Open in app

Following

596K Followers

This is your **last** free member-only story this month. Upgrade for unlimited access.

HANDS-ON TUTORIALS

Simulate Real-life Events in Python Using SimPy

Simulate a Restaurant with Hungry Customers and a Limited Supply of Food

Khuyen Tran 4 days ago · 6 min read ★

Motivation

As a restaurant manager, you want to approximate how much food your restaurant needs to prepare for tomorrow. You know how many customers come to your restaurant per day and the average time it takes to serve one customer. However, it is challenging to put all of these variables into one calculation.

Wouldn't it be great if you can simulate this event using Python?

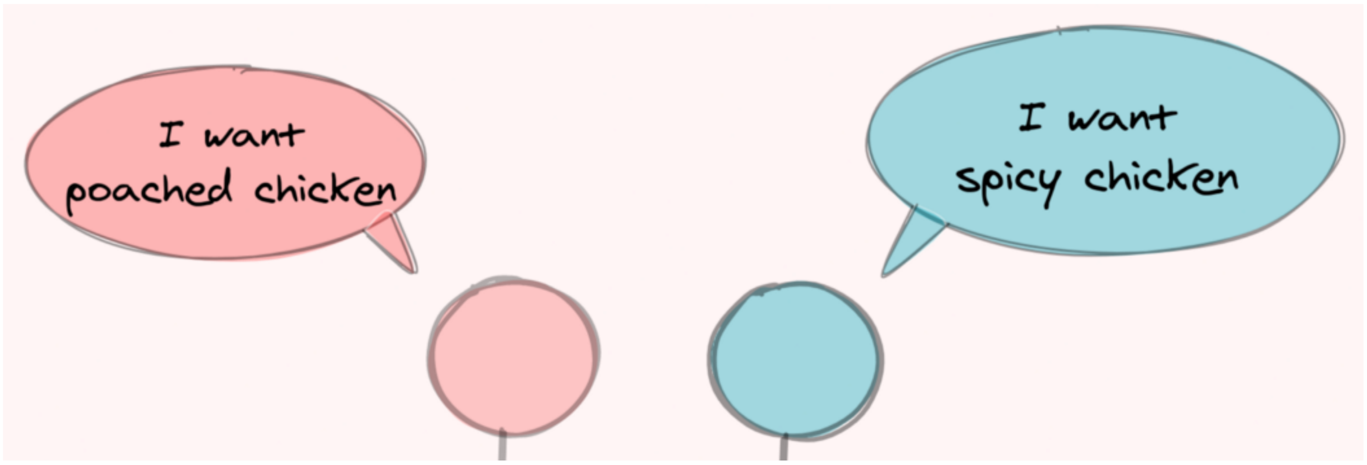


Image by Author

That is when SimPy comes in handy.

What is SimPy?

SimPy is a Python library that enables you to simulate real-life events. It can model active components such as customers, vehicles, or agents.

To install SimPy, type:

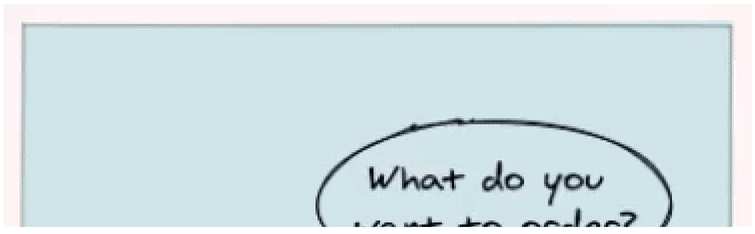
```
pip install simpy
```

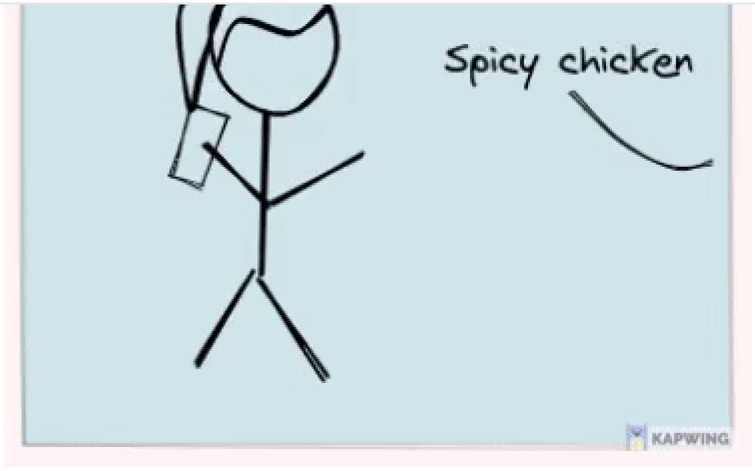
Get Started — Simulate Waiters

In SimPy, the behaviors of active components such as customers or vehicles are modeled with *processes*. These processes live in an *environment*. They interact with the environment and with each other via *events*.

