Assignment 3: Chatbot

Due: 10/29 (Monday) 6am

Overview

This assignment is completed in groups of 2-4 students.

Your task for this assignment is a build a chatbot for Slack that attempts a task that requires a human connection. Some possibilities include:

- Teaching someone a specific topic (eg. understanding and applying Bayes rule, understanding and applying Boolean logic, etc.)
- Counseling someone who is struggling emotionally (eg. from a bad breakup, from a lost of a loved one, etc.)
- Debating someone on a contentious topic (eg. abortion rights, gun rights, euthanasia, etc.)

Note that this is *intentionally difficult, if not impossible* to do perfectly. One of the learning goals of this assignment is discovering the limits chatbots - but also to see how far we can push against them. The constraint that your chatbot should tap into human connections is part of the challenge, and asks whether chatbots can do more than scheduling a hair salon appointment.

As you are designing the chatbot, keep in mind that these are topics where "multiple choice" responses are not appropriate, nor the simple parroting we've seen with ELIZA. In fact, these are situations where saying the wrong thing may cause more harm than not saying anything at all.

Some initial design questions to consider:

- What are the advantages of a conversational in your situation, and how are you taking advantage of them?
- What kind of

You will be writing this chatbot in Python. Some starter code has been provided, although you are free to ignore it and/or to use other frameworks if you want.

Deliverables

In addition to the working code, submitted as a link to GitHub, you will also submit the following:

1. A screen-capture video of your chatbot in action, which showcases the main design features of your chatbot. Please use a native screen-capturing software instead of filming shaky video with your phone.
2. A blog posts, written individually, describing the design process. As before, this should describe the overall arc of how you developed the final product, including the brainstorm, iteration, and final polishing. Additionally, your blog post should include the following:
    - Transcripts from user tests and what you learned from them

- Description of technical limitations that prevented your chatbot from responding correctly

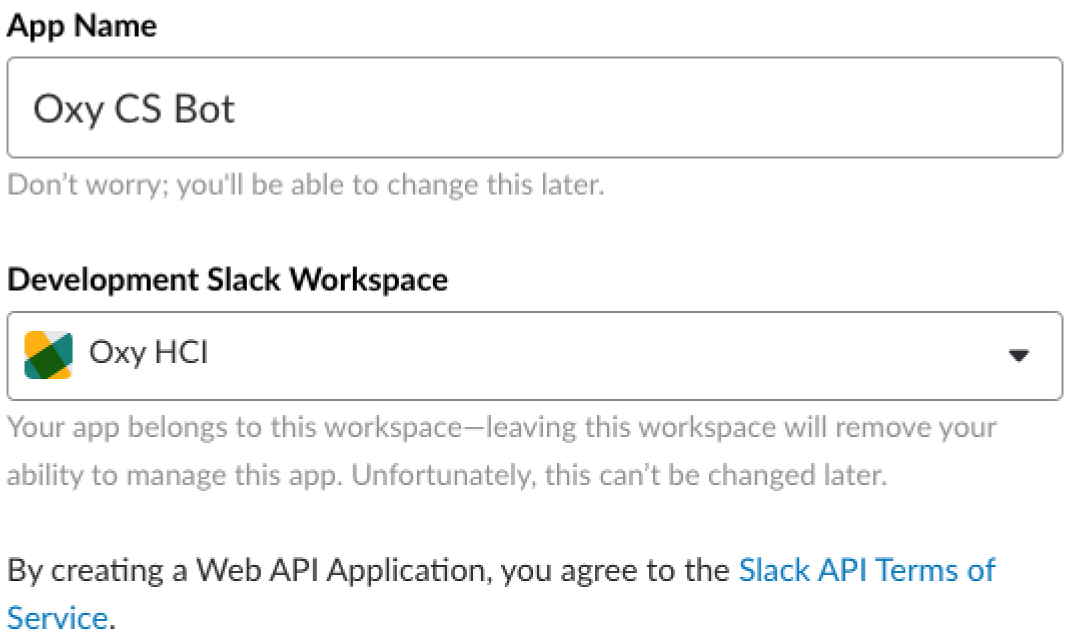
Setting up your chatbot

Setting up your chatbot is a fairly involved process, but only one person needs to do this.

Slack

Slack is a instant messaging platform, mostly used by tech companies to manage teams. You should have received an invitation to join the Oxy HCI workspace, or you can click here to join.

