

Lab 1

In this exercise we will build a Django project that will display the message “Hello, World” on the homepage.

Step 1 – Using Windows Command Line Terminal for Windows

- a) In Windows Command Line create a folder called **djangoprojects**, change into that directory and create another directory called **lab1** and change into that directory using the following four commands:

> **mkdir djangoprojects**

> **cd djangoprojects**

> **mkdir lab1**

> **cd lab1**

- b) Python itself comes with venv for managing environments which we will use for our projects. To set up our virtual environment, type the following command:

> **python -m venv env**

To activate our virtual environment, type the following command:

> **env\scripts\activate.bat**

When the environment is activated the name of this environment (env) is displayed in the terminal inside parenthesis as shown below:

```
(env) C:\Users\pmagee\djangoprojects\lab1>
```

Step 2 –Install Django

- a) To install Django type the following command:

> **python -m pip install django**

- b) To see the installation, type the following command:

> **pip freeze**

Here you will see the version of Django that has been installed along with some other files

- c) To create a new Django project, type the following command and do not forget the full stop at the end of the command:

> **django-admin startproject lab1hello .**

- d) Type the following command to run the project:

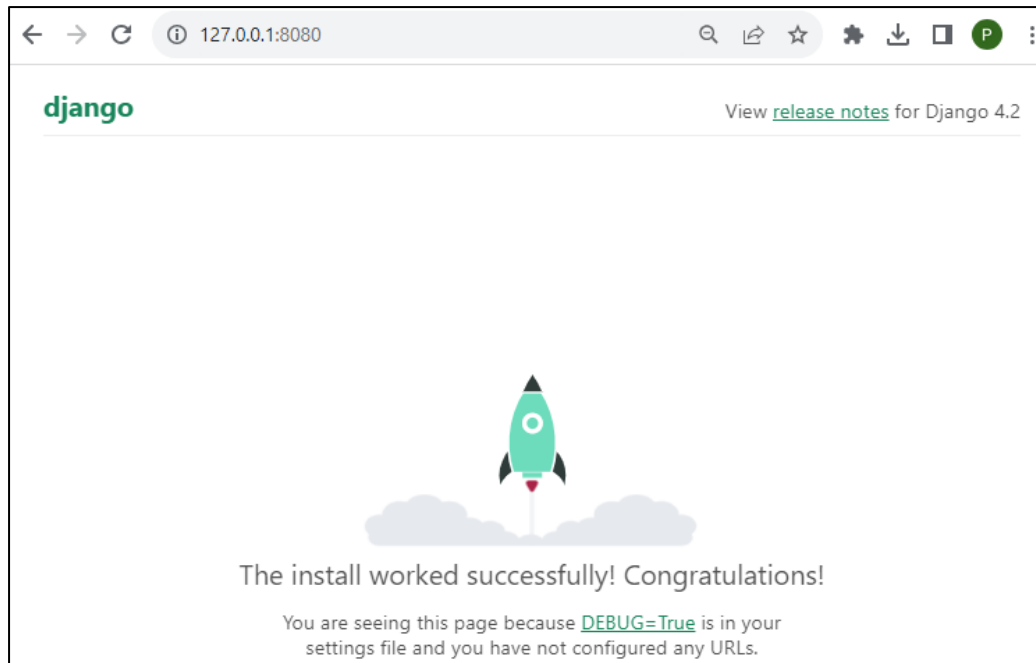
> **python manage.py runserver 8080**

When this command is executed, we see the following output indicating that the web server is running:

```
$ python manage.py runserver
Watching for file changes with StatReloader
```

- e) Open a browser and type the following url: <http://localhost:8080>.

- f) The following screen is the default web page displayed indicating that the project is running successfully:



Create an app

Django uses the concept of projects and apps to keep the code organised. A single Django project contains one or more apps within it that all work together to form a web application. Therefore, the command for a new Django project is **startproject**. For example, a real-world Django e-commerce site for selling books might have one app for user authentication, another app for payments, and a third app to deal with book listing details. Each app deals with an isolated piece of functionality.

