Web Framework (Part 1)

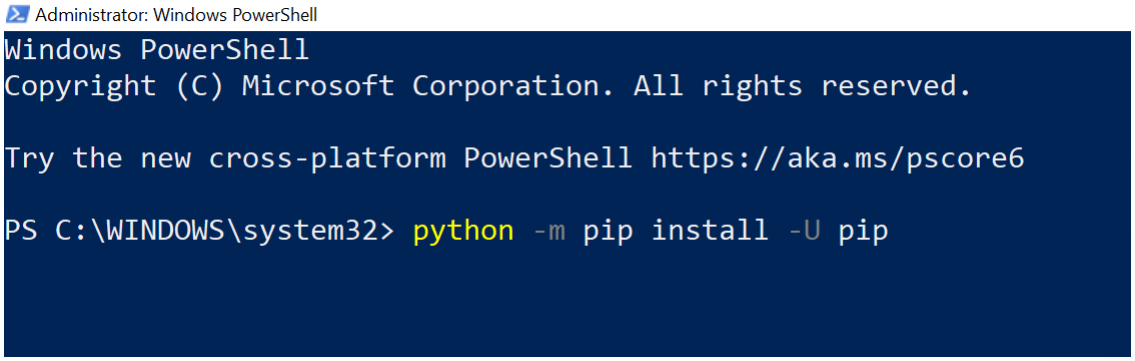
Building Interactive Software

Initial Setup

Preparing your machine

Run the Windows PowerShell command line prompt with an administrator rights.

With Python already installed previously, run the following command:

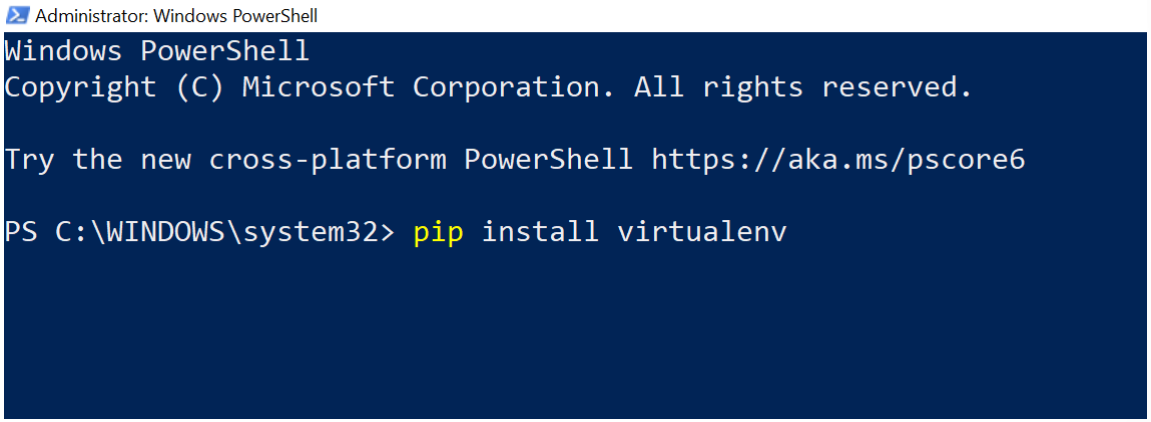


This will either install pip or tell you that it is already up to date. Pip will assist with the installation of our other requirements.

Initial Setup

Preparing your machine (Continued)

Next, from the command line, run the following:



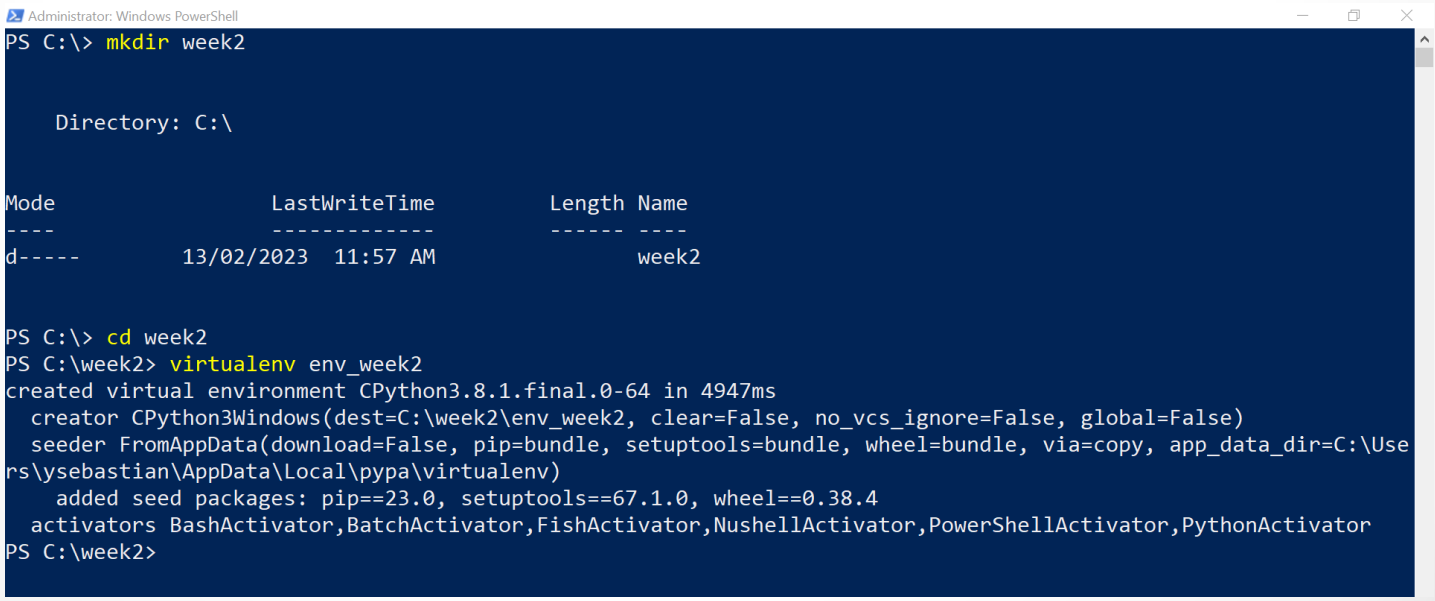
This will install the python virtual environment tool onto your machine and allow us to create distinct working environments without the risk of affecting other projects we are working on. Once completed, your machine is ready to use and you should **not** have to run these commands again.