All chatbots are Slack "apps", so you'll need to create one. Go to https://api.slack.com/apps and create an app. You can call your app whatever you want, but make sure you set Oxy HCI as the development workspace.



App Name

Oxy CS Bot

Don't worry; you'll be able to change this later.

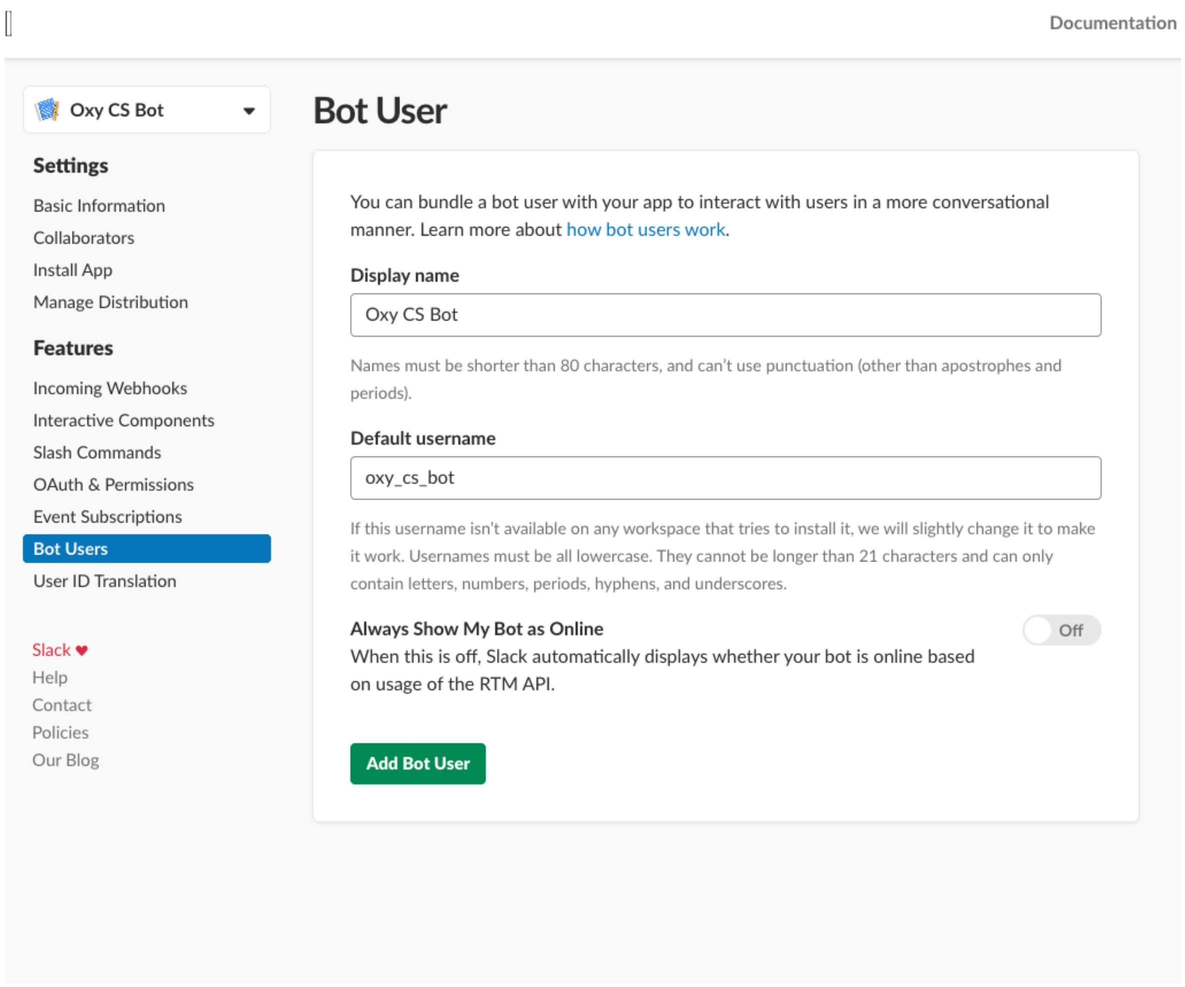
Development Slack Workspace

Oxy HCI

Your app belongs to this workspace—leaving this workspace will remove your ability to manage this app. Unfortunately, this can't be changed later.

By creating a Web API Application, you agree to the Slack API Terms of Service.

