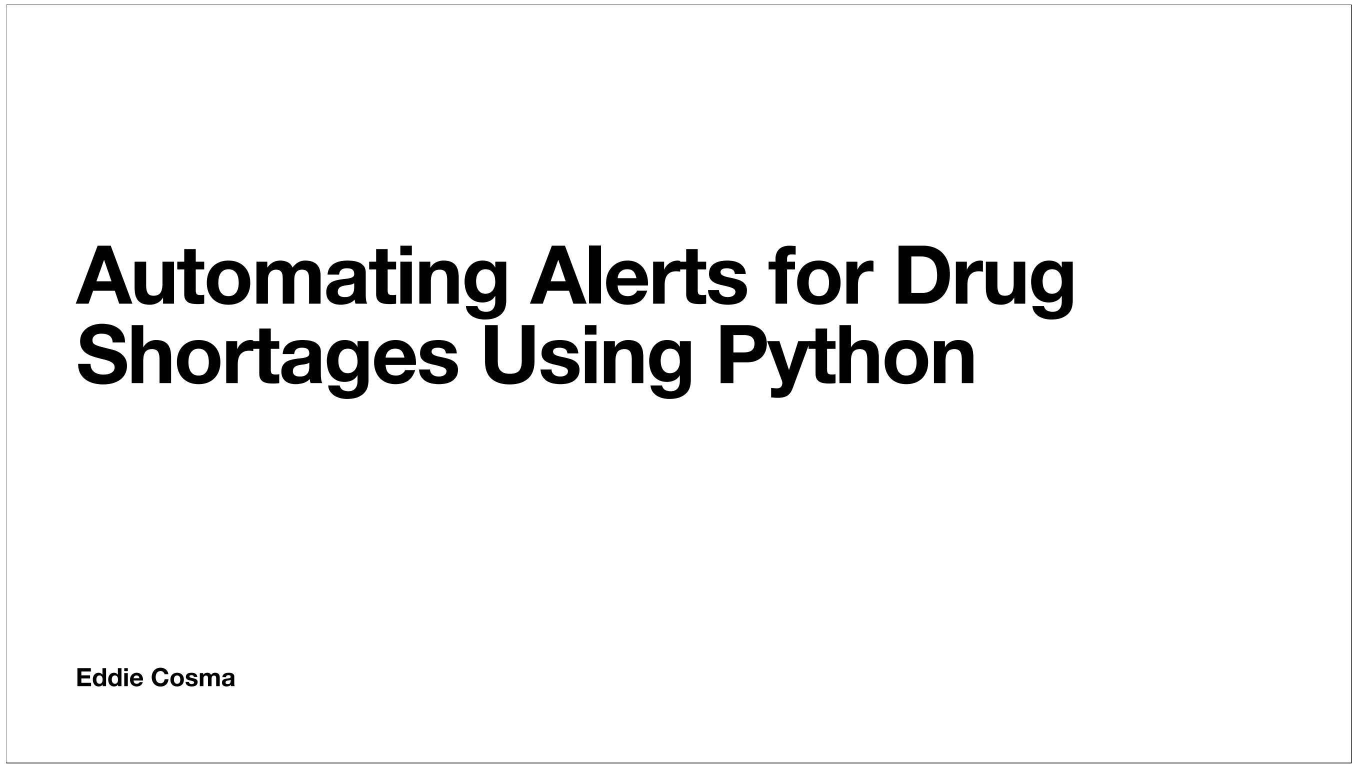


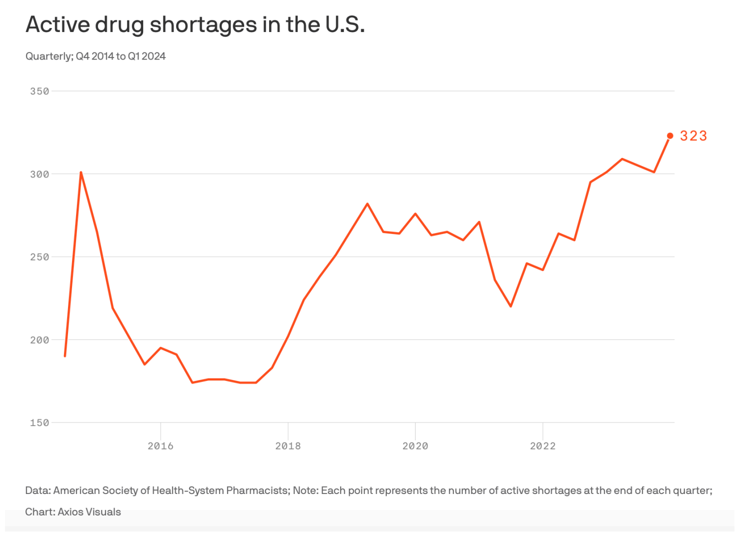
Automating Alerts for Drug Shortages Using Python

Eddie Cosma

Intro
- Pharmacist at MetroHealth since 2018
- 3.5 years staffing inpatient pharmacy, including through COVID
- For last almost 3 years been in informatics

Problem

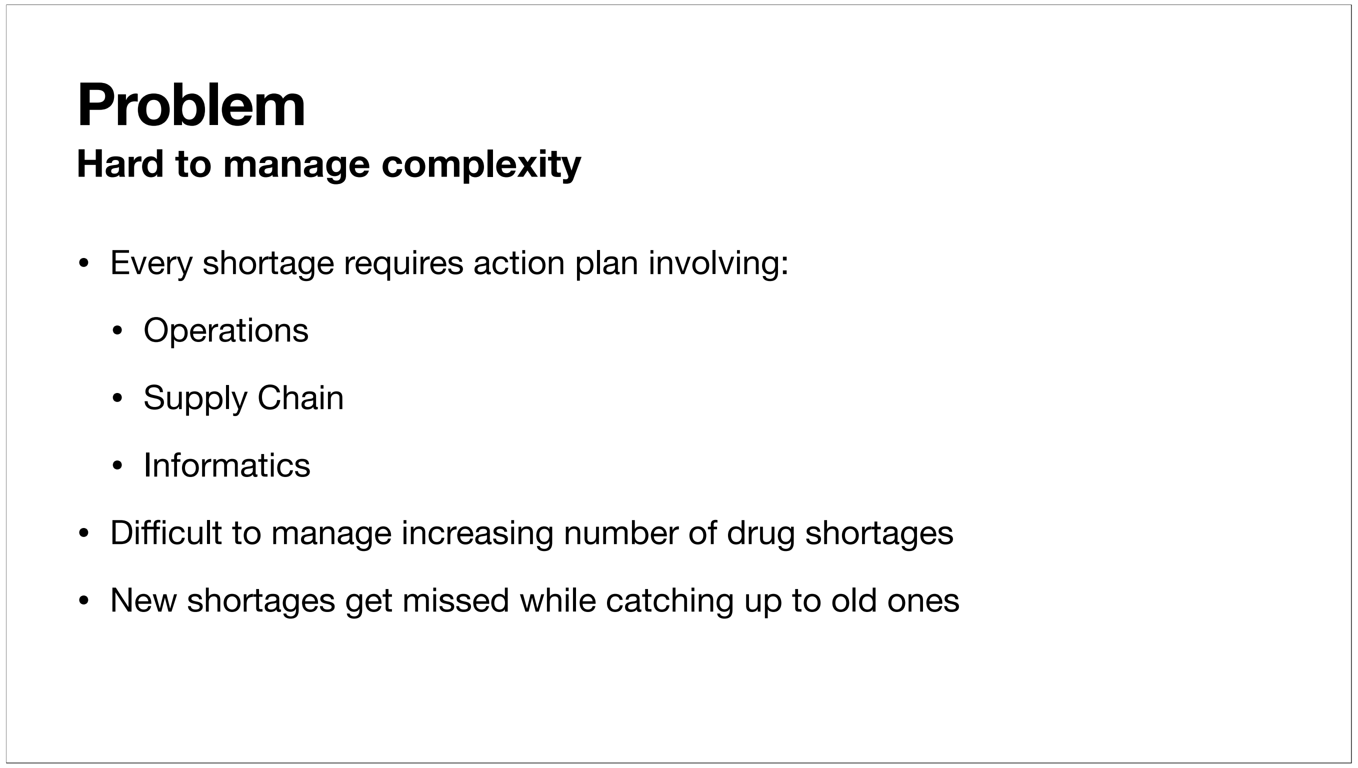
Increasing number of drug shortages


Active drug shortages in the U.S.

Quarterly; Q4 2014 to Q1 2024

Data: American Society of Health-System Pharmacists; Note: Each point represents the number of active shortages at the end of each quarter; Chart: Axios Visuals

- During time staffing, shortages rose rapidly, especially with COVID
- [Note 2018 and 2020 on graph]
- With increasing shortages comes increased complexity

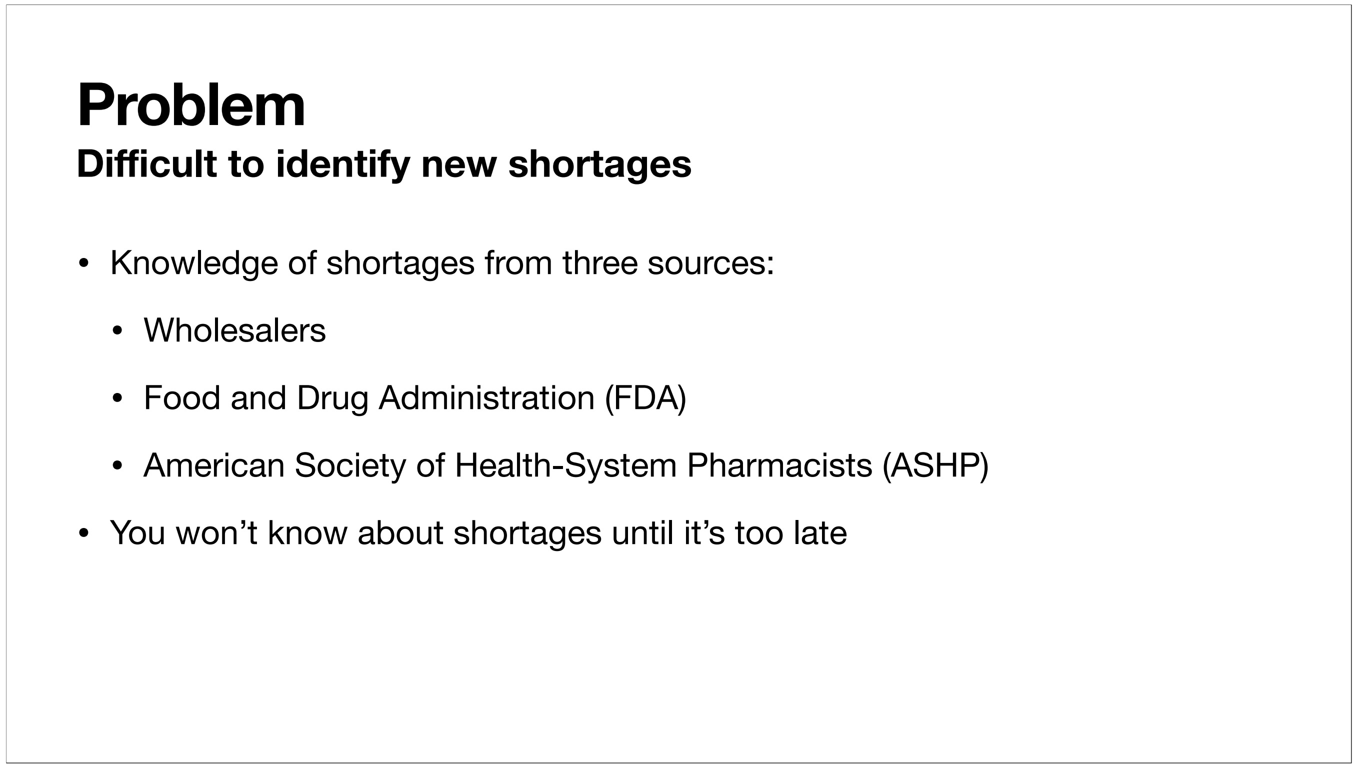


Problem

Hard to manage complexity

- Every shortage requires action plan involving:
  - Operations
  - Supply Chain
  - Informatics
- Difficult to manage increasing number of drug shortages
- New shortages get missed while catching up to old ones

- How do we get information on active shortages?

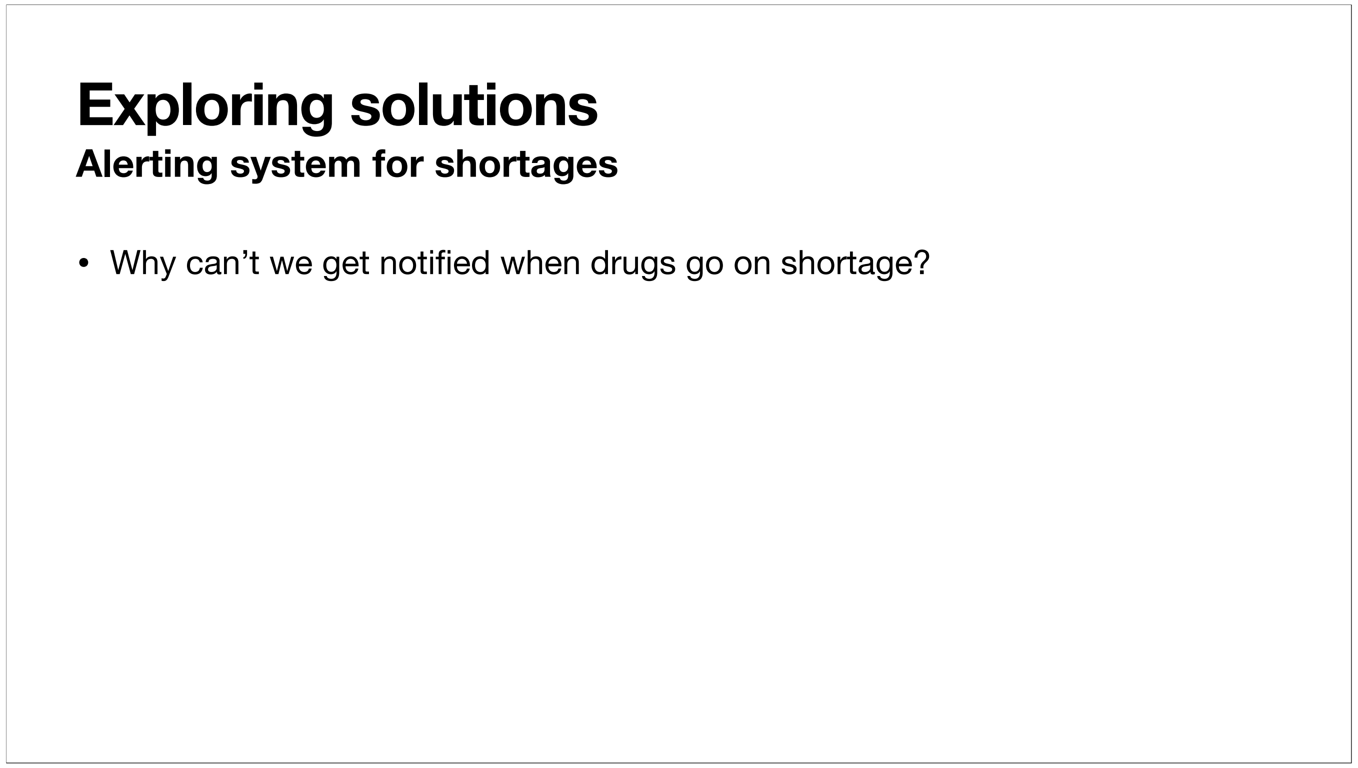


Problem

Difficult to identify new shortages

- Knowledge of shortages from three sources:
  - Wholesalers
  - Food and Drug Administration (FDA)
  - American Society of Health-System Pharmacists (ASHP)
- You won’t know about shortages until it’s too late

- Too late by the time you go to place an order and the drug is no longer available