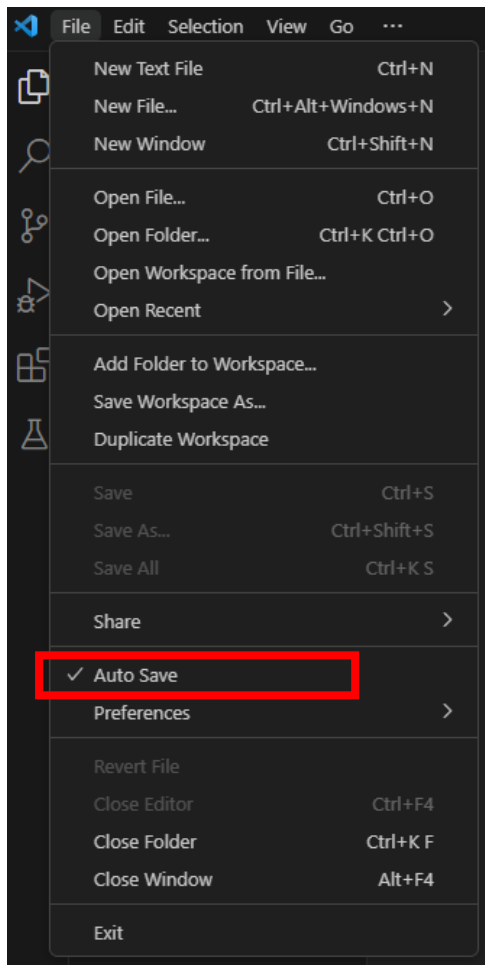
We will create our first app called **pages**. From the command line, quit the server with Control+BREAK or Control+c. Then use the startapp command as shown below:

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

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

Launch the VS Code IDE and open your project in this IDE by clicking on **File>Open Folder**.

Before you enter any code, you should make sure that the autosave option in VS Code. Is checked as shown below:



In VS Code, open the **settings.py** file and scroll down to `INSTALLED_APPS` where you will see six built-in Django apps already there.

1. Add the new **pages** app at the bottom by typing the code shown below.

```
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  ]
```

Don't forget the comma at the end.

We should always add our own local apps at the bottom because Django will read our `INSTALLED_APPS` in top down order. Therefore, the internal admin app is loaded

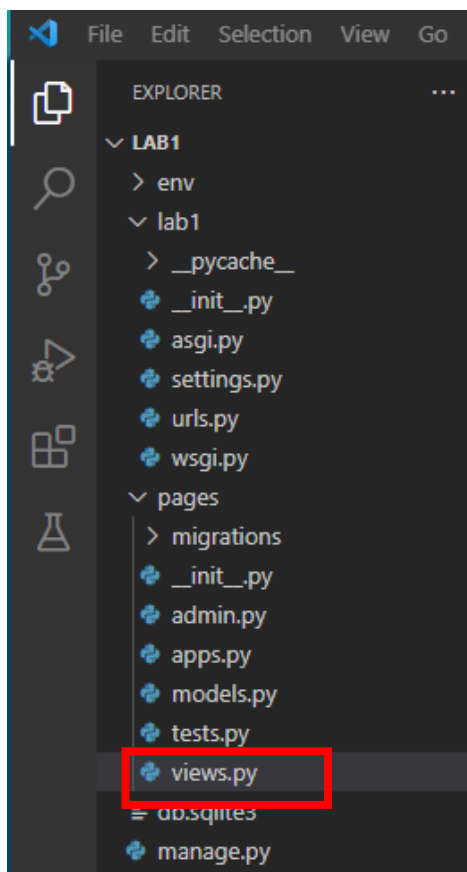
first, then auth, and so on. We want the core Django apps to be available since it is quite likely our own apps will rely on their functionality.

