## What is Django?

Django is a Python framework that makes it easier to create web sites using Python.

Django takes care of the difficult stuff so that you can concentrate on building your web applications.

Django emphasizes reusability of components, also referred to as DRY (Don't Repeat Yourself), and comes with ready-to-use features like login system, database connection and CRUD operations (Create Read Update Delete).

## How does Django Work?

Django follows the MVT design pattern (Model View Template).

* Model - The data you want to present, usually data from a database.
* View - A request handler that returns the relevant template and content - based on the request from the user.
* Template - A text file (like an HTML file) containing the layout of the web page, with logic on how to display the data.



## Model

The model provides data from the database.

In Django, the data is delivered as an Object Relational Mapping (ORM), which is a technique designed to make it easier to work with databases.

The most common way to extract data from a database is SQL. One problem with SQL is that you have to have a pretty good understanding of the database structure to be able to work with it.

Django, with ORM, makes it easier to communicate with the database, without having to write complex SQL statements.

The models are usually located in a file called models.py.

## View

A view is a function or method that takes http requests as arguments, imports the relevant model(s), and finds out what data to send to the template, and returns the final result.

The views are usually located in a file called views.py.



## Template

A template is a file where you describe how the result should be represented.

Templates are often .html files, with HTML code describing the layout of a web page, but it can also be in other file formats to present other results, but we will concentrate on .html files.

Django uses standard HTML to describe the layout, but uses Django tags to add logic:

<h1>My Homepage</h1>

<p>My name is {{ firstname }}.</p>

The templates of an application is located in a folder named templates.

What we have to do?

When you have installed Django and created your first Django web application, and the browser requests the URL, this is basically what happens:

1. Django receives the URL, checks the urls.py file, and calls the view that matches the URL.
2. The view, located in views.py, checks for relevant models.
3. The models are imported from the models.py file.
4. The view then sends the data to a specified template in the template folder.
5. The template contains HTML and Django tags, and with the data it returns finished HTML content back to the browser.

## Django Requires Python

To check if your system has Python installed, run this command in the command prompt: python --version

If Python is installed, you will get a result with the version number, like this

Python 3.9.2

If you find that you do not have Python installed on your computer, then you can download it for free from the following website: <https://www.python.org/>

# Django - Create Virtual Environment

## Virtual Environment

It is suggested to have a dedicated virtual environment for each Django project, and one way to manage a virtual environment is [venv](https://docs.python.org/3/tutorial/venv.html), which is included in Python.

The name of the virtual environment is your choice, in this tutorial we will call it myworld.

Type the following in the command prompt, remember to navigate to where you want to create your project:



Visual studio terminal:

In terminal:

python -m venv myworld

May take a little time but a pop-up will show up asking about whether you want to switch to virtual environment

Start a new terminal window

This will set up a virtual environment, and create a folder named "myworld" with subfolders and files, like this:



Then you have to activate the environment, by typing this command:



In Visual Studio terminal:

Activating the virtual enviornment

myworld\Scripts\activate.bat

Once the environment is activated, you will see this result in the command prompt:



## Install Django

Now, that we have created a virtual environment, we are ready to install Django.

Note: Remember to install Django while you are in the virtual environment!

Django is installed using pip, with this command:



In visual studio code:

Go to your folder

Ctrl+shift+x

Type python: install python extension

In Visual Studio terminal

Pip install Django

Which will give a result that looks like this (at least on my Windows machine):





Now you have installed Django in your new project, running in a virtual environment!

## Check Django Version

You can check if Django is installed by asking for its version number like this:

(myworld) C:\Users\*Your Name*>django-admin --version If Django is installed, you will get a result with the version number:

4.1.2

## First Project

Once you have come up with a suitable name for your Django project, like mine: my\_tennis\_club, navigate to where in the file system you want to store the code (in the virtual environment), I will navigate to the myworld folder, and run this command in the command prompt:

django-admin startproject my\_tennis\_club Django creates a my\_tennis\_club folder on my computer, with this content:



