CHAINING AND TEEING

Chaining Iterables `itertools.chain(*args)` → lazy iterator

This is analogous to sequence concatenation

but not the same!

→ dealing with iterables (including iterators)

→ chaining is itself a lazy iterator

We can manually chain iterables this way: `iter1 iter2 iter3`

```
for it in (iter1, iter2, iter3):
    yield from it
```

Or, we an use `chain` as follows:

```
for item in chain(iter1, iter2, iter3):
    print(item)
```

Variable number of positional arguments – each argument must be an iterable

Chaining Iterables

What happens if we want to chain from iterables contained inside another, single, iterable?

```
l = [iter1, iter2, iter3]

chain(l)  → l
```

What we really want is to chain `iter1`, `iter2` and `iter3`

We can try this using unpacking: `chain(*l)`

→ produces chained elements from `iter1`, `iter2` and `iter3`

BUT unpacking is eager – not lazy!

If `l` was a lazy iterator, we essentially iterated through `l` (not the sub iterators), just to unpack!

This could be a problem if we really wanted the entire chaining process to be lazy

Chaining Iterables `itertools.chain.from_iterable(it)` → lazy iterator

We could try this approach:

```
def chain_lazy(it):
    for sub_it in it:
        yield from sub_it
```

Or we can use `chain.from_iterable`

```
chain.from_iterable(it)
```

This achieves the same result

→ iterates lazily over `it`

→ in turn, iterates lazily over each iterable in `it`

"Copying" Iterators `itertools.tee(iterable, n)`

Sometimes we need to iterate through the same iterator multiple times, or even in parallel

We could create the iterator multiple times manually

```
iters = []
for _ in range(10):
    iters.append(create_iterator())
```

Or we can use `tee` in `itertools`

→ returns independent iterators in a tuple

`tee(iterable, 10)` → `(iter1, iter2, …, iter10)`

all different objects

Teeing Iterables

One important to thing to note

The elements of the returned tuple are lazy iterators

→ always!

→ even if the original argument was not

```
l = [1, 2, 3, 4]

tee(l, 3)    → (iter1, iter2, iter3)
```

all lazy iterators

not lists!

Coding Exercises