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).

Django is especially helpful for database driven websites.

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.

URLs

Django also provides a way to navigate around the different pages in a website.

When a user requests a URL, Django decides which *view*it will send it to.

This is done in a file called urls.py.

So, What is Going On?

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 can do a lot more than this, but this is basically what you will learn in this tutorial, and are the basic steps in a simple web application made with Django.

Django History

Django was invented by Lawrence Journal-World in 2003, to meet the short deadlines in the newspaper and at the same time meeting the demands of experienced web developers.

Initial release to the public was in July 2005.

Latest version of Django is 4.0.3 (March 2022).

To install Django, you must have [Python](https://www.w3schools.com/python/default.asp) installed, and a package manager like [PIP](https://www.w3schools.com/python/python_pip.asp).

PIP is included in Python from version 3.4.

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/>

PIP

To install Django, you must use a package manager like PIP, which is included in Python from version 3.4.

To check if your system has PIP installed, run this command in the command prompt:

pip --version

If PIP is installed, you will get a result with the version number.

For me, on a windows machine, the result looks like this:

pip 20.2.3 from c:\python39\lib\site-packages\pip (python 3.9)

If you do not have PIP installed, you can download and install it from this page: <https://pypi.org/project/pip/>

Virtual Environment

It is suggested to have a dedicated virtual environment for each Django project, and in the [next chapter](https://www.w3schools.com/django/django_create_virtual_environment.php) you will learn how to create a virtual environment, and then install Django in it.

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" \t "_blank), 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:

Windows:

py -m venv myworld

Unix/MacOS:

python -m venv myworld

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

myworld  
  Include  
  Lib  
  Scripts  
  pyvenv.cfg

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

Windows:

myworld\Scripts\activate.bat

Unix/MacOS:

source myworld/bin/activate

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

Windows:

(myworld) C:\Users\*Your Name*>

Unix/MacOS:

(myworld) ... $

**Note:** You must activate the virtual environment every time you open the command prompt to work on your project.

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:

Windows:

(myworld) C:\Users\*Your Name*>py -m pip install Django

Unix/MacOS:

(myworld) ... $ python -m pip install Django

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

Collecting Django  
  Downloading Django-4.0.3-py3-none-any.whl (8.0 MB)  
      |████████████████████████████████| 8.0 MB 2.2 MB/s  
Collecting sqlparse>=0.2.2  
  Using cached sqlparse-0.4.2-py3-none-any.whl (42 kB)  
Collecting asgiref<4,>=3.4.1  
  Downloading asgiref-3.5.0-py3-none-any.whl (22 kB)  
Collecting tzdata; sys\_platform == "win32"  
  Downloading tzdata-2021.5-py2.py3-none-any.whl (339 kB)  
      |████████████████████████████████| 339 kB 6.4 MB/s  
Installing collected packages: sqlparse, asgiref, tzdata, Django  
Successfully installed Django-4.0.3 asgiref-3.5.0 sqlparse-0.4.2 tzdata-2021.5  
WARNING: You are using pip version 20.2.3; however, version 22.3 is available.  
You should consider upgrading via the 'C:\Users\*Your Name*\myworld\Scripts\python.exe -m pip install --upgrade pip' command.

That's it! Now you have installed Django in your new project, running in a virtual environment!

Windows, Mac, or Unix?

You can run this project on either one. There are some small differences, like when writing commands in the command prompt, Windows uses py as the first word in the command line, while Unix and MacOS use python:

Windows:

py --version

Unix/MacOS:

python --version

In the rest of this tutorial, we will be using the Windows command.

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

My 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:

my\_tennis\_club  
    manage.py  
    my\_tennis\_club/  
        \_\_init\_\_.py  
        asgi.py  
        settings.py  
        urls.py  
        wsgi.py

These are all files and folders with a specific meaning, you will learn about some of them later in this tutorial, but for now, it is more important to know that this is the location of your project, and that you can start building applications in it.