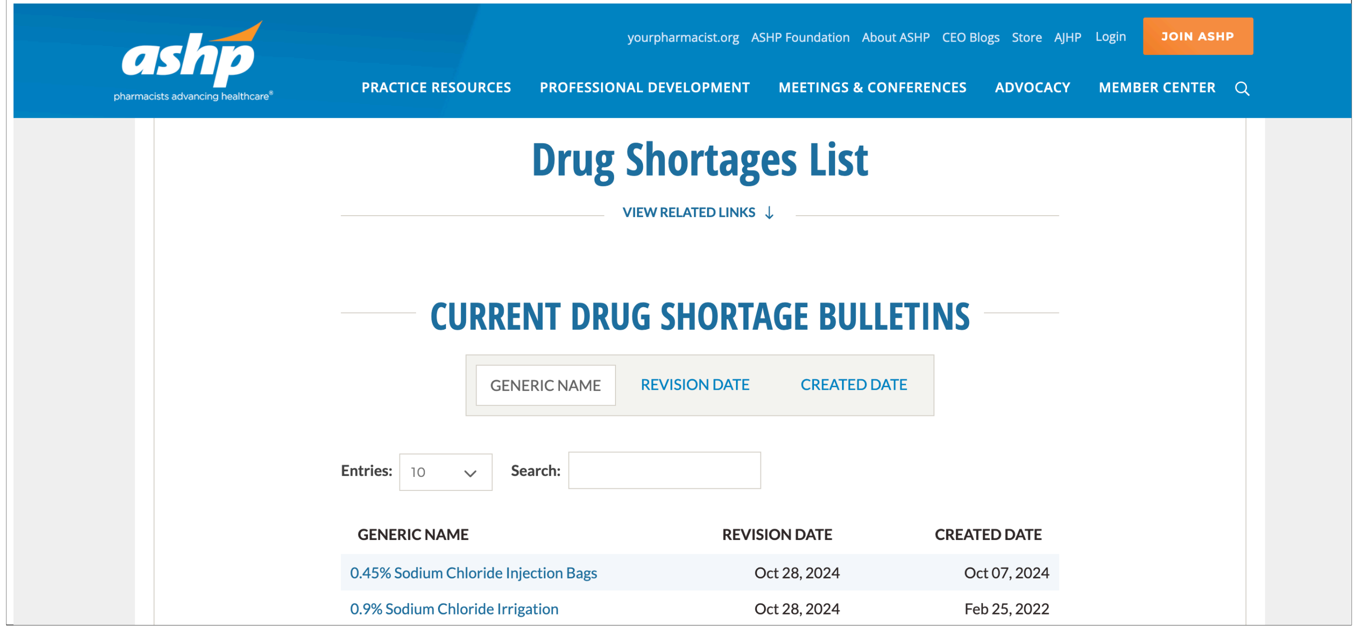
Exploring solutions

Alerting system for shortages

- Why can’t we get notified when drugs go on shortage?

- Shortage we missed at Metro; idea from Steph specifically based on ASHP shortages list

Exploring source data


ashp
pharmacists advancing healthcare®
yourpharmacist.org ASHP Foundation About ASHP CEO Blogs Store AJHP Login
JOIN ASHP
PRACTICE RESOURCES PROFESSIONAL DEVELOPMENT MEETINGS & CONFERENCES ADVOCACY MEMBER CENTER
VIEW RELATED LINKS
GENERIC NAME REVISION DATE CREATED DATE
Entries: 10
Search:

Drug Shortages List

CURRENT DRUG SHORTAGE BULLETINS

GENERIC NAME	REVISION DATE	CREATED DATE
0.45% Sodium Chloride Injection Bags	Oct 28, 2024	Oct 07, 2024
0.9% Sodium Chloride Irrigation	Oct 28, 2024	Feb 25, 2022

Demo

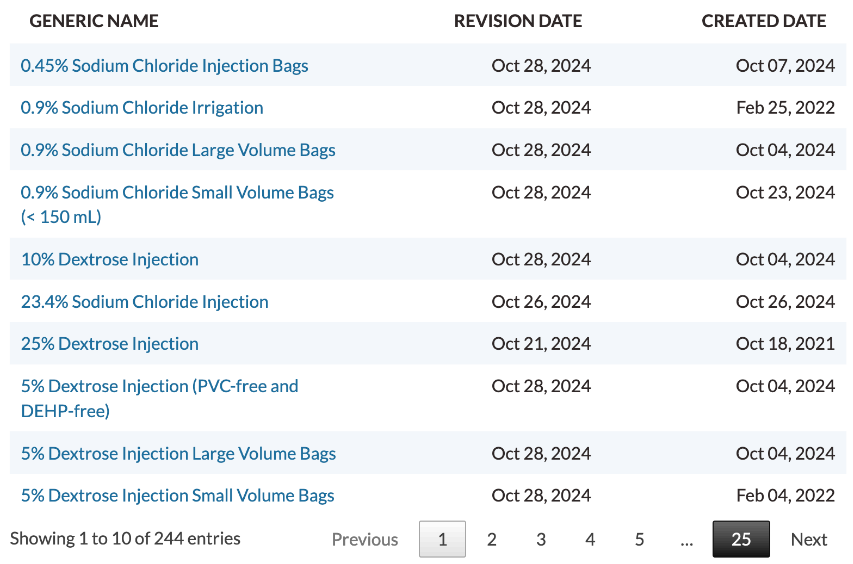
Exploring solutions

Alerting system for shortages

- Can we scrape this?

Scraping ASHP

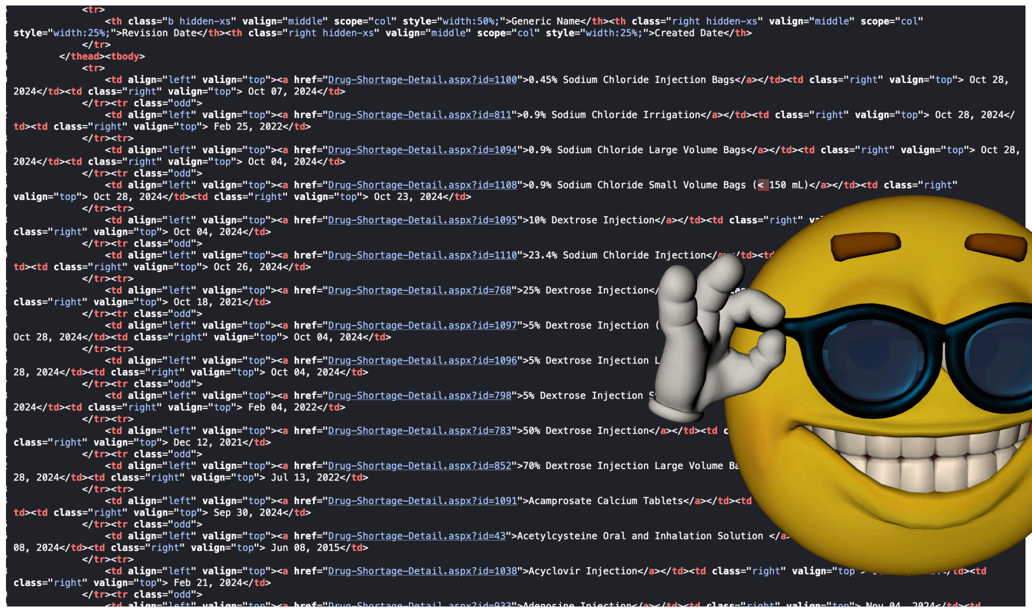
- Pagination = Sad
- But maybe…



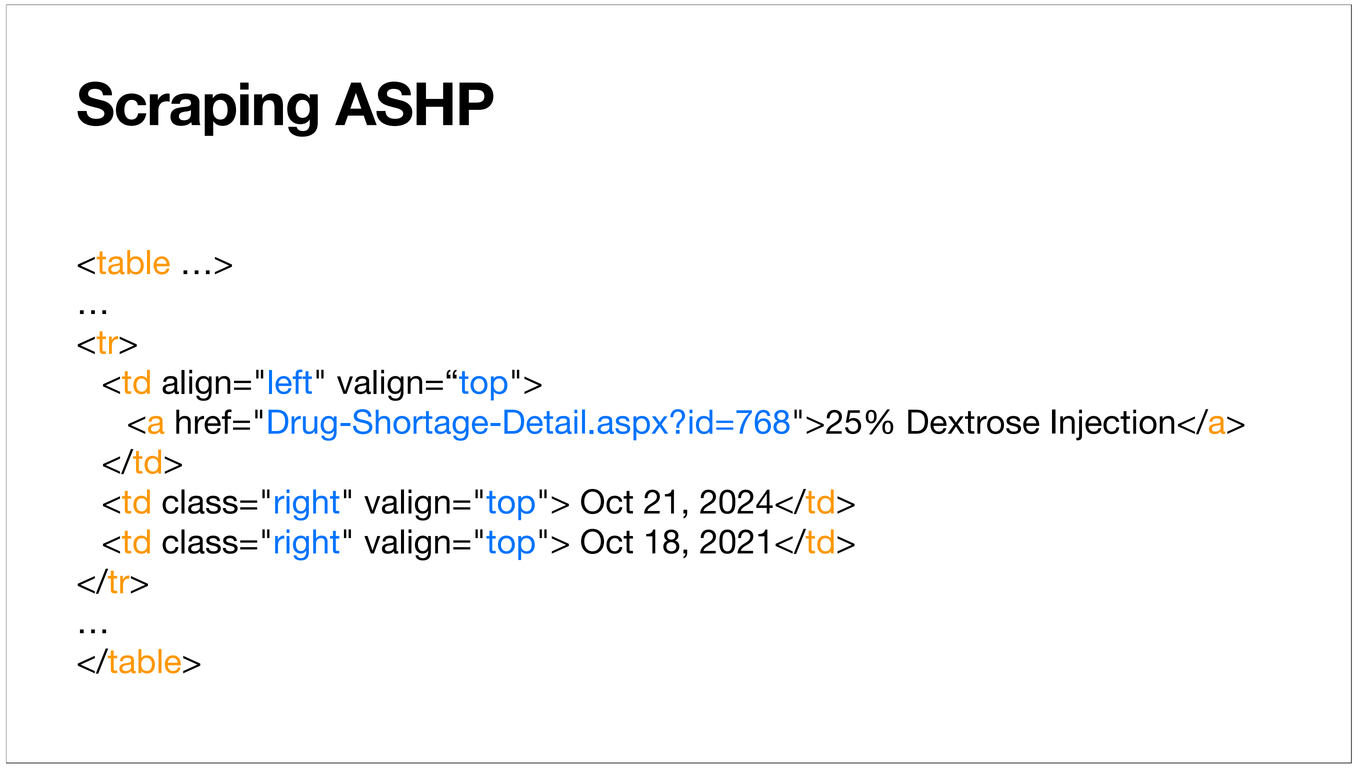
GENERIC NAME	REVISION DATE	CREATED DATE
0.45% Sodium Chloride Injection Bags	Oct 28, 2024	Oct 07, 2024
0.9% Sodium Chloride Irrigation	Oct 28, 2024	Feb 25, 2022
0.9% Sodium Chloride Large Volume Bags	Oct 28, 2024	Oct 04, 2024
0.9% Sodium Chloride Small Volume Bags (< 150 mL)	Oct 28, 2024	Oct 23, 2024
10% Dextrose Injection	Oct 28, 2024	Oct 04, 2024
23.4% Sodium Chloride Injection	Oct 26, 2024	Oct 26, 2024
25% Dextrose Injection	Oct 21, 2024	Oct 18, 2021
5% Dextrose Injection (PVC-free and DEHP-free)	Oct 28, 2024	Oct 04, 2024
5% Dextrose Injection Large Volume Bags	Oct 28, 2024	Oct 04, 2024
5% Dextrose Injection Small Volume Bags	Oct 28, 2024	Feb 04, 2022

Showing 1 to 10 of 244 entries

Scraping ASHP



```
            <tr>
                <th class="b hidden-xs" valign="middle" scope="col" style="width:50%;">Generic Name</th><th class="right hidden-xs" valign="middle" scope="col"
style="width:25%;">Revision Date</th><th class="right hidden-xs" valign="middle" scope="col" style="width:25%;">Created Date</th>
            </tr>
        </thead><tbody>
            <tr>
                <td align="left" valign="top"><a href="Drug-Shortage-Detail.aspx?id=1100">0.45% Sodium Chloride Injection Bags</a></td><td class="right" valign="top"> Oct 28,
2024</td><td class="right" valign="top"> Oct 07, 2024</td>
            </tr><tr class="odd">
                <td align="left" valign="top"><a href="Drug-Shortage-Detail.aspx?id=811">0.9% Sodium Chloride Irrigation</a></td><td class="right" valign="top"> Oct 28, 2024</
td><td class="right" valign="top"> Feb 25, 2022</td>
            </tr><tr>
                <td align="left" valign="top"><a href="Drug-Shortage-Detail.aspx?id=1094">0.9% Sodium Chloride Large Volume Bags</a></td><td class="right" valign="top"> Oct 28,
2024</td><td class="right" valign="top"> Oct 04, 2024</td>
            </tr><tr class="odd">
                <td align="left" valign="top"><a href="Drug-Shortage-Detail.aspx?id=1108">0.9% Sodium Chloride Small Volume Bags (< 150 mL)</a></td><td class="right"
valign="top"> Oct 28, 2024</td><td class="right" valign="top"> Oct 23, 2024</td>
            </tr><tr>
                <td align="left" valign="top"><a href="Drug-Shortage-Detail.aspx?id=1095">10% Dextrose Injection</a></td><td class="right" v
class="right" valign="top"> Oct 04, 2024</td>
            </tr><tr class="odd">
                <td align="left" valign="top"><a href="Drug-Shortage-Detail.aspx?id=1110">23.4% Sodium Chloride Injection</
td><td class="right" valign="top"> Oct 26, 2024</td>
            </tr><tr>
                <td align="left" valign="top"><a href="Drug-Shortage-Detail.aspx?id=768">25% Dextrose Injection</
class="right" valign="top"> Oct 18, 2021</td>
            </tr><tr class="odd">
                <td align="left" valign="top"><a href="Drug-Shortage-Detail.aspx?id=1097">5% Dextrose Injection (
Oct 28, 2024</td><td class="right" valign="top"> Oct 04, 2024</td>
            </tr><tr>
                <td align="left" valign="top"><a href="Drug-Shortage-Detail.aspx?id=1096">5% Dextrose Injection L
28, 2024</td><td class="right" valign="top"> Oct 04, 2024</td>
            </tr><tr class="odd">
                <td align="left" valign="top"><a href="Drug-Shortage-Detail.aspx?id=798">5% Dextrose Injection S
2024</td><td class="right" valign="top"> Feb 04, 2022</td>
            </tr><tr>
                <td align="left" valign="top"><a href="Drug-Shortage-Detail.aspx?id=783">50% Dextrose Injection</a></td><td c
class="right" valign="top"> Dec 12, 2021</td>
            </tr><tr class="odd">
                <td align="left" valign="top"><a href="Drug-Shortage-Detail.aspx?id=852">70% Dextrose Injection Large Volume Ba
28, 2024</td><td class="right" valign="top"> Jul 13, 2022</td>
            </tr><tr>
                <td align="left" valign="top"><a href="Drug-Shortage-Detail.aspx?id=1091">Acamprosate Calcium Tablets</a></td><td
td><td class="right" valign="top"> Sep 30, 2024</td>
            </tr><tr class="odd">
                <td align="left" valign="top"><a href="Drug-Shortage-Detail.aspx?id=43">Acetylcysteine Oral and Inhalation Solution </a
08, 2024</td><td class="right" valign="top"> Jun 08, 2015</td>
            </tr><tr>
                <td align="left" valign="top"><a href="Drug-Shortage-Detail.aspx?id=1038">Acyclovir Injection</a></td><td class="right" valign="top           </td><td
class="right" valign="top"> Feb 21, 2024</td>
            </tr><tr class="odd">
                <td align="left" valign="top"><a href="Drug-Shortage-Detail.aspx?id=933">Adenosine Injection</a></td><td class="right" valign="top"> Nov 04, 2024</td><td
```



Scraping ASHP

```
<table …>
…
<tr>
  <td align="left" valign=“top">
    <a href="Drug-Shortage-Detail.aspx?id=768">25% Dextrose Injection</a>
  </td>
  <td class="right" valign="top"> Oct 21, 2024</td>
  <td class="right" valign="top"> Oct 18, 2021</td>
</tr>
…
</table>
```

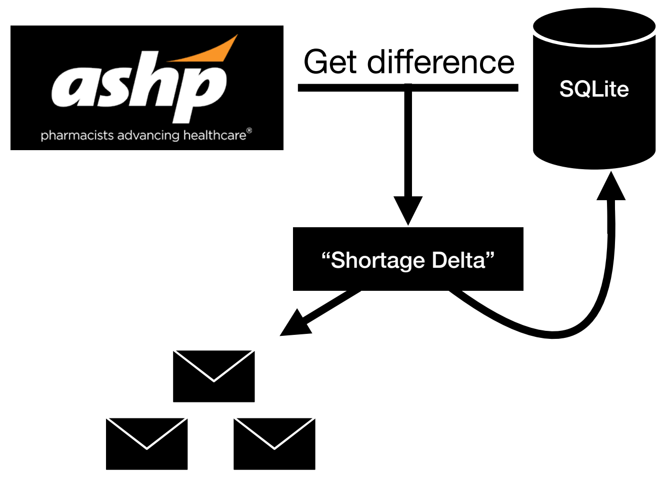
Specifically there are two elements we care about:

- Name of the drug between <a> tags (i.e., “25% Dextrose Injection”)
- ID in the href of the <a> tag (i.e., 768) - Is this a unique value? Probably.