Setup Work Environment

Preparing your environment

Each project should be given its own virtual environment.

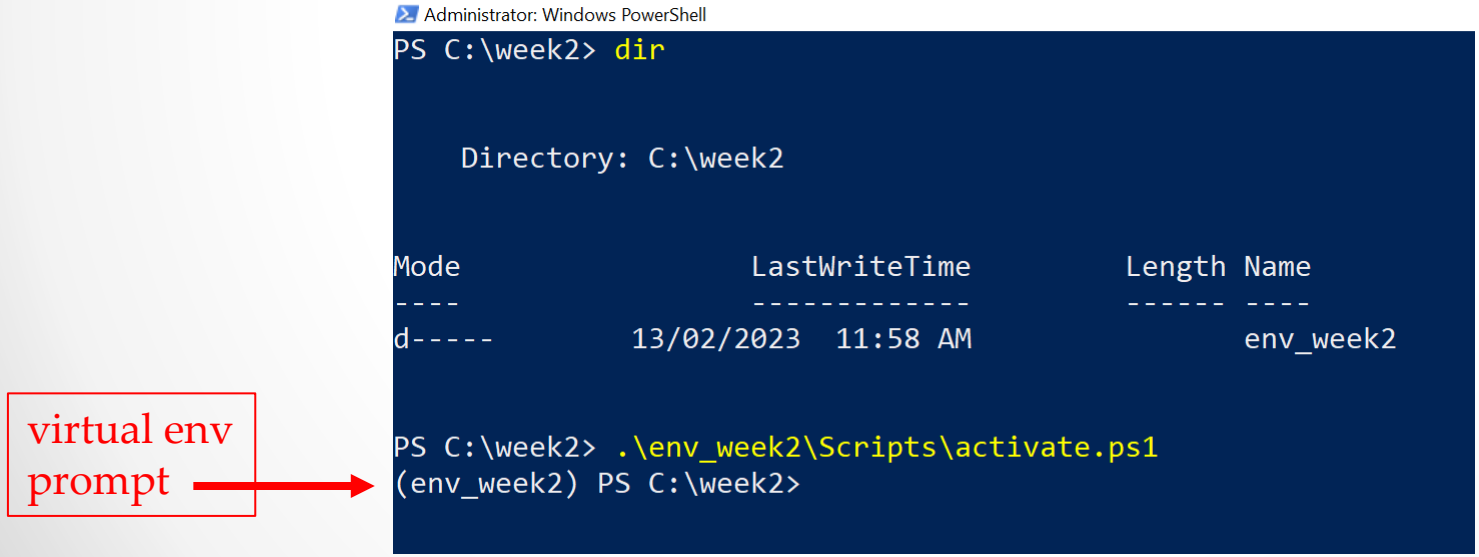
Create a new folder for your new Django project (preferably under C:\ drive for easy navigation). Then, open the Windows PowerShell command prompt in that location. For example:



Setup Work Environment

Preparing your environment (Continued)

Next, activate your newly created environment by running the following (note your path may be different). Once run, the environment name will appear at the beginning of the prompt, indicating that the virtual environment is running and ready.



Setup Work Environment

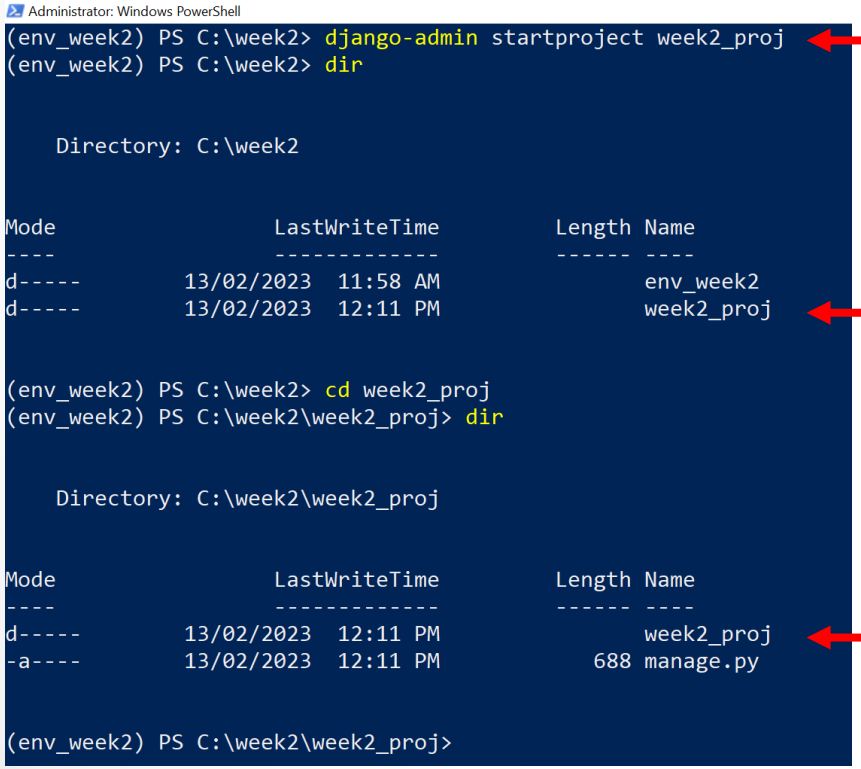
Preparing your environment (Continued)

