CS2204 Homework: Baking Pizzas ¶

Objectives

- Understand Python generators
- Build a supply chain model
- Work with physical time
- Eat more pizzas

Background

In this assignment, you are building a supply chain model for *PizzaPy, Co.* using Python generators. Each ingredient and the intermediate workflow steps will be implemented as generator functions. The goal of this exercise is to have a better understanding of how iteration works in Python and how iterators *pull* data from each other on-demand. The ingredients and intermediate products are measured in simple (fictional) quantities. E.g., a pizza *dough* will need one *flour,* one *yeast,* one *salt_, and one _water.*

Tasks

You need to finish the `pizza.py` source file. Note: you can use the simple test code provided at the end of the file (by uncommenting those lines). This example shows you how to use all the generator functions together. You may want to start with a smaller subset of this testing code by using and iterating over your generators separately (e.g., just do an exercise with the `flour_sack()` generator by using it in a `for` loop as shown below). Feel free to change/add/delete any testing code. The outputs of this testing code are not going to be graded. The validator program will use your code directly.

1. Implement the `flour_sack` generator function. This should be a relatively simple exercise. This generator should produce a finite amount of `"flour"` strings. The number of generated strings is provided as a parameter for the generator function. The expected behavior of this function is this:

   ```
   for flour in flour_sack(5):
       print(flour)
   ```

   This should print the following:

   ```
   flour
   flour
   flour
   flour
   flour
   ```

2. Implement the `yeast_jar`, `salt_shaker`, `sauce_container` generator functions. These are all the same as the previous one, except for the generated strings (see the source code documentation strings for details). Don't feel bad by using copy & paste this time.

3. Implement the `water_faucet` generator function. This is even simplier because we have an *unlimited* supply of water resources (this pizza chain is not in California).
4. Implement the `dough_maker` generator function. This function receives four iterators (created by the generator functions, above) and should produce a `"dough"` string by consuming exactly one element from each of these iterators. *Hint:* you may find the `zip()` function with a `for` loop very useful to shorten your code. If you run out of any of the ingredients, you should stop generating `"dough"`.
5. Implement the `cheese_grater` generator function. Living in an ideal world, we also have *unlimited* cheese. However, there is a *throughput* limit on how fast we can make shredded cheese. The `throughput` parameter of this function tells you how many `"cheese"` strings you are allowed to generate in a single second. Initially, you can produce up-to `throughput` number of elements, but after that, you should generate a new element if in the past 1.0 second less then `throughput` number were produced. If this is not the case, you should wait (`sleep`) in this generator to maintain the rate. *Hint:* you definitely want to use the `time.time()` and `time.sleep()` functions, and you also need to keep track of the producing times of the `"cheese"` items at least for the last second (e.g., in a list). **Note:** this is the hardest part of the assignment. You will be given (generous) partial credit if you cannot solve the rate control problem.
6. Implement the `pizza_preparator` generator function, which is very similar to `dough_maker`, but requires a `dough`, a `sauce`, and a `cheese` iterator and should produce `"raw_pizza"` strings. Also, because we have an unlimited amount of cheese (albeit at a limited rate), the pizza preparator should use **5** cheese elements for each raw pizza. Yum.
7. Finally, implement the `oven` generator function, which consumes `raw_pizza` items and generates `"pizza"` strings. There is a single parameter for this function (`baking_time`). The generator should wait for this amount of seconds before producing a pizza from a raw pizza. Again, you should use the `time.sleep()` function, but the code should be much simpler than for `cheese_grater`.

Hints

If you decide to use the `next()` function inside one of your generators (you don't need to do so to solve the assignment), there is an important caveat. If you are already in a generator, you cannot simply rely on the fact that the `next()` function call on another iterator will generate a `StopIteration` by itself, and it would be automatically propagated from your generator, too (a detailed, but probably too technical explanation can be found here (https://www.python.org/dev/peps/pep-0479/)). So, either avoid using `next()`, or use it with the following way (try/except block) inside a generator:

```
def example_generator(another_iterator):
    while True:
    try:
        item = next(another_iterator)
    except StopIteration:
        return
```

This code transforms a `StopIteration` exception to a simple `return` from your generator, ending the iteration.

When you need to consume more than one element from an iterator, the `itertools.islice` function may be useful. For example, to consume every 7th element from a `range()` iterator, you can use this code:

```
from itertools import islice
for item in islice(range(30), 6, None, 7):
    print(item)
```

this should print

```
6
13
20
27
```

The meaning of `islice(range(30), 6, None, 7)` is that you want to take every `7th` element from `range(30)` starting at index `6`. It is important to always get the last element (`6`) of each seven-element segment to make sure you won't use a *partial* segment at the end.

Grading

You can use the attached `validator.py` program to check your work (and the instructor's original mistakes). It will also estimate your final score for the homework. The program is in the same folder as your homework assignment. Open the `validator.py` script and run it in the Spyder environment to track your progress.

Penalties

Points will be deducted if you fail to set `__author__` variable (-10 pts) and for **each PEP 8 style errors** (-1 pt for each) in your program.

Submission

Please, upload the final version of the following file(s) (**and only those files**) to Brightspace:

- `pizza.py`