Day 1

Summary

- Get pipenv installed
- Clone your repo
  - (If you cloned the Hello-Django repo, delete the file requirements.txt!)
- Go to your repo root directory
- pipenv --three
- pipenv install
- pipenv shell
- django-admin startproject djorg .
- django-admin startapp notes
- ./manage.py runserver
- ./manage.py showmigrations
- ./manage.py migrate
- ./manage.py runserver
- Add model to notes/models.py
- Add 'notes' to INSTALLED_APPS in djorg/settings.py
- ./manage.py showmigrations
- ./manage.py makemigrations
- ./manage.py showmigrations
- ./manage.py migrate
- ./manage.py shell
  - from notes.models import Note
  - n = Note(title=”example”, content=”This is a test.”)
  - n.save()
  - exit()
- ./manage.py shell
  - from notes.models import Note
  - x = Note.objects.all()
  - x[0]
  - x[0].content
  - exit()
- pipenv install python-decouple
- Add config information to settings.py and .env

To Start a Django Project and App:

Check Python version and install or upgrade if less than 3.5.x
Check Pip version and install or upgrade if less than 10.x

- Mac/Linux - Option A: `sudo -H pip3 install --upgrade pip`
- Mac/Linux - Option B: `brew upgrade python`
- PC - `python -m pip install --upgrade pip`

Check Pipenv version and install or upgrade if less than 2018.x

Create a repo in github
- With README
- With Python .gitignore

Clone repo on to local machine

In the terminal, navigate to root folder of repo

Create pipenv virtual environment with `pipenv --three`
- The --three option tells it to use Python3
- This is similar to using npm/yarn

Verify that the Pipfile was created in the root of the repo

Activate pipenv with `pipenv shell`
- You should see the command line change to the name of your repo/folder followed by a dash and a random string
- We are using pipenv because it is newer and more robust. Uses a lockfile similar to npm/yarn. Easier to get into and out of shell
- To get back in, use `pipenv shell` from the root directory of the project

Once you are in the virtual environment, install django with `pipenv install django`
- We are using a virtual environment instead of installing globally because installing globally would be like using npm/yarn install globally and installing all the packages on everything

Add pipfile and pipfile.lock to the repo with `git add Pipfile*` and commit with `git commit -m "added pipfiles"

Start a project with `django-admin startproject [name_of_project] .`
- Replace [nameofproject] with the name of your project
- The . tells it to create the project in the current directory. Otherwise, it would create a project in a subdirectory called [name_of_project]. We don't need that because we want the repo folder to be the root

Verify that the [name_of_project] folder was created and has boilerplate files such as __init__.py
The project is what it was named above. A project is made up of a collection of apps. It can have many.

Create an app with `django-admin startapp [name_of_app]`
- For the first project, we are naming the app notes

- Name it differently as appropriate if you are following this to set up, but working on something else.

Verify that the [name_of_app] subdirectory has been created

Test by navigating to the project folder root/[name_of_project] and running `python manage.py runserver`
- This should launch the animated rocket default page
- Take note of the warning about unapplied migrations. We will fix that in a moment

Django makes it easier to make changes to databases. This is called migration(s)

Run `python manage.py showmigrations`. This will show a list of outstanding changes that need to occur

To take a closer look at what is being done, run `python manage.py sqlmigrate [package_name] [migration_id], for example, `python manage.py sqlmigrate admin 0001_initial`
- This will display a large number of sql commands that may not make sense if you are not yet familiar with sql
- This doesn't actually do anything. It just displays info
- These are all the data structures that your python code has created. Django turns this into sql tables, etc. for you. (If you've ever done this manually, you know how awesome that is :))

To actually run the migrations, use `python manage.py migrate`

Check them by showing migrations again - `python manage.py showmigrations`
- The list of migrations should show an `x` for each item

Run the server again and confirm that the migration warning is not present. There won't be a change to the actual page that renders

----
Adding data and doing setup for the notes app

In the `notes` folder, open `models.py`

Create a class called `notes` that inherits from `models.Model` - `class Note(models.Model):`

This gives our new class access to all of the built-in functionality in `models.Model`

Think about the data that we need for standard web notes functionality. We might want a title, body, some timestamps, etc. We can use the docs to find the types of things we can add:

https://docs.djangoproject.com/en/2.0/ref/models/

Add the following variables:

```
title = models.CharField(max_length=200)
content = models.TextField(blank=True)
created_at = models.DateTimeField(auto_now_add=True)
last_modified = models.DateTimeField(auto_now=True)
```
Add any additional fields you would like as well.

We also need something to serve as a unique identifier for each record. We’ll use something called a UUID for this: https://en.wikipedia.org/wiki/Universally_unique_identifier

Add a uuid to serve as a key for each record.
- First, import the library - `from uuid import uuid4`
- Second, add the field - `id = models.UUIDField(primary_key=True, default=uuid4, editable=False)
    - Primary key is how the database tracks records
    - Default calls a function to randomly generate a unique identifier
    - We make editable false because we never want to change the key
    - Put it at the top of the list of fields because it’s sort of like the index for the record

----
Next, we need to tell the project that the app exists. Open `settings.py` from the project folder.

Find the section for `INSTALLED_APPS` and add `’notes’`, or other apps as appropriate.

In the console, check for migrations again with `python manage.py showmigrations`. The notes app should show up in the list now, but it has no migrations.

Run `python manage.py makemigrations` to generate them. If you get an error that there are no changes to make, double-check that you have saved `models.py`.

Show migrations again to make sure they appear, then do the migration - `python manage.py migrate`

Manage.py has its own shell. Run `python manage.py shell` to bring up a Python repl.

- The input line should change to `>>>`

Import the notes class into the repl - `from notes.models import Note`

Create a new note with `n = Note(title="example", content="This is a test.")`

Check by the name of your variable to make sure worked - `n'
We can use a built in function from models to save this to the database - `n.save()`

Exit the terminal, then restart it - `exit()` then `python manage.py shell`

We have to import the `Note` class again, using the same command as before

There is another built in method that will retrieve all existing objects of a class - `Note.objects.all()`

Use this to save the data back into a variable named `b` and explore.

Take a look at that secret key in `settings.py`

We want to move this out of the settings file so that it doesn't get checked into source control. We'll move it to another file, which everyone on the project will need a copy of, but it won't be in the repo itself.

We're going to make use of a module called Python Decouple by installing it in the virtual environment = `pipenv install python-decouple`

Once it's installed, we can bring it into `settings.py` with `from decouple import config`

Pull up the docs for Python Decouple and take a look at the usage and rationale. There is an example for how to use it with Django. Follow that to remove the key from settings.

We should also move `DEBUG` to the config file. Because the file is a string, and `DEBUG` expects a bool, we need to cast it - `config('DEBUG', cast=bool)`

We do this not for security, but so that it can be changed as needed on a development machine, without modifying the source code

Don't forget to add it to `.env` as well.

Test to make sure it still works and debug as needed.

Before moving on, verify that `.env` is in `.gitignore` and commit.

Code to generate new secret keys from a Python REPL:

```
import random

''.join([random.SystemRandom().choice('abcdefghijklmnopqrstuvwxyz012345
6789!@#$%^&*(-_=+)') for i in range(50)])  # All one line!
```