For every new virtual environment we setup, we will need to make sure we install Django. With your virtual environment active, install the latest Django version:

Setup Project

Preparing your Django project

Create a new Django project:



Command:
Django-admin startproject <proj. name>

Project container

Project Python package

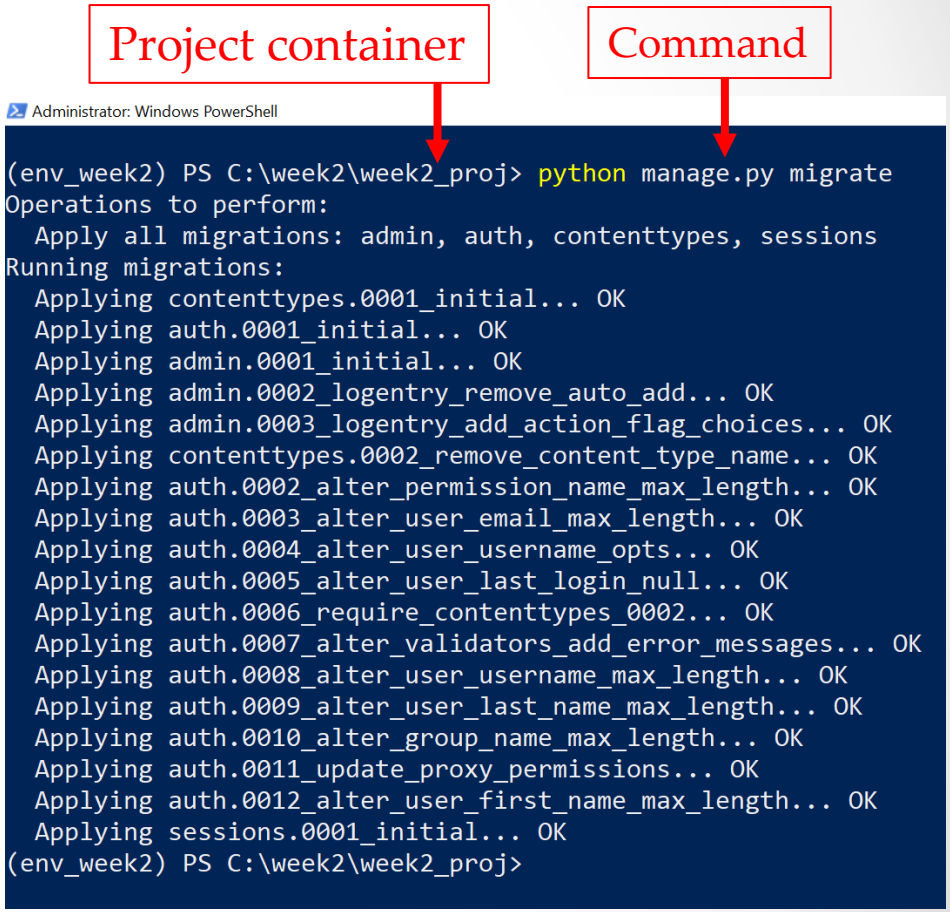
Setup Project

Preparing your Django project (Continued)

We also need to create the database that Django uses to store information about your project.

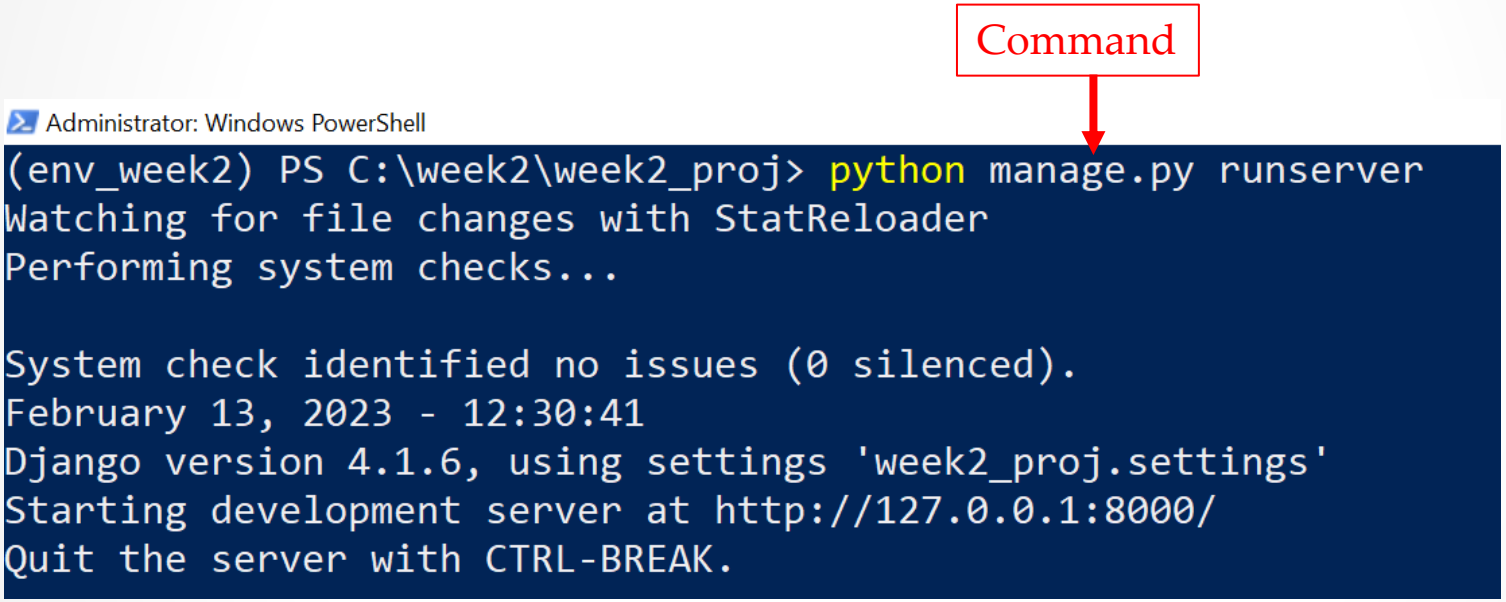
Navigate into the project container folder.