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:

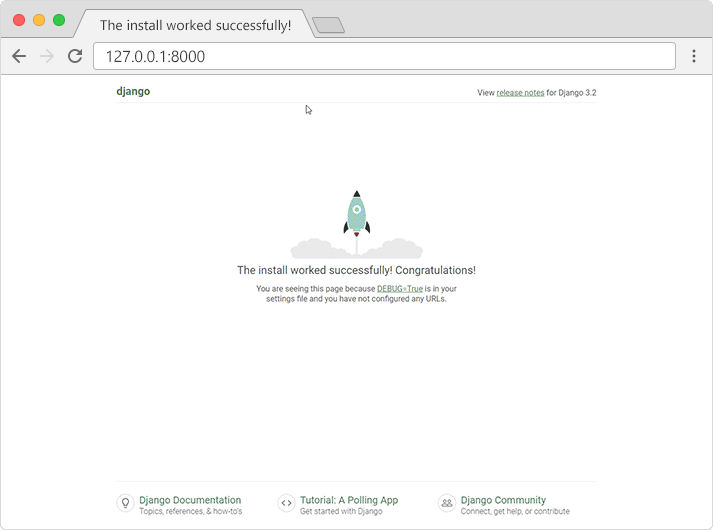
py manage.py runserver

Which will produce this result:

Watching for file changes with StatReloader  
Performing system checks...  
  
System check identified no issues (0 silenced).  
  
You have 18 unapplied migration(s). Your project may not work properly until you apply the migrations for app(s): admin, auth, contenttypes, sessions.  
Run 'python manage.py migrate' to apply them.  
October 27, 2022 - 13:03:14  
Django version 4.1.2, using settings 'my\_tennis\_club.settings'  
Starting development server at http://127.0.0.1:8000/  
Quit the server with CTRL-BREAK.

Open a new browser window and type [127.0.0.1:8000](http://127.0.0.1:8000/) in the address bar.

The result:



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:

my\_tennis\_club  
    manage.py  
    my\_tennis\_club/  
    members/  
        migrations/  
            \_\_init\_\_.py  
        \_\_init\_\_.py  
        admin.py  
        apps.py  
        models.py  
        tests.py  
        views.py

These are all files and folders with a specific meaning. You will learn about most of them later in this tutorial.

First, take a look at the file called views.py.

This is where we gather the information we need to send back a proper response.

## Views

Django views are Python functions that take http requests and return 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

# Create your views here.

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):

return HttpResponse("Hello world!")

**Note:** The name of the view does not have to be the same as the application.

I call it members because I think it fits well in this context.

This is a simple example on how to send a response back to the browser.

But how can we execute the view? Well, we must call the view via a URL.

## 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')),

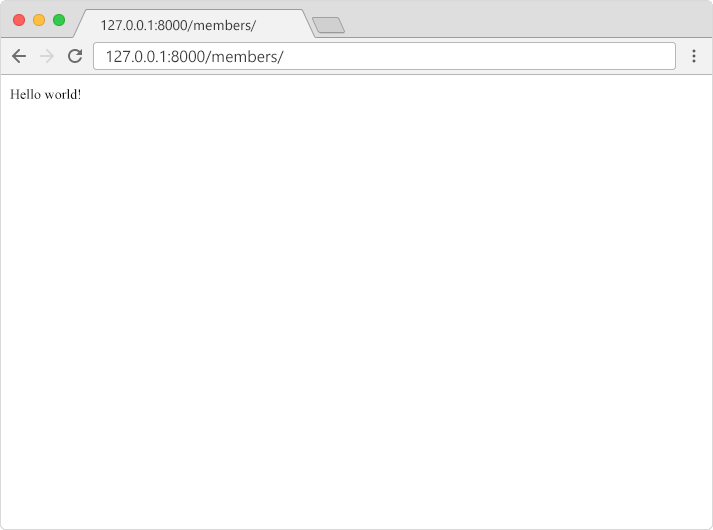
path('admin/', admin.site.urls),

]

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

py manage.py runserver

In the browser window, type [127.0.0.1:8000/members/](http://127.0.0.1:8000/members/) in the address bar.