To understand these concepts better, let's try to model a waiter (a process) who serves in a restaurant (an environment). Some basic things that waiters do are:

- Taking orders from customers
- Giving orders to the cooks
- Serving food to customers





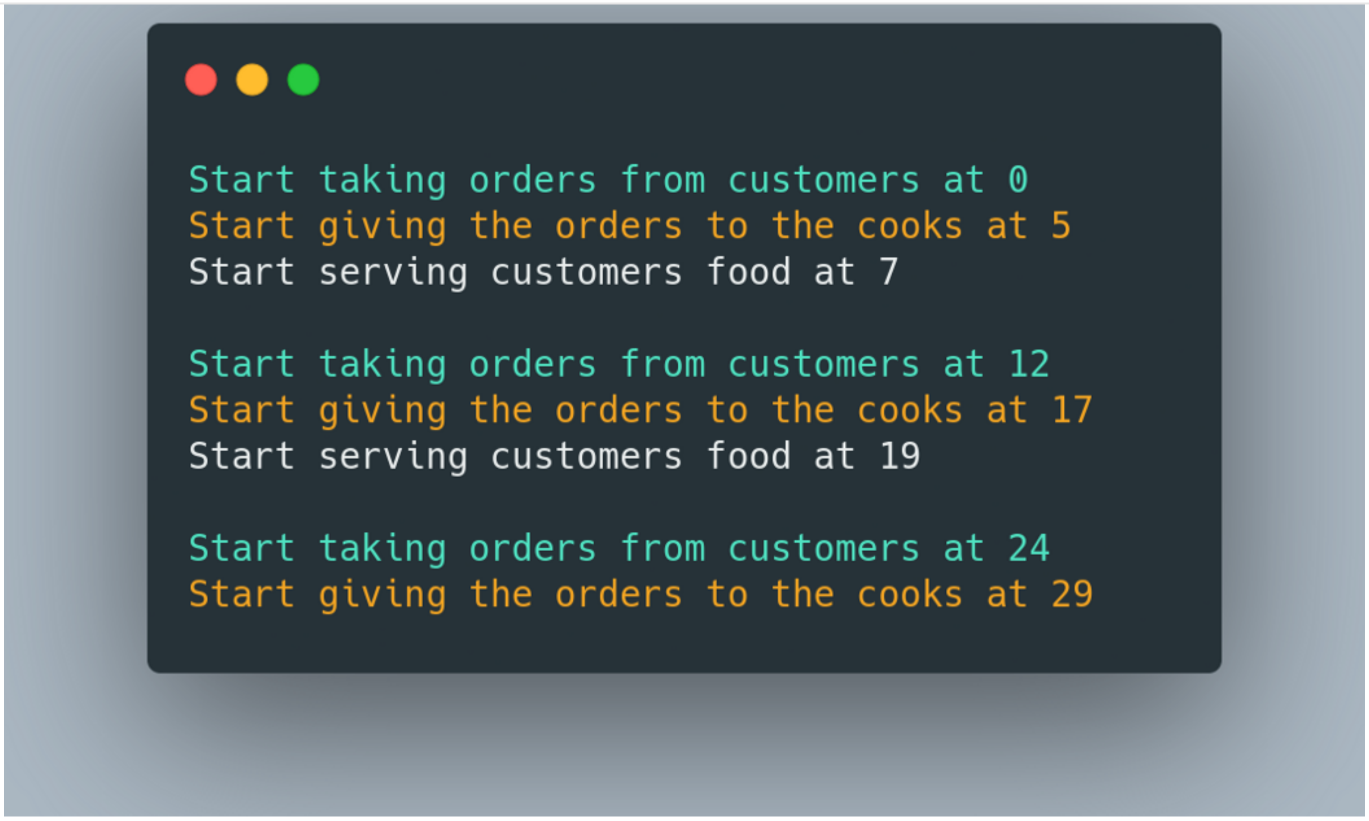
GIF by Author

Here, the waiter is a process. The waiter lives in an `env` . The current simulation time is `env.now` . The `env.timeout(duration)` method models the time it takes to finish one activity.

```python
import simpy

def waiter(env):
    while True: # Simulate until the time limit
        print(f"Start taking orders from customers at {env.now}")
        take_order_duration = 5
        yield env.timeout(take_order_duration) # models duration

        print(f'Start giving the orders to the cooks at {env.now}')
        give_order_duration = 2
        yield env.timeout(give_order_duration)

        print(f'Start serving customers food at {env.now}\n')
        serve_order_duration = 5
        yield env.timeout(serve_order_duration)

env = simpy.Environment() # the environment where the waiter lives
env.process(waiter(env)) # pass the waiter to the environment
env.run(until=30) # Run simulation until 30s
```

waiter.py hosted with ♡ by GitHub view raw

Output:



```
Start taking orders from customers at 0
Start giving the orders to the cooks at 5
Start serving customers food at 7

Start taking orders from customers at 12
Start giving the orders to the cooks at 17
Start serving customers food at 19

Start taking orders from customers at 24
Start giving the orders to the cooks at 29
```

Image by Author

Cool! We are able to simulate some basic things a waiter does. The outputs above make sense since it takes around:

- 5s to take the orders from the customers (`take_order_duration = 5`)
- 2s to give the orders to the cooks (`give_order_duration = 2`)
- 5s to serve food to the customers (`serve_order_duration = 5`)

Note that we don't see "Start serving customers food at 31" because we only run the simulation for `30` s.

Resource — Simulate Customers

We can simulate customers using the same logic as above. However, since there is a limited number of customers that can be served in the restaurant, we will also model the restaurant with a limited capacity.

into the restaurant only when there are **fewer than 2** customers in the restaurant.