Once this is done, go to the Bot Users section of the app settings page, and click Add Bot User. Here you can configure what username the bot will have.



Bot User

You can bundle a bot user with your app to interact with users in a more conversational manner. Learn more about how bot users work.

Display name

Oxy CS Bot

Names must be shorter than 80 characters, and can't use punctuation (other than apostrophes and periods).

Default username

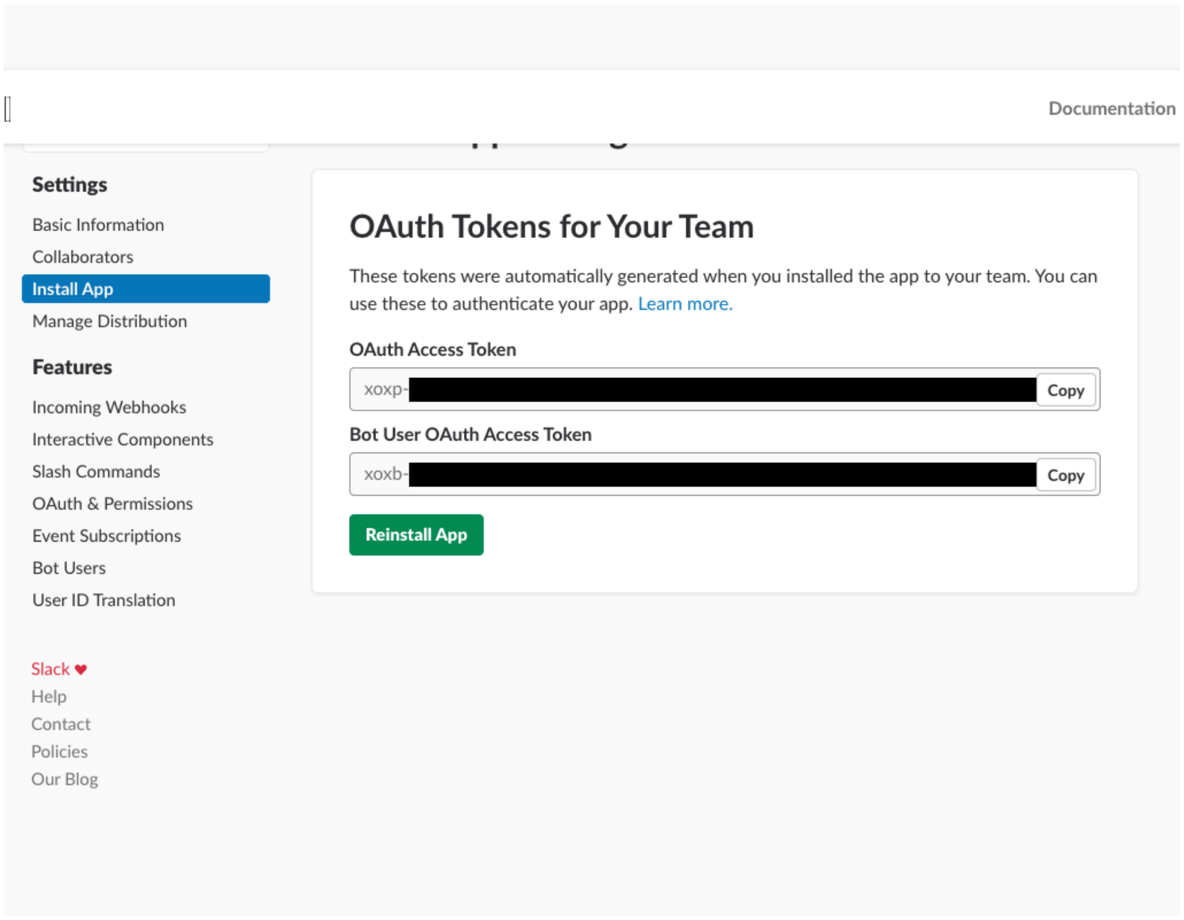
oxy_cs_bot

If this username isn't available on any workspace that tries to install it, we will slightly change it to make it work. Usernames must be all lowercase. They cannot be longer than 21 characters and can only contain letters, numbers, periods, hyphens, and underscores.

Always Show My Bot as Online Off

When this is off, Slack automatically displays whether your bot is online based on usage of the RTM API.

