Django

Django is a very popular Python framework among web developers for a few different reasons. It comes with most of the essentials for web dev already included, including plug-ins for user-friendly features. Django is *fast*, in both development (utilizing MVT architecture) and processing, with high speed network transmission and content delivery. It follows DRY principles, which is extremely important in Python, and is very scalable, also thanks to its MVT architecture. With built-in automated encryption and a robust community of contributors, it's obvious why Django is so popular among Python developers.

5 companies that use Django:

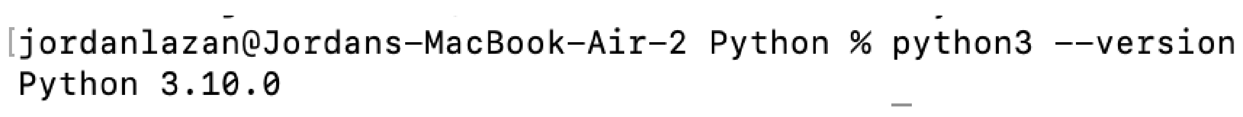
- **Instagram**: A social media photo and video sharing platform; Instagram uses Django because of the high traffic and ample content delivery involved, plus for error reporting (using a Django app called Sentry)
- **Dropbox**: A file hosting service with over 600 million users; Dropbox uses Djangoto ensure high performance at great scale
- **Pinterest**: A photo-sharing platform that allows users to "pin" liked pictures and discover new "pins"; Pinterest uses Django for its many user-friendly features, and to be able to scale as the site traffic and user engagement grows
- **Spotify**: An audio streaming platform with music, podcasts, audiobooks, and more; Spotify uses Django for its backend to build powerful libraries and use analytics to provide customized user interactions
- **YouTube**: A video-sharing platform; YouTube uses Django to implement new features and upgrades as quickly as possible, which is very important in such a content-heavy service

Scenarios

1. You need to develop a web application with multiple users
    a. In this case, I would probably use Django. Being a website with users, there will be a front-end and a back-end (which probably stores information, at the very least user information if not more data). Also, the project could scale as time goes on, which is a strength of Django thanks to its speed.
2. You need fast deployment and the ability to make changes as you proceed.
    a. Here I would definitely use Django. Django is very speedy when it comes to deployment and content, and allows developers to make changes easily since they don't have to worry about the framework as much as they might with other frameworks.
3. You need to build a very basic application, which doesn't require any database access or file operations.
    a. I would not use Django in this instance. Django comes with all its packages and plug-ins out of box, and I doubt I would need all this power for a simple application. It would be easier to either use a different, smaller-scale framework, or build the application on my own.
4. You want to build an application from scratch and want a lot of control over how it works.

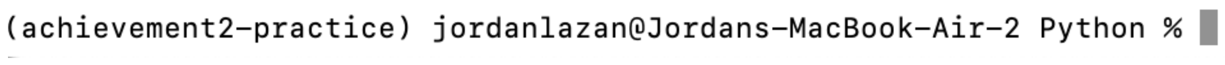
    a. Here, I again probably would not use Django. While in many cases a strength, Django must be used in a very particular manner, which can take away some control over how the application works. If I want control over everything, I probably don’t want to incorporate Django.
5. You’re about to start working on a big project and are afraid of getting stuck and needing additional support.
    a. Being a large project, Django sounds helpful. I can work on my back-end code, and front-end user experience, and allow Django to do all the middle work for me. Additionally, with such a large community of developers and contributors, I know that there will be plenty of support out there for Django if I do get stuck.

Step 5 - Check Python Version



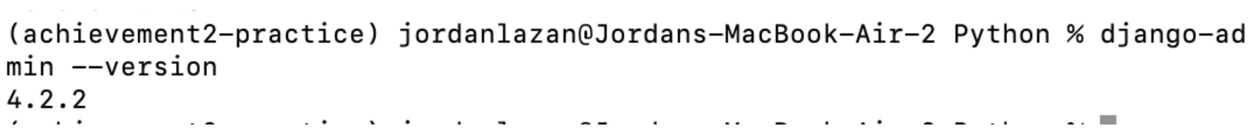
```
[jordanlazan@Jordans-MacBook-Air-2 Python % python3 --version
 Python 3.10.0
```

Step 7 - Create New Virtual Environment



```
(achievement2-practice) jordanlazan@Jordans-MacBook-Air-2 Python %
```

Step 8 - Check Django Version



```
(achievement2-practice) jordanlazan@Jordans-MacBook-Air-2 Python % django-ad
min --version
4.2.2
```