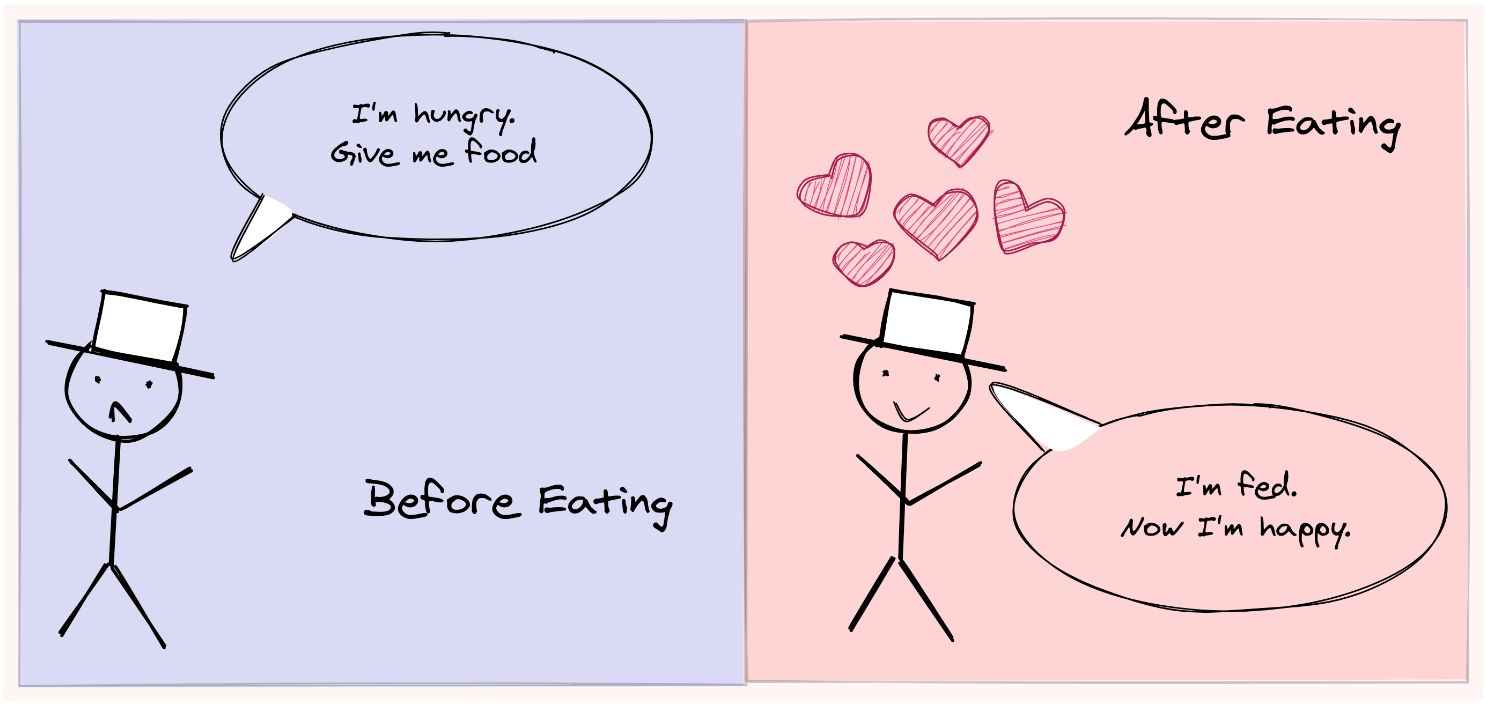


Image by Author

```
import simpy
from random import random, seed


def customer(env, name, restaurant, **duration):
    while True:
        yield env.timeout(random()*10) # There is a new customer between 0 and 10 minutes
        print(f"{name} enters the restaurant and for the waiter to come at {round(env.now, 2)}")
        with restaurant.request() as req:
            yield req

            print(f"Sits are available. {name} get sitted at {round(env.now, 2)}")
            yield env.timeout(duration['get_sitted'])

            print(f"{name} starts looking at the menu at {round(env.now, 2)}")
            yield env.timeout(duration['choose_food'])

            print(f'Waiters start getting the order from {name} at {round(env.now, 2)}')
            yield env.timeout(duration['give_order'])
```

Open in app

