LAZY ITERABLES

Lazy Evaluation

This is often used in class properties

properties of classes may not always be populated when the object is created

value of a property only becomes known when the property is requested - deferred

Example

```
class Actor:
    def __init__(self, actor_id):
        self.actor_id = actor_id
        self.bio = lookup_actor_in_db(actor_id)
        self.movies = None

    @property
    def movies(self):
        if self.movies is None:
            self.movies = lookup_movies_in_db(self.actor_id)
        return self.movies
```

Application to Iterables

We can apply the same concept to certain iterables

We do not calculate the next item in an iterable until it is actually requested

Example

iterable → `Factorial(n)`

- will return factorials of consecutive integers from `0` to `n-1`
- do not pre-compute all the factorials
- wait until `next` requests one, then calculate it

This is a form of lazy evaluation

Application to Iterables

Another application of this might be retrieving a list of forum posts

`Posts` might be an iterable

- each call to `next` returns a list of 5 posts (or some page size)
- but uses lazy loading

→ every time `next` is called, go back to database and get next 5 posts

Application to Iterables → Infinite Iterables

Using that lazy evaluation technique means that we can actually have infinite iterables

Since items are not computed until they are requested

we can have an infinite number of items in the collection

Don't try to use a for loop over such an iterable
unless you have some type of exit condition in your loop

→ otherwise infinite loop!

Lazy evaluation of iterables is something that is used a lot in Python!

We'll examine that in detail in the next section on generators

Code Exercises