## Run the Django Project

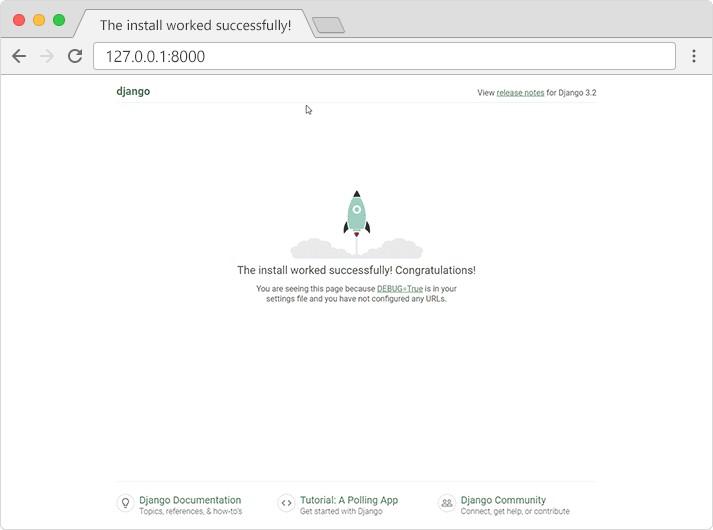
Now that you have a Django project, you can run it, and see what it looks like in a browser. Navigate to the /my\_tennis\_club folder and execute this command in the command prompt:

py manage.py runserver

Which will produce this result:



Open a new browser window and type 127.0.0.1:8000 in the address bar. The result:



# Django Create App

## What is an App?

An app is a web application that has a specific meaning in your project, like a home page, a contact form, or a members database.

In this tutorial we will create an app that allows us to list and register members in a database.

But first, let's just create a simple Django app that displays "Hello World!".



## Create App

I will name my app members.

Start by navigating to the selected location where you want to store the app, in my case the

my\_tennis\_club folder, and run the command below.

If the server is still running, and you are not able to write commands, press [CTRL] [BREAK], or [CTRL] [C] to stop the server and you should be back in the virtual environment.

py manage.py startapp members

Django creates a folder named members in my project, with this content:



# Django Views

## Views

Django views are Python functions that takes http requests and returns http response, like HTML documents.

A web page that uses Django is full of views with different tasks and missions. Views are usually put in a file called views.py located on your app's folder.

There is a views.py in your members folder that looks like this:

### my\_tennis\_club/members/views.py:

from django.shortcuts import render



Find it and open it, and replace the content with this:

### my\_tennis\_club/members/views.py:

from django.shortcuts import render from django.http import HttpResponse

def members(request):



# Django URLs

## URLs

Create a file named urls.py in the same folder as the views.py file, and type this code in it:

### my\_tennis\_club/members/urls.py:

from django.urls import path from . import views

urlpatterns = [

path('members/', views.members, name='members'),

]



The urls.py file you just created is specific for the members application. We have to do some routing in the root directory my\_tennis\_club as well. This may seem complicated, but for now, just follow the instructions below.

There is a file called urls.py on the my\_tennis\_club folder, open that file and add the include module in the import statement, and also add a path() function in the urlpatterns[] list, with arguments that will route users that comes in via 127.0.0.1:8000/.

Then your file will look like this:

### my\_tennis\_club/my\_tennis\_club/urls.py:

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

urlpatterns = [

path('', include('members.urls')),

path('admin/', admin.site.urls),

]



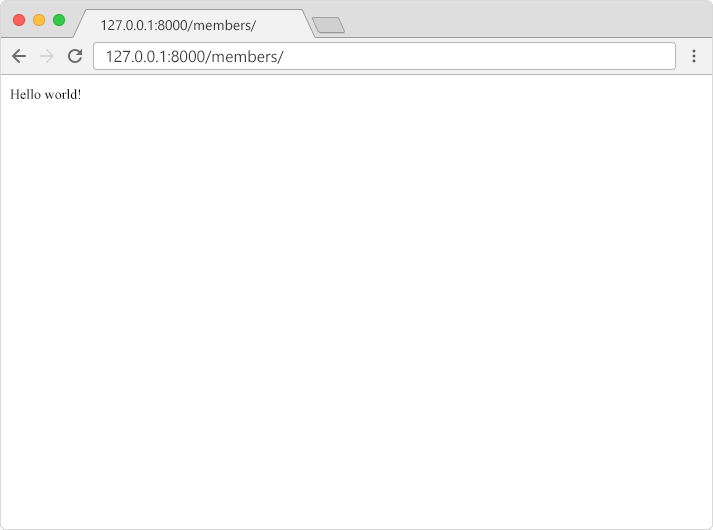




If the server is not running, navigate to the /my\_tennis\_club folder and execute this command in the command prompt:



In the browser window, type 127.0.0.1:8000/members/ in the address bar.