```python

            print(f'{name} starts eating at {round(env.now, 2)}')
            yield env.timeout(duration['eat'])

            print(f'{name} starts paying at {round(env.now, 2)}')
            yield env.timeout(duration['pay'])

            print(f'{name} leaves at {round(env.now, 2)}')


seed(1)
env = simpy.Environment()

# Model restaurant that can only allow 2 customers at once
restaurant = simpy.Resource(env, capacity=2)

```

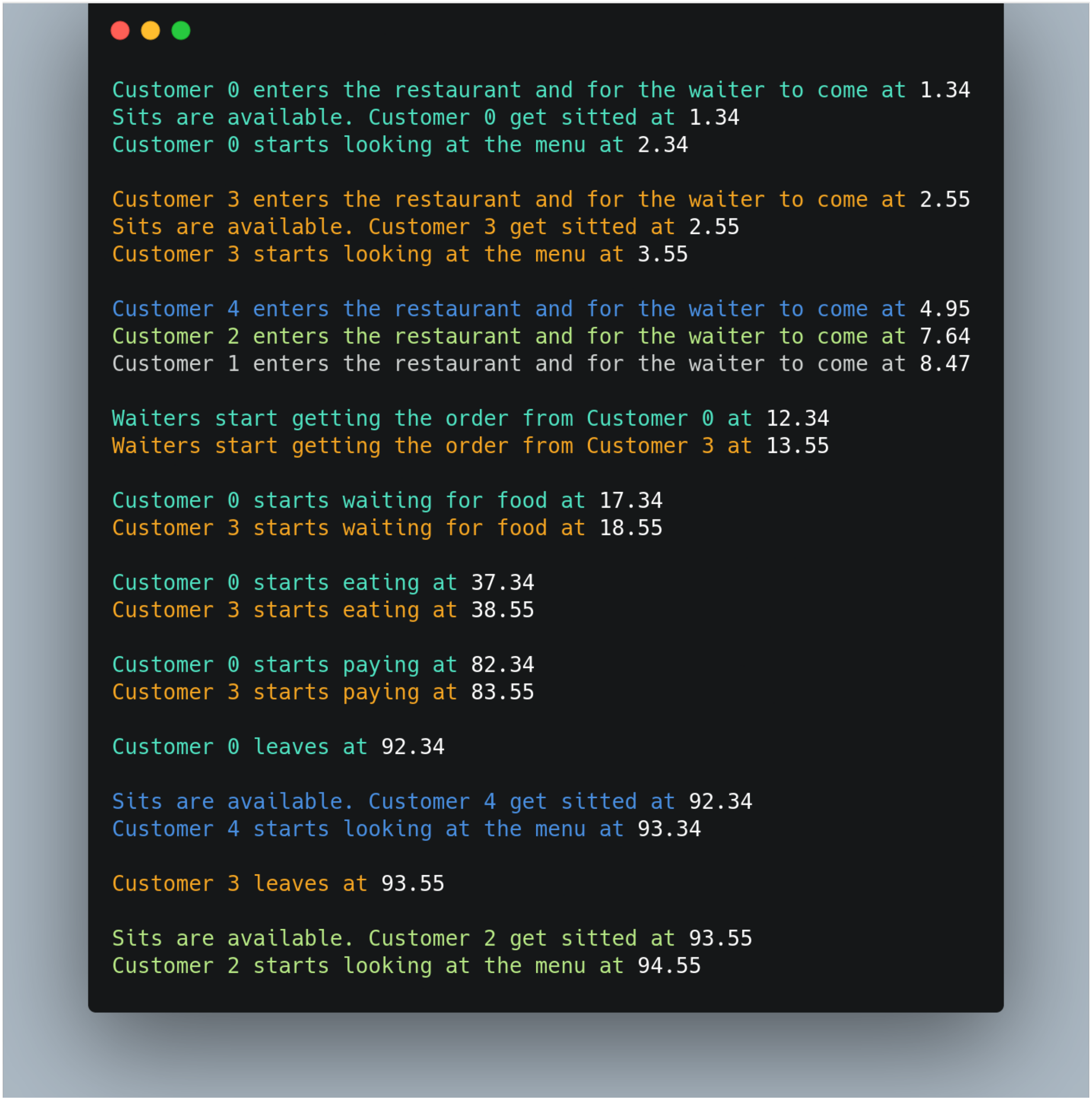
Now we can specify the duration of each activity and simulate 5 customers. Note that the order of customers and their numbers are unrelated.

```python
durations = {'get_sitted': 1, 'choose_food': 10, 'give_order': 5, 'wait_for_food': 20, 'eat': 45,

for i in range(5):
    env.process(customer(env, f'Customer {i}', restaurant, **durations))

env.run(until=95)
```

