Tango with Django (with Stijn)
→ https://docs.djangoproject.com

Flask: Microframework, good for no models and quick web stuffz. / http://flask.pocoo.org/

- Stijn used for APIs, you may not need models.

PIP: Python Package Index → use virtualenv to manage various Django installations and versions. / http://pypi.python.org/pypi/pip Equivalent of Ruby's RVM.

- Create virtual environment
  - `mkvirtualenv <name of project> --no-site-packages`
- Activate the virtual environment
  - `source env/bin/activate`
- Set up alias!
  - Edit .bash_profile
    - `alias workon='source env/bin/activate'`
  - Now to activate your project environment, type:
    - `workon <name of project>`
- Install Django & other infrastructure
  - `pip install django`

Django extensions: very useful! Stijn will come back to this later.

requirements.txt

- To get a list of all installed, type `pip freeze`.
- `pip freeze > requirements.txt` → Pipe your contents into requirements.txt
- `pip install -r requirements.txt` → Installs all app requirements / dependencies.

For larger projects:

- common.txt
- development.txt → Django extensions, debugging
- production.txt → gunicorn: Server that is only used in production.

Deployment: can be as hard or as easy as you want.

- BASICS: Do it in the very same way you install something locally.
- Use different servers:
  - Frontend → serve lots of requests at the same time and protect backend app. If frontend getting hammered, only send a couple requests to backedn. Ideal frontend server is **nginx**.
  - Backend → use gunicorn port of Unicorn in Ruby world (!!) instead of `python manage.py runserver`. Install with pip.
    - Uwgsi / Nginx module that replaces gunicorn.
- What about **Git**? If you have shell access on server, just SSH login to server, pull your repository. Create virtualenv there. Use this env to install requirements. Run server

with gunicorn.
    - Ways to make this easier: Fabric, like Capistrano. → Deployment tool, abstraction of this parent bullet.
    - `pip install fabric`
    - Fabric just consists of file with functions in it: RUN, which will run something on remote server specified. And LOCAL.
    - Docs: http://docs.fabfile.org/en/1.2.2/index.html
    - http://www.slideshare.net/andymccurdy/python-deployment-with-fabric [Slide 17]
    - `fabfile.py` in project directory.
    - `fab list` → list of available methods, which are the ones you've written.

Fixtures!
Often when you are writing an app, you will need test data. Most people use Django admin and enter information in there -- which serves as test data. But problem is when you delete your db or someone else installs your app, or if you update your models and re-generate your database.

- South: Migration tool / http://south.aeracode.org/
- `fixture_gen.py` → https://github.com/kzhu91/roundtable/blob/master/issues/fixture_gen.py
- Tool called Fixture generator.

Two ways to create fixtures:

1. *Standard but sucky*

    `python manage.py dumpdata` → dumps JSON.
    Then pump this into a file: `initial.json`.
    `python manage.py loaddata initial.json`

Annoyances: you don't have control over what's in your fixture or flexibility to change.
Hence, fixture generator!

2. Better way: use fixture generator → https://github.com/alex/django-fixture-generator

    ```
    pip install fixturegenerator
    python manage.py generatefixture
    ```

Uses a decorator -- function that changes function into another function.
If you've installed fixture-generator and added to `INSTALLED_APPS` in `settings.py`, issues.test_users. It will generate same JSON, but now, instead of having to edit the JSON, you can just very easily edit the `fixture_gen.py`.

Then you can just type `fab reset`. Now you have a fresh db with the initial data.

South: Database migrations
Tad different if you're using from existing app or from a fresh start.

New project, new app. Then install South (with pip).
`pip install south`

In `settings.py`, we add South to INSTALLED_APPS.

python manage.py schemamigration
--initial → pumps to file (?)
--auto → shows your migrations?

Other cool stuff

- Hate writing regular expressions when specifying Django urls? Try the **surlex** module.
- Deploying stuff? Learn about **NGINX**, **gunicorn**, **upstart** and **fabric**.
- Want some help debugging? Install **django-extensions**, which gives you two new management commands: `python manage.py runserver_plus` and `python manage.py shell_plus`. `runserver_plus` is just like `runserver`, but it will give you an interactive debugging console right in the browser whenever you make an error, so you can investigate what's going wrong. `shell_plus` is like the regular `shell` command, but it preloads all your models, so you don't have to import anything before you can start playing around with your models and your data.

An overview of some commands

```
# create a virtual environment
mkdir myproj; cd myproj
virtualenv env --no-site-packages
# activate it
source env/bin/activate
# (a shortcut, because you'll use this a lot -- add to your ~/.bashrc)
alias workon="source env/bin/activate"
# installing software with pip
pip install django; pip install south; pip install django-extensions;
# list all the software in your virtual environment
pip freeze
# save that software into a requirements file
pip freeze > requirements.txt
# install software from a requirements file
pip install -r requirements.txt
# generate fixtures from a populated database
```

```
python manage.py dumpdata
# generate fixtures with django-fixture-generator
python manage.py generate_fixture issues.test_users
# working with south
python manage.py schemamigration myapp --initial
python manage.py schemamigration myapp --auto
# a list of all the commands in your Fabric fabfile
fab --list
```