



Task: Introduction to Django

www.hyperiondev.com



Introduction

Welcome to The Introduction to Django Task!

This task will introduce some foundational concepts in back-end Web Development with Django. Django is a high-level Python Web framework that encourages rapid development and clean, pragmatic design. Built by experienced developers, it takes care of much of the hassle of Web development, so you can focus on writing your app without needing to reinvent the wheel.



**CONNECT WITH
YOUR MENTOR**

You don't have to take our courses alone! This course has been designed to be taken with an online mentor who marks your submitted code by hand and supports you to achieve your career goals on a daily basis.

To access this mentor support, simply navigate to www.hyperiondev.com/support.

Why Django?

If this is your first venture into Web development with the Python programming language, you're probably wondering why you should learn Django when there are approximately 20 Web development frameworks available in Python. Well, for the most part, there are high-level and low-level frameworks, and Django is the highest level framework, making it easier to build better Web apps more quickly and with less code.

Setting up Django

If you don't already have Django installed, you can open your command prompt and use PIP. PIP stands for 'PIP Installs Packages'. A side fact here - PIP is actually a recursive acronym. (Who says that programmers are not funny?).

Navigate to the 'Scripts' folder in the directory that was created when you installed Python. E.g

```
$ cd C:\Users\Hyperion\AppData\Local\Programs\Python\Python36-32\Scripts
```

Type the following command in your Command Line and hit ENTER:

```
$ pip install Django
```

This should work if you have Python successfully installed, however, if it doesn't work, please feel free to contact your mentor for additional support. Alternatively, you can visit <https://djangoproject.com/download/> to discover the different ways in which you can get Django.

Once you have Django installed, you're ready to start your project. However, you'll first have to auto-generate some code that establishes a Django project – a collection of settings for an instance of Django, including database configuration, Django-specific options and application-specific settings.

If you are developing your app on a Windows machine, first add a path to the django-admin.exe to your environment variables. See the document, "FAQs" in this task folder for help to do this.

Now navigate to a directory where you would like to start your Web development and open a Command Window in that directory.

To start your project, type the following command in the command window you just opened:

```
$ django-admin startproject hyperion
```

The command will set up a new project called hyperion for you. You can name your project however you see fit but, you'll need to avoid naming projects after built-in Python or Django components. In particular, this means you should avoid using names like "django" (which will conflict with Django itself) or "test" (which conflicts with a built-in Python package).

If you have a look at the project you just generated, you will see that it has the following scaffolding:

hyperion/ The outer hyperion/ root directory is just a container for your project. Its name doesn't matter to Django; you can rename it to anything you like.

- ❖ **manage.py:** A command-line utility that lets you interact with this Django project in various ways.
- ❖ **hyperion/** The inner hyperion/ directory is the actual Python package for your project. Its name is the Python package name you'll need to use to import anything inside it (e.g. hyperion.urls).
 - **__init__.py** An empty file that tells Python that this directory should be considered a Python package.
 - **settings.py** Settings/configuration for this Django project. Perhaps one of the most important elements inside this directory is the "INSTALLED APPS". (We will get to this shortly). Another thing to note here is your SECRET_KEY - keep this a secret! This is used for session encryption. If someone gains access to this key, they will be able to decrypt the session and act as an admin on your site.
 - **urls.py** Controls what is served, based on url patterns (regular expressions).
 - **wsgi.py** A Python script used to help run your development server and deploy your project to a production environment.

Initial Project Test

Now that you have generated your project, you may want to run an initial test to make sure that it is in working order. Open a command window inside the directory where your

manage.py file is located and run the following command:

```
$ python manage.py runserver
```

This starts up the Django development server, which is a lightweight web server built entirely in Python. It is included in Django for testing purposes. This means that development is rapid and doesn't involve configuring and restarting the server as most changes are made, until you are ready for production. Go to <http://127.0.0.1:8000/> via your web browser and you will see a "Welcome to Django" page.

If you want the site to run on a different port, then you are welcome to change the runserver command, e.g. for port 8080 we run:

```
$ python manage.py runserver 8080
```

Note: To quit the server, you can simply enter CTRL-BREAK (CTRL + C) or exit the command window.

Creating your First Web App

A Django project is a collection of configurations and applications that have been put together to make a given website. A Django Application exists to perform a particular function. A project can contain multiple applications and an application can be in multiple projects. To create an app, we run the following command from within the same directory that our manage.py is in:

```
$ python manage.py startapp webapp
```

The command will generate an app called webapp for you. Remember not to choose an app name that may conflict with your project name or with a Python package.