Views and URLConfs

In Django, views determine what content is displayed on a given page while URLConfs determine where that content is going. A URLconf is like a table of contents for your Django-powered Web site. Basically, it's a mapping between URLs and the view functions that should be called for those URLs. It's how you tell Django, "For this URL, call this code.

When a user requests a specific page, like the homepage, the URLConf maps 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.

In VS Code, open the file **views.py** which is sitting inside the **pages** folder as shown below:



Delete the following line of code at the top of the file.

```
pages > views.py
1  from django.shortcuts import render
```

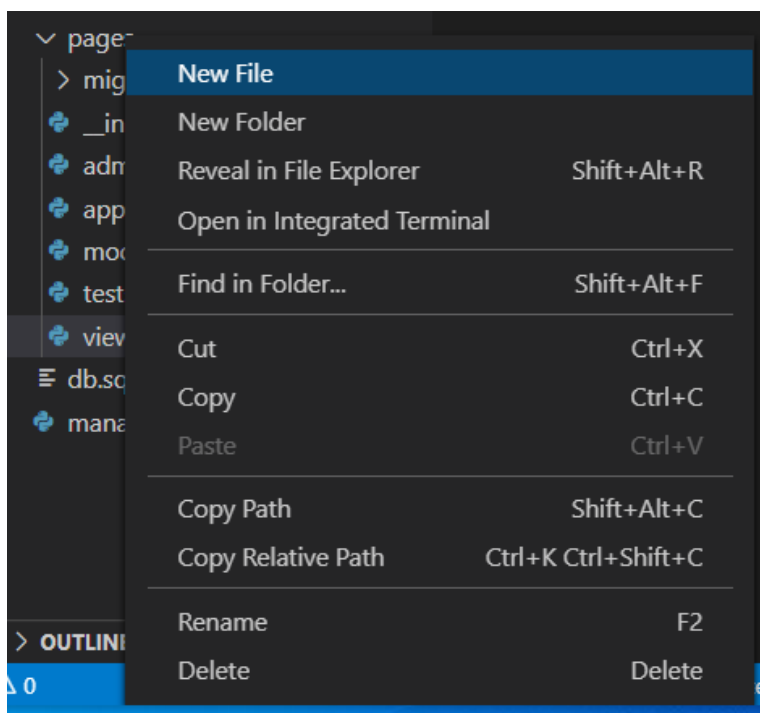
Code

Add the following code to this file

```
pages > views.py
1  from django.http import HttpResponse
2
3  # Create your views here.
4  def homePageView(request):
5      return HttpResponse('Hello World!')
```


In this snippet of code whenever the view function `homePageView` is called, we want to return the text “Hello, World!”. To make that happen we have imported the built-in `HttpResponse` method so we can return a response object to the user. We created a function called `homePageView` that accepts the request object and returns a response with the string Hello, World!.

Next, we need to configure our urls. Within the **pages** app, create a new `urls.py` file by right clicking on the pages folder and selecting the option New File.



Name this file `urls.py` and add in the following code:

Code:

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

On the top line we import `path` from Django to power our URLpattern and on the next line we import our views. By using the period `.views` we reference the current directory, which is our pages app containing both `views.py` and `urls.py`.

Our URLpattern 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 update our `lab1hello/urls.py` file. It is common to have multiple apps within a single Django project, like pages here, and they each need their own dedicated URL path.

```
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')),
22  ]
```

1

2

1. We have imported **include** on the second line next to path
2. We have also created a new URLpattern for our pages app

Now whenever a user visits the homepage at / they will first be routed to the pages app and then to the homePageView view.

To confirm everything works as expected, save the project in VS Code and restart our Django server using the following command:

> **python manage.py runserver 8080**

If you refresh the browser for <http://127.0.0.1:8080/> it now displays the text “Hello, world!”



To stop the server hit Ctrl+BREAK or Ctrl+c

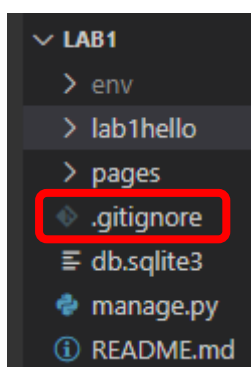
Now that we have our first Django project created, we will push our code to a GitHub repository.

