SELECTING AND FILTERING

The filter function

You should already be familiar with the filter function → `filter(predicate, iterable)`

→ returns all elements of `iterable` where `predicate(element)` is `True`

`predicate` can be `None` – in which case it is the identity function `f(x) → x`

→ in other words, truthy elements only will be retained

→ `filter` returns a lazy iterator

We can achieve the same result using generator expressions:

```
(item for item in iterable if pred(item))
```

predicate is not `None`

```
(item for item in iterable if item)
```

or

```
(item for item in iterable if bool(item))
```

predicate is `None`

Example

```
filter(lambda x: x < 4, [1, 10, 2, 10, 3, 10])        → 1, 2, 3

filter(None, [0, '', 'hello', 100, False])            → 'hello', 100
```

→ remember that `filter` returns a (lazy) iterator

itertools.filterfalse

This works the same way as the `filter` function

but instead of retaining elements where the predicate evaluates to `True`

it retains elements where the predicate evaluates to `False`

Example

```
filterfalse(lambda x: x < 4, [1, 10, 2, 10, 3, 10])     → 10, 10, 10


filterfalse(None, [0, '', 'hello', 100, False])          → 0, '', False
```

→ `filterfalse` returns a (lazy) iterator

`itertools.compress`

No, this is not a compressor in the sense of say a zip archive!

It is basically a way of filtering one iterable, using the truthiness of items in another iterable

```
     data = ['a',   'b', 'c', 'd', 'e']
              ↕       ↕    ↕    ↕    ↕
selectors = [True, False, 1,   0]   None
```

```
compress(data, selectors)   → a, c
```

→ `compress` returns a (lazy) iterator

`itertools.takewhile`

```
takewhile(pred, iterable)
```

The `takewhile` function returns an iterator that will yield items while `pred(item)` is Truthy

→ at that point the iterator is exhausted

even if there are more items in the iterable whose predicate would be truthy

```
takewhile(lambda x: x < 5, [1, 3, 5, 2, 1])
```
→ `1, 3`

→ `takewhile` returns a (lazy) iterator

`itertools.dropwhile`

```
dropwhile(pred, iterable)
```

The `dropwhile` function returns an iterator that will start iterating (and yield all remaining elements) once `pred(item)` becomes Falsy

```
dropwhile(lambda x: x < 5, [1, 3, 5, 2, 1])
```
→ `5, 2, 1`

→ `dropwhile` returns a (lazy) iterator

Code Exercises