# Django Templates

## Templates

In the [Django Intro](https://www.w3schools.com/django/django_intro.php) page, we learned that the result should be in HTML, and it should be created in a template, so let's do that.

Create a templates folder inside the members folder, and create a HTML file named

### myfirst.html.

The file structure should be like this:





Open the HTML file and insert the following:

### my\_tennis\_club/members/templates/myfirst.html:

<!DOCTYPE html>

<html>

<body>

<h1>Hello World!</h1>

<p>Welcome to my first Django project!</p>

</body>





## Modify the View

Open the views.py file and replace the members view with this:

### my\_tennis\_club/members/views.py:

from django.http import HttpResponse from django.template import loader

def members(request):

template = loader.get\_template('myfirst.html')





## Change Settings

To be able to work with more complicated stuff than "Hello World!", We have to tell Django that a new app is created.

This is done in the settings.py file in the my\_tennis\_club folder.

Look up the INSTALLED\_APPS[] list and add the members app like this:

### my\_tennis\_club/my\_tennis\_club/settings.py:

INSTALLED\_APPS = [

'django.contrib.admin', 'django.contrib.auth', 'django.contrib.contenttypes', 'django.contrib.sessions', 'django.contrib.messages', 'django.contrib.staticfiles', 'members'





Then run this command:

py manage.py migrate Which will produce this output:

Operations to perform:

Apply all migrations: admin, auth, contenttypes, sessions Running migrations:

Applying contenttypes.0001\_initial... OK Applying auth.0001\_initial... OK Applying admin.0001\_initial... OK

Applying admin.0002\_logentry\_remove\_auto\_add... OK Applying admin.0003\_logentry\_add\_action\_flag\_choices... OK Applying contenttypes.0002\_remove\_content\_type\_name... OK Applying auth.0002\_alter\_permission\_name\_max\_length... OK Applying auth.0003\_alter\_user\_email\_max\_length... OK Applying auth.0004\_alter\_user\_username\_opts... OK

Applying auth.0005\_alter\_user\_last\_login\_null... OK Applying auth.0006\_require\_contenttypes\_0002... OK

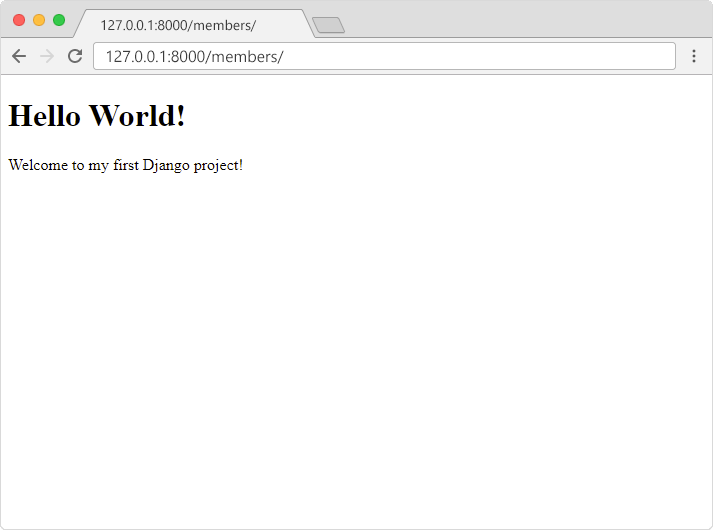
Applying auth.0007\_alter\_validators\_add\_error\_messages... OK Applying auth.0008\_alter\_user\_username\_max\_length... OK Applying auth.0009\_alter\_user\_last\_name\_max\_length... OK



Start the server by navigating to the /my\_tennis\_club folder and execute this command:

py manage.py runserver In the browser window, type 127.0.0.1:8000/members/ in the address bar.

