Custom Sequence Types

Part 1

Creating our own Sequence types

We will cover Abstract Base Classes later in this course, so we'll revisit this topic again

At it's most basic, an immutable sequence type should support two things:

returning the length of the sequence *(technically, we don't even really need that!)*

given an index, returning the element at that index

If an object provides this functionality, then we should in theory be able to:

retrieve elements by index using square brackets [ ]

iterate through the elements using Python's native looping mechanisms

e.g. for loops, comprehensions

How Python does it

Remember that sequence types are iterables, but not all iterables are sequence types

Sequence types, at a minimum, implement the following methods:

`__len__` `__getitem__`

At its most basic, the `__getitem__` method takes in a single integer argument – the index

However, it may also choose to handle a `slice` type argument

So how does this help when iterating over the elements of a sequence?

The `__getitem__` method

The `__getitem__` method should return an element of the sequence based on the specified index

or raise an `IndexError` exception if the index is out of bounds

(and may, but does not have to, support negative indices and slicing)

Python's `list` object implements the `__getitem__` method:

```
my_list = ['a', 'b', 'c', 'd', 'e', 'f']

my_list.__getitem__(0) → 'a'
my_list.__getitem__(1) → 'b'
my_list.__getitem__(-1) → 'f'
my_list.__getitem__(slice(None, None, -1))
    → ['f', 'e', 'd', 'c', 'b', 'a']
```

The `__getitem__` method

But if we specify an index that is out of bounds:

`my_list.__getitem__(100)` → `IndexError`

`my_list.__getitem__(-100)` → `IndexError`

All we really need from this `__getitem__` method is the ability to

- return an element for a valid index
- raise an `IndexError` exception for an invalid index

Also remember, that sequence indices start at `0`

- i.e. we always know the index of the first element of the sequence

Implementing a `for` loop

So now we know: sequence indexing starts at `0`

`__getitem__(i)` will return the element at index `i`

`__getitem__(i)` will raise an `IndexError` exception when `i` is out of bounds

```
my_list = [0, 1, 2, 3, 4, 5]

for item in my_list:
    print(item ** 2)
```

```
index = 0

while True:
    try:
        item = my_list.__getitem__(index)
    except IndexError:
        break
    print(item ** 2)
    index += 1
```

The point is that if the object implements `__getitem__`

we can iterate through it using a `for` loop, or even a comprehension

The __len__ Method

In general sequence types support the Python built-in function `len()`

To support this all we need to do is implement the `__len__` method in our custom sequence type

```
my_list = [0, 1, 2, 3, 4, 5]
```

```
len(my_list) → 6
my_list.__len__() → 6
```

Writing our own Custom Sequence Type

to implement our own custom sequence type we should then implement:

- `__len__`
- `__getitem__`

At the very least `__getitem__` should:

- return an element for a valid index `[0, length-1]`
- raise an `IndexError` exception if index is out of bounds

Additionally we can choose to support:

- negative indices — `i < 0` → `i = length - i`
- slicing — handle `slice` objects as argument to `__getitem__`

Code Exercises