

Lab 2 - Templates

In this exercise we will create a Django project with an app called **pages** that has two pages - a homepage and an about page. We will also use Django's class-based views and templates to build this small web application.

Open **Windows Command Line**, move into the directory called **django projects** and create a new Django project called **lab2**

```
> mkdir lab2
```

Move into the **lab2** directory

```
> cd lab2
```

Run the following command to create a virtual environment

```
> python -m venv env
```

To activate our virtual environment, type the following command:

```
> env\scripts\activate.bat
```

You should now see parentheses around the name of your current directory on your command line which indicates the virtual environment is activated:

```
(env) C:\Users\pmagee\django projects\lab2>
```

Install Django using the following command:

```
> python -m pip install django
```

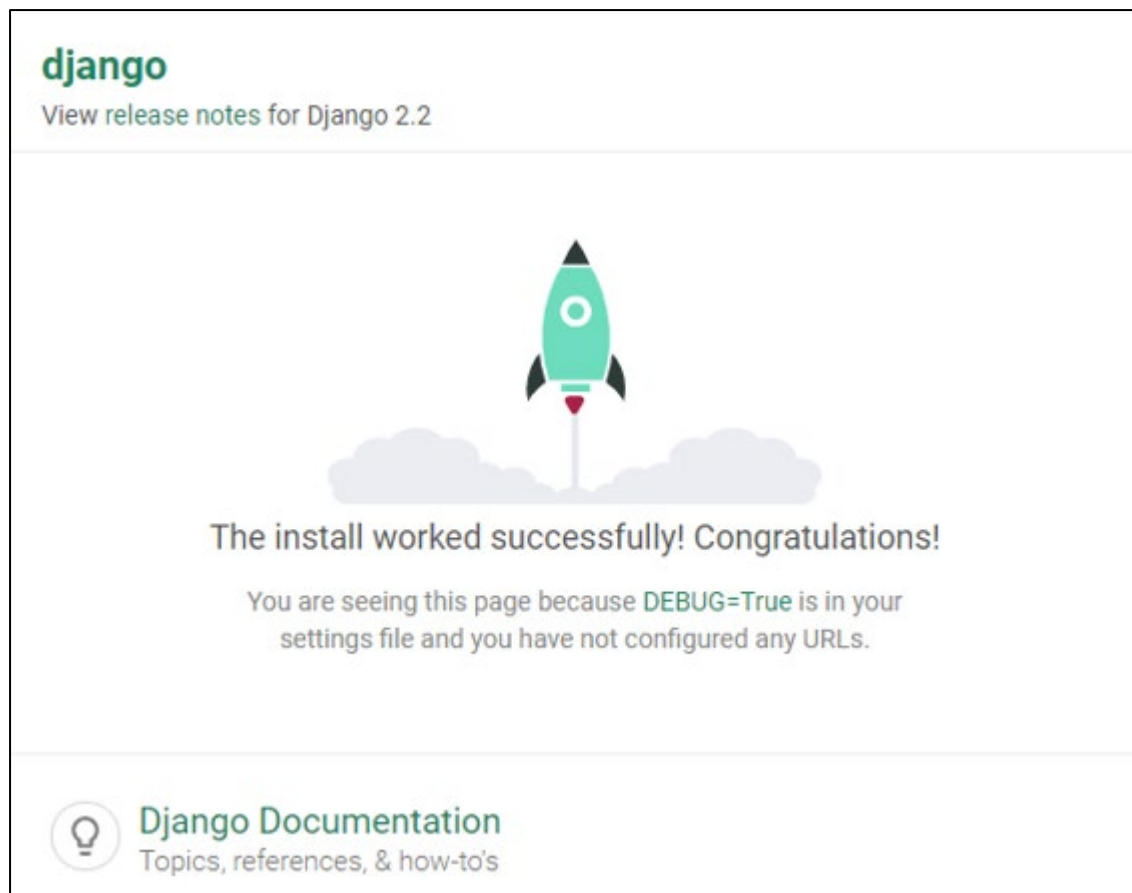
Create a new Django project called **pagesproj** with the following command. Don't forget the period (fullstop) at the end:

```
> django-admin startproject pagesproj .
```

To verify that the Django project works type the following command:

```
> python manage.py runserver 8080
```