Your project scaffolding should now be adjusted to the following:

```
❖ hyperion
  > manage.py
  > hyperion/
    ■ __init__.py
    ■ settings.py
    ■ urls.py
    ■ Wsgi.py
```

➤ **webapp/**

- **migrations**
- **__init__.py:** This file indicates to the Python interpreter that the directory is a Python package.
- **admin.py:** Stores the representation of a model in the admin interface.
- **apps.py:** Contains a registry of installed applications that stores configuration and provides introspection. It also maintains a list of available models.
- **models.py:** Describes database structures and metadata.
- **tests.py:** A Python script used to help run your development server and deploy your project to a production environment.
- **views.py:** Handles what the end-user "views" or interacts with.

Once you have generated a new application, you have to install it. This is easily done by adding the name of your application to the `INSTALLED_APPS` list in your `settings.py` file.

Once you've installed your application, your `INSTALLED_APPS` list in your `settings.py` file should look like this:

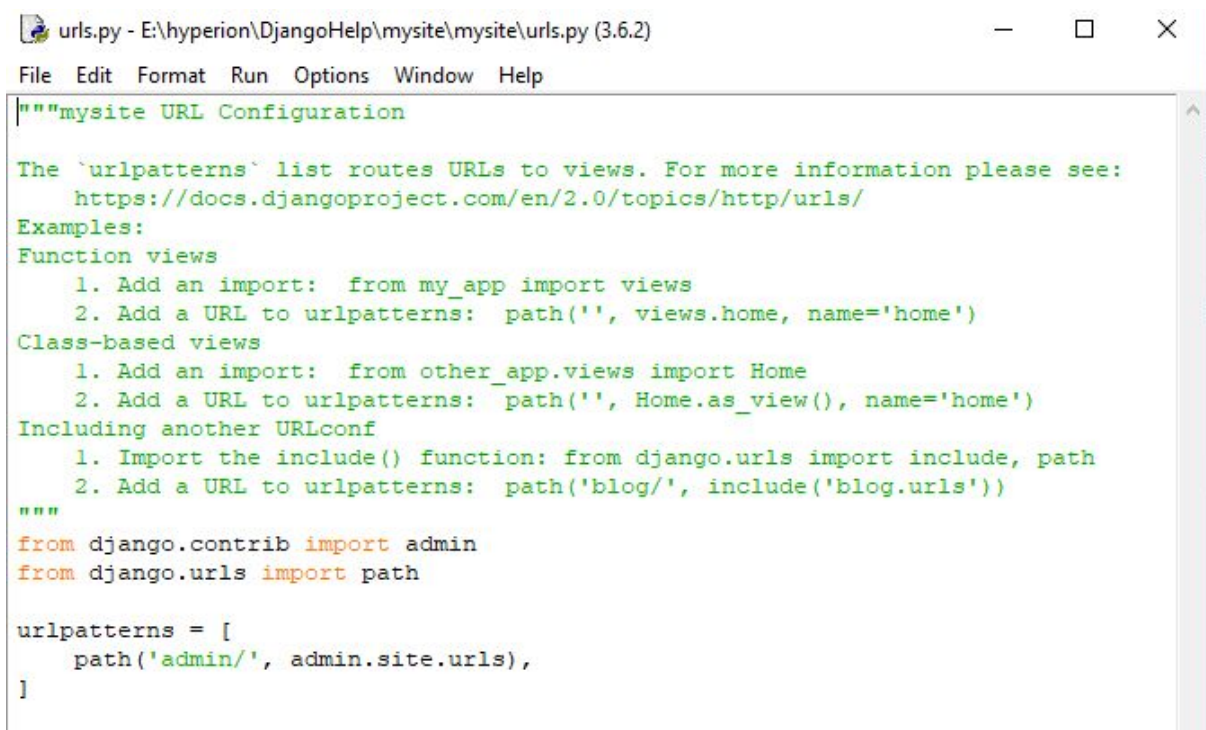
```
# Application definition

INSTALLED_APPS = [
    'webapp',
    'django.contrib.admin',
    'django.contrib.auth',
    'django.contrib.contenttypes',
    'django.contrib.sessions',
    'django.contrib.messages',
    'django.contrib.staticfiles',
]
```

Notice that 'webapp' is added at the beginning of the list. It doesn't really matter where in the list you append your app name, just as long as you follow the correct Python syntax!