Then, run the command line utility **manage.py** as follows:



Run Django Server

Finally, test run the Django Server with the following command:

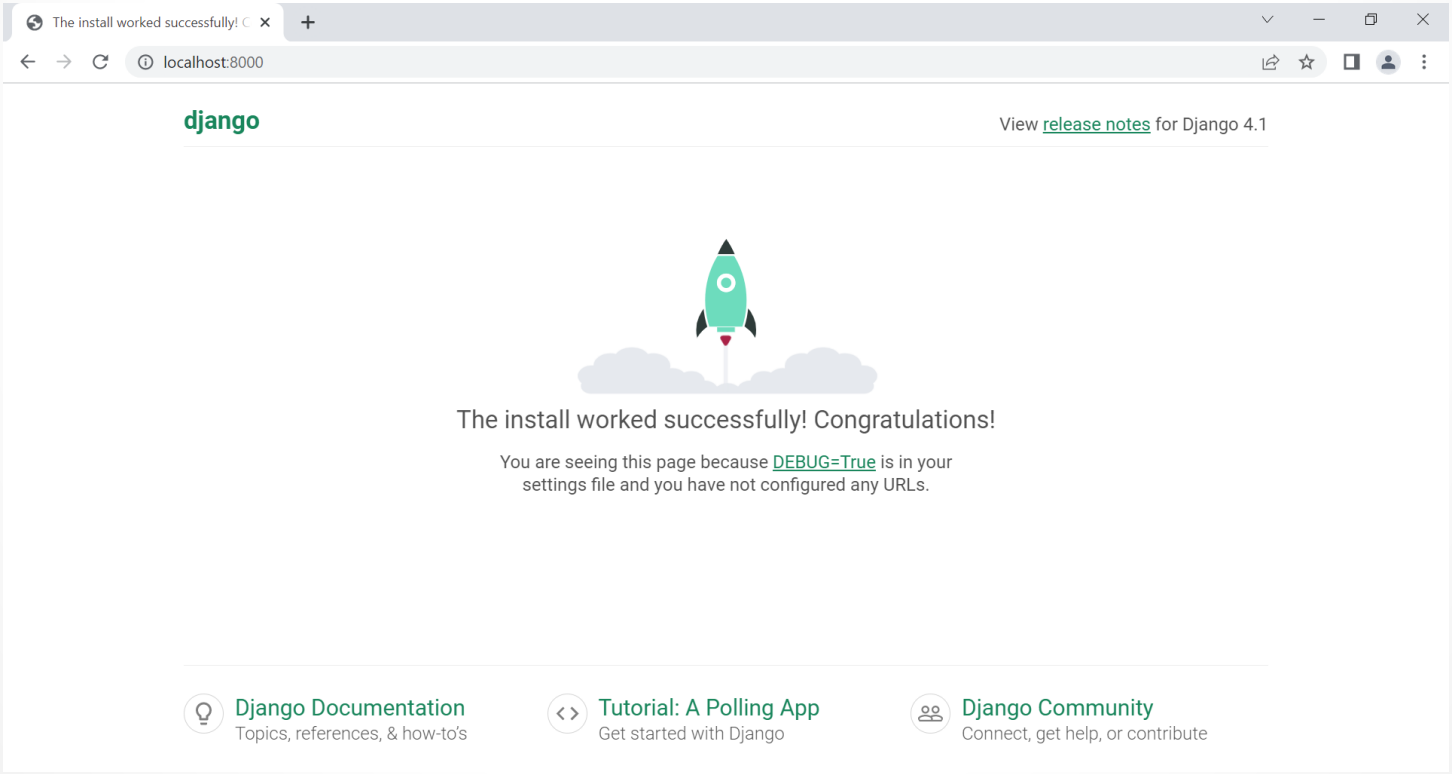


Note: While the server is running, you will not be able to use that command prompt window for anything else.