Now go to the Install App section, and click Install App to Workspace. After you authorize the bot, you should be shown a screen with a Bot User OAuth Access Token. Keep this token (and the OAuth Access Token) secret! Anyone with this token can pretend to be your bot.


Documentation

Settings
- Basic Information
- Collaborators
- Install App
- Manage Distribution

Features
- Incoming Webhooks
- Interactive Components
- Slash Commands
- OAuth & Permissions
- Event Subscriptions
- Bot Users
- User ID Translation

Slack ♥
Help
Contact
Policies
Our Blog

OAuth Tokens for Your Team

These tokens were automatically generated when you installed the app to your team. You can use these to authenticate your app. Learn more.

OAuth Access Token

xoxp-[redacted] Copy

Bot User OAuth Access Token

xoxb-[redacted] Copy

Reinstall App

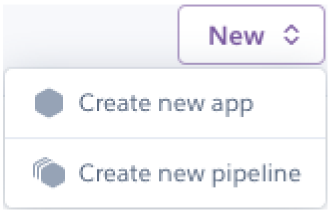
Keep the Bot User OAuth Access Token handy; we will need this in a bit.

GitHub

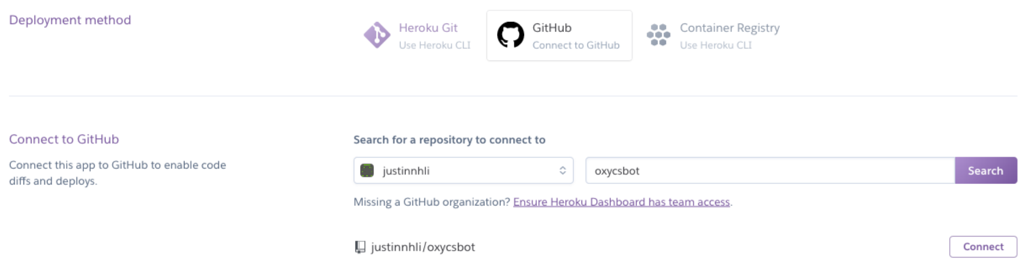
An example chatbot is available on my GitHub. To create your own repository, you can either fork it, or download a zipped copy and add that to your own new repository.

Heroku

To get your chatbot running 24/7, we will be hosting it on Heroku. After you create an account, you can create a new app from the dashboard.



Once you have named your app, you should be dropped into the Deploy tab in Heroku. Follow the instructions to connect your Heroku account to GitHub, then select the repository you just created.


Deployment method

Connect to GitHub

Connect this app to GitHub to enable code diffs and deploys.

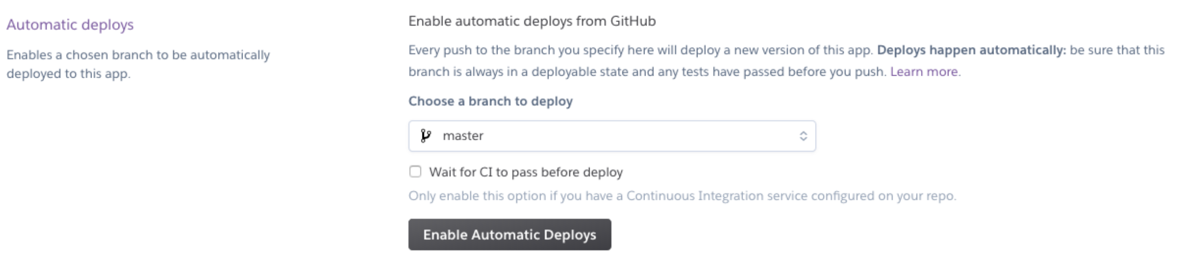
Search for a repository to connect to

justinnhli | oxycsbot | Search

Missing a GitHub organization? Ensure Heroku Dashboard has team access.

justinnhli/oxycsbot | Connect

Enabling automatic deploys means that your bot will automatically update when you push to the GitHub repository.


Automatic deploys

Enables a chosen branch to be automatically deployed to this app.

Enable automatic deploys from GitHub

Every push to the branch you specify here will deploy a new version of this app. **Deploys happen automatically:** be sure that this branch is always in a deployable state and any tests have passed before you push. Learn more.