## 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:

my\_tennis\_club  
    manage.py  
    my\_tennis\_club/  
    members/  
        templates/  
            myfirst.html

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>

</html>

## 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')

return HttpResponse(template.render())

## 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:

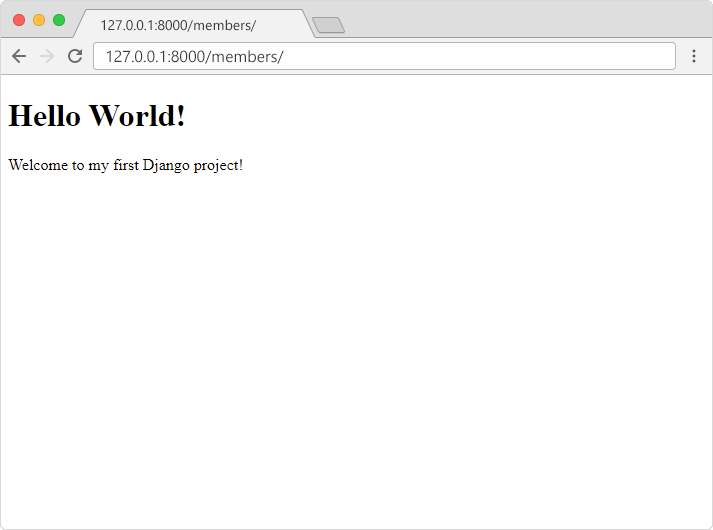
Operations to perform:  
  Apply all migrations: admin, auth, contenttypes, sessions  
Running migrations:  
  Applying contenttypes.0001\_initial... OK  
  Applying auth.0001\_initial... OK  
  Applying admin.0001\_initial... OK  
  Applying admin.0002\_logentry\_remove\_auto\_add... OK  
  Applying 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  
  Applying auth.0010\_alter\_group\_name\_max\_length... OK  
  Applying auth.0011\_update\_proxy\_permissions... OK  
  Applying auth.0012\_alter\_user\_first\_name\_max\_length... OK  
  Applying sessions.0001\_initial... OK  
  
(myworld) C:\Users\Your Name\myworld\my\_tennis\_club>

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/](http://127.0.0.1:8000/members/) in the address bar.

The result should look like this:



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 process.

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)

lastname = 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.

## SQLite Database

When we created the Django project, we got an empty SQLite database.

It was created in the my\_tennis\_club root folder, and has the filename db.sqlite3.

By default, all Models created in the Django project will be created as tables in this database.

## 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:

Migrations for 'members':  
  members\migrations\0001\_initial.py  
    - Create model Member  
  
(myworld) C:\Users\Your Name\myworld\my\_tennis\_club>

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:

Operations to perform:  
  Apply all migrations: admin, auth, contenttypes, members, sessions  
Running migrations:  
  Applying members.0001\_initial... OK  
  
(myworld) C:\Users\Your Name\myworld\my\_tennis\_club>

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:

BEGIN;  
--  
-- Create model Member  
--  
CREATE TABLE "members\_member" ("id" integer NOT NULL PRIMARY KEY AUTOINCREMENT, "firstname" varchar(255) NOT NULL, "lastname" varchar(255) NOT NULL); COMMIT;

## Add Records

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:

Python 3.9.2 (tags/v3.9.2:1a79785, Feb 19 2021, 13:44:55) [MSC v.1928 64 bit (AMD64)] on win32  
Type "help", "copyright", "credits" or "license" for more information.  
(InteractiveConsole)  
>>>

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

>>> from members.models import Member

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

>>> Member.objects.all()

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

<QuerySet []>

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:

>>> member = Member(firstname='Emil', lastname='Refsnes')  
>>> member.save()

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

>>> Member.objects.all().values()

Hopefully, the result will look like this:

<QuerySet [{'id': 1, 'firstname': 'Emil', 'lastname': 'Refsnes'}]>

## Add Multiple Records

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

>>> member1 = Member(firstname='Tobias', lastname='Refsnes')  
>>> member2 = Member(firstname='Linus', lastname='Refsnes')  
>>> member3 = Member(firstname='Lene', lastname='Refsnes')  
>>> member4 = Member(firstname='Stale', lastname='Refsnes')  
>>> member5 = Member(firstname='Jane', lastname='Doe')  
>>> members\_list = [member1, member2, member3, member4, member5]  
>>> for x in members\_list:  
>>>   x.save()

Now there are 6 members in the Member table:

>>> Member.objects.all().values()  
<QuerySet [{'id': 1, 'firstname': 'Emil', 'lastname': 'Refsnes'},  
{'id': 2, 'firstname': 'Tobias', 'lastname': 'Refsnes'},  
{'id': 3, 'firstname': 'Linus', 'lastname': 'Refsnes'},  
{'id': 4, 'firstname': 'Lene', 'lastname': 'Refsnes'},  
{'id': 5, 'firstname': 'Stale', 'lastname': 'Refsnes'},  
{'id': 6, 'firstname': 'Jane', 'lastname': 'Doe'}]>

## Update Records

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

>>> from members.models import Member  
>>> x = Member.objects.all()[4]

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:

>>> x.firstname = "Stalikken"  
>>> x.save()

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

>>> Member.objects.all().values()

Hopefully, the result will look like this:

<QuerySet [{'id': 1, 'firstname': 'Emil', 'lastname': 'Refsnes'},  
{'id': 2, 'firstname': 'Tobias', 'lastname': 'Refsnes'},  
{'id': 3, 'firstname': 'Linus', 'lastname': 'Refsnes'},  
{'id': 4, 'firstname': 'Lene', 'lastname': 'Refsnes'},  
{'id': 5, 'firstname': 'Stalikken', 'lastname': 'Refsnes'},  
{'id': 6, 'firstname': 'Jane', 'lastname': 'Doe'}]>

## Delete Records

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

>>> from members.models import Member  
>>> x = Member.objects.all()[5]

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:

>>> Member.objects.all().values()  
<QuerySet [{'id': 1, 'firstname': 'Emil', 'lastname': 'Refsnes'},  
{'id': 2, 'firstname': 'Tobias', 'lastname': 'Refsnes'},  
{'id': 3, 'firstname': 'Linus', 'lastname': 'Refsnes'},  
{'id': 4, 'firstname': 'Lene', 'lastname': 'Refsnes'},  
{'id': 5, 'firstname': 'Stalikken', 'lastname': 'Refsnes'}]>

## 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:

py manage.py makemigrations members

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:

py manage.py makemigrations members  
You are trying to add a non-nullable field 'joined\_date' to members without a default; we can't do that (the database needs something to populate existing rows).  
Please select a fix:  
 1) Provide a one-off default now (will be set on all existing rows with a null value for this column)  
 2) Quit, and let me add a default in models.py  
Select an option:

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

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(null=True)

joined\_date = models.DateField(null=True)

And make the migration once again:

py manage.py makemigrations members

Which will result in this:

Migrations for 'members':  
  members\migrations\0002\_member\_joined\_date\_member\_phone.py  
    - Add field joined\_date to member  
    - Add field phone to member

Run the migrate command:

py manage.py migrate

Which will result in this output:

Operations to perform:  
  Apply all migrations: admin, auth, contenttypes, members, sessions  
Running migrations:  
  Applying members.0002\_member\_joined\_date\_member\_phone... OK  
  
(myworld) C:\Users\Your Name\myworld\my\_tennis\_club>

## Insert Data

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

First we enter the Python Shell:

py manage.py shell

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

Python 3.9.2 (tags/v3.9.2:1a79785, Feb 19 2021, 13:44:55) [MSC v.1928 64 bit (AMD64)] on win32  
Type "help", "copyright", "credits" or "license" for more information.  
(InteractiveConsole)  
>>>

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

>>> from members.models import Member  
>>> x = Member.objects.all()[0]  
>>> x.phone = 5551234  
>>> x.joined\_date = '2022-01-05'  
>>> x.save()

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:

>>> Member.objects.all().values()

The result should look like this:

<QuerySet [  
{'id': 1, 'firstname': 'Emil', 'lastname': 'Refsnes', 'phone': 5551234, 'joined\_date': datetime.date(2022, 1, 5)},  
{'id': 2, 'firstname': 'Tobias', 'lastname': 'Refsnes', 'phone': None, 'joined\_date': None},  
{'id': 3, 'firstname': 'Linus', 'lastname': 'Refsnes', 'phone': None, 'joined\_date': None},  
{'id': 4, 'firstname': 'Lene', 'lastname': 'Refsnes', 'phone': None, 'joined\_date': None},  
{'id': 5, 'firstname': 'Stalikken', 'lastname': 'Refsnes', 'phone': None, 'joined\_date': None}]>