Data processing

- What do we do with this?
- We need a way to track **changes** to the list
    - New shortages
    - Resolved shortages

Functionality overview

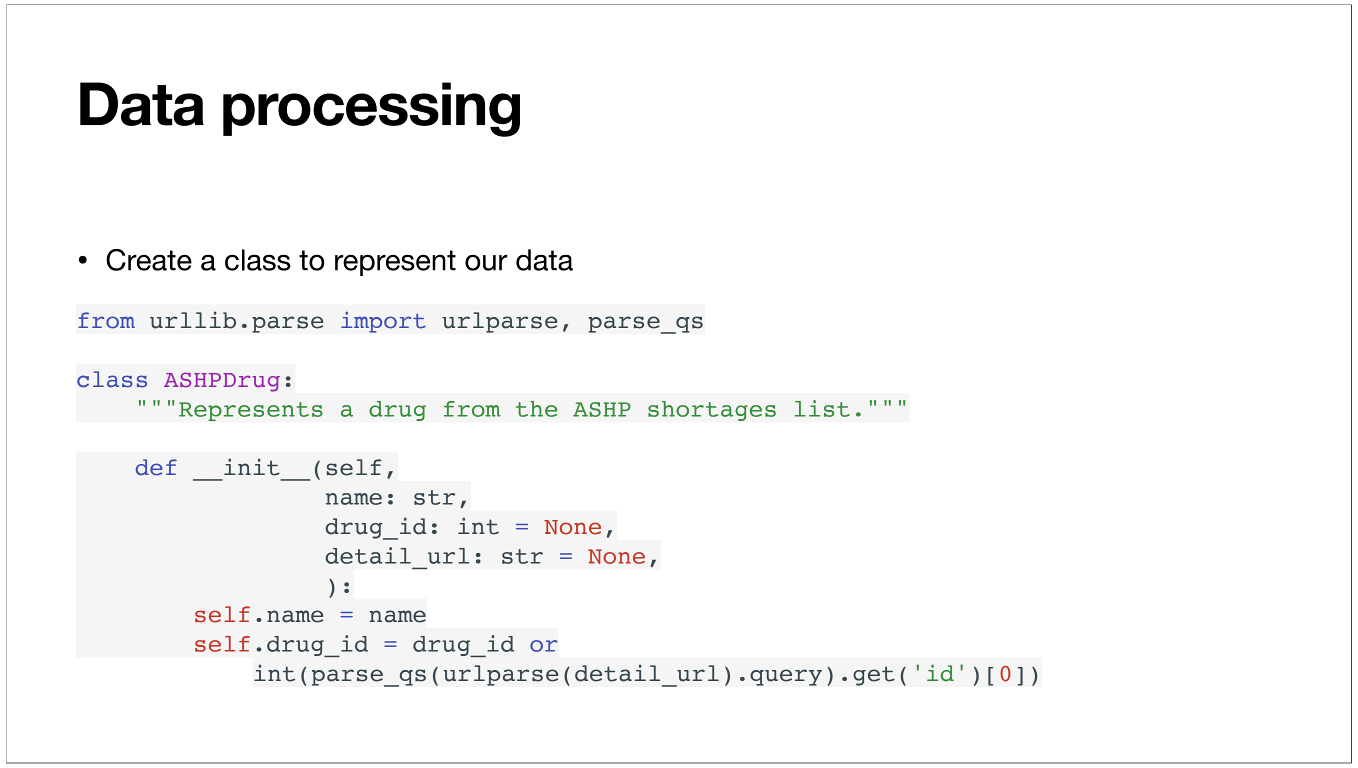


Database model

- Let’s create the simplest model ever with SQLAlchemy…

```
class Drug(Base):
    """The drug model."""

    __tablename__ = 'drug'
    id = Column(Integer, primary_key=True)
    name = Column(String(255), nullable=False)
```

- We can store this data in a SQLite database

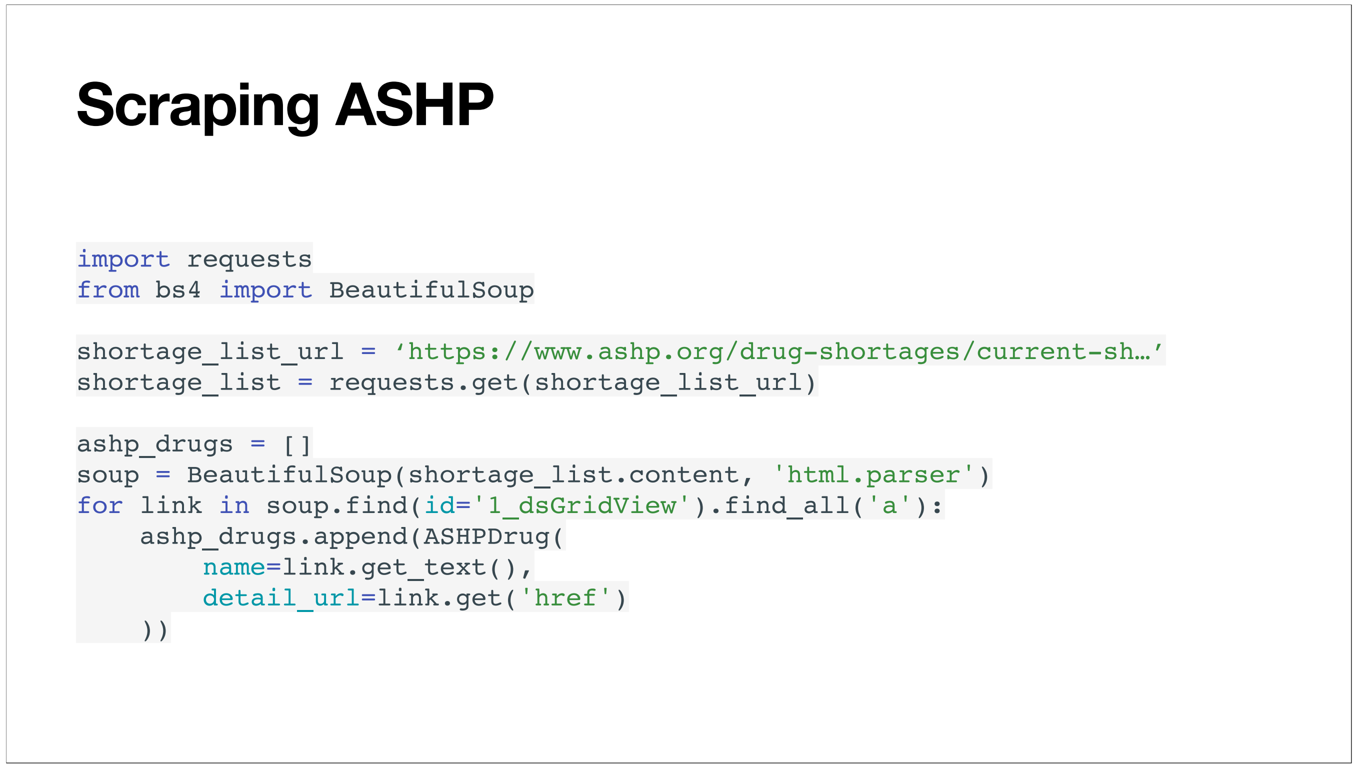


Data processing

- Create a class to represent our data

```python
from urllib.parse import urlparse, parse_qs

class ASHPDrug:
    """Represents a drug from the ASHP shortages list."""

    def __init__(self,
                 name: str,
                 drug_id: int = None,
                 detail_url: str = None,
                 ):
        self.name = name
        self.drug_id = drug_id or
            int(parse_qs(urlparse(detail_url).query).get('id')[0])
```

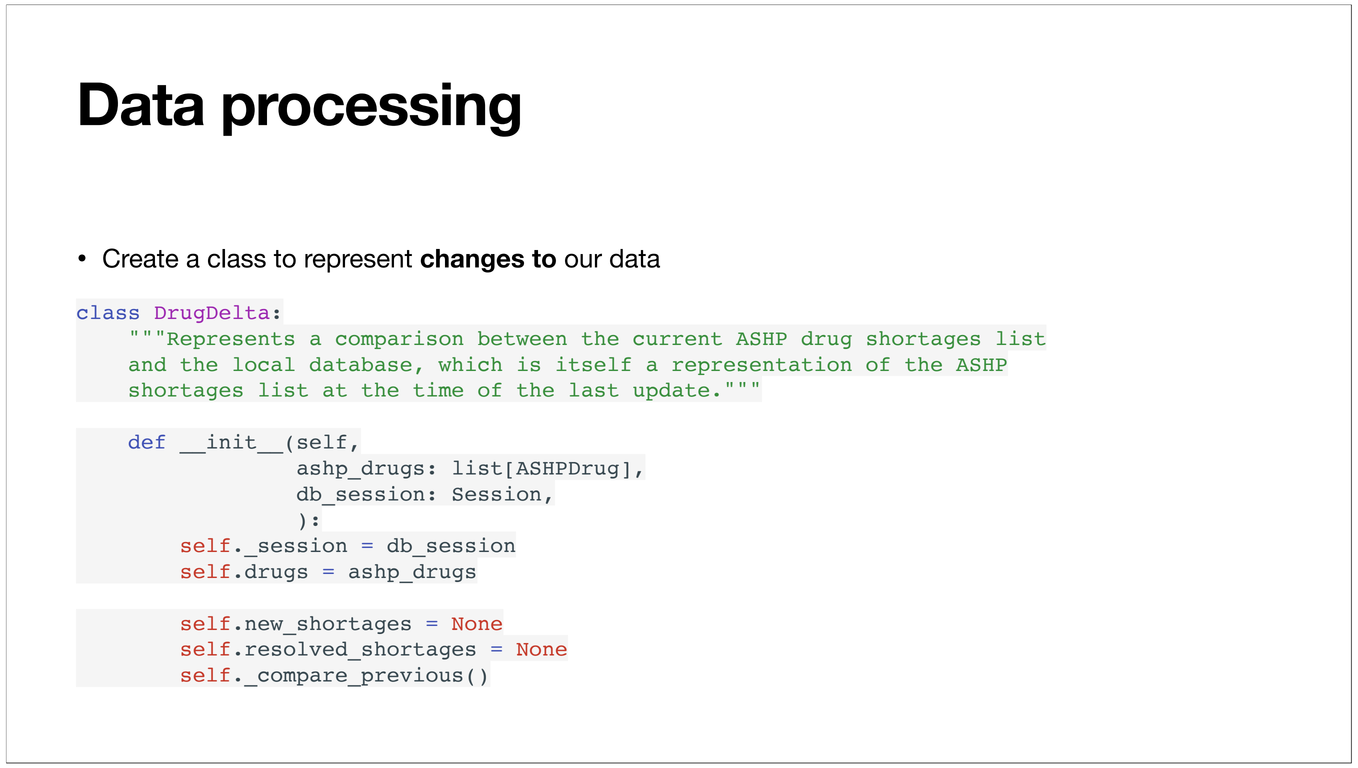
Spaghetti code at the bottom is what accepts the detail URL from the ASHP page and gets the shortage ID



Scraping ASHP

```python
import requests
from bs4 import BeautifulSoup

shortage_list_url = ‘https://www.ashp.org/drug-shortages/current-sh…’
shortage_list = requests.get(shortage_list_url)

ashp_drugs = []
soup = BeautifulSoup(shortage_list.content, 'html.parser')
for link in soup.find(id='1_dsGridView').find_all('a'):
    ashp_drugs.append(ASHPDrug(
        name=link.get_text(),
        detail_url=link.get('href')
    ))
```

1. Import requests and beautiful soup
2. Do a get request on the main shortage URL
3. parse out the link element and create a list of instantiated drug classes

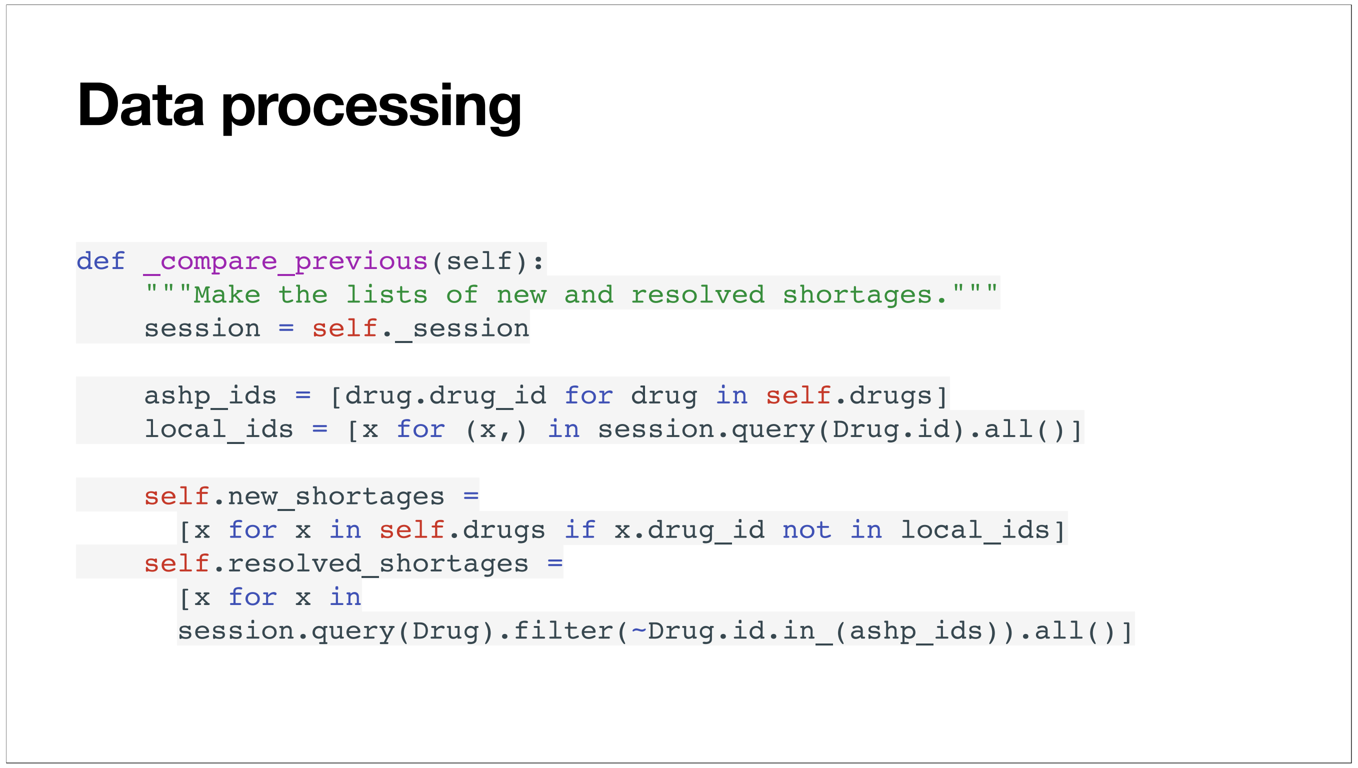


Data processing

- Create a class to represent **changes to** our data

```python
class DrugDelta:
    """Represents a comparison between the current ASHP drug shortages list
    and the local database, which is itself a representation of the ASHP
    shortages list at the time of the last update."""

    def __init__(self,
                 ashp_drugs: list[ASHPDrug],
                 db_session: Session,
                 ):
        self._session = db_session
        self.drugs = ashp_drugs

        self.new_shortages = None
        self.resolved_shortages = None
        self._compare_previous()
```

