# **Creating an app**

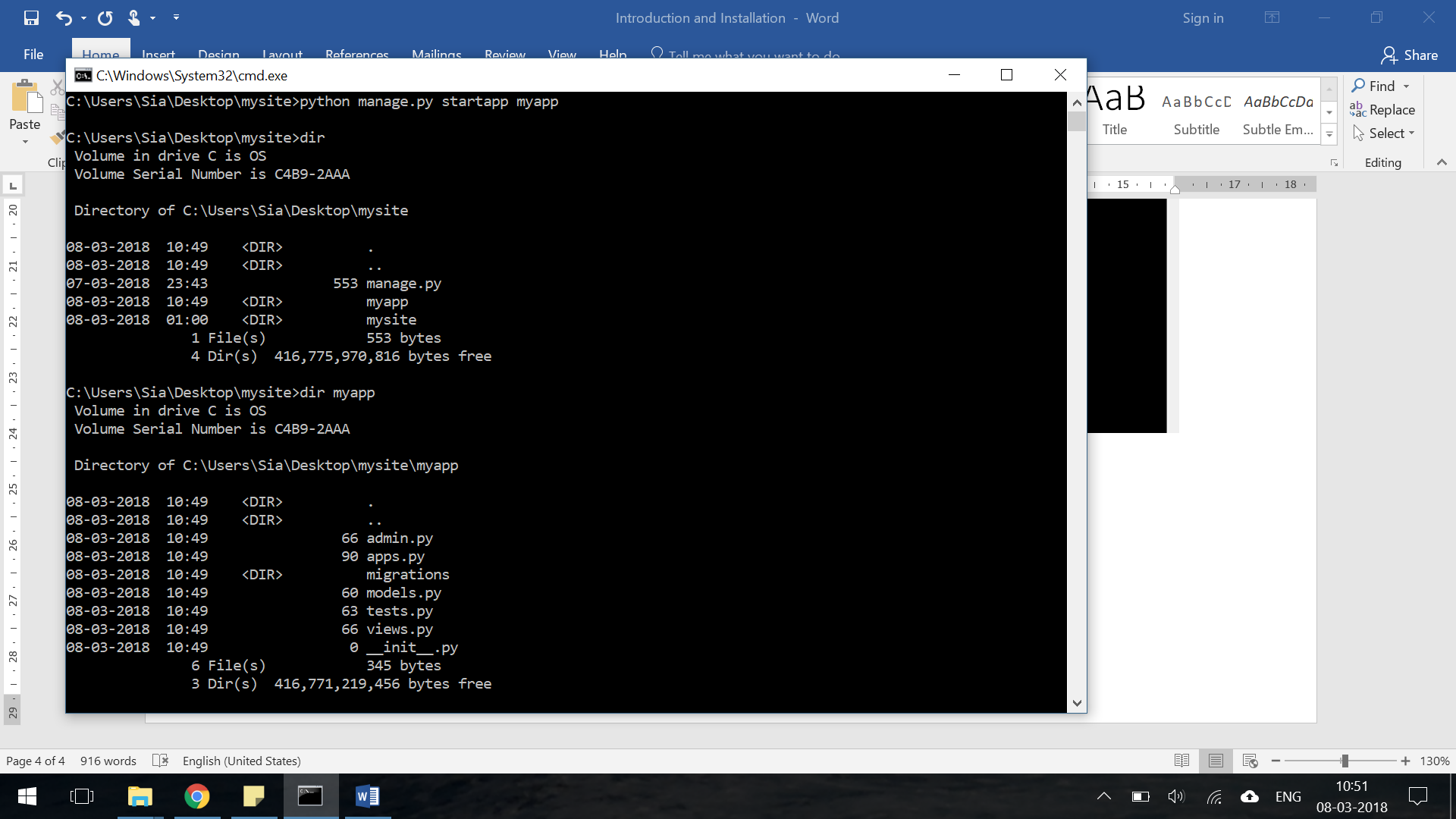
Now that your environment – a “project” – is set up, you’re set to start doing work.

Each application you write in Django consists of a Python package that follows a certain convention. Django comes with a utility that automatically generates the basic directory structure of an app, so you can focus on writing code rather than creating directories.