If you visit <http://127.0.0.1:8080/> you should see the familiar Django welcome page



Create an app called **pages**. From the command line, quit the server with **Control+c**. Then use the startapp command as shown below:

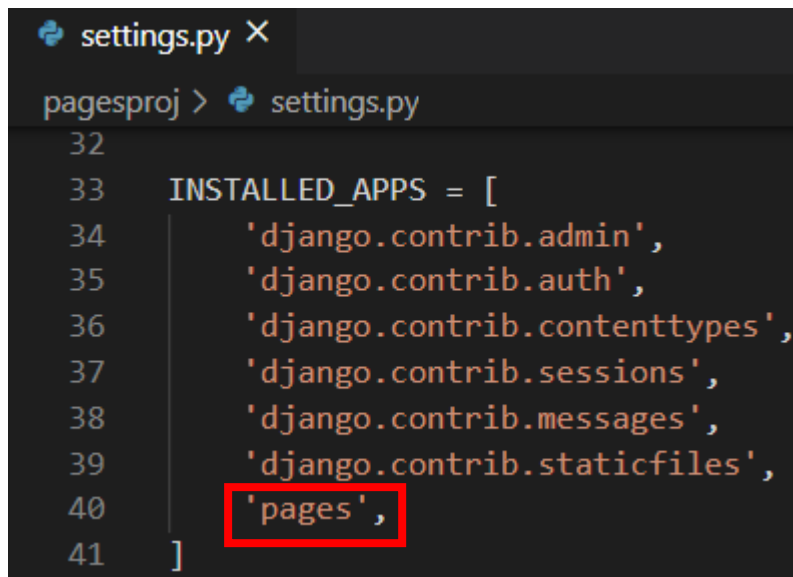
```
> python manage.py startapp pages
```

Settings.py

Even though our new app exists within the Django project, Django doesn't "know" about it until we explicitly add it.

1. Launch the VS Code IDE and open your project in this IDE.
2. In your text editor open the **settings.py** file and scroll down to **INSTALLED_APPS** where you will see six built-in Django apps already there.
3. Add the new **pages** app at the bottom.

Code



```
settings.py X
pagesproj > settings.py
32
33 INSTALLED_APPS = [
34     'django.contrib.admin',
35     'django.contrib.auth',
36     'django.contrib.contenttypes',
37     'django.contrib.sessions',
38     'django.contrib.messages',
39     'django.contrib.staticfiles',
40     'pages',
41 ]
```

Templates

To create HTML files in Django we use templates. A template is a text file that consists of static parts of the desired HTML output as well as some special syntax describing how dynamic content will be inserted. In Django we use templates so that individual HTML files can be served by a view to a web page specified by the URL.

In the previous lab where we created a Django project to display the message “Hello, World” on the page, we had the phrase hardcoded into a **views.py** file as a string. That technically works but doesn’t scale well! A better approach is to link a view to a template, thereby separating the information contained in each.

Before we create a template, we must decide where exactly we will place it in our project structure. We will create a single project-level templates directory and place all templates within there. By making a small change to our **settings.py** file we can tell Django to also look in this directory for templates.

First, quit the running server with the Control+BREAK or Control+c command. Then create a directory called **templates**.

> **mkdir templates**

In VS Code right-click on the **templates** folder and create a new file called **home.html** and add in the following code:

Code

```
templates > <> home.html > ...  
1 <!--templates/home.html-->  
2 <h1>Home page</h1>
```

Next, we need to update **settings.py** to tell Django the location of our new **templates** directory. This is a one-line change to the setting **'DIRS'** within **TEMPLATES**.

```
55 TEMPLATES = [  
56     {  
57         'BACKEND': 'django.template.backends.django.DjangoTemplates',  
58         'DIRS': [str(BASE_DIR.joinpath('templates'))],  
59         'APP_DIRS': True,  
60         'OPTIONS': {  
61             'context_processors': [  
62                 'django.template.context_processors.debug',  
63                 'django.template.context_processors.request',  
64                 'django.contrib.auth.context_processors.auth',  
65                 'django.contrib.messages.context_processors.messages',  
66             ],  
67         },  
68     },  
69 ]
```

The next step is to configure our **URL** and **view**.

Class-Based Views

Classes are a fundamental part of Python. In our view we will use the Django built-in **TemplateView** to display our template.