Run Django Server

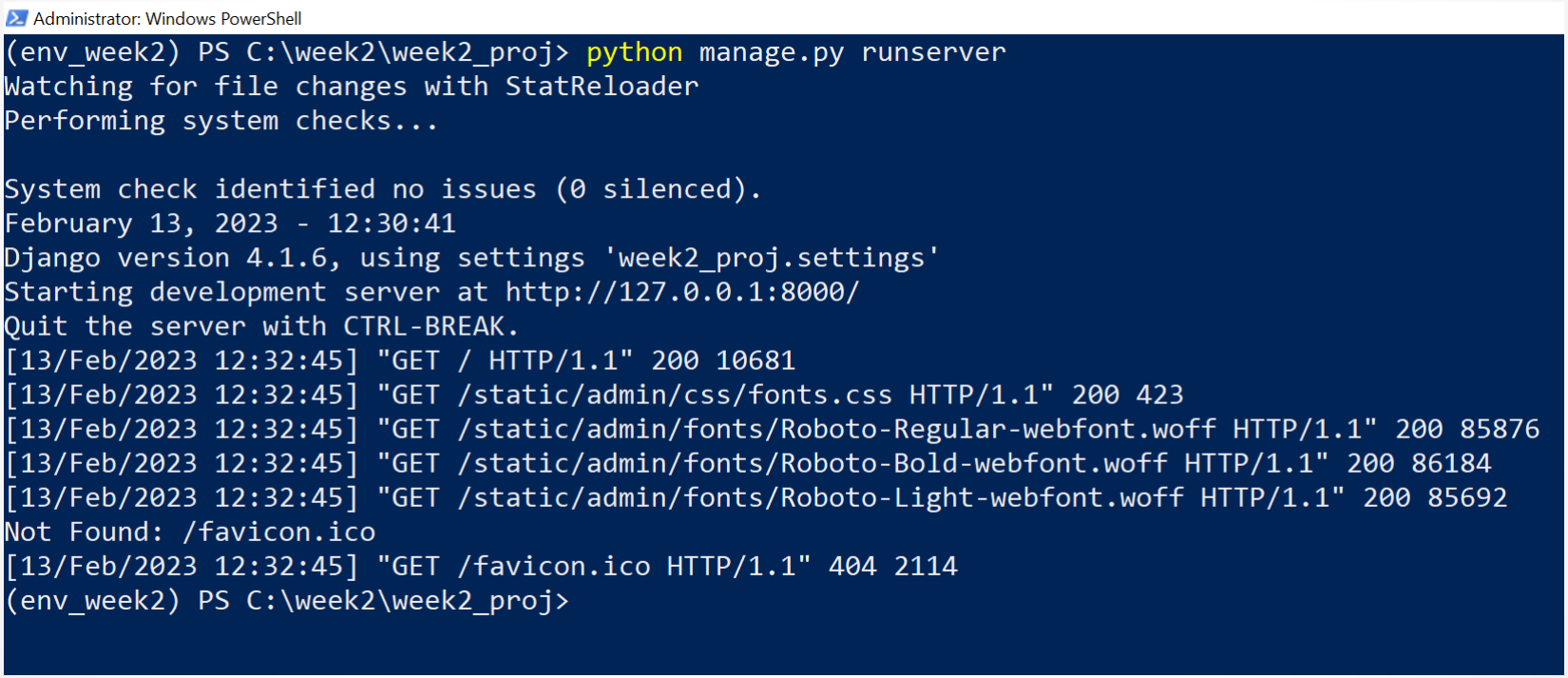
Open your web browser to localhost port 8000. http://localhost:8000/

You should see a similar page:



Stop Django Server

To stop a Django server, hit Ctrl + C on the command line windows:



Note: You can't explicitly see the Ctrl + C on the window.

Exit Virtual Env

To exit a virtual environment, use the following command:

```
[13/Feb/2023 12:32:45] "GET /favicon.ico HTTP/1.1" 404 2114
(env_week2) PS C:\week2\week2_proj> deactivate
PS C:\week2\week2_proj>
```

Once deactivated, the virtualenv prompt disappears.

Django Hello World

Now, we will add some lines of code to create a simple Django website that says “Hello world!”

Django Hello World

`views.py`

Create a new file `views.py` in the same directory as the `urls.py` file (i.e. inside the Python project package folder) with the following content:

```
from django.http import HttpResponse

def hello(request):
    return HttpResponse("Hello world!")
```

Django Hello World

`urls.py`

This Python file controls the URL Configuration for your project. By adding entries into the configuration we can control which URLs link to which resources on the server.

Think of it as "Table of Contents" for your project.

It contains one URL by default:

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

Django Hello World

`urls.py`

Edit and save the `urls.py` file, to reflect the changes below.

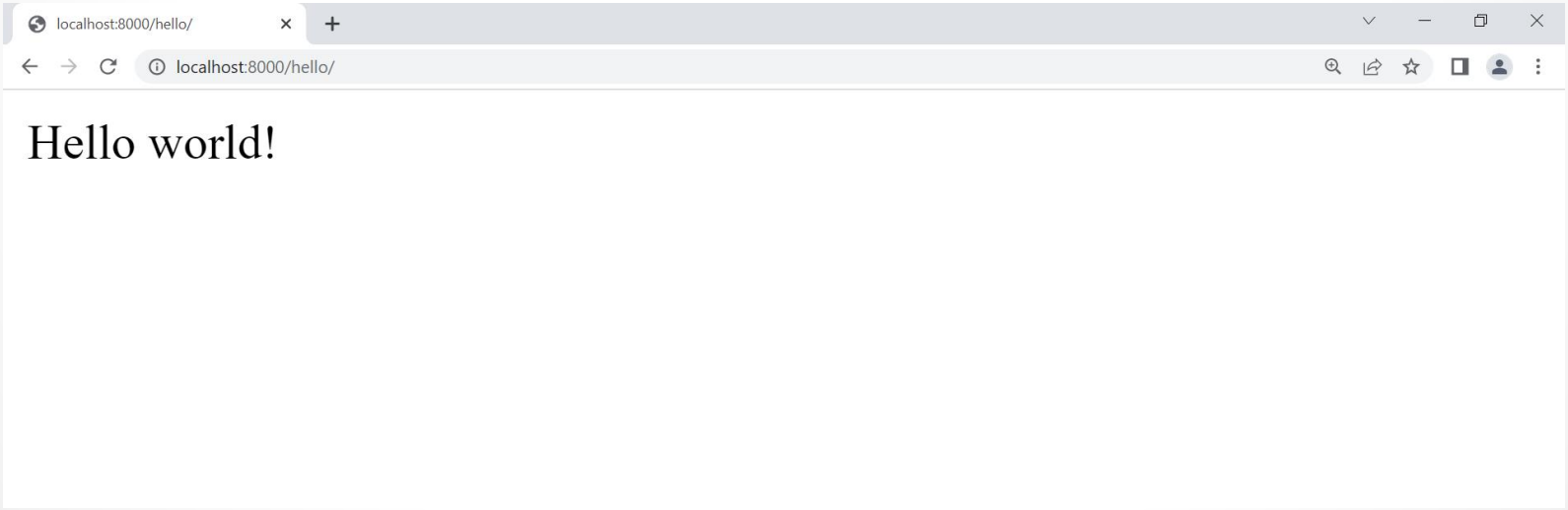
```
from django.contrib import admin
from django.urls import path, re_path
from . import views

urlpatterns = [
    re_path(r'^admin/', admin.site.urls),
    re_path(r'^hello/?$', views.hello),
]
```