customer.py hosted with ♡ by GitHub view raw

Output:

Open in app



```
Customer 0 enters the restaurant and for the waiter to come at 1.34
Sits are available. Customer 0 get sitted at 1.34
Customer 0 starts looking at the menu at 2.34

Customer 3 enters the restaurant and for the waiter to come at 2.55
Sits are available. Customer 3 get sitted at 2.55
Customer 3 starts looking at the menu at 3.55

Customer 4 enters the restaurant and for the waiter to come at 4.95
Customer 2 enters the restaurant and for the waiter to come at 7.64
Customer 1 enters the restaurant and for the waiter to come at 8.47

Waiters start getting the order from Customer 0 at 12.34
Waiters start getting the order from Customer 3 at 13.55

Customer 0 starts waiting for food at 17.34
Customer 3 starts waiting for food at 18.55

Customer 0 starts eating at 37.34
Customer 3 starts eating at 38.55

Customer 0 starts paying at 82.34
Customer 3 starts paying at 83.55

Customer 0 leaves at 92.34

Sits are available. Customer 4 get sitted at 92.34
Customer 4 starts looking at the menu at 93.34

Customer 3 leaves at 93.55

Sits are available. Customer 2 get sitted at 93.55
Customer 2 starts looking at the menu at 94.55
```

Image by Author

Cool! From the output, we can see that:

- New customers come in at random times

Pretty cool, isn't it?

Put Everything Together — Simulate a Restaurant with a Limited Supply of Food

As a manager, you think 10 items per option of food are enough to feed your customers. Let's test this assumption by applying what we have learned so far.

