what we just did!

```
class SeqIterator:
    def __init__(self, seq):
        self.seq = seq
        self.index = 0

    def __iter__(self):
        return self

    def __next__:
        try:
            item = self.seq[self.index]
            self.index += 1
            return item
        except IndexError:
            raise StopIteration()
```

Calling `iter()`

So when `iter(obj)` is called:

Python first looks for an `__iter__` method

→ if it's there, use it

→ if it's not

look for a `__getitem__` method

→ if it's there create an iterator object and return that

→ if it's not there, raise a `TypeError` exception (not iterable)

Testing if an object is iterable

Sometimes (very rarely!)

you may want to know if an object is iterable or not

But now you would have to check if they implement

`__getitem__` or `__iter__`

and that `__iter__` returns an iterator

Easier approach:

```
try:
    iter(obj)
except TypeError:
    # not iterable
    <code>
else:
    # is iterable
    <code>
```

Code Exercises