Django Hello World

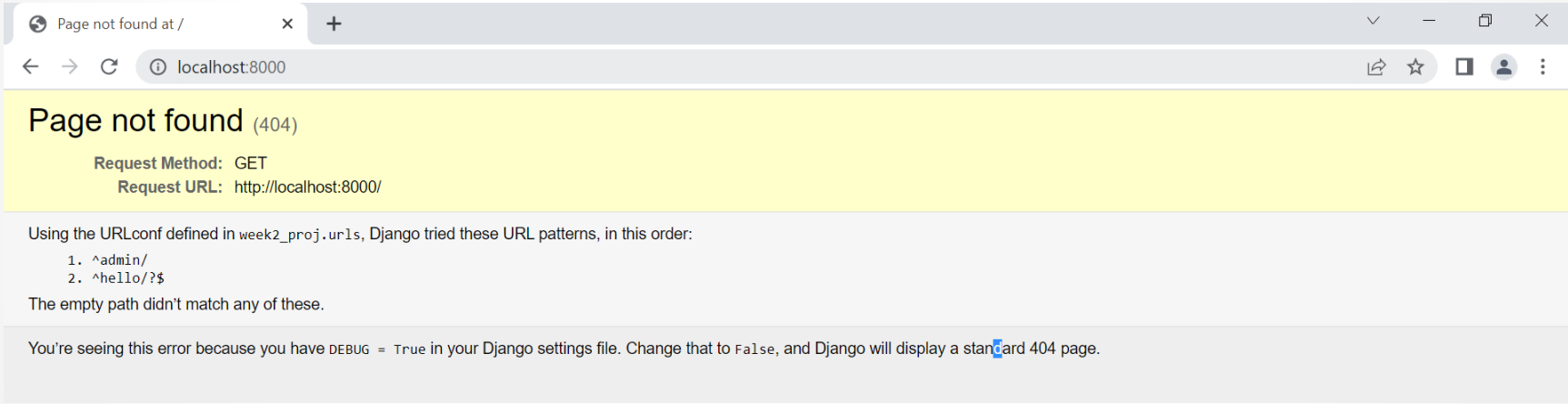
Ensuring that the virtual environment is active, restart the Django server. Open this URL on your web browser http://localhost:8000/hello/

You should see the following output:



Django Hello World

Open the URL on your web browser http://localhost:8000/. You will see the error below. Why?



Fix the problem by adding a suitable URL entry in `urls.py`, calling a function named **home** from `views.py`. The **home** function writes the word "Homepage" on the browser in response to http://localhost:8000/ .

Uniform Resource Locator

A Uniform Resource Locator (URL) is a reference to a resource that specifies where to find it and how to interact with it.

scheme:[//[user[:password]@]**host**[:port]][/**path**][?query][#fragment]

http://libguides.cdu.edu.au/az.php?s=36055

Uniform Resource Locator

scheme:[//[user[:password]@]host[:**port**]][/path][?query][#fragment]

A port is always associated with an IP address of a host and the protocol type of the communication. When not specified, the default port number for the protocol is used.

The first 1024 port numbers are called the well-known port numbers.

http://libguides.cdu.edu.au:**80**/az.php?s=36055

Uniform Resource Locator

Django Server

Hosting resources on your local machine:

localhost

127.0.0.1

Port 8000

http://localhost:8000[/path]

Regular Expression

Regular Expression (also referred to as RegEx) is a formal way of describing a search pattern. That search pattern can then be applied to a string to determine if a match is found.

Django uses RegEx syntax to match a URL's path component with the appropriate views and resources to display.

Regular Expression

URL: http://localhost:8000/
RegEx: "^$"

URL: http://localhost:8000/books
RegEx: "^books$"

URL: http://localhost:8000/books/django
RegEx: "^books" – What happens here?

Regular Expression

Many online tools assist with the compilation and testing of your Regular Expressions syntax.