Delete the top line of code in **pages/views.py** and update this file using the code below:

Code

```
views.py ×  
pages > views.py  
1 # pages/views.py  
2 from django.views.generic import TemplateView  
3  
4 class HomePageView(TemplateView):  
5     template_name = 'home.html'
```

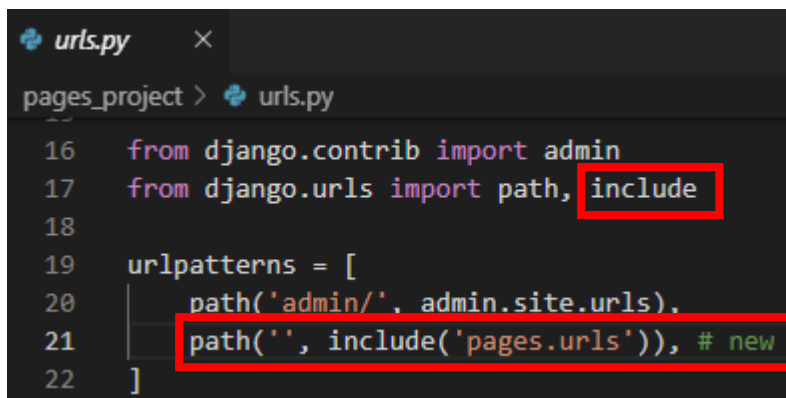
Note that we have capitalized our view since it is now a Python class. Classes, unlike functions, should always be capitalized. The **TemplateView** already contains all the logic needed to display our template, we just need to specify the template's name.

URLs

The last step is to update our URLConfs. We to make updates in two locations; 1)we update the **pages_project/urls.py** file to point at our **pages** app and 2) within the **pages** app we match the view to the route.

Let's start with the **pages_project/urls.py** file.

Code

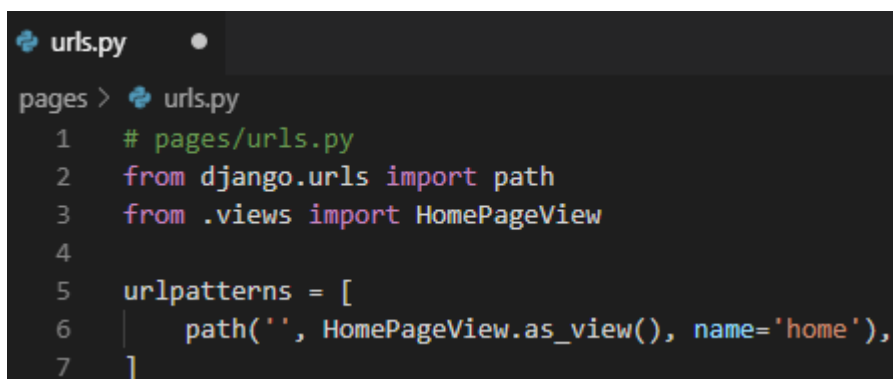


```
urls.py
pages_project > urls.py
16 from django.contrib import admin
17 from django.urls import path, include
18
19 urlpatterns = [
20     path('admin/', admin.site.urls),
21     path('', include('pages.urls')), # new
22 ]
```

We add include on the second line to point to the existing URL in the **pages** app.

Next in VS Code, create a **urls.py** file inside the **pages** app and add the following code.

Code

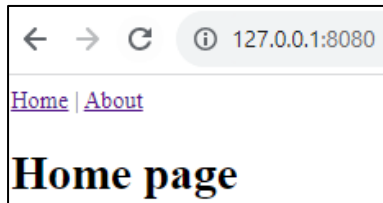


```
urls.py
pages > urls.py
1 # pages/urls.py
2 from django.urls import path
3 from .views import HomePageView
4
5 urlpatterns = [
6     path('', HomePageView.as_view(), name='home'),
7 ]
```

This pattern is almost identical to what we did in Lab1 with one major difference.

When using Class-Based Views, you always add **as_view()** at the end of the view name.

If you start up the web server with **python manage.py runserver** and navigate to <http://127.0.0.1:8080/> you can see the new homepage.



Add an About Page

The process for adding an about page is very similar to what we just did. We will create a new template file, a new view, and a new url route.