Django uses the concept of projects and apps to keep code clean and readable. A single Django project contains one or more apps within it that all work together to power a web application. This is why the command for a new Django project is startproject! For example, a real-world Django e-commerce site might have one app for user authentication, another app for payments, and a third app to power item listing details. Each focus on an isolated piece of functionality.

**Projects vs. apps**

What’s the difference between a project and an app? An app is a Web application that does something – e.g., a Weblog system, a database of public records or a simple poll app. A project is a collection of configuration and apps for a particular website. A project can contain multiple apps. An app can be in multiple projects.



Django has created a *myapp* directory with the following files:

├── myapp

│ ├── \_\_init\_\_.py

│ ├── admin.py

│ ├── apps.py

│ ├── migrations

│ │ └── \_\_init\_\_.py

│ ├── models.py

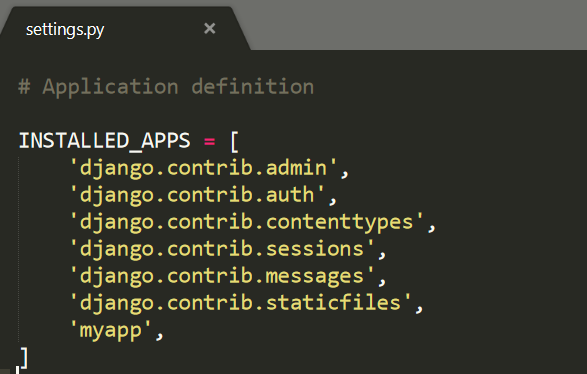
│ ├── tests.py

│ └── views.py

Let’s review what each new myapp app file does:

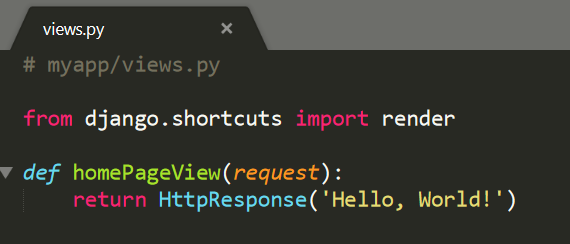
* admin.py is a configuration file for the built-in Django Admin app
* apps.py is a configuration file for the app itself
* migrations/ keeps track of any changes to our models.py file so our database and models.py stay in sync
* models.py is where we define our database models, which Django automatically translates into database tables
* tests.py is for our app-specific tests
* views.py is where we handle the request/response logic for our web app

Even though our new app exists within the Django project, Django doesn’t “know” about it until we explicitly add it. We do that in the file *mysite/settings.py*. We need to find *INSTALLED\_APPS* and add a line containing ‘myapp’, just above]. So, the final product should look like this:

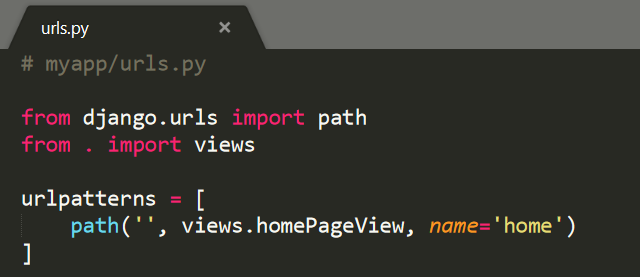


**Views and URLConfs**

In Django, Views determine what content is displayed on a given page while URLConfs determine where that content is going. When a user requests a specific page, like the homepage, the URLConf uses a regular expression to map that request to the appropriate view function which then returns the correct data. In other words, our view will output the text “Hello, World” while our url will ensure that when the user visits the homepage they are redirected to the correct view.



Basically, we’re saying whenever the view function homePageView is called, return the text “Hello, World!” More specifically, we’ve imported the built-in HttpResponse method so we can return a response object to the user. We’ve created a function called homePageView that accepts the request object and returns a response with the string Hello, World! Now we need to configure our URLs. Within the ‘myapp’ directory, create a new urls.py file.



On the top line we import path from Django to power our URL pattern and on the next line we import our views. The period used here from. import views means reference the current directory, which is our pages app containing both views.py and urls.py. Our URL pattern has three parts:

* a Python regular expression for the empty string ''
* specify the view which is called homePageView
* add an optional url name of 'home'

In other words, if the user requests the homepage, represented by the empty string '' then use the view called homePageView. The last step is to configure our project-level urls.py file too.

# 