Choose a branch to deploy

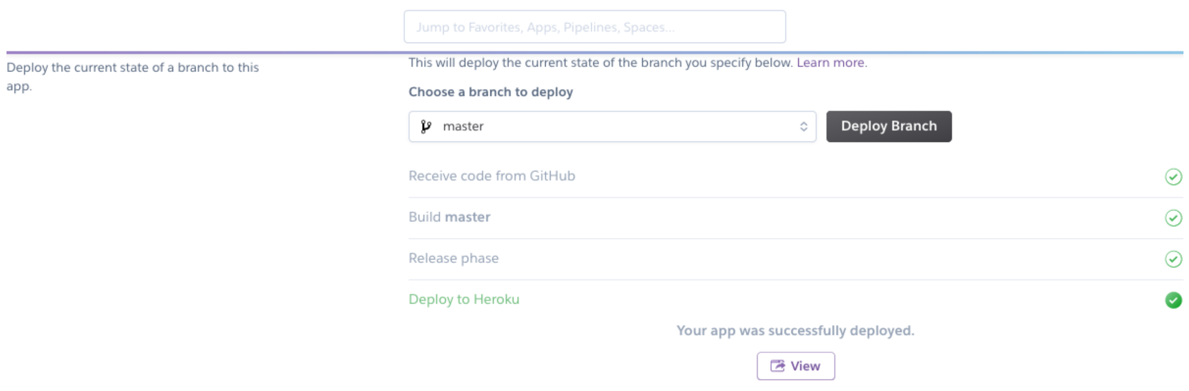
master

Wait for CI to pass before deploy

Only enable this option if you have a Continuous Integration service configured on your repo.

Enable Automatic Deploys

You will still need to manually deploy your chatbot for the first time.



Deploy the current state of a branch to this app.

This will deploy the current state of the branch you specify below. Learn more.

Choose a branch to deploy

master | Deploy Branch

Receive code from GitHub ✓

Build master ✓

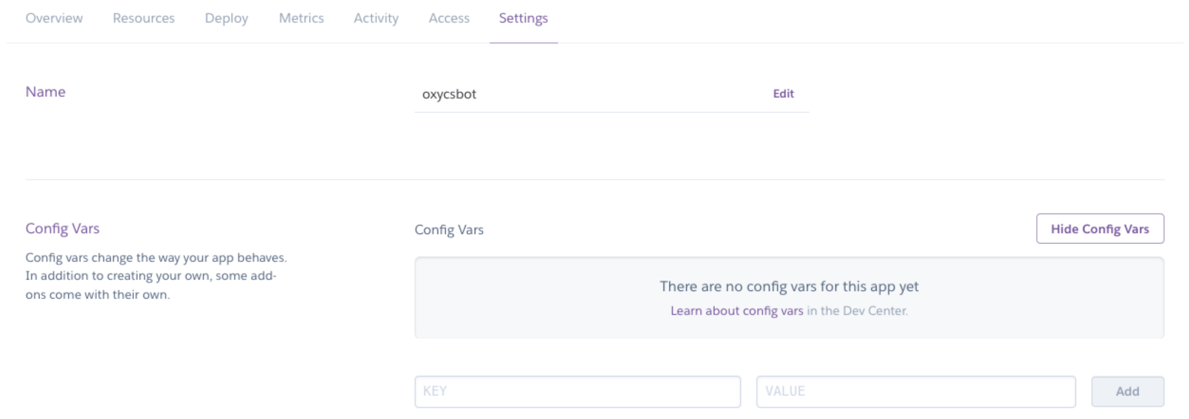
Release phase ✓

Deploy to Heroku ✓

Your app was successfully deployed.

View

When the app has been successfully deployed, go the Settings tab and look for the Config Vars section. Click on Reveal Config Vars.



Name | oxycsbot | Edit

Config Vars

Config vars change the way your app behaves. In addition to creating your own, some add-ons come with their own.

Config Vars | Hide Config Vars

There are no config vars for this app yet

Learn about config vars in the Dev Center.

KEY | VALUE | Add

Remember that Bot User OAuth Access Token from Slack? This is where you need it. Add a config var where the key is "TOKEN" (all upper case, no quotes) and the value is the Bot User OAuth Access Token.

Finally, go the Resources tab. Edit the "worker dyna" by clicking on the edit icon on the right. Flip the switch to on, then confirm the change.