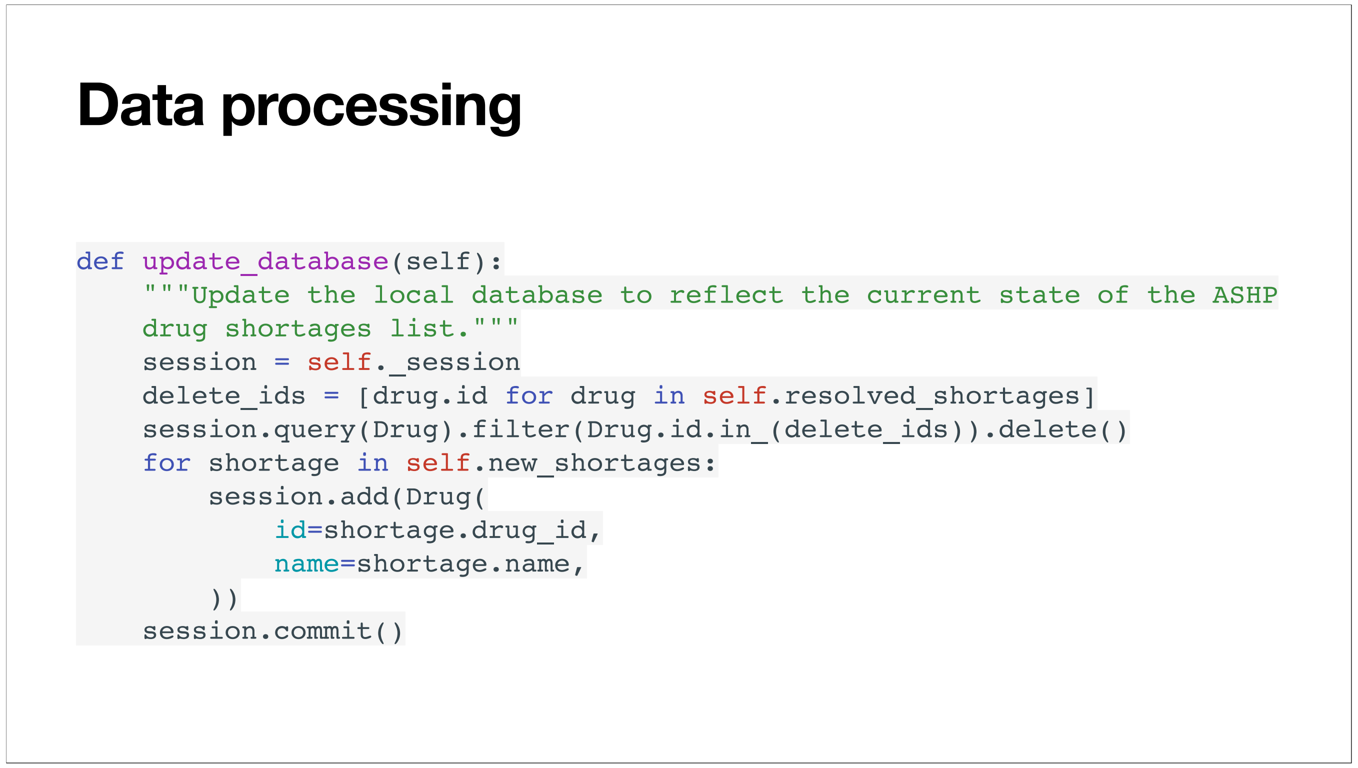
- Take the list of drugs we just created and an active database session
- Instantiate some lists for new shortages and resolved shortages
- Call a function called _compare_previous



Data processing

```python
def _compare_previous(self):
    """Make the lists of new and resolved shortages."""
    session = self._session

    ashp_ids = [drug.drug_id for drug in self.drugs]
    local_ids = [x for (x,) in session.query(Drug.id).all()]

    self.new_shortages =
      [x for x in self.drugs if x.drug_id not in local_ids]
    self.resolved_shortages =
      [x for x in
      session.query(Drug).filter(~Drug.id.in_(ashp_ids)).all()]
```

- Compare IDs from the ASHP website with the IDs from our database. Names we got earlier will be used for the emails we’ll send out



Data processing

```python
def update_database(self):
    """Update the local database to reflect the current state of the ASHP
    drug shortages list."""
    session = self._session
    delete_ids = [drug.id for drug in self.resolved_shortages]
    session.query(Drug).filter(Drug.id.in_(delete_ids)).delete()
    for shortage in self.new_shortages:
        session.add(Drug(
            id=shortage.drug_id,
            name=shortage.name,
        ))
    session.commit()
```

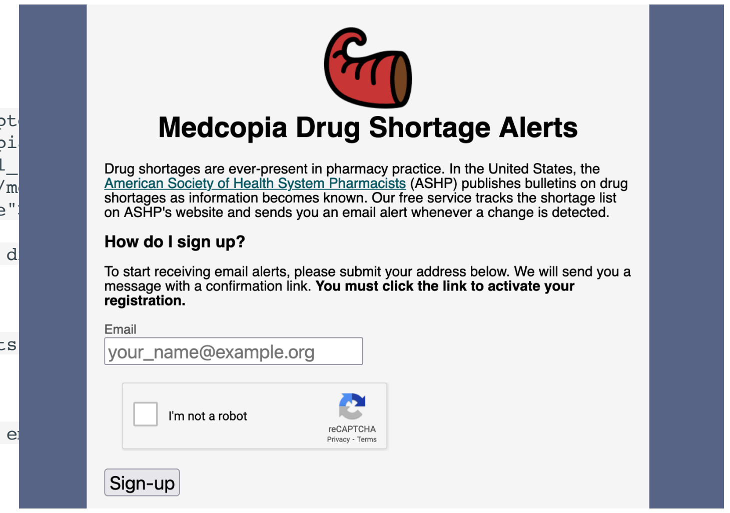
- To wrap things up, we'll create a function to update the database with the updated list
- Two parts:
    - Delete the rows for resolved shortages
    - Insert rows for new shortages
- We will call this later when we put it all together
- In the meantime…

Sending the emails

- What do we need to achieve?
  - Allow users to subscribe and unsubscribe
  - Generate emails from template
  - Send emails without looking like spam

Registration page

```
{% extends 'base.html' %}

{% block head %}
    <script src="https://www.google.com/recapt
    <meta property="og:title" content="Medcopi
    <meta property="og:image" content="{{ url_
    <meta property="og:url" content="https://m
    <meta property="og:type" content="website"
    <meta property="og:description"
          content="Get email alerts any time d
list.">
{% endblock %}

{% block title %}Medcopia Drug Shortage Alerts

{% block content %}
    {% if registrant %}
        <div class="registered">Confirmation e
    {% endif %}
...
```



Medcopia Drug Shortage Alerts

Drug shortages are ever-present in pharmacy practice. In the United States, the American Society of Health System Pharmacists (ASHP) publishes bulletins on drug shortages as information becomes known. Our free service tracks the shortage list on ASHP's website and sends you an email alert whenever a change is detected.

How do I sign up?

To start receiving email alerts, please submit your address below. We will send you a message with a confirmation link. **You must click the link to activate your registration.**

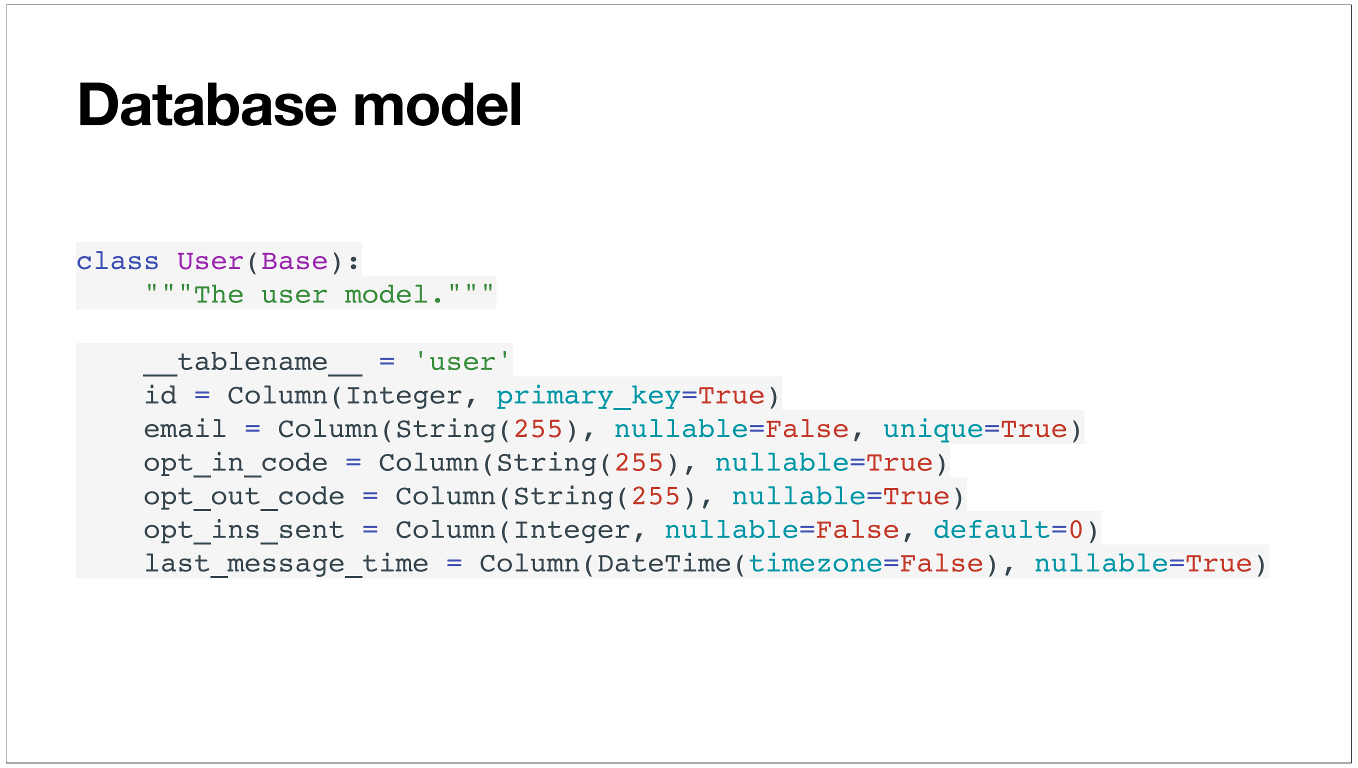
Email

your_name@example.org

I used Flask to create a simple subscription form for the mailing list. On the backend, Flask uses Jinja templates, which is an HTML templating language.

Everyone familiar with Jinja?

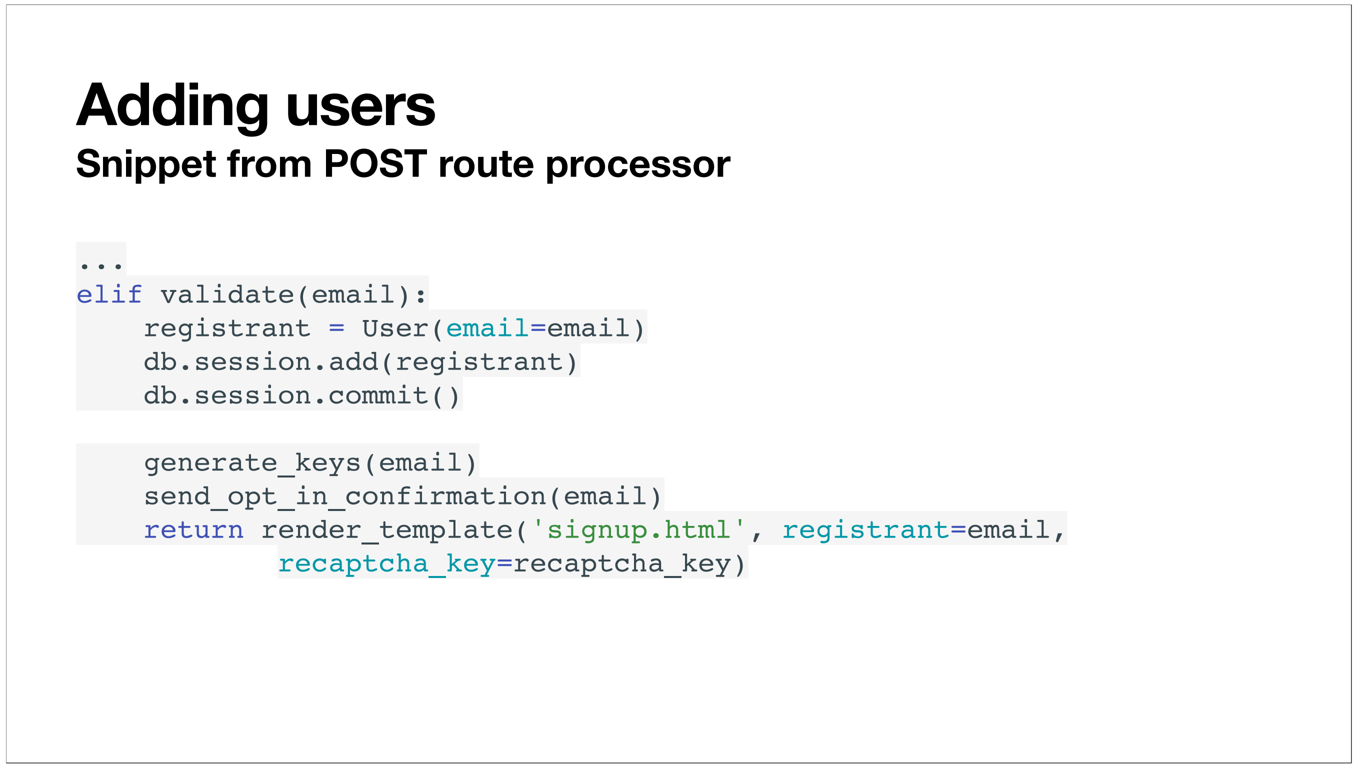
This will come back later.



