CODING NOMADS

16) Representing Content Lesson

Generating Fake Data with Python Faker

14 min to complete · By Brandon Gigous

Contents

Note: This lesson builds on a project that's been gradually worked on throughout this course.

At this point of your Python + Flask project, you probably have *some* data displayed in your index page, but what if you need more data? It is nice to see how your app behaves and displays data when used more extensively with more content. Generating dummy data, aka mock data or fake data, would be an ideal way to do that. In this lesson, you'll learn to generate fake data using Python Faker.

Python Faker: Random Data Generator

Dictionary.com defines "faker" as "a person who fakes." When you let two oddly-humanoid hedgehogs loose in a forest, this might happen:

While these two are too busy throwing insults at each other, you can learn about the benefits of being a faker.

Sometimes it's helpful to generate data for testing, meaning you can "fake" some of the content in your website for testing and development. This kind of thing isn't to be used in a production release of your website, as you want *real* people making *real* content once your webapp is ready for the world.

To help you generate fake data for testing, you can use the Python ***Faker library***. Python Faker is a fake data generator that allows you to generate and add dummy data to the database. For this exercise, you'll generate fake users and fake compositions, so you have plenty of data to work with.

Install Python Faker

You can install Python Faker using pip:

```
(env) $ pip install faker
```

Only Use Dummy Data in Development

As mentioned earlier, this module won't be used in production as it's only needed during development. Knowing this, how are you supposed to keep it as a dependency of your project while keeping it completely out of production?

Say you set your app to run a production release. You'd set your `FLASK_CONFIG` environment variable to `"production"` and do the other necessary steps, and you're all set, right? (You'll learn about those other steps later.) You'd probably be fine, but you'll want to make sure your app knows *absolutely nothing* about the Python Faker module. If you cloned your own flask-webdev repo and tried to start it from scratch, you'd install all its dependencies with `pip install -r requirements.txt`. But this file might have Faker listed as a dependency! Not good for a production release.

Separate `requirements.txt` into Multiple Files

The solution is to separate your `requirements.txt` file into multiple files: one for your development configuration, another for production, and maybe even another for testing. To prevent repeating yourself and help you avoid mistakes like using one version of a package in one configuration and a different version of the same package in another, you can create a *common* requirements file. We'll show you how to do this in the next video.

To better organize, you can put all these files in a folder. **Before you add Faker to your project's list of dependencies**, take a look at this slightly improved file structure and read on before implementing it:

```
flask-webdev
├── app/
├── ...
└── requirements/
    ├── common.txt
    ├── dev.txt
    └── prod.txt
```

Here, a new top-level `requirements/` is created alongside `app/`. What was the `requirements.txt` file is now called `common.txt`. The two new files, `dev.txt` and `prod.txt`, then list dependencies for the development configuration and production configuration, respectively.

Now, go ahead and `git mv` your `requirements.txt` to `requirements/common.txt`. To create `dev.txt`, you can run this command, which will also add the Faker dependency to the file (assuming you are using Bash or similar as your terminal shell):

```
(env) $ pip freeze | grep Faker= > dev.txt
```

Then edit `dev.txt` to look something like this (your Faker version may vary):

```
-r common.txt
Faker==4.0.2
```

The entire content of `dev.txt` should contain those two lines, but what does the `-r` mean? It just means that all the dependencies that are in `common.txt` should also be included in `dev.txt`. Since you also want the same dependencies in `common.txt` to also be in `prod.txt`, you can edit `prod.txt` to contain only `-r common.txt`.

Going forward, you'll want to make sure you follow this new dependency organization and keep any development-only packages out of `common.txt`.

Get Mentorship

Become The Faker

> *You must be shapeless, formless, like water. When you pour water into a cup, it becomes the cup. When you pour water into a bottle, it becomes the bottle. When you pour water into a teapot, it becomes the teapot. Water can drip, and it can crash. Become like water, my friend. — Bruce Lee*

With that inspiring quote from martial arts master Mr. Lee, it's time to *become the faker*. Create a new `app/faker.py` file, and get ready to add fake users and compositions.

Info: After you make these, you *can* try them out in a Flask shell session, but you will have *a lot* of data displayed in your index page, which may be slow. Tune in to the next lesson to learn about paginating so you can show only a few compositions at a time!

Faking Users

Create a new `users()` function that will generate fake users:

```
from sqlalchemy.exc import IntegrityError
from faker import Faker
from . import db
from .models import User

def users(count=20):
    fake = Faker()
    i = 0
    while i < count:
        u = User(email=fake.email(),
                 username=fake.user_name(),
                 password='password',
                 confirmed=True,
                 name=fake.name(),
                 location=fake.city(),
                 bio=fake.text(),
                 last_seen=fake.past_date())
        db.session.add(u)
        try:
            db.session.commit()
            i += 1
        except IntegrityError:
            db.session.rollback()
```

Using the Python `faker.Faker` class, you can generate all sorts of dummy data, like emails, usernames, and even dates in the past, using Faker's random fake data generator. The best part? At first glance, the generated data actually *looks* real. The `users()` function will loop until it generates and commits to the database a certain `count` of users.

However, sometimes a `Faker` object can generate a username or email that already exists. In that case, it's best to catch the `IntegrityError` exception that SQLAlchemy would otherwise throw, and then do a `db.session.rollback()` to undo the duplicate user.

Faking Compositions

In the same file, make a new `compositions()` function to generate fake compositions submitted by your fake users.

```
from random import randint
from .models import Composition
import string

def compositions(count=100):
    fake = Faker()
    user_count = User.query.count()
    for i in range(count):
        u = User.query.offset(randint(0, user_count - 1)).first()
        c = Composition(release_type=randint(0,2),
                        title=string.capwords(fake.bs()),
                        description=fake.text(),
                        timestamp=fake.past_date(),
                        artist=u)
        db.session.add(c)
    db.session.commit()
```

This is pretty similar to generating fake users, but this time, you don't have to worry about possible duplicates. Because *some* user submits a composition, you can pick a user at random as the artist for each new composition. To do that, you can utilize the `offset()` query filter, which discards the first *n* number of results given as an argument. In this case, *n* is a random number between 0 and the number of users, less one.

Want to know my favorite Python Faker function? It's gotta be `bs()` which stands for [BEEEEP] and generates some cool-sounding titles for each composition. "Orchestrate Bricks-and-Clicks Portals" is but one example of this powerful and *totally inconspicuous* name generator.

How does it feel to be a faker? Not a faker in the personal or professional sense, of course, but a faker in terms of generating content? It shouldn't feel like cheating. This is something developers often do in order to generate data for testing, and develop web apps more quickly. Before you go about generating all this fake data, it's best to be able to *paginate* it so you, your users, nor your browser are inundated with posts.

Summary: Using Python Faker Random Data Generator

- The Python Faker library can help you generate lots of dummy data and add it to the database.
- To keep dummy data out of production, separate your `requirements.txt` file up into multiple files: one for your development configuration, production, and another for testing. And to prevent repeating yourself and help you avoid mistakes, you can create a *common* requirements file.
- Your `dev.txt` will look something like this (your Faker version may vary):

```
-r common.txt
Faker==4.0.2
```

- Using the `faker.Faker` class, you can generate all sorts of dummy data, like emails, usernames, and even dates in the past, using Faker's random data generator.
- Sometimes, a `Faker` object can generate duplicate data. It's best to catch the `IntegrityError` exception that SQLAlchemy would otherwise throw, and then do a `db.session.rollback()` to undo the duplicate user.

Previous Next → Video: Python Faker