The following site allows you to test your syntax on text you provide, and includes a number of resources to assist you with syntax.

http://regexr.com/

Symbol	Matches
. (dot)	Any single character
\d	Any single digit
[A-Z]	Any character between A and Z (uppercase)
[a-z]	Any character between a and z (lowercase)
[A-Za-z]	Any character between a and z (case-insensitive)
+	One or more of the previous expression (e.g., \d+ matches one or more digits)
[^/]+	One or more characters until (and not including) a forward slash
?	Zero or one of the previous expression (e.g., \d? matches zero or one digits)
*	Zero or more of the previous expression (e.g., \d* matches zero, one or more than one digit)
{1,3}	Between one and three (inclusive) of the previous expression (e.g., \d{1,3}matches one, two or three digits)

Web Framework (Part 2)

Building Interactive Software

Revision

What is the single URL Pattern string to match the following 3 example URLs and send them to the same page?

1. http://.../uni/sci101/info
2. http://.../uNi/sci505/info
3. http://.../UNI/sci999/info/

^[Uu][Nn][Ii]/sci\d{3}/info/?$

Revision

What is the single URL Pattern string to match the following example URLs and capture the id# for use within our Django program?

1. http://.../uni/sci101/topic/id1234/

^uni/sci101/topic/id(\d+)/

Templates

What is a Django Template?

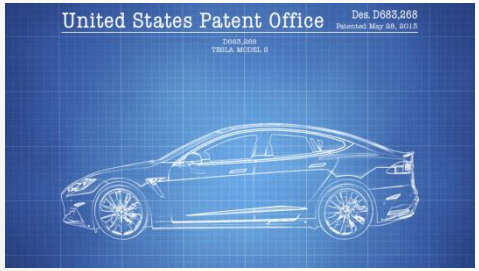
Django template is a string of text that is intended to separate the presentation of a document from its data.

A template defines placeholders and various bits of basic logic (template tags) that regulate how the document should be displayed.

Templates

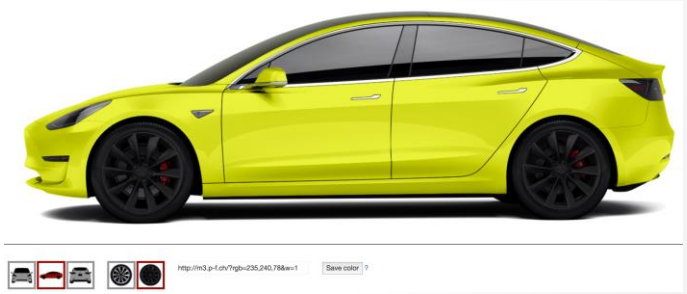
Scenario: Buying a Car

Context



Template

Resulting Output



Templates

```
<html>
<head>
    <title>Ordering Notice</title>
</head>
<body>
    <h1>Ordering Notice</h1>
    <p>Dear {{ person_name }},</p>
    <p>Thanks for placing an order from {{ company }}. It's scheduled to ship on
{{ ship_date|date:"F j, Y" }}.</p>
    <p>Here are the items you've ordered:</p>
    <ul>
        {% for item in item_list %}
        <li>{{ item }}</li>{% endfor %}
    </ul>
    {% if ordered_warranty %}
    <p>Your warranty information will be included in the packaging.</p>
    {% else %}
    <p>
        You didn't order a warranty, so you're on your own when the products
inevitably stop working.
    </p>
    {% endif %}
    <p>Sincerely,<br />{{ company }}</p>
</body>
</html>
```

Templates

Examples of Variables:

```
{{ person_name }}
{{ company }}
{{ item }}
```

Examples of Template Tag:

```
{% for item in item_list %} … {% endfor %}
{% if ordered_warranty %} … {% else %} … {% endif %}
```

Examples of Filters:

```
{{ ship_date|date:"F j, Y" }}
```

Templates

If/else tag

The `{% if %}` tag evaluates a variable, and if that variable is "True" **(i.e., it exists, is not empty, and is not a false Boolean value)** the system will display everything between `{% if %}` and `{% endif %}`

Logical operators are valid:

```
and          {% if tea and biscuits %}
or           {% if milk or sugar %}
not          {% if not honey %}
```

Templates

for tag

The `{% for %}` tag allows you to loop over each item in a sequence. As in Python's `for` statement, the syntax is for X in Y, where Y is the sequence to loop over and X is the name of the variable to use for a particular cycle of the loop.

Each time through the loop, the template system will render everything between `{% for %}` and `{% endfor %}`.

Templates

for tag

```
item_list = ['Apples', 'Pears']
```

Python Code

```
<ul>
    {% for item in item_list %}
    <li>{{ item }}</li>{% endfor %}
</ul>
```

Template

```
<ul>
    <li>Apples</li>
    <li>Pears</li>
</ul>
```

Rendered Output

Templates

Filters

Template filters are simple ways of altering the value of variables before they are displayed. Filters use a pipe character, like this:

```
{{ cust_name|lower }}
```

This example would cause the contents of the variable `cust_name` to be converted to lower case before being rendered.

Templates

Filters

The Django Project documentation lists many of the different filters. Some more common/useful filters include:

- date – Used to format a date string.
- lower – Convert to lowercase.
- upper – Convert to uppercase.
- title – Convert to title case.
- length – display the length of the variable.

Django App

A Django ***project*** is organized as a group of individual ***apps*** that work together to make the project work as a whole.

This allows the project to be broken up into logical components which assists with development and maintainability.

Django App

Projects vs. apps

What's the difference between a project and an app? An app is a web application that does something – e.g., a blog system, a database of public records or a small poll app. A project is a collection of configuration and apps for a particular website. A project can contain multiple apps. An app can be in multiple projects.