Git Hub Account Set Up

See slides on Moodle

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 do not 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:



Copy and paste the following into the .gitignore file:

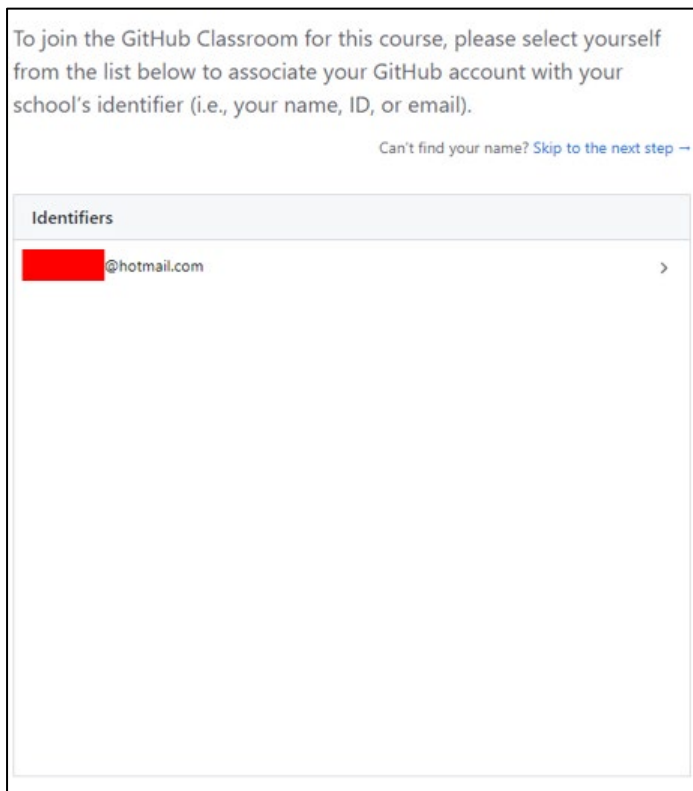
```
# Environments
env/
```

Upload of Project to GitHub Classroom

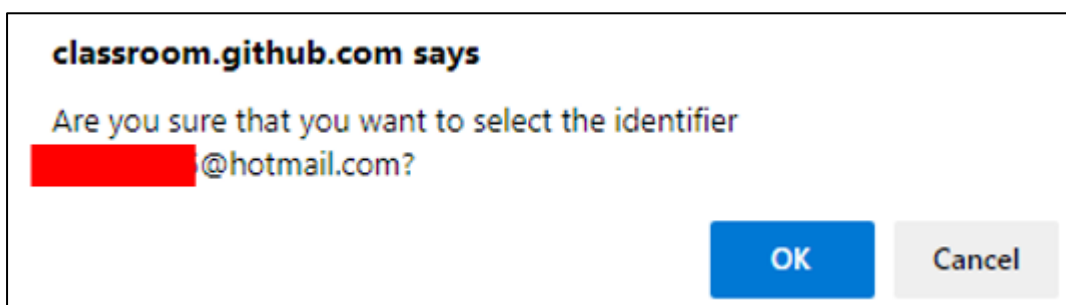
Log into GitHub before you proceed. Go to Moodle and click on the link similar to the one shown below:

Lab 1 Upload GitHub Classroom - Due Mon 25th Sep 6pm

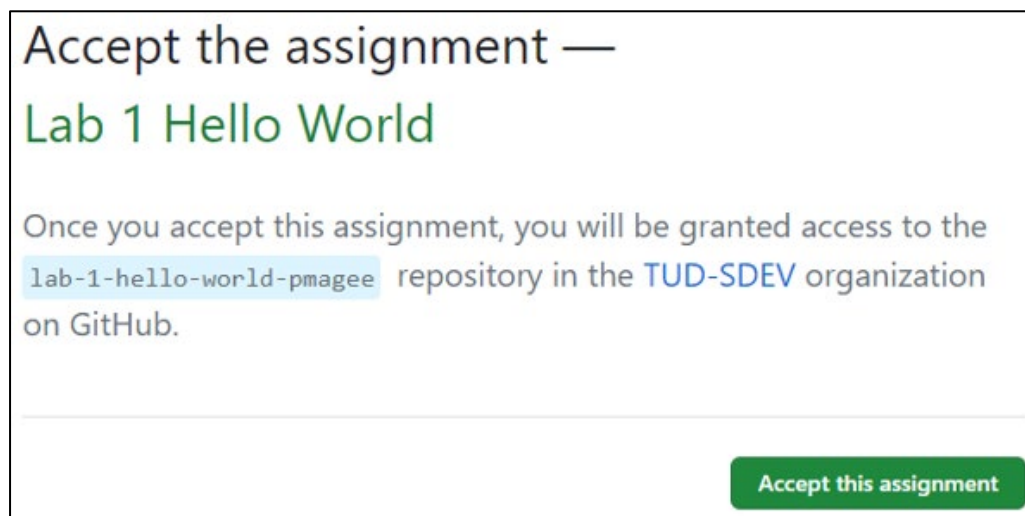
You will then be presented with a screen like the one shown below asking you to join the SDEV3 classroom. Find your email from the list shown on the screen:



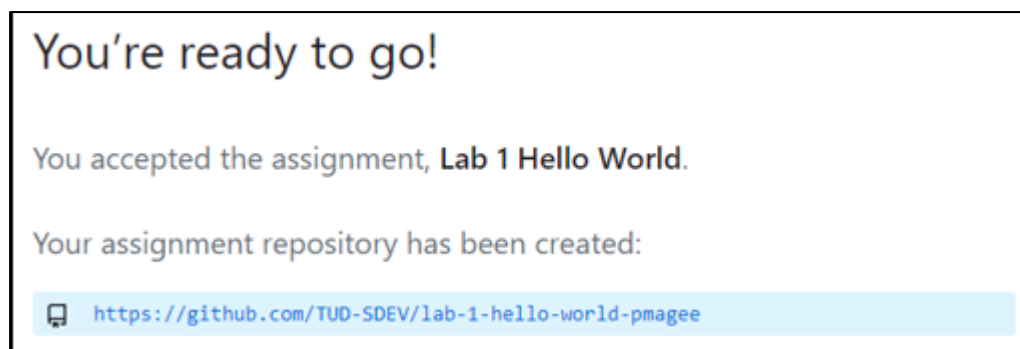
When you select your email address, you are asked to confirm it by clicking the OK button:



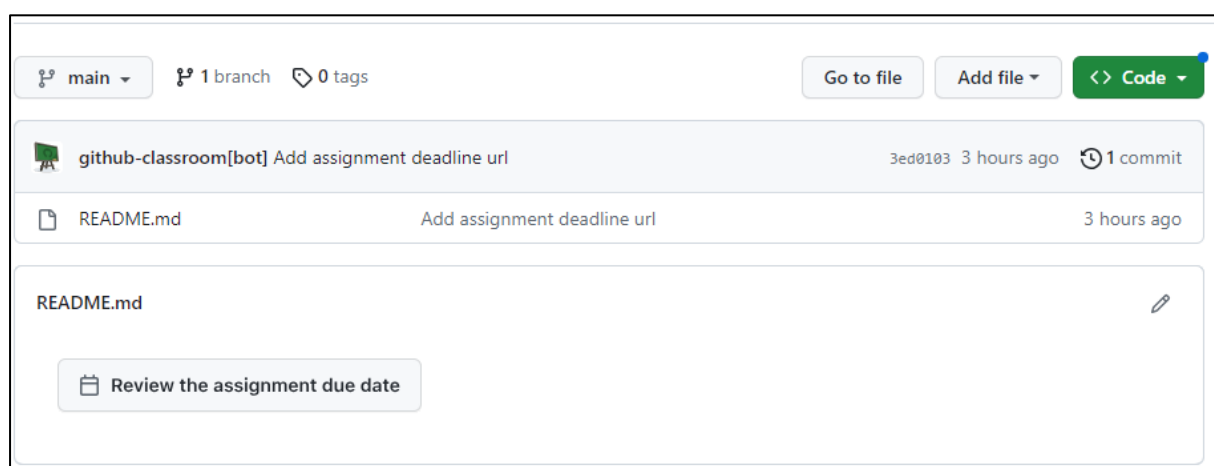
Next you should see a screen like the following asking you to accept the assignment.