The result should look like this:



# Django Models

A Django model is a table in your database.



## Django Models

Up until now in this tutorial, output has been static data from Python or HTML templates.

Now we will see how Django allows us to work with data, without having to change or upload files in the prosess.

In Django, data is created in objects, called Models, and is actually tables in a database.



## Create Table (Model)

To create a model, navigate to the models.py file in the /members/ folder.

Open it, and add a Member table by creating a Member class, and describe the table fields in it:

### my\_tennis\_club/members/models.py:

from django.db import models class Member(models.Model):

firstname = models.CharField(max\_length=255)



The first field, firstname, is a Text field, and will contain the first name of the members. The second field, lastname, is also a Text field, with the member's last name.

Both firstname and lastname is set up to have a maximum of 255 characters.





## Migrate

Now when we have described a Model in the models.py file, we must run a command to actually create the table in the database.

Navigate to the /my\_tennis\_club/ folder and run this command:

py manage.py makemigrations members Which will result in this output:



Django creates a file describing the changes and stores the file in the /migrations/ folder:

### my\_tennis\_club/members/migrations/0001\_initial.py:

# Generated by Django 4.1.2 on 2022-10-27 11:14 from django.db import migrations, models

class Migration(migrations.Migration): initial = True

dependencies = [

]

operations = [

migrations.CreateModel( name='Member', fields=[

('id', models.BigAutoField(auto\_created=True, primary\_key=True, serialize=False, verbose\_name='ID')),

('firstname', models.CharField(max\_length=255)), ('lastname', models.CharField(max\_length=255)),

],

),

]



Note that Django inserts an id field for your tables, which is an auto increment number (first record gets the value 1, the second record 2 etc.), this is the default behavior of Django, you can override it by describing your own id field.

The table is not created yet, you will have to run one more command, then Django will create and execute an SQL statement, based on the content of the new file in the

/migrations/ folder.

Run the migrate command:

py manage.py migrate Which will result in this output:



Now you have a Member table in you database!



## View SQL

As a side-note: you can view the SQL statement that were executed from the migration above. All you have to do is to run this command, with the migration number:

py manage.py sqlmigrate members 0001 Which will result in this output:



# Django Insert Data

## Add Records

The Members table created in the [previous chapter](https://www.w3schools.com/django/django_models.php) is empty.

We will use the Python interpreter (Python shell) to add some members to it. To open a Python shell, type this command:

py manage.py shell Now we are in the shell, the result should be something like this:



At the bottom, after the three >>> write the following:



Hit [enter] and write this to look at the empty Member table:



This should give you an empty QuerySet object, like this:



A QuerySet is a collection of data from a database.

Read more about QuerySets in the [Django QuerySet](https://www.w3schools.com/django/django_queryset.php) chapter. Add a record to the table, by executing these two lines:



Execute this command to see if the Member table got a member:



Hopefully, the result will look like this:





## Add Multiple Records

You can add multiple records by making a list of Member objects, and execute .save() on each entry:



Now there are 6 members in the Member table:



# Django Update Data

## Update Records

To update records that are already in the database, we first have to get the record we want to update:



x will now represent the member at index 4, which is "Stale Refsnes", but to make sure, let us see if that is correct:

>>> x.firstname This should give you this result:

'Stale' Now we can change the values of this record:



Execute this command to see if the Member table got updated:

>>> Member.objects.all().values() Hopefully, the result will look like this:





# Django Delete Data

## Delete Records

To delete a record in a table, start by getting the record you want to delete:



x will now represent the member at index 5, which is "Jane Doe", but to make sure, let us see if that is correct:

>>> x.firstname This should give you this result:

'Jane' Now we can delete the record:

>>> x.delete() The result will be:

(1, {'members.Member': 1})



Which tells us how many items were deleted, and from which Model.

If we look at the Member Model, we can see that 'Jane Doe' is removed from the Model:



# Django Update Model

## Add Fields in the Model

To add a field to a table after it is created, open the models.py file, and make your changes:

### my\_tennis\_club/members/models.py:

from django.db import models class Member(models.Model):

firstname = models.CharField(max\_length=255) lastname = models.CharField(max\_length=255) phone = models.IntegerField()

joined\_date = models.DateField()



As you can see, we want to add phone and joined\_date to our Member Model.