Create a Restaurant

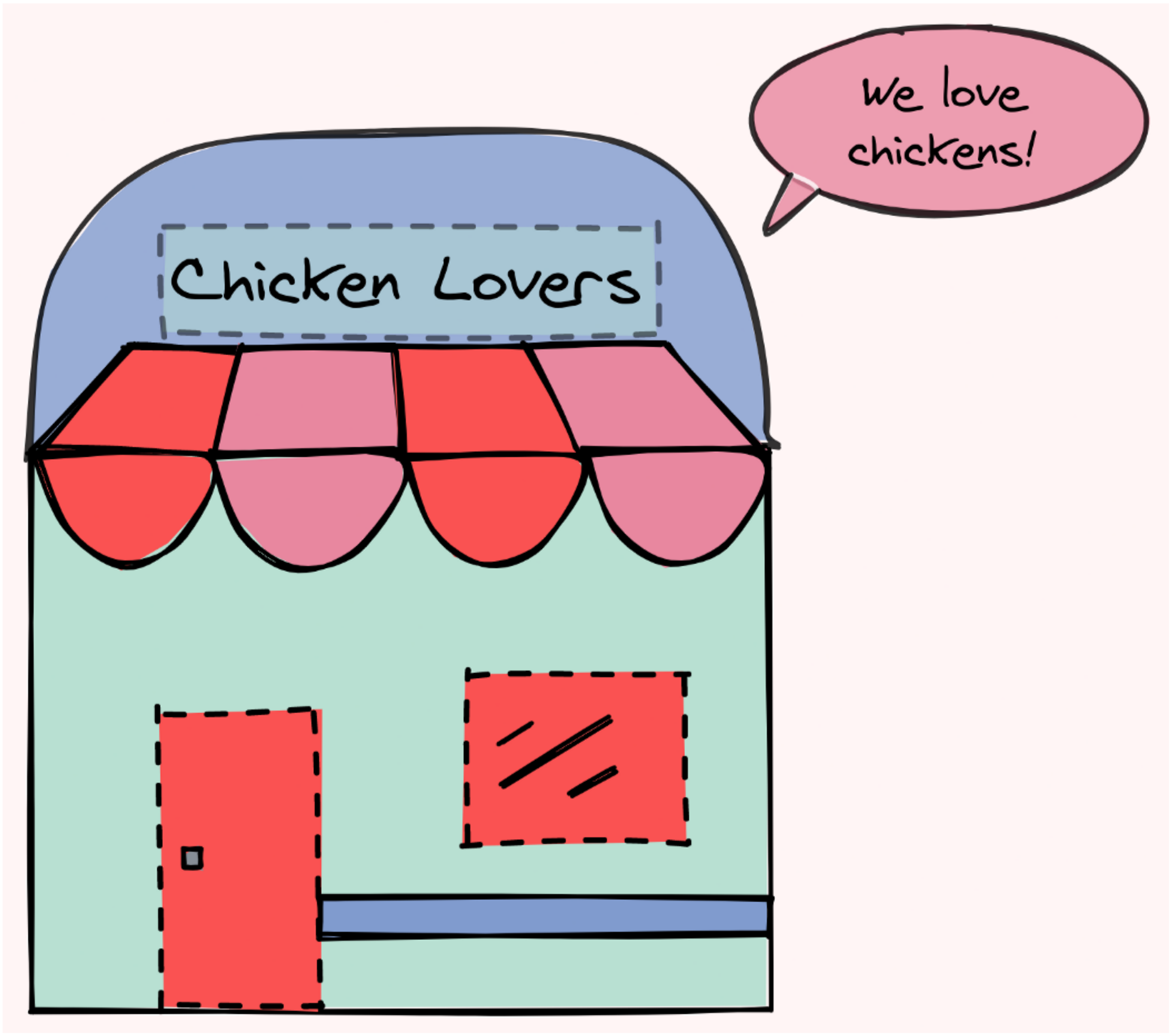


Image by Author

Open in app

- Only takes take-out orders
- There is only one staff member
- The customers are waiting in line to order food on the menu. The more items a customer orders, the longer the customer needs to wait.

The attributes of this restaurant are:

- `staff` : a resource with the capacity 1 (the staff can only serve one customer at a time)
- `foods` : options of food the restaurant offers
- `available` : number of items per option of food
- `run_out` : events when each option of food runs out
- `when_run_out` : the time when each option of food runs out
- `rejected_customers` : number of customers who leave the line because their food option runs out

```python
from collections import namedtuple

NUM_ITEMS = 10 # Number of items per food option

staff = simpy.Resource(env, capacity=1)
foods = ['Spicy Chicken', 'Poached Chicken', 'Tomato Chicken Skillet',
         'Honey Mustard Chicken']
available = {food: NUM_ITEMS for food in foods}
run_out = {food: env.event() for food in foods}
when_run_out = {food: None for food in foods}
rejected_customers = {food: 0 for food in foods}

Restaurant = namedtuple('Restaurant', 'staff, foods, available,'
                        'run_out, when_run_out, rejected_customers')
restaurant = Restaurant(staff, foods, available, run_out,
                        when_run_out, rejected_customers)
```