Database model

```python
class User(Base):
    """The user model."""

    __tablename__ = 'user'
    id = Column(Integer, primary_key=True)
    email = Column(String(255), nullable=False, unique=True)
    opt_in_code = Column(String(255), nullable=True)
    opt_out_code = Column(String(255), nullable=True)
    opt_ins_sent = Column(Integer, nullable=False, default=0)
    last_message_time = Column(DateTime(timezone=False), nullable=True)
```

- Back to the database—we want to create a user class.
- We need
    - opt-in code
    - opt-out code
    - some way to track how many emails we’ve sent so we don’t exceed limits

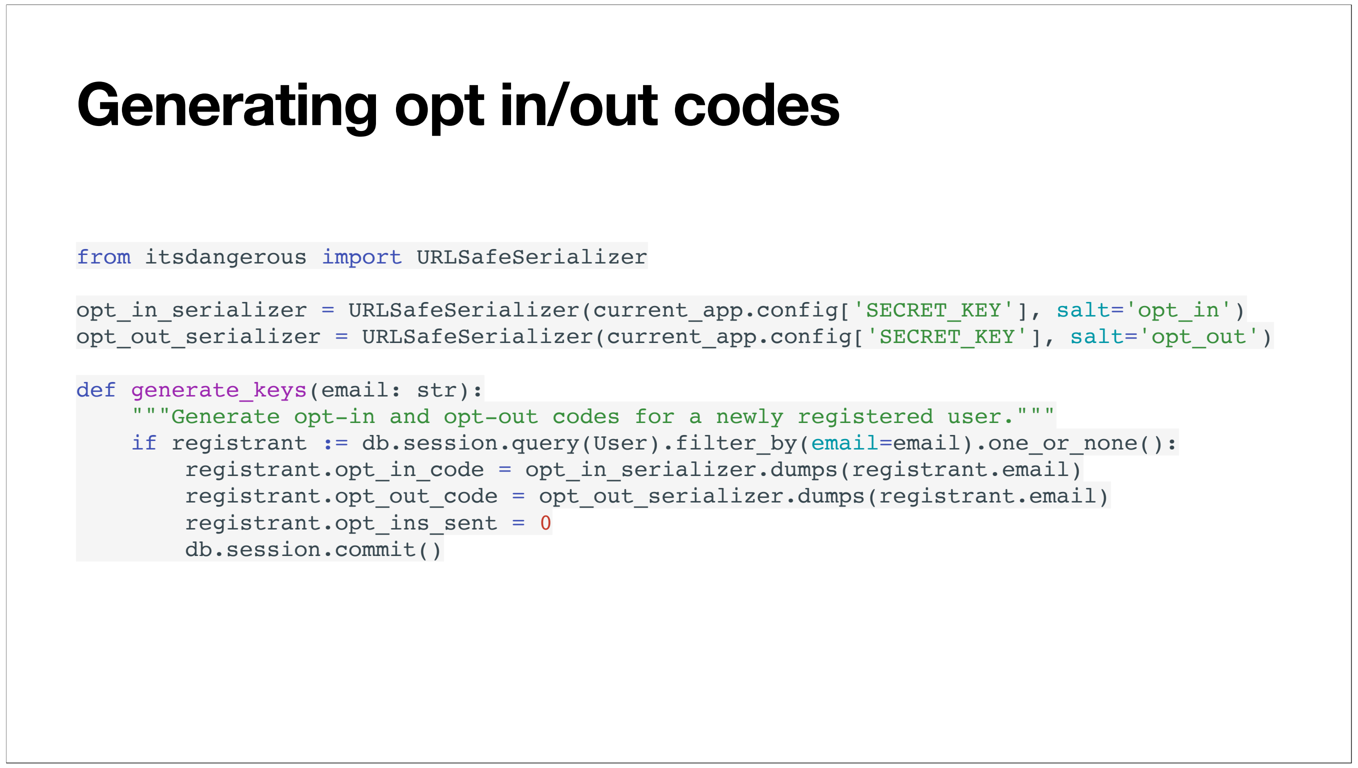


Adding users

Snippet from POST route processor

```
...
elif validate(email):
    registrant = User(email=email)
    db.session.add(registrant)
    db.session.commit()

    generate_keys(email)
    send_opt_in_confirmation(email)
    return render_template('signup.html', registrant=email,
            recaptcha_key=recaptcha_key)
```

- The Flask endpoint is more complicated than this
- Bottom line: if user creates a POST request, we:
    - Parse their email from the request
    - Generate the opt-in and opt-out keys
    - Send a confirmation email
    - Render a page showing a successful subscription

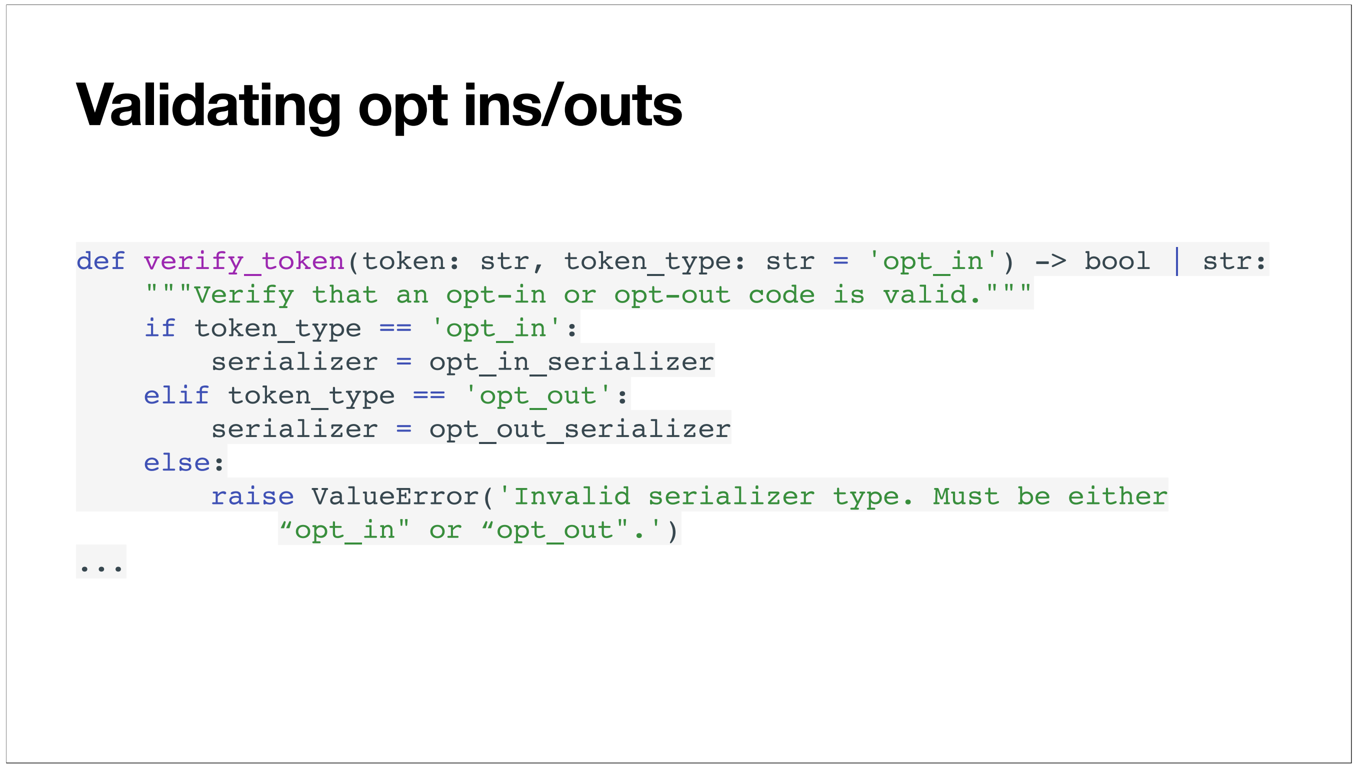


Generating opt in/out codes

```python
from itsdangerous import URLSafeSerializer

opt_in_serializer = URLSafeSerializer(current_app.config['SECRET_KEY'], salt='opt_in')
opt_out_serializer = URLSafeSerializer(current_app.config['SECRET_KEY'], salt='opt_out')

def generate_keys(email: str):
    """Generate opt-in and opt-out codes for a newly registered user."""
    if registrant := db.session.query(User).filter_by(email=email).one_or_none():
        registrant.opt_in_code = opt_in_serializer.dumps(registrant.email)
        registrant.opt_out_code = opt_out_serializer.dumps(registrant.email)
        registrant.opt_ins_sent = 0
        db.session.commit()
```

This is how we generate those keys.

- Probably not necessary
- itsdangerous (part of Flask) has a Serializer class that can serialize and sign data
    - There is a URLSafeSerializer that can be used to pass data in GET requests
- I created two serializers using the Flask secret key, salted them differently depending on the type of serializer, and serialized the user’s email with both
- Committed to database

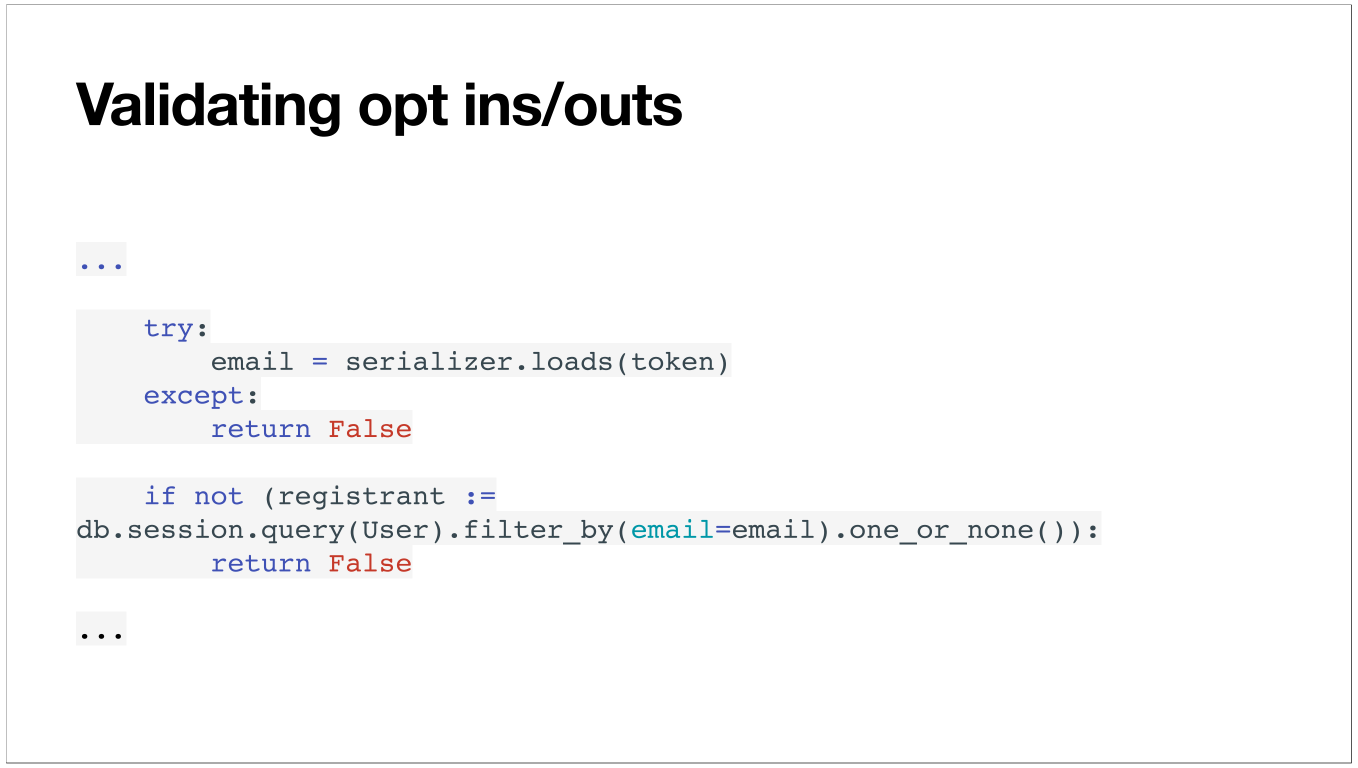


Validating opt ins/outs

```python
def verify_token(token: str, token_type: str = 'opt_in') -> bool | str:
    """Verify that an opt-in or opt-out code is valid."""
    if token_type == 'opt_in':
        serializer = opt_in_serializer
    elif token_type == 'opt_out':
        serializer = opt_out_serializer
    else:
        raise ValueError('Invalid serializer type. Must be either
            “opt_in" or “opt_out".')
...
```

To verify the opt-in and opt-out keys, we can reverse the serialization using our secret key

First we choose a serializer

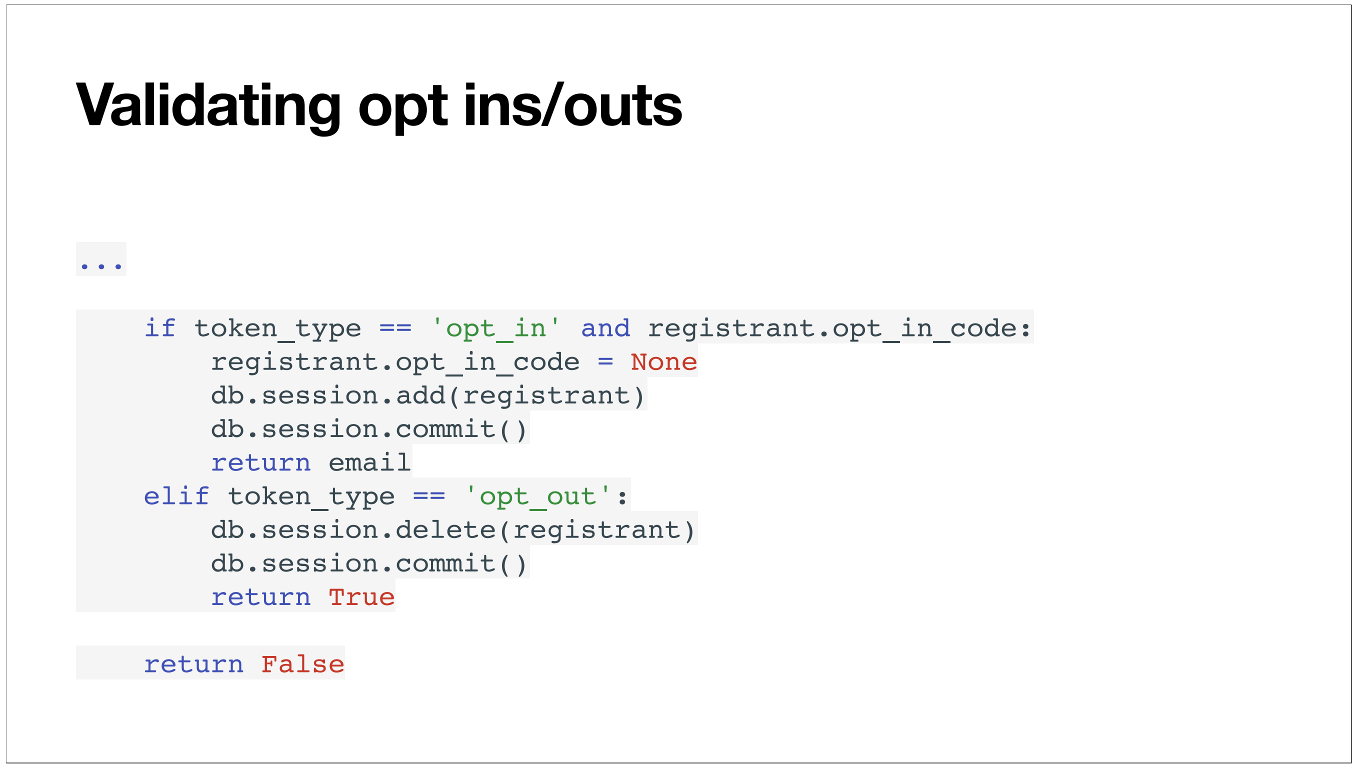


Validating opt ins/outs

```
...

    try:
        email = serializer.loads(token)
    except:
        return False

    if not (registrant :=
db.session.query(User).filter_by(email=email).one_or_none()):
        return False

...
```

Then we try to decode the email and get the corresponding user

If the user is not registered, we quit



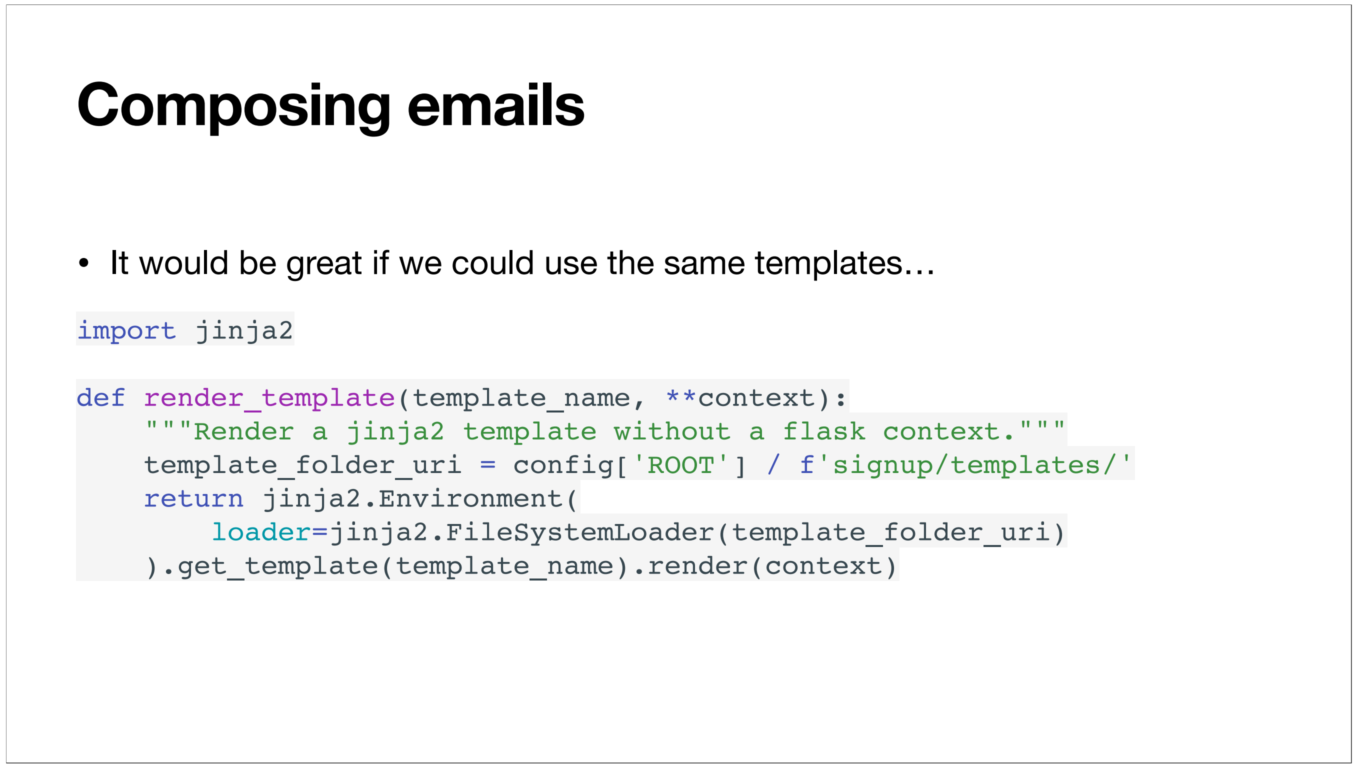
Validating opt ins/outs

```
...

    if token_type == 'opt_in' and registrant.opt_in_code:
        registrant.opt_in_code = None
        db.session.add(registrant)
        db.session.commit()
        return email
    elif token_type == 'opt_out':
        db.session.delete(registrant)
        db.session.commit()
        return True

    return False
```

Finally, we do something…

If we have an opt-in code, we wipe it out. If opt-in code = NULL, user is active.