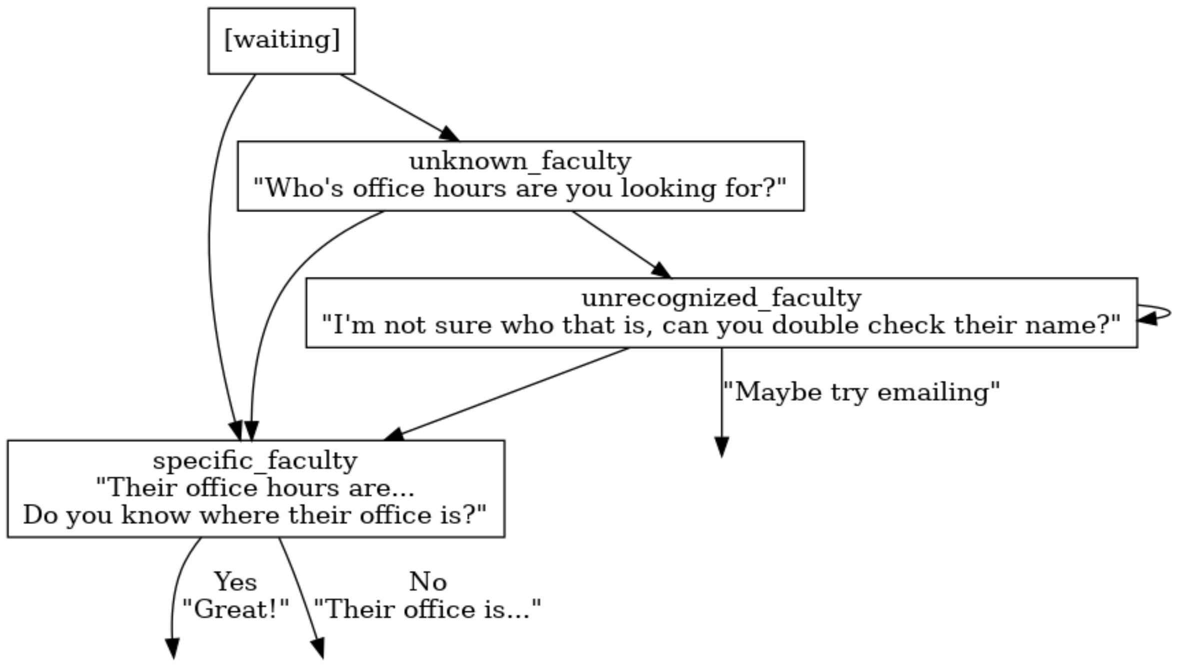
worker
python3 slackbot.py
$0.00
Confirm
Cancel

Testing

Viola! Your bot should be ready. To test it, create a new channel in Slack, and invite your bot user to that channel. Address a message to them; the bot should respond.

Example Chatbot: OxyCSBot

The starter code is available on GitHub as a repository and as a zip file. The function-by-function documentation is in the source code, which I won't repeat here. Instead, let's look at an example chatbot, OxyCSBot, that can direct students to professor's office hours. First, a flow chart of how this bot should react:


[waiting]
unknown_faculty
"Who's office hours are you looking for?"
unrecognized_faculty
"I'm not sure who that is, can you double check their name?"
"Maybe try emailing"
specific_faculty
"Their office hours are...
Do you know where their office is?"
Yes
"Great!"
No
"Their office is..."

This flow chart already defines all the necessary states: the default `waiting` state, and `unknown_faculty`, `unrecognized_faculty`, and `specific_faculty` states. This means we need to write seven state functions (we don't need an `on_enter` function for the default `waiting` state), and several `finish` functions that would quit the flow:

- `respond_from_waiting`
- `on_enter_specific_faculty`
- `respond_from_specific_faculty`
- `on_enter_unknown_faculty`
- `respond_from_unknown_faculty`
- `on_enter_unrecognized_faculty`

- `respond_from_unrecognized_faculty`
- `finish_location`
- `finish_success`
- `finish_fail`
- `finish_confused`
- `finish_thanks`

The last two `finish` functions, for `confused` and `thanks`, are generic and don't show up on the flow chart.

For the tags, we probably want to recognize the name of each professor. This is the main determiner for whether to go to the `specific_faculty` or `unknown_faculty` state. So the code looks like this:

```
for professor in self.PROFESSORS:
    if professor in tags:
        self.professor = professor
        return self.go_to_state('specific_faculty')
return self.go_to_state('unknown_faculty')
```

The rest of the code is not much harder than this, so feel free to look through and see how the bot works.