We also want to create a customer who is waiting in line. A customer will:

- Leave if there is not enough food left
- Order food if there is enough food left

If there is no food left after the customer orders a particular kind of food, we will declare that the chosen food has run out.

```python
def customer(env, food, num_food_order, restaurant):
    """Customer tries to order a certain number of a particular food,
    if that food ran out, customer leaves. If there is enough food left,
    customer orders food."""

    with restaurant.staff.request() as customer:

        # If there is not enough food left, customer leaves
        if restaurant.available[food] < num_food_order:
            restaurant.rejected_customers[food] +=1
            return

        # If there is enough food left, customer orders food
        restaurant.available[food] -= num_food_order
        # The time it takes to prepare food
        yield env.timeout(10*num_food_order)

        # If there is no food left after customer orders, trigger run out event
        if restaurant.available[food] == 0:
            restaurant.run_out[food].succeed()
            restaurant.when_run_out[food] = env.now

        yield env.timeout(2)
```

restaurant.py hosted with ♡ by GitHub view raw

Two new concepts in the code above are:

- `restaurant.staff.request` : Request usage of the resource. In this case, the customer requests a service from the staff member.

of a food option.

There are new customers until the simulation reaches the time limit.

```python
import random

def customer_arrivals(env, restaurant):
    """Create new customers until the simulation reaches the time limit"""
    while True:
        yield env.timeout(random.random()*10)

        # Choose a random food choice from the menu
        food = random.choice(restaurant.foods)

        # Number of a food choice the customer orders
        num_food_order = random.randint(1,6)

        if restaurant.available[food]:
            env.process(customer(env, food, num_food_order, restaurant))
```

restaurant.py hosted with ♡ by GitHub view raw

Run the Simulation

Now we have created the customers and the restaurant, let's run the simulation. We will print messages when the restaurant runs out of a food option.

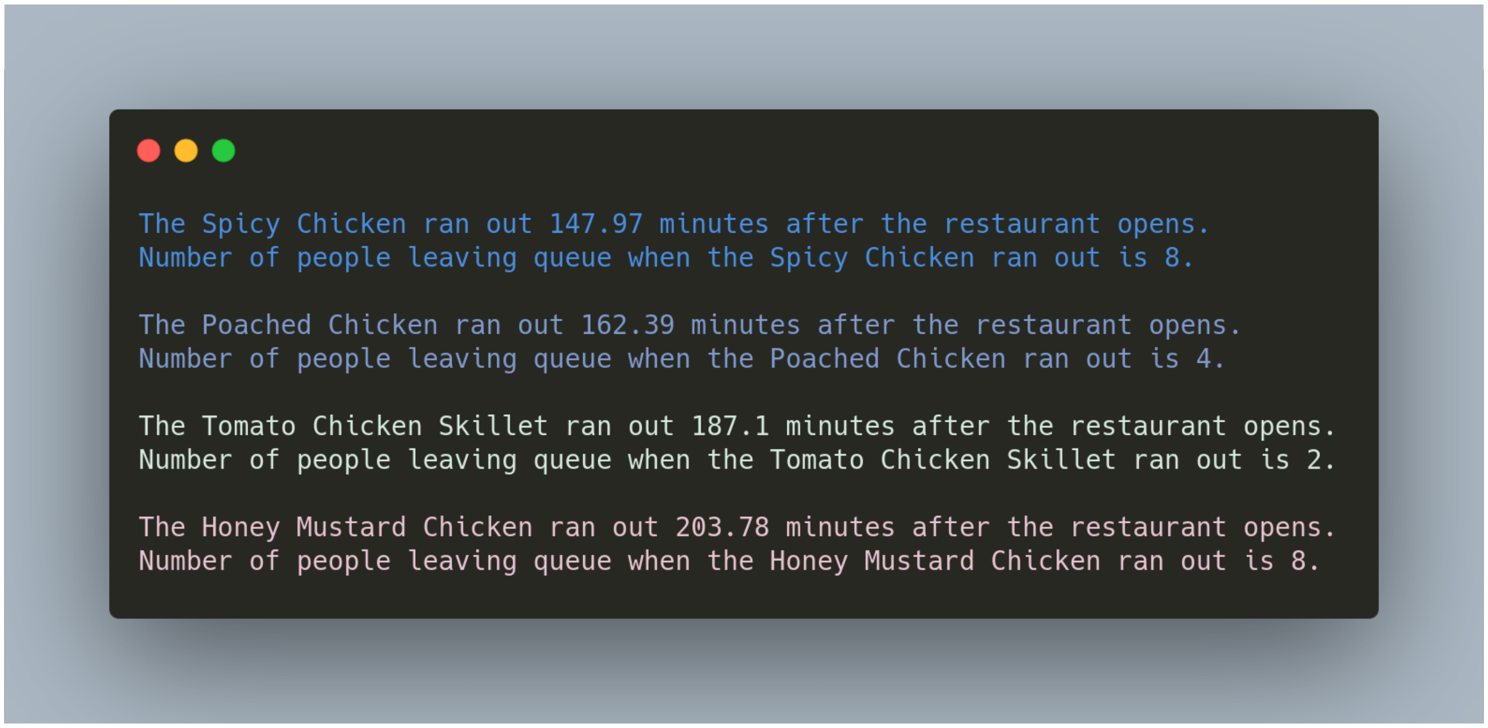
```python
import random

RANDOM_SEED = 20
SIM_TIME  = 240

random.seed(RANDOM_SEED)
env = simpy.Environment()

# Start process and run
env.process(customer_arrivals(env, restaurant))
env.run(until=SIM_TIME)

for food in foods:
    if restaurant.run_out[food]:
```