In VS code create a new template called **about.html** inside the templates folder.

Open this new file and add a simple headline to the body.

Code

```
<> about.html ×
templates > <> about.html > ...
1  <!-- templates/about.html -->
2  <h1>About page</h1>
```

In **pages/views.py** add the following code to create a new view for the page:

Code

```
views.py ×
pages > views.py
6
7
8  class AboutPageView(TemplateView):
9      template_name = 'about.html'
10
```

And then connect it to a URL **/about** by adding the following code to **pages/urls.py**

Code

```
urls.py
pages > urls.py
1  # pages/urls.py
2  from django.urls import path
3  from .views import HomePageView, AboutPageView
4
5  urlpatterns = [
6      path('about/', AboutPageView.as_view(), name='about'),
7      path('', HomePageView.as_view(), name='home'),
8  ]
```

Make sure that the server is still running and navigate to **http://127.0.0.1:8000/about** and you can see our new “About page”.



Extending Templates

One of the main advantages of using templates is their ability to be extended. If you think about most websites, there is content that is repeated on every page (header, footer, etc). In Django we can have one place for our header code that is inherited by all other templates.

We will create a **base.html** file which contains a header with links to our two pages. We could name this file anything but using **base.html** is a common convention.

In VS Code, right-click on the templates folder and create a new template page called **base.html**.

Django has a minimal templating language for adding links and basic logic in our templates. You can see the full list of built-in template tags in the official documentation.

Template tags take the form of **{% something %}** where the “something” is the template tag itself.

To add URL links in our project we can use the built-in **url** template tag which takes the URL pattern name as an argument. We can use the optional URL names that we created in the **urls.py** file. The **url** tag uses these names to automatically create links for us.

The URL route for our homepage is called **home** therefore to configure a link to it we would use the following: `{% url 'home' %}`.

In VS Code, enter the following code into the file **base.html** making sure to avoid any typos.

```
<> base.html •
templates > <> base.html > ...
1  <!-- templates/base.html -->
2  <header>
3  |   <a href="{% url 'home' %}">Home</a> | <a href="{% url 'about' %}">About</a>
4  </header>
5  {% block content %}
6  {% endblock content %}
```

At the bottom we have added a block tag called **content**. Blocks can be overwritten by child templates via inheritance. While it is optional to name our closing **endblock** we have added it in as it helps with readability, especially in larger template files.

Update the files **home.html** and **about.html** to extend the **base.html** template. That means we can reuse the same code from one template in another template. The Django templating language comes with an **extends** method that we can use for this.

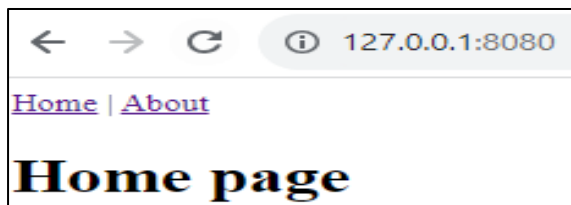
Code

```
<> home.html •
templates > <> home.html > ...
1  <!-- templates/home.html -->
2  {% extends 'base.html' %}
3
4  {% block content %}
5  <h1>Homepage</h1>
6  {% endblock content %}
```


Code

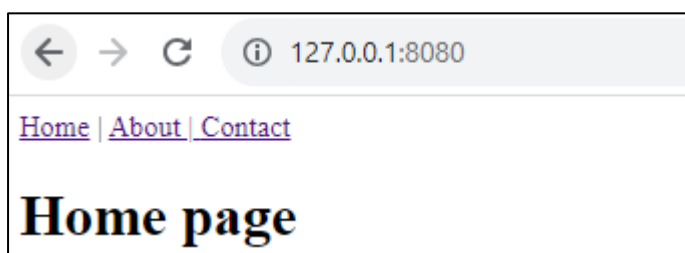
```
<> about.html ×
templates > <> about.html > ...
1  <!-- templates/about.html -->
2  {% extends 'base.html' %}
3
4  {% block content %}
5  <h1>About page</h1>
6  {% endblock content %}
```

If you open the web pages again at <http://127.0.0.1:8080/> and <http://127.0.0.1:8080/about> you will see the header is now included in both locations.



As you will see the steps for creating the home and about pages are very similar. Try to create a new html file called **contact.html** using the steps provided for creating the other pages.