This is a change in the Model's structure, and therefor we have to make a migration to tell Django that it has to update the database:



Which, in my case, will result in a prompt, because I try to add fields that are not allowed to be null, to a table that already contains records.

As you can see, Django asks if I want to provide the fields with a specific value, or if I want to stop the migration and fix it in the model:



I will select option 2, and open the models.py file again and allow NULL values for the two new fields:



from django.db import models class Member(models.Model):

firstname = models.CharField(max\_length=255) lastname = models.CharField(max\_length=255) phone = models.IntegerField(null=True) joined\_date = models.DateField(null=True)



And make the migration once again:

py manage.py makemigrations members Which will result in this:



Run the migrate command:

py manage.py migrate Which will result in this output:





## Insert Data

We can insert data to the two new fields with the same approach as we did in the [Update](https://www.w3schools.com/django/django_update_data.php) [Data chapter](https://www.w3schools.com/django/django_update_data.php):

First we enter the Python Shell:



Now we are in the shell, the result should be something like this:



At the bottom, after the three >>> write the following (and hit [enter] for each line):



This will insert a phone number and a date in the Member Model, at least for the first record, the four remaining records will for now be left empty. We will deal with them later in the tutorial.

Execute this command to see if the Member table got updated:



The result should look like this:





**PostgreSQL**

The psycopg2 package is a driver that is necessary for PostgreSQL to work in Python.

**Installation**

pip install psycopg2-binary

**Output**

Collecting psycopg2-binary

Downloading psycopg2\_binary-2.9.9-cp312-cp312-win\_amd64.whl.metadata (4.6 kB)

Downloading psycopg2\_binary-2.9.9-cp312-cp312-win\_amd64.whl (1.2 MB)

---------------------------------------- 1.2/1.2 MB 3.4 MB/s eta 0:00:00

Installing collected packages: psycopg2-binary

Successfully installed psycopg2-binary-2.9.9

Download

<https://www.postgresql.org/ftp/pgadmin/pgadmin4/v8.5/windows/>

Download the file and install it.

Complete initial setup either using (SQL Shell)psql or pgAdmin (right-click on server and click Register→ in connections hostname default is localhost)

You can write a query:

In PgAdmin in servers→ postgreSQL 15/16 (select it connection established) Databases → postgres → right click → Query tool

In SQL Shell simply press enter till password is asked → enter the password →

you are now connected → if successfully done you ger postgres=# dialog to write your query.

To connect to django Project:

In settings.py → in the DATABASES section → right now default sqlite is mentioned change it to:

DATABASES = {

'default': {

'ENGINE': 'django.db.backends.postgresql',

'NAME': 'database name default(postgres)',

'USER': 'Your username',

'PASSWORD': 'Your password',

'HOST': 'your host default(localhost)',

'PORT': 'default(5432)'

}

}

Once we have done the changes in settings.py, we must run a migration in our virtual environment, before the changes will take place:

py manage.py makemigrations or python manage.py makemigrations

py manage.py migrate or python manage.py migrate

In models section:

from django.db import models

class Teacher(models.Model):

name = models.CharField(max\_length=80)

age = models.IntegerField()

After applying makemigrations and given that your app is registered(it is added in the INSTALLED APPS section in settings.py) and necessary package is installed (psycog2) run migrate.

The created model(table) can be seen at: in databases → your database(default postgres) → Schema → public → Tables

PostgreSQL queries that you can run in the query Tool or SQL Shell:

As you can see they are the same as the SQL queries.

CREATE TABLE

CREATE TABLE students(name VARCHAR(255), rollno INT);

INSERT

INSERT INTO students(name, rollno) VALUES ('Krishna', 1);

FIND

SELECT \* FROM students;

SELECT \* FROM students WHERE name='Krishna';

DELETE

DELETE FROM students WHERE name='Jane';

<https://stackpython.medium.com/how-to-start-django-project-with-a-database-postgresql-aaa1d74659d8>