Open in app

```
        print(f'Number of people leaving queue when the {food} ran out is '
        f'{restaurant.rejected_customers[food]}.\n')
```

restaurant.py hosted with ♡ by GitHub view raw

Output:



```
The Spicy Chicken ran out 147.97 minutes after the restaurant opens.
Number of people leaving queue when the Spicy Chicken ran out is 8.

The Poached Chicken ran out 162.39 minutes after the restaurant opens.
Number of people leaving queue when the Poached Chicken ran out is 4.

The Tomato Chicken Skillet ran out 187.1 minutes after the restaurant opens.
Number of people leaving queue when the Tomato Chicken Skillet ran out is 2.

The Honey Mustard Chicken ran out 203.78 minutes after the restaurant opens.
Number of people leaving queue when the Honey Mustard Chicken ran out is 8.
```

Image by Author

Cool! Imagine 1 second in SimPy is 1 minute in real life. If the simulation runs for 240 minutes (4 hours), and there are 10 items per food choice, then the restaurant will run out of each food choice in fewer than 240 minutes.

There is also a total of 22 customers leaving the restaurant hungry. Thus, if we want to feed all of our customers in 240 minutes, we might need to prepare more than 10 items per food option.

How to Improve this Example?

Note that the example above is only a simplified version of a real restaurant. To make the example closer to a real event in a restaurant, you might want to add more interactions.

Check out the examples in SimPy's documentation for other inspirations.

Conclusion

Congratulations! You have just learned how to use SimPy to simulate real-life events. I hope this will give you the motivation to simulate the events you are interested in.

The source code for this article can be found here:

khuyentran1401/Data-science

Collection of useful data science topics along with code and articles - khuyentran1401/Data-science

github.com

I like to write about basic data science concepts and play with different algorithms and data science tools. You could connect with me on LinkedIn and Twitter.

Star this repo if you want to check out the codes for all of the articles I have written. Follow me on Medium to stay informed with my latest data science articles like these:

How to Create Mathematical Animations like 3Blue1Brown Using Python

Leverage your Python Skills to Create Beautiful Mathematical Animations

towardsdatascience.com

Streamlit and spaCy: Create an App to Predict Sentiment and Word Similarities with Minimal Domain…

All it Takes is 10 Lines of Code!

towardsdatascience.com

Open in app

Use your Domain Knowledge to Label your Data

towardsdatascience.com

How to Create Fake Data with Faker

You can either Collect Data or Create your Own Data

towardsdatascience.com

Sign up for The Variable

By Towards Data Science

Every Thursday, the Variable delivers the very best of Towards Data Science: from hands-on tutorials and cutting-edge research to original features you don't want to miss. Take a look.

Get this newsletter

Emails will be sent to javimartinalonso@gmail.com. Not you?

Simulation Python Data Science Hands On Tutorials Editors Pick

About Help Legal

Get the Medium app

Download on the App Store

GET IT ON Google Play

Open in app