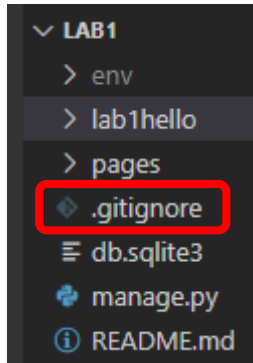
When you run the server, you should then see a “Contact Us” link on the home page which when clicked brings us to **contact.html**.





Exclude virtual environment from Git

A .gitignore file is a text file that tells Git which files or folders to ignore in a project. We don't want the virtual environment files to be part of our version control. Create a file called .gitignore in VS Code at the location shown below:



Add the following to the .gitignore file:

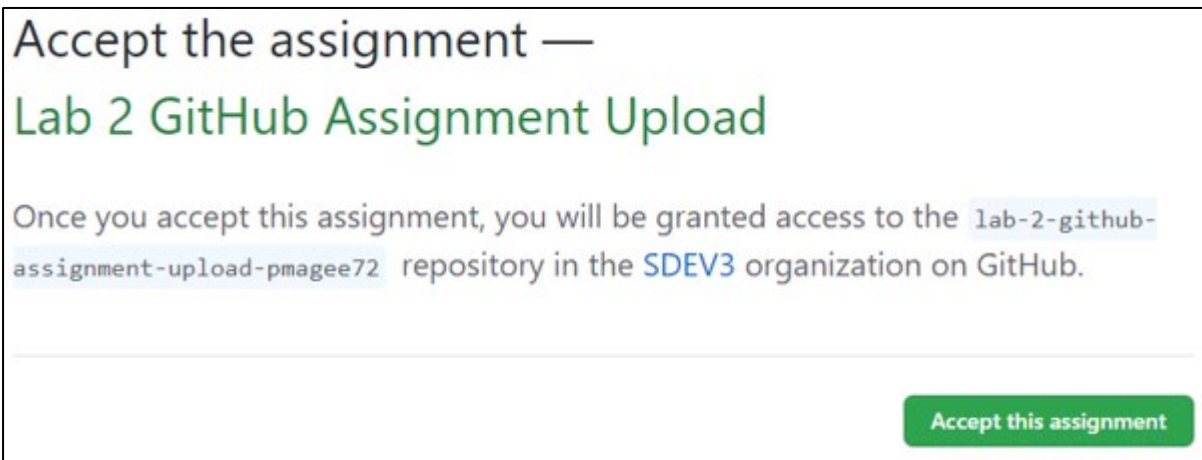
```
# Environments
env/
```

Upload of Project to GitHub Classroom

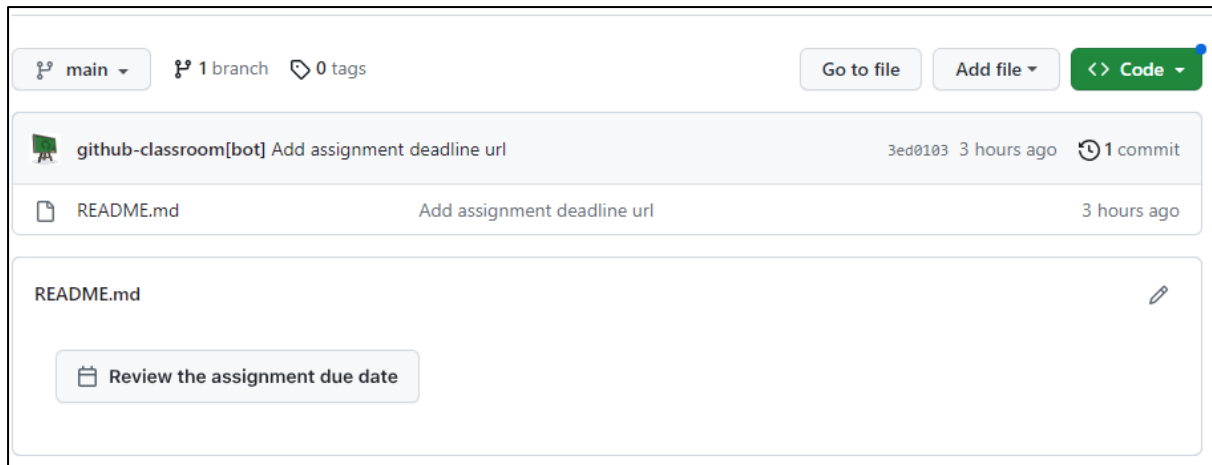
Log into GitHub before you proceed. Go to Moodle and click on the link similar to the one shown below:

Lab 2 Upload GitHub Classroom - Due Wed 27th Sep 6pm

Next you should see a screen like the following asking you to accept the assignment.



