Lab 6: Generators

Learning objectives:

- Understand how generators work
- Gain experience writing generators in Python

You will make use of generators in homework 2, and as such the aim of this lab is to give you practice using them. You may **not** use any built-in higher-order iterators (e.g. `map`, `zip`, `filter`) on this lab.
Use the following commands to download and unpack the distribution code:

```
$ wget https://eecs390.github.io/lab/lab06/starter-files.tar.gz
$ tar xzf starter-files.tar.gz
```

1. *The map pattern.* Implement the `map_nary()` generator, which takes in a function and any number of iterables and yields the n-way mapping of the given function over the items in the given iterables. The given function must be invocable on the same number of arguments as the number of iterables passed in.

   Note: In Python, the `**` operator can be used to raise a number to a power (e.g. `2 ** 3`).

```
>>> m = map_nary(
...     lambda x, y: y ** x,
...     [1, 2, 3],
...     [6, 7, 8]
... )
>>> next(m) # produces 6 ** 1
6
>>> next(m) # produces 7 ** 2
49
>>> next(m) # produces 8 ** 3
512
>>> next(m)
Traceback (most recent call last):
...
StopIteration
```

   The generated sequence should end as soon as any of the iterables passed in raises a `StopIteration`.

```
>>> concat = map_nary(
...     lambda a, b, c: a + b + c,
...     ['hello', 'world'],
...     ['foo', 'bar'],
...     ['baz']
... )
>>> next(concat)
'hellofoobaz'
>>> next(concat)
Traceback (most recent call last):
...
StopIteration
```

   Starter code for this problem can be found in `generators.py`. To run the doctests on your implementation:

```
$ python3 -m doctest generators.py
```

2. *The zip pattern.* Implement the `zippy()` generator, which takes in any number of iterables and yields tuples of the given iterables' elements.

   ```
   >>> def naturals():
   ...     num = 0
   ...     while True:
   ...         yield num
   ...     num += 1
   >>> offset = naturals()
   >>> next(offset)
   0
   >>> nats = zippy(naturals(), offset, naturals())
   >>> next(nats)
   (0, 1, 0)
   >>> next(nats)
   (1, 2, 1)
   >>> next(nats)
   (2, 3, 2)
   ```

   The generated sequence should end as soon as any of the given iterables raises a StopIteration.

   ```
   >>> zipped = zippy([8, 9, 10])
   >>> next(zipped)
   (8,)
   >>> next(zipped)
   (9,)
   >>> next(zipped)
   (10,)
   >>> next(zipped)
   Traceback (most recent call last):
   ...
   StopIteration
   ```

   Starter code for this problem can be found in `generators.py`. To run the doctests on your implementation:

   ```
   $ python3 -m doctest generators.py
   ```

3. *The filter pattern.* Implement the `filter_multiple()` generator, which takes in an iterable and any number of predicate functions and yields an element from that iterable only if **all** predicates return True when called on that element.

   ```
   >>> def naturals():
   ...     num = 0
   ...     while True:
   ...         yield num
   ...     num += 1
   >>> evens_over_ten = filter_multiple(
   ...     naturals(),
   ...     lambda n: n > 10,
   ...     lambda n: n % 2 == 0
   ... )
   >>> next(evens_over_ten)
   12
   >>> next(evens_over_ten)
   14
   >>> next(evens_over_ten)
   16
   ```

   If `any=True` is provided as a keyword argument, an element should be yielded if **any** of the predicates return True when called on that element.

```
>>> multiples_of_3_or_5 = filter_multiple(
...     naturals(),
...     lambda n: n % 3 == 0,
...     lambda n: n % 5 == 0,
...     any=True
... )
>>> next(multiples_of_3_or_5)
0
>>> next(multiples_of_3_or_5)
3
>>> next(multiples_of_3_or_5)
5
>>> next(multiples_of_3_or_5)
6
```

Starter code for this problem can be found in `generators.py`. To run the doctests on your implementation:

```
$ python3 -m doctest generators.py
```