If we have an opt-out code, we delete the row from the user table



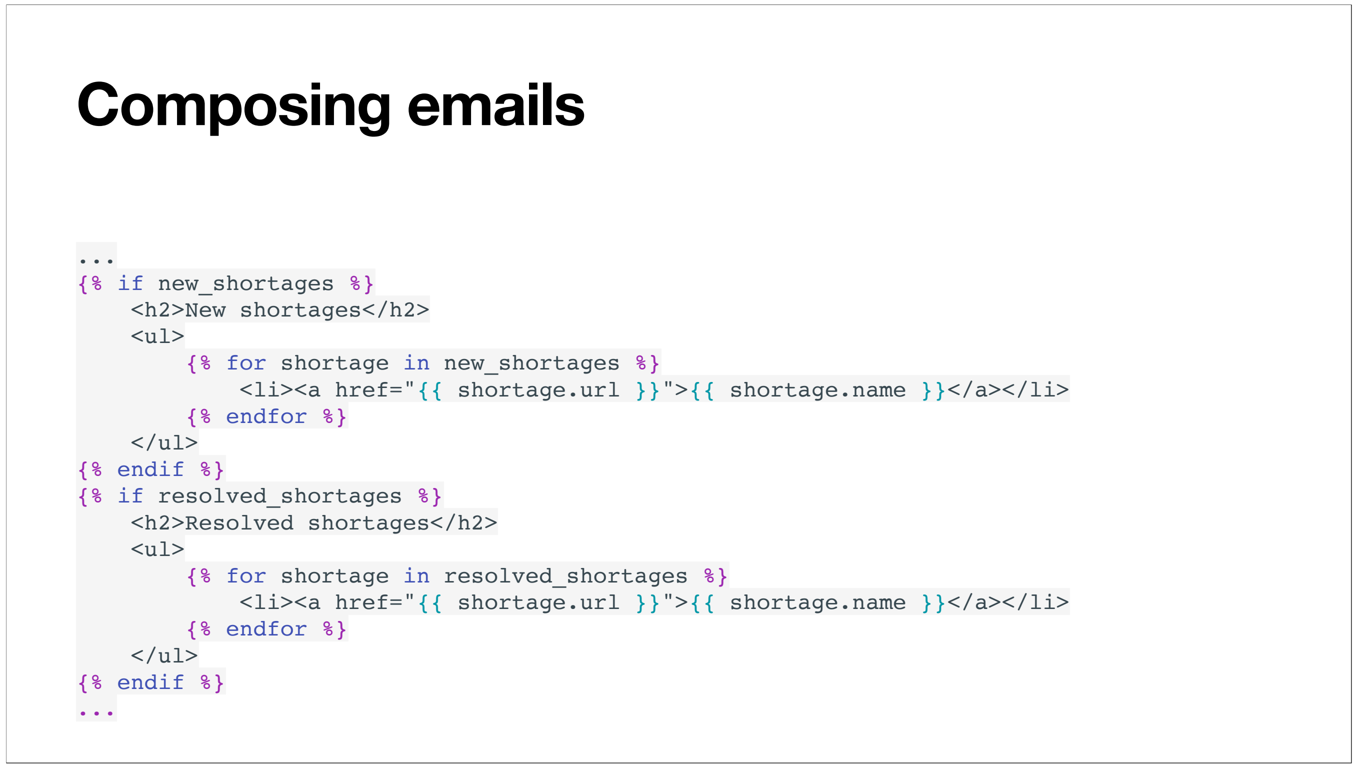
Composing emails

- It would be great if we could use the same templates…

```python
import jinja2

def render_template(template_name, **context):
    """Render a jinja2 template without a flask context."""
    template_folder_uri = config['ROOT'] / f'signup/templates/'
    return jinja2.Environment(
        loader=jinja2.FileSystemLoader(template_folder_uri)
    ).get_template(template_name).render(context)
```

Now that we have a list of users, we need to actually compose some emails

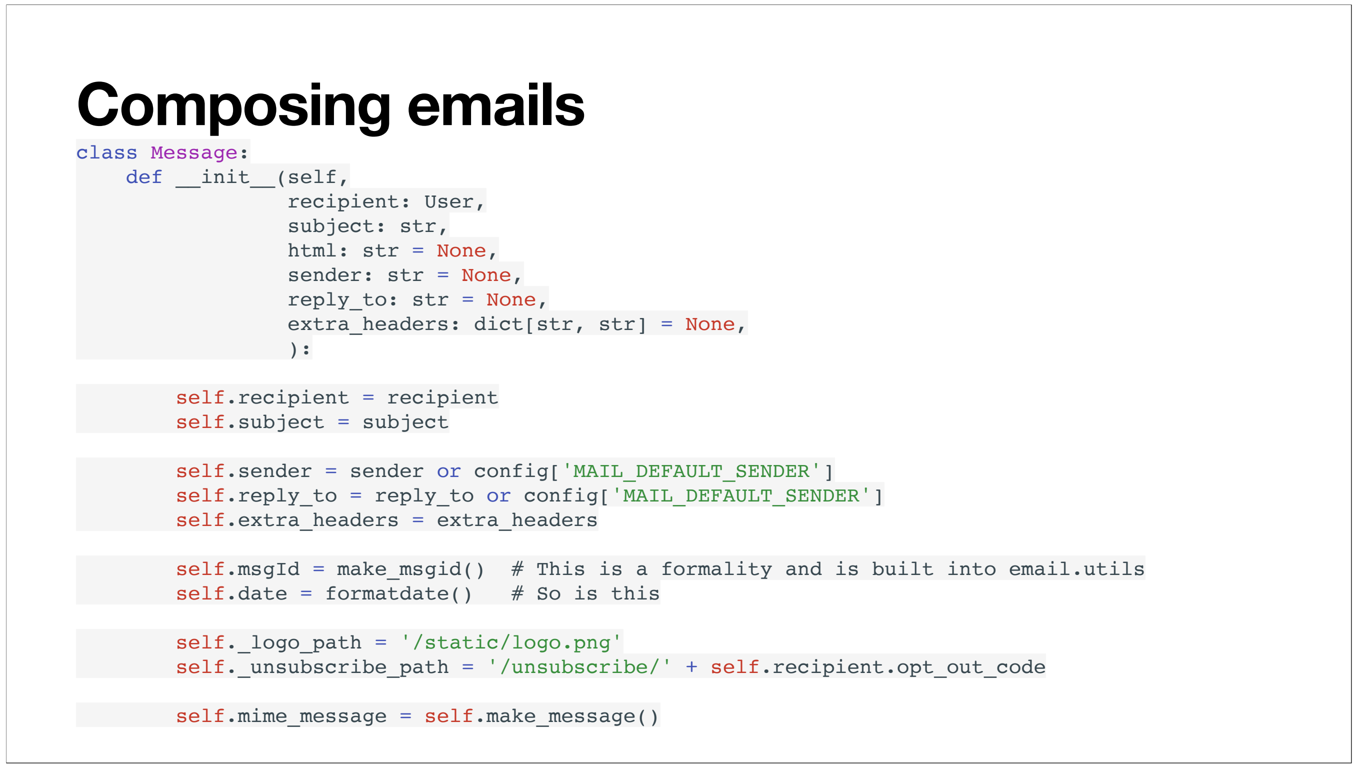
We are already using Jinja for Flask…



Composing emails

```
...
{% if new_shortages %}
    <h2>New shortages</h2>
    <ul>
        {% for shortage in new_shortages %}
            <li><a href="{{ shortage.url }}">{{ shortage.name }}</a></li>
        {% endfor %}
    </ul>
{% endif %}
{% if resolved_shortages %}
    <h2>Resolved shortages</h2>
    <ul>
        {% for shortage in resolved_shortages %}
            <li><a href="{{ shortage.url }}">{{ shortage.name }}</a></li>
        {% endfor %}
    </ul>
{% endif %}
...
```

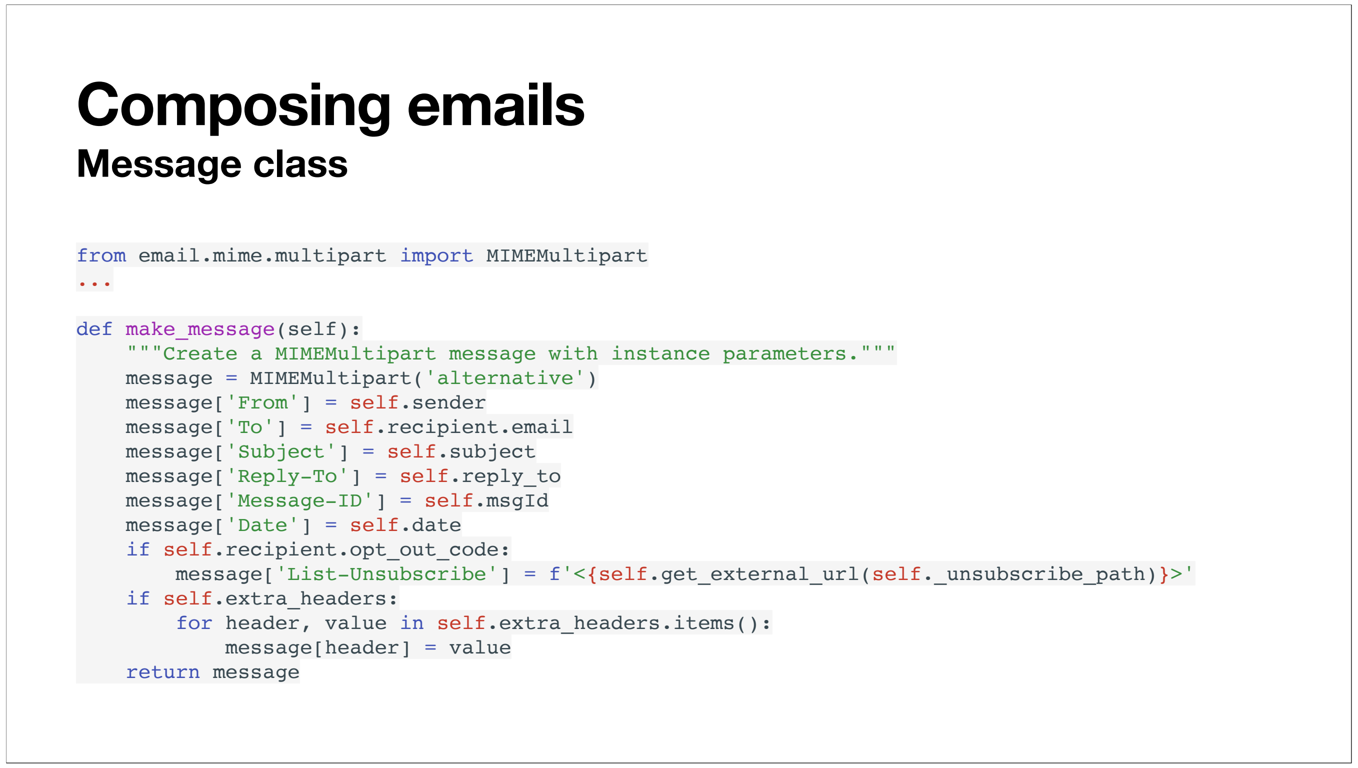
I created a Jinja template that will dynamically list the new and resolved shortages from the class instance we populated earlier



Composing emails

```python
class Message:
    def __init__(self,
                 recipient: User,
                 subject: str,
                 html: str = None,
                 sender: str = None,
                 reply_to: str = None,
                 extra_headers: dict[str, str] = None,
                 ):

        self.recipient = recipient
        self.subject = subject

        self.sender = sender or config['MAIL_DEFAULT_SENDER']
        self.reply_to = reply_to or config['MAIL_DEFAULT_SENDER']
        self.extra_headers = extra_headers

        self.msgId = make_msgid()  # This is a formality and is built into email.utils
        self.date = formatdate()   # So is this

        self._logo_path = '/static/logo.png'
        self._unsubscribe_path = '/unsubscribe/' + self.recipient.opt_out_code

        self.mime_message = self.make_message()
```

- Let’s make this usable by writing a class to represent an email message…
- There are other ways to do this. I used native libraries because why not.
- [Review code]
- At the end, we create a MIMEMultipartMessage using the mime_message method