Once you have accepted the assignment, refresh the screen and you are presented with a link to your repository for lab 2 as shown below:

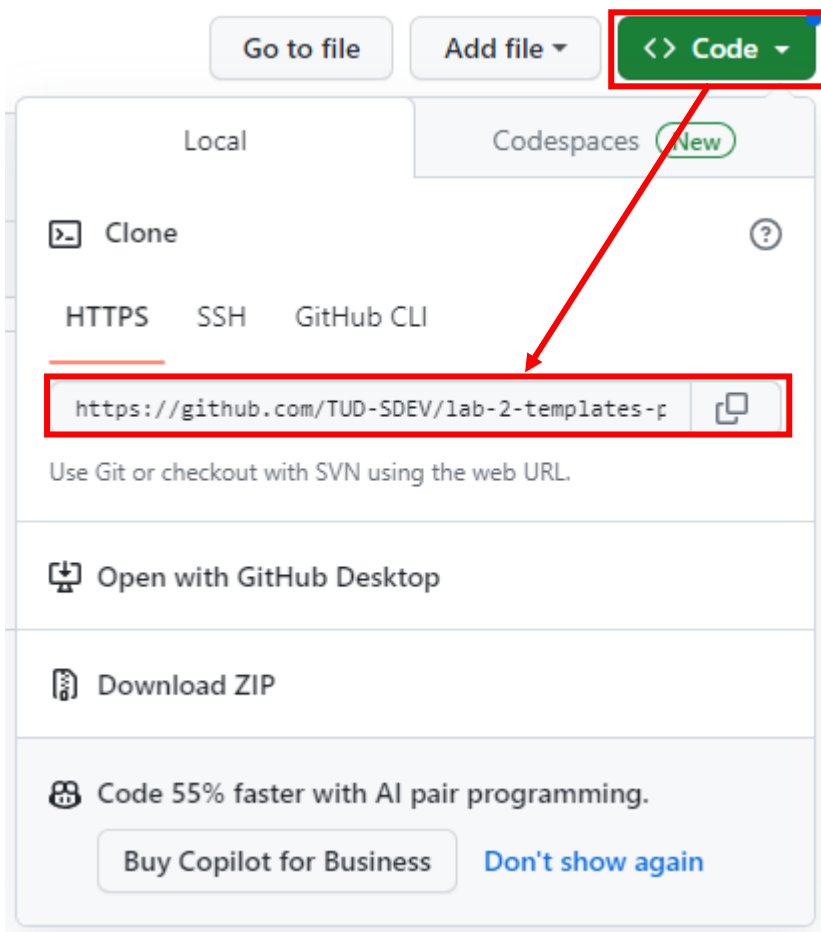


In Windows Command line make sure that you are in the **django** projects folder (not the **lab2** folder)

Use the following command to clone this repo to your local computer:

git clone <repo name>

You can get the repo details by clicking on the Code button as shown below:



After the clone command is finished you have a new folder called **lab-2-templates-
<username>** inside the **django** projects folder

In Windows Explorer copy the **lab2** folder into this **lab-2-hello-world-<username>** folder

Go to Windows Command Line and move into the **lab-2-hello-world-<username>** folder

Save a snapshot of the current project state with the following command:

git add -A



Commit the changes along with a suitable message:

git commit -m "initial commit"

Update the remote repository with the local commits:

git push -u origin main

Go to your GitHub page and refresh the page to see your local code now hosted online. Edit the **README** file in GitHub to include your name and id number.

 lab2	initial commit	now
 README.md	Add assignment deadline url	yesterday

To sync this file with your local project, use the following command:

git pull

The updated README file should now be part of your local project.

To deactivate the virtual environment type, deactivate as shown below:

deactivate

To exit out of Windows Command Line type exit:

exit