Once you have accepted the assignment, refresh the screen and you are presented with a link to your repository for lab 1 as shown below:



When you click on the repository link, you are taken to the repository like the following:

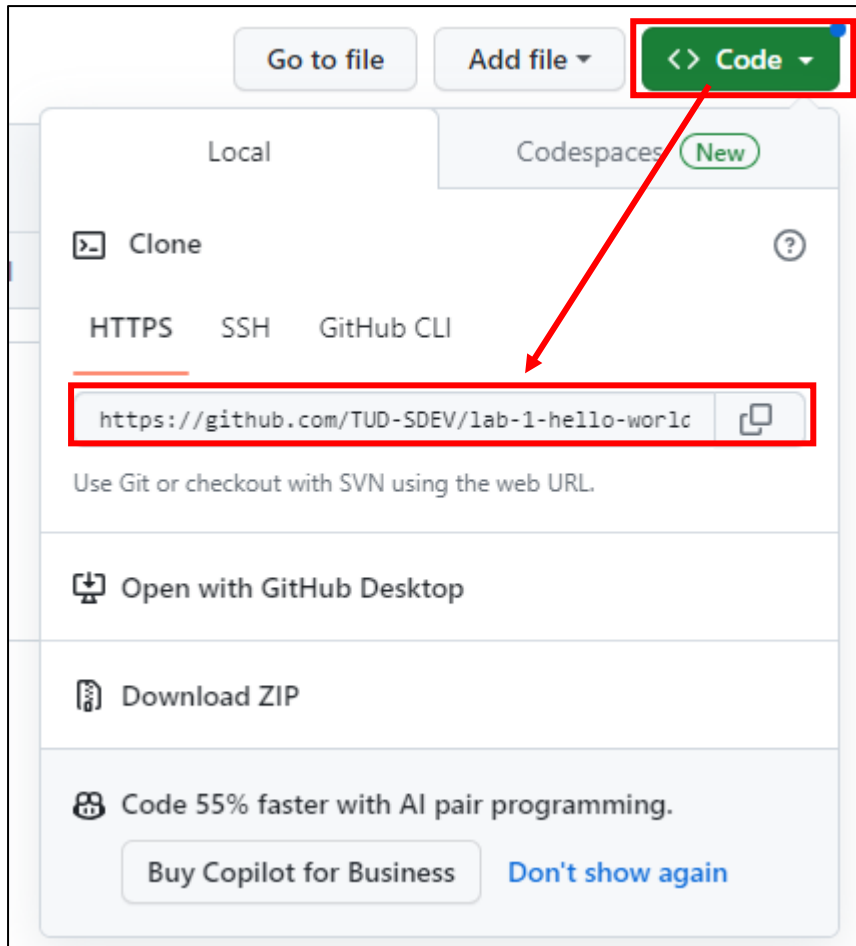


In Windows Command line make sure that you are in the **django**projects folder (not the **lab1** folder)

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-1-hello-world-
<username>** inside the **django**projects folder

In Windows Explorer copy the **lab1** folder into this **lab-1-hello-world-
<username>** folder

Go to Windows Command Line and move into the **lab-1-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.

 lab1	Uploading lab1	3 hours ago
 README.md	Add assignment deadline url	4 hours ago

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