After installing your application, the next task is to add a url that will point to your app. To do this, open `urls.py`. This is in the same directory as your `settings.py` file. Once you have

that opened, you will see the following:



```
urls.py - E:\hyperion\DjangoHelp\mysite\mysite\urls.py (3.6.2)
File Edit Format Run Options Window Help

"""mysite URL Configuration

The `urlpatterns` list routes URLs to views. For more information please see:
    https://docs.djangoproject.com/en/2.0/topics/http/urls/
Examples:
Function views
    1. Add an import:  from my_app import views
    2. Add a URL to urlpatterns:  path('', views.home, name='home')
Class-based views
    1. Add an import:  from other_app.views import Home
    2. Add a URL to urlpatterns:  path('', Home.as_view(), name='home')
Including another URLconf
    1. Import the include() function: from django.urls import include, path
    2. Add a URL to urlpatterns:  path('blog/', include('blog.urls'))
"""
from django.contrib import admin
from django.urls import path

urlpatterns = [
    path('admin/', admin.site.urls),
]
```

In the urlpatterns list, append the following item:

```
path('webapp/', include('webapp.urls'))
```

Your urls.py file should now have the following code:

```
from django.contrib import admin
from django.urls import path, include

urlpatterns = [
    path('admin/', admin.site.urls),
    path('webapp/', include('webapp.urls'))
]
```

Note that *include* was imported from `django.urls` at the top. Also, we've referenced a `urls` file which apparently is in `webapp`, however, if you have a look at the files inside your `webapp` directory, you will see that a `urls.py` file doesn't exist. This means that we have to create it. But, before we create it, remember the changes that we made to `settings.py` and `urls.py` in our configuration directory. These changes must be made every time we create a new app. To summarise:

1. Install the app in your `INSTALLED_APPS` list in `settings.py`
2. Append a url that will point to your application in `urls.py`

Moving along, create a new Python file called `urls.py` inside your `webapp` directory. Inside

it, copy and paste the following code:

```
from django.urls import path, include
from . import views

urlpatterns = [
    path('', views.index),
]
```

We have referenced a function called index from views.py. This function doesn't exist yet, so let's open views.py and define this function.

```
def index(request):
    return HttpResponse("<h2>Hello World<h2>")
```

At the top of views.py, you have to import HttpResponse from django.shortcuts, or else your function will not work:

```
from django.shortcuts import HttpResponse
```

Our web application now has a view. Start up your development server to see the output:

```
$ python manage.py runserver
```

Go to <http://127.0.0.1:8000/webapp/>. You should see the text "Hello World!" in h2 sized heading.

Rendering HTML

HTML code can be placed as a parameter of the HttpResponse function, however, most of the time our web page will have a little more than just the "Hello World" text, therefore, we have to figure a way in which we can load HTML files into our function. Edit the function in views.py:

```
def index(request):
    return render(request, 'webapp/index.html')
```

Inside the webapp directory, create a new folder called templates. Inside templates, create a new folder called webapp. Now, inside webapp, insert one of the HTML files that

you created in the previous tasks. All HTML templates for webapp will be stored here (in templates/webapp). Your webapp directory should now have the following scaffolding:

```
❖ webapp/
  ➤ migrations
  ➤ templates/
    ■ webapp/
      ● index.html
  ➤ __init__.py
  ➤ admin.py
  ➤ apps.py
  ➤ models.py
  ➤ tests.py
  ➤ views.py
```

Note: the home page of any website is generally named index, however, if you want to load any of the other HTML files you created make sure that you adjust your function accordingly.

Loading Static Files

If you have rendered your HTML files correctly and started up your development server, you will notice that your web pages excludes any stylesheets, images etc.

You will need to load these in by creating a new folder in your app directory called static. You then have to place the following code right at the beginning of your relevant HTML file:

```
{%load staticfiles%}
```

Also, you have to reference your items accordingly (for example an image tag would be):

```

```

The image.jpg is in another folder in static called assets.

Compulsory Task

Follow these steps:

- Create a new project called mySite.
- Start an application called personal. Our end-goal will be set up an “about you” dynamic website.
- Create a template for your app that will display some information about you.
- Render your template and map a URL to it.
- Render two additional HTML files that you have previously created and set up URLs for them.
- Install your app, and setup a url for it.
- Create a new batch file called **runserver.bat** in your manage.py directory. Write code that will automate the process of starting the development server.



**SHARE YOUR
THOUGHTS WITH US**

Hyperion strives to provide internationally-excellent course content that helps you achieve your learning outcomes. Think the content of this task, or this course as a whole, can be improved or think we've done a good job? [Click here](#) to share your thoughts anonymously.