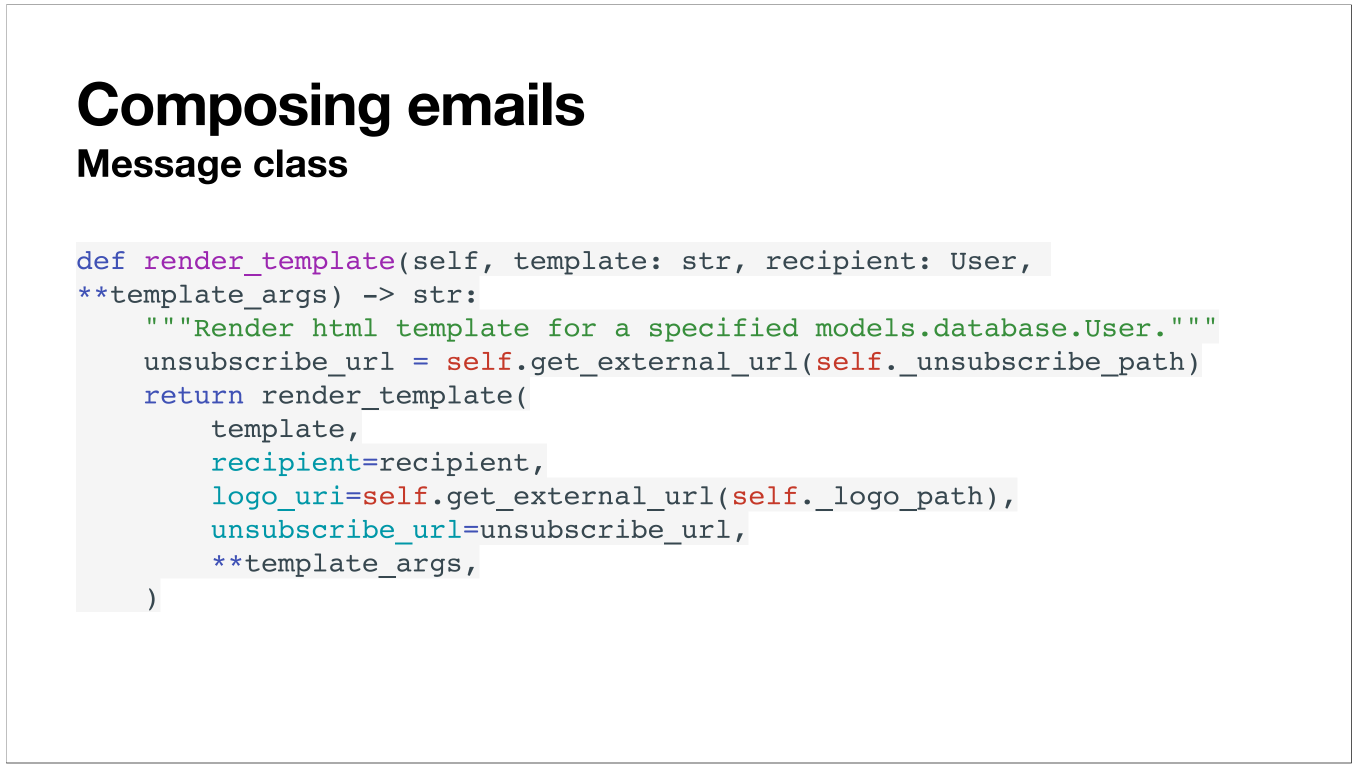


Composing emails

Message class

```python
from email.mime.multipart import MIMEMultipart
...

def make_message(self):
    """Create a MIMEMultipart message with instance parameters."""
    message = MIMEMultipart('alternative')
    message['From'] = self.sender
    message['To'] = self.recipient.email
    message['Subject'] = self.subject
    message['Reply-To'] = self.reply_to
    message['Message-ID'] = self.msgId
    message['Date'] = self.date
    if self.recipient.opt_out_code:
        message['List-Unsubscribe'] = f'<{self.get_external_url(self._unsubscribe_path)}>'
    if self.extra_headers:
        for header, value in self.extra_headers.items():
            message[header] = value
    return message
```

- Python has this built in: from import email.mime.multipart import MIMEMultipart
- Add our email headers
- Return a message that will be stored in our mime_message instance variable

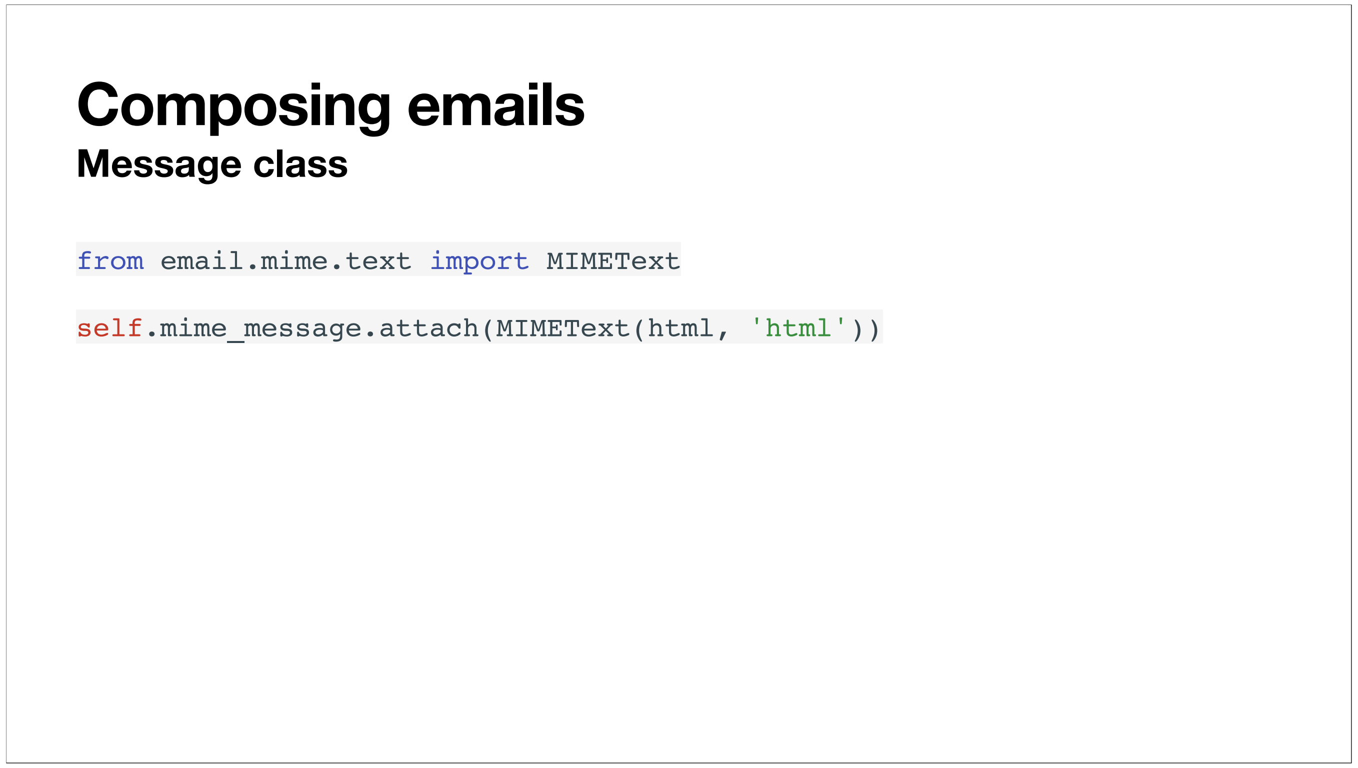


Composing emails

Message class

```python
def render_template(self, template: str, recipient: User,
**template_args) -> str:
    """Render html template for a specified models.database.User."""
    unsubscribe_url = self.get_external_url(self._unsubscribe_path)
    return render_template(
        template,
        recipient=recipient,
        logo_uri=self.get_external_url(self._logo_path),
        unsubscribe_url=unsubscribe_url,
        **template_args,
    )
```

We can use our previous render_template **function** to create a render template method for our class. This will take the variables we instantiated earlier and use them to generate the HTML to actually send with the email.



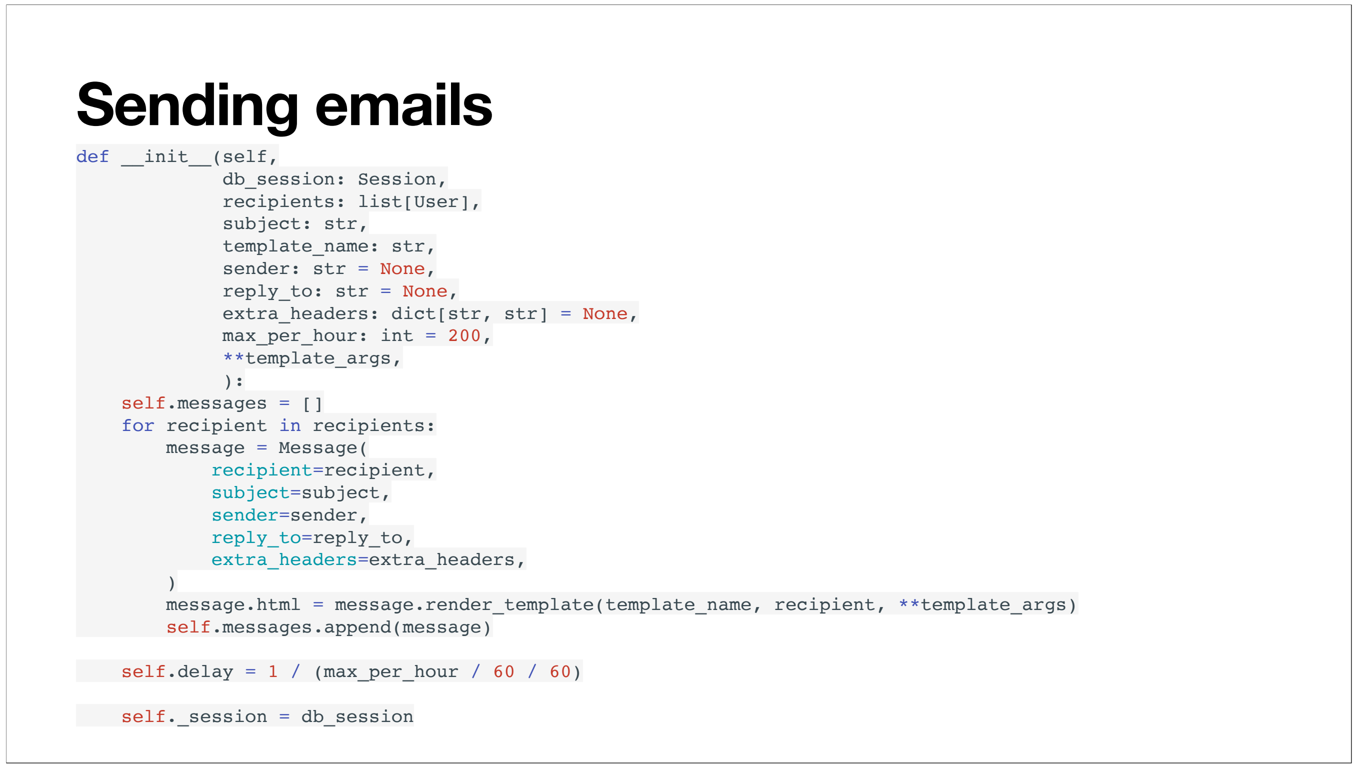
Composing emails

Message class

```python
from email.mime.text import MIMEText

self.mime_message.attach(MIMEText(html, 'html'))
```

Finally, we can attach the HTML we generated to the mime_message

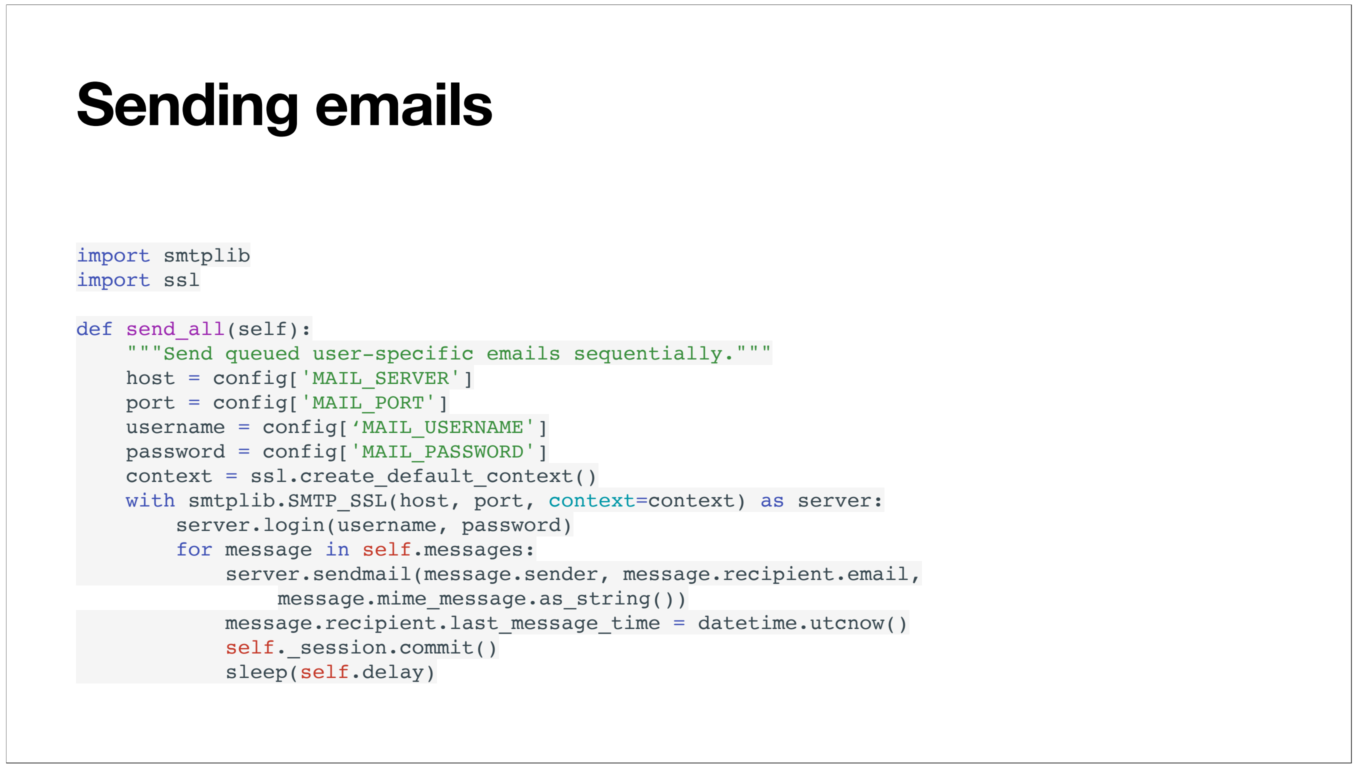
Do this by importing MIMEText and calling attach



Sending emails

```python
def __init__(self,
             db_session: Session,
             recipients: list[User],
             subject: str,
             template_name: str,
             sender: str = None,
             reply_to: str = None,
             extra_headers: dict[str, str] = None,
             max_per_hour: int = 200,
             **template_args,
             ):
    self.messages = []
    for recipient in recipients:
        message = Message(
            recipient=recipient,
            subject=subject,
            sender=sender,
            reply_to=reply_to,
            extra_headers=extra_headers,
        )
        message.html = message.render_template(template_name, recipient, **template_args)
        self.messages.append(message)

    self.delay = 1 / (max_per_hour / 60 / 60)

    self._session = db_session
```

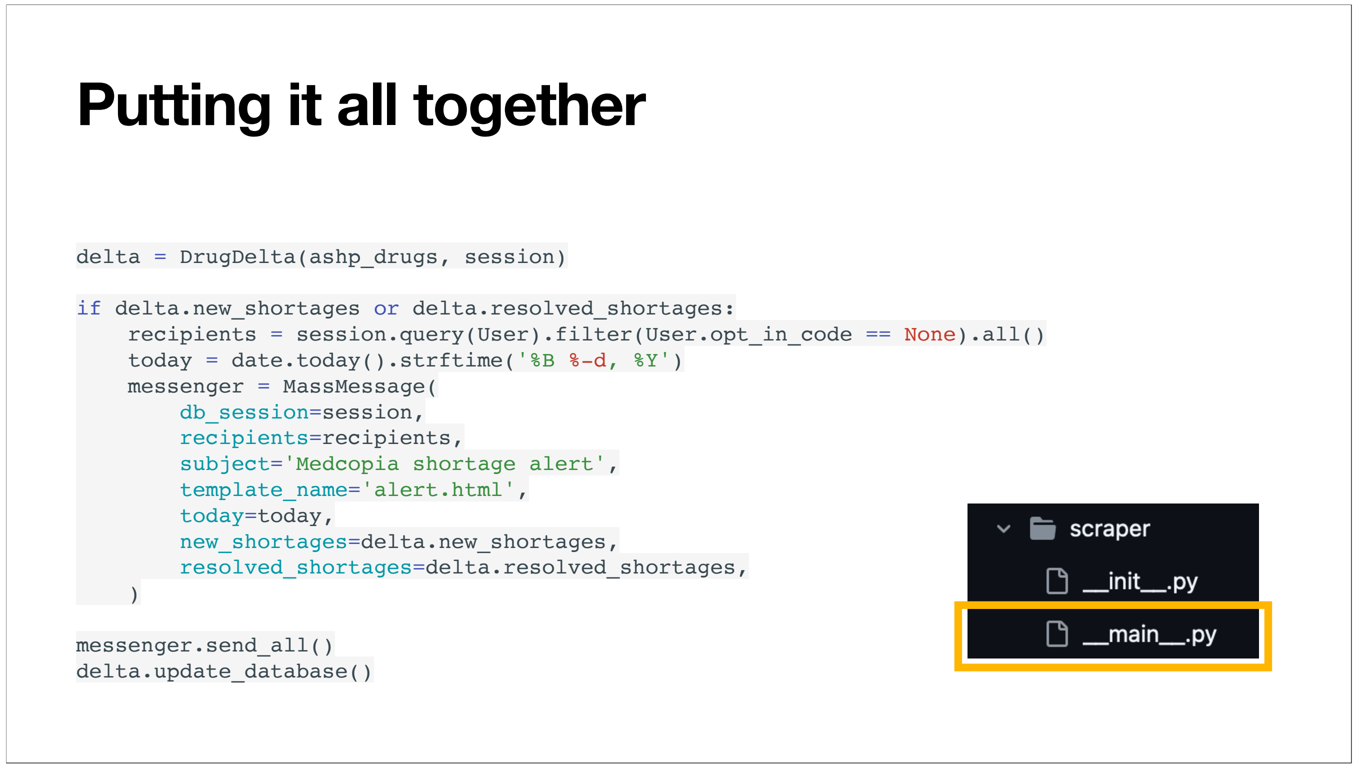
- Finally we create a class to handle each mass mailing.
- Cut off but it’s called MassMessage
- Pass a list of messages and render a unique template for each
- Add a delay
    - e.g., My mailer allows 300 emails per hour. Divide by 60 to get 0.083 emails per second. Invert it to get seconds between each email. Use this to sleep between sends.



