Larger Applications

Imagine a simple flask application structure that looks like this:

```
/yourapplication
    yourapplication.py
    /static
        style.css
    /templates
        layout.html
        index.html
        login.html
        ...
```

While this is fine for small applications, for larger applications it's a good idea to use a package instead of a module. The tutorial is structured to use the package pattern, see the example code.

Simple Packages

To convert that into a larger one, just create a new folder `yourapplication` inside the existing one and move everything below it. Then rename `yourapplication.py` to `__init__.py`. (Make sure to delete all `.pyc` files first, otherwise things would most likely break)

You should then end up with something like that:

```
/yourapplication
    /yourapplication
        __init__.py
        /static
            style.css
        /templates
            layout.html
            index.html
            login.html
            ...
```

But how do you run your application now? The naive `python yourapplication/__init__.py` will not work. Let's just say that Python does not want modules in packages to be the startup file. But that is not a big problem, just add a new file called `setup.py` next to the inner `yourapplication` folder with the following contents:

```
from setuptools import setup

setup(
    name='yourapplication',
    packages=['yourapplication'],
    include_package_data=True,
    install_requires=[
        'flask',
    ],
)
```

In order to run the application you need to export an environment variable that tells Flask where to find the application instance:

```
$ export FLASK_APP=yourapplication
```

If you are outside of the project directory make sure to provide the exact path to your application directory. Similarly you can turn on the development features like this:

```
$ export FLASK_ENV=development
```

In order to install and run the application you need to issue the following commands:

```
$ pip install -e .
$ flask run
```

What did we gain from this? Now we can restructure the application a bit into multiple modules. The only thing you have to remember is the following quick checklist:

1. the *Flask* application object creation has to be in the `__init__.py` file. That way each module can import it safely and the *__name__* variable will resolve to the correct package.
2. all the view functions (the ones with a **`route()`** decorator on top) have to be imported in the `__init__.py` file. Not the object itself, but the module it is in. Import the view module **after the application object is created**.

Here's an example `__init__.py`:

```
from flask import Flask
app = Flask(__name__)

import yourapplication.views
```

And this is what `views.py` would look like:

```
from yourapplication import app

@app.route('/')
def index():
    return 'Hello World!'
```

You should then end up with something like that:

```
/yourapplication
    setup.py
    /yourapplication
        __init__.py
        views.py
        /static
            style.css
        /templates
            layout.html
            index.html
            login.html
            ...
```

Circular Imports:

Every Python programmer hates them, and yet we just added some: circular imports (That's when two modules depend on each other. In this case `views.py` depends on `__init__.py`). Be advised that this is a bad idea in general but here it is actually fine. The reason for this is that we are not actually using the views in `__init__.py` and just ensuring the module is imported and we are doing that at the bottom of the file.

There are still some problems with that approach but if you want to use decorators there is no way around that. Check out the Becoming Big section for some inspiration how to deal with that.

Working with Blueprints

If you have larger applications it's recommended to divide them into smaller groups where each group is implemented with the help of a blueprint. For a gentle introduction into this topic refer to the Modular Applications with Blueprints chapter of the documentation.