Blogful project

This project is to build a blog using Flask with features of login and others.

Create the model

```
1. import datetime
2.
3. from sqlalchemy import Column, Integer, String, Text, DateTime
4.
5. class Entry(Base):
6.     __tablename__ = "entries"
7.
8.     id = Column(Integer, primary_key=True)
9.     title = Column(String(1024))
10.     content = Column(Text)
11.     datetime = Column(DateTime,
  default=datetime.datetime.now)
12.
13. Base.metadata.create_all(engine)
```

Add some data as:

```
from blog.database import session, Entry

@manager.command
def seed():
    content = """Lorem ipsum dolor sit amet, consectetur adipisicing elit, sed do eiusmod tempor incididunt ut labore et dolore magna aliqua. Ut enim ad minim veniam, quis nostrud exercitation ullamco laboris nisi ut aliquip ex ea commodo consequat. Duis aute irure dolor in reprehenderit in voluptate velit esse cillum dolore eu fugiat nulla pariatur. Excepteur sint occaecat cupidatat non proident, sunt in culpa qui officia deserunt mollit anim id est laborum."""

    for i in range(25):
        entry = Entry(
            title="Test Entry #{}".format(i),
```

```
            content=content
        )
        session.add(entry)
    session.commit()
```

Display the entries as:

```
from flask import render_template

from . import app
from .database import session, Entry

@app.route("/")
def entries():
    entries = session.query(Entry)
    entries = entries.order_by(Entry.datetime.desc())
    entries = entries.all()
    return render_template("entries.html",
        entries=entries
    )
```

Adding pagination

Although the view works fine for 25 entries, what about if you have thousands? It is unlikely that anyone will need to see all of these entries at once, and it will make rendering far slower. So try modifying the view slightly to introduce pagination.

> You can use all of this data to calculate a number of pieces of information about the pagination:
>
> - `start` - The index of the first item that you should see
> - `end` - The index of the last item that you should see
> - `total_pages` - The total number of pages of content
> - `has_next` - Whether there is a page after the current one
> - `has_prev` - Whether there is a page before the current one

Adding blog entries

Create the `/entry/add` page of your app. You should see the form for adding an entry. You should be able to post the new entries to entries table.

Expand your code to add the following features:

- `/entry/<id>`
  - Allows you to view a single entry
  - Should be accessed by clicking on the title of an entry
  - Should use the `render_entry` macro to display the entry
- `/entry/<id>/edit`
  - Allows you to edit an entry
  - Should be accessed via a link in the metadata div
  - Should display a similar form to the `add_entry` view
  - The form should be prepopulated with the existing entry data
- `/entry/<id>/delete`
  - Allows you to delete an entry
  - Should be accessed via a link in the metadata div
  - Should display buttons asking you to confirm or cancel the deletion
- `/?limit=20` and `/page/2?limit=20`
  - Allows you to customize the number of entries per page
  - Should be accessed via a link or drop-down "Entries per page:"
  - Use `request.args.get` to find out what was entered
  - Be sure to cope with the ridiculous cases (eg limit=0, limit=99999999, limit=3.14, limit=spam), eg by reverting to the default limit

Authentication using Flask-login

Next you will be building a login system based on the Flask-Login module which will allow you to limit the people who are able to make changes to the blog.

Adding a user, follow the model as:

```
from flask.ext.login import UserMixin

class User(Base, UserMixin):
    __tablename__ = "users"

    id = Column(Integer, primary_key=True)
    name = Column(String(128))
    email = Column(String(128), unique=True)
    password = Column(String(128))
```

Setting up the login system (login.py)

```
from flask.ext.login import LoginManager

from . import app
from .database import session, User

login_manager = LoginManager()
login_manager.init_app(app)

login_manager.login_view = "login_get"
login_manager.login_message_category = "danger"

@login_manager.user_loader
def load_user(id):
    return session.query(User).get(int(id))
```

Protecting resources

The login system is to actually protect the resources which require authentication. This is done using the `login_required` decorator from Flask-Login.

Try visiting the `/entry/add` page whilst not logged in. You should find that you are redirected to the login page. Then when you log in you should be redirected to the add entry form, and be able to post content.

Database migrations using Flask-Migrate

You already have a load of information stored in the database which you don't want to lose by recreating the entire schema. So how do you transition between the old database models and the new one?

Use migrations to do so.