Sending emails

```python
import smtplib
import ssl

def send_all(self):
    """Send queued user-specific emails sequentially."""
    host = config['MAIL_SERVER']
    port = config['MAIL_PORT']
    username = config[‘MAIL_USERNAME']
    password = config['MAIL_PASSWORD']
    context = ssl.create_default_context()
    with smtplib.SMTP_SSL(host, port, context=context) as server:
        server.login(username, password)
        for message in self.messages:
            server.sendmail(message.sender, message.recipient.email,
                message.mime_message.as_string())
            message.recipient.last_message_time = datetime.utcnow()
            self._session.commit()
            sleep(self.delay)
```

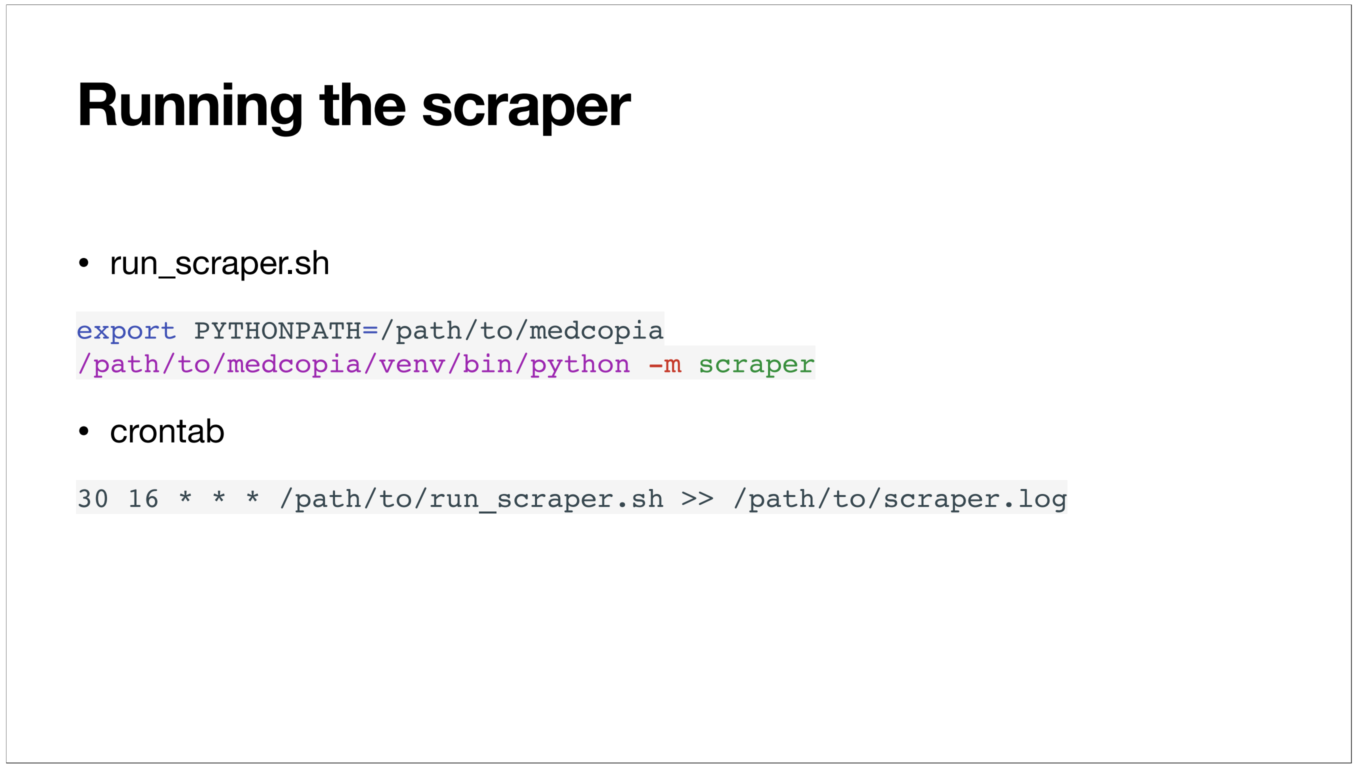
SLEEP function at the end



Putting it all together

```
delta = DrugDelta(ashp_drugs, session)

if delta.new_shortages or delta.resolved_shortages:
    recipients = session.query(User).filter(User.opt_in_code == None).all()
    today = date.today().strftime('%B %-d, %Y')
    messenger = MassMessage(
        db_session=session,
        recipients=recipients,
        subject='Medcopia shortage alert',
        template_name='alert.html',
        today=today,
        new_shortages=delta.new_shortages,
        resolved_shortages=delta.resolved_shortages,
    )

messenger.send_all()
delta.update_database()
```

Finally we can put everything all together.
- Module with dunder main will:
    - Do the scraping (code shown earlier)
    - Create a DrugDelta
    - If there are shortages, create a MassMessage
    - Send all the emails
    - Update the database



Running the scraper

- run_scraper.sh

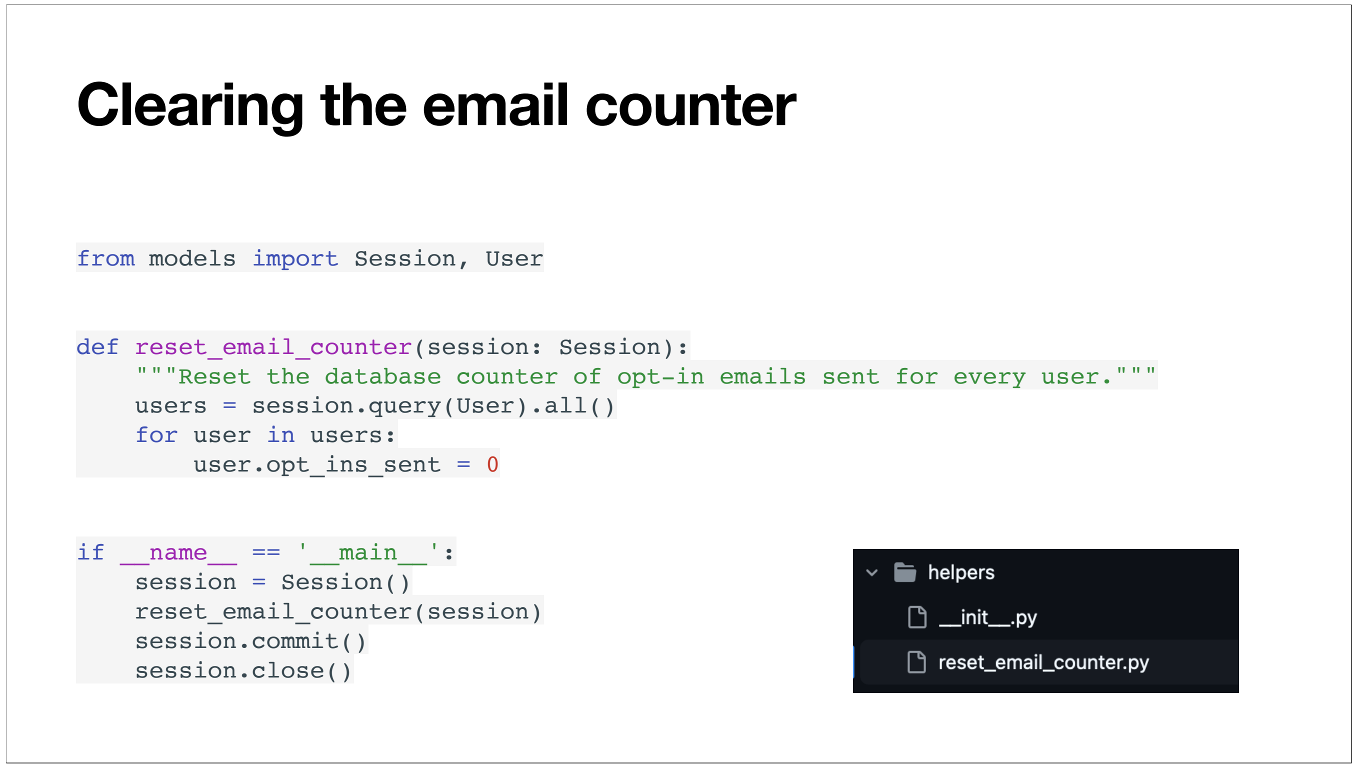
```
export PYTHONPATH=/path/to/medcopia
/path/to/medcopia/venv/bin/python -m scraper
```

- crontab

```
30 16 * * * /path/to/run_scraper.sh >> /path/to/scraper.log
```

I put it in a shell script.

It runs as a cron job daily at 4:30p.

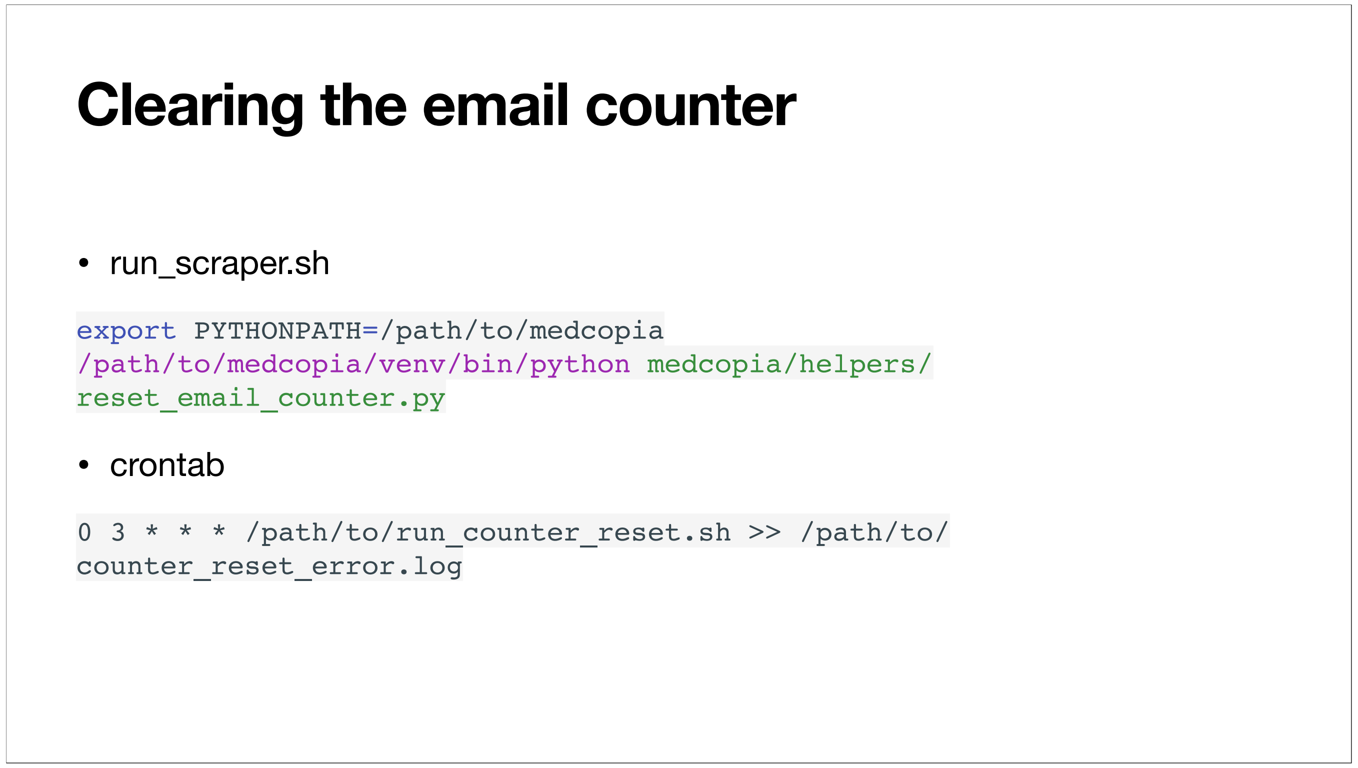


Clearing the email counter

```python
from models import Session, User


def reset_email_counter(session: Session):
    """Reset the database counter of opt-in emails sent for every user."""
    users = session.query(User).all()
    for user in users:
        user.opt_ins_sent = 0


if __name__ == '__main__':
    session = Session()
    reset_email_counter(session)
    session.commit()
    session.close()
```

Last but not least, we want to reset the number of opt-ins sent daily.

This is just a helper script.



Clearing the email counter

- run_scraper.sh

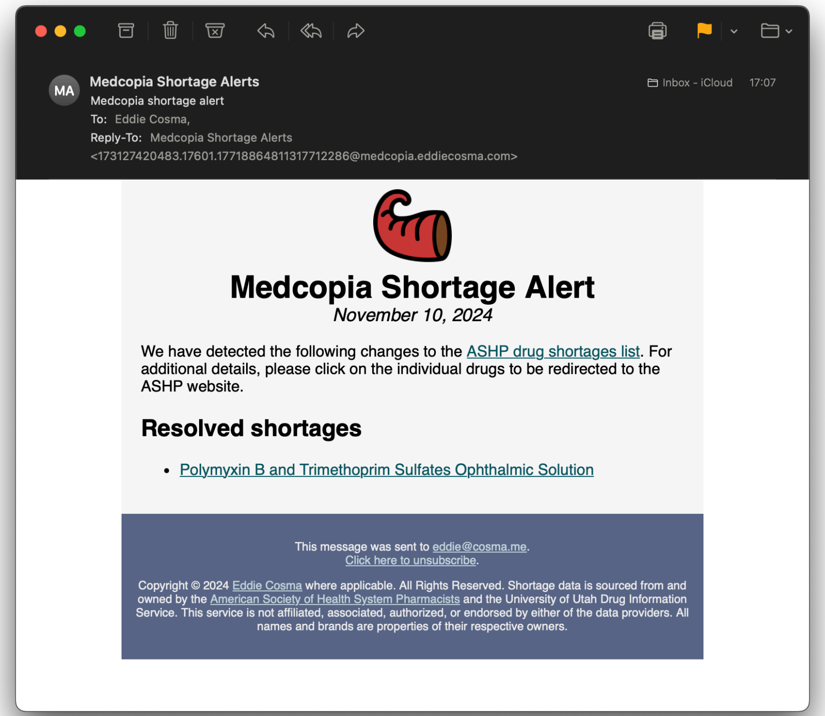
```
export PYTHONPATH=/path/to/medcopia
/path/to/medcopia/venv/bin/python medcopia/helpers/reset_email_counter.py
```

- crontab

```
0 3 * * * /path/to/run_counter_reset.sh >> /path/to/counter_reset_error.log
```

Also called by a shell script and built as a cron job.

Victory



MA Medcopia Shortage Alerts
Medcopia shortage alert
To: Eddie Cosma,
Reply-To: Medcopia Shortage Alerts
<173127420483.17601.17718864811317712286@medcopia.eddiecosma.com>

Inbox - iCloud 17:07

Medcopia Shortage Alert

November 10, 2024

We have detected the following changes to the ASHP drug shortages list. For additional details, please click on the individual drugs to be redirected to the ASHP website.

Resolved shortages

- Polymyxin B and Trimethoprim Sulfates Ophthalmic Solution

This message was sent to eddie@cosma.me.
Click here to unsubscribe.

The end result.