From: https://docs.djangoproject.com/en/4.1/intro/tutorial01/

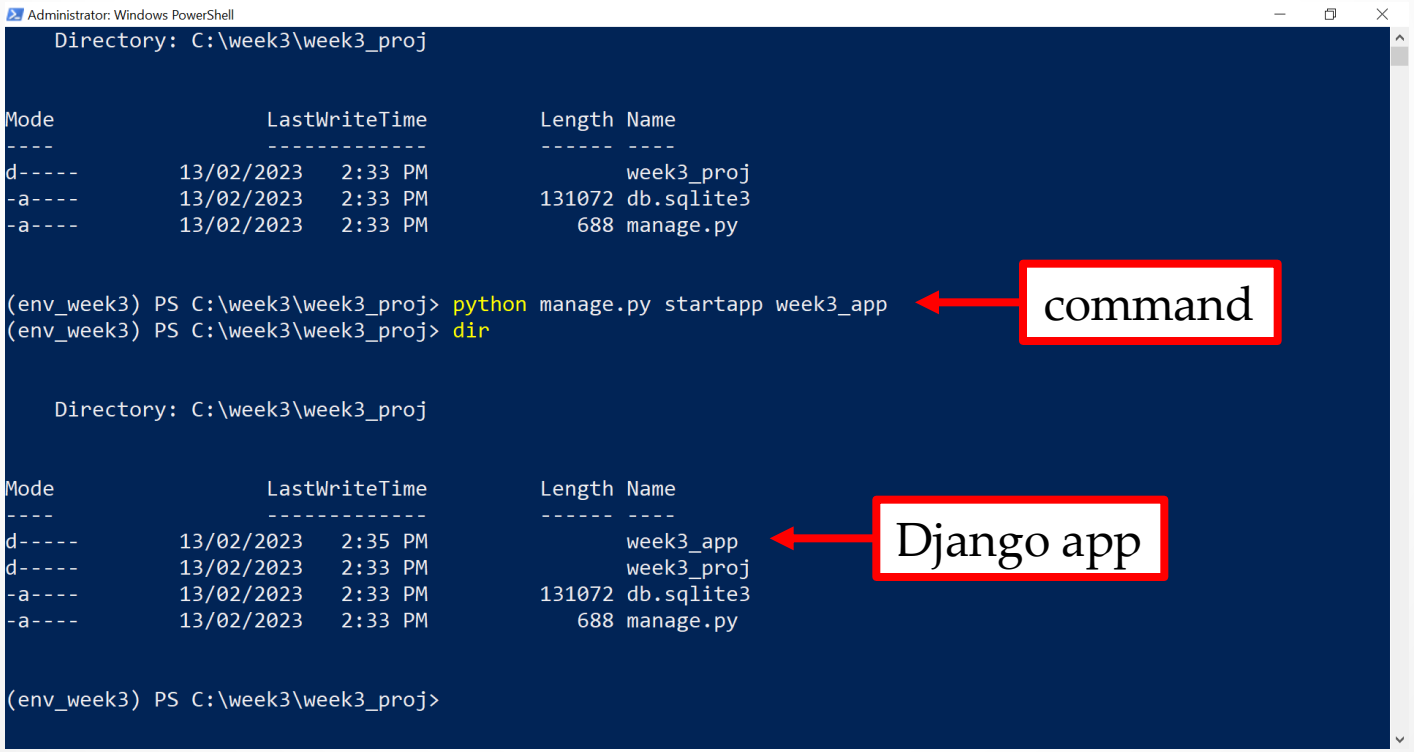
Create an App

Repeat the steps learned last week to create a new virtual environment and Django project. Briefly, those steps are:

1. Run Windows PowerShell (as an administrator)
2. Create and move into a new directory `week3`
3. `virtualenv env_week3`
4. `env_week3\Scripts\activate.ps1`
5. `pip install django`
6. `django-admin startproject week3_proj`
7. `python manage.py migrate`

Create an App

Make sure your virtual environment is active. Then, **from the directory that contains the `manage.py` file** run the following command:



Create an App

Now you should have a Django ***project Python package*** folder (i.e. the ***default*** Web app) called `week3_proj` and a Django ***app*** folder called `week3_app`.

Next, we must register the new app with the current Django project. Edit the **`settings.py`** file found inside the ***default*** Web app `week3_proj` folder.

Create an App

settings.py

```
--snip--
INSTALLED_APPS = [
    --snip--
    'django.contrib.staticfiles',

    # apps created by me
    'week3_app',
]
--snip—
```

Edit the file, and include the new entry in the INSTALLED_APPS list.

Hello World Page

Our app is now setup and configured. Let us recreate the Hello World example from last week.

Our app gives us some advantages over last week.

We no longer need to separately create the views.py file. The new app automatically creates this file for our use.

Also, we can now use Templates to handle serving HTML files for us.

Hello World Page

In the default Web app folder `week3_proj`, edit the **`urls.py`**:

```
from django.contrib import admin
from django.urls import path, re_path
from week3_app import views

urlpatterns = [
    path('admin/', admin.site.urls),
    re_path(r'^hello/?$', views.hello),
]
```

Hello World Page

Inside the app folder `week3_app`, edit **`views.py`**:

```
from django.shortcuts import render

# Create your views here.
def hello(request):
    return render(request, 'week3_app/hello.html')
```

Hello World Page

By default, Django apps store their HTML Templates in a subfolder of the app folder which is named '`templates`'. We have to add this folder manually.

Create the '`templates`' folder now.

In our example, the folder location is:

```
...\week3_app\templates\
```

Hello World Page

Now that we have a templates folder, we can place our HTML Templates here, or sort them into other additional subfolders.

Our example is:

```
'...\templates\week3_app\hello.html'
```

So create a new **subfolder** called 'week3_app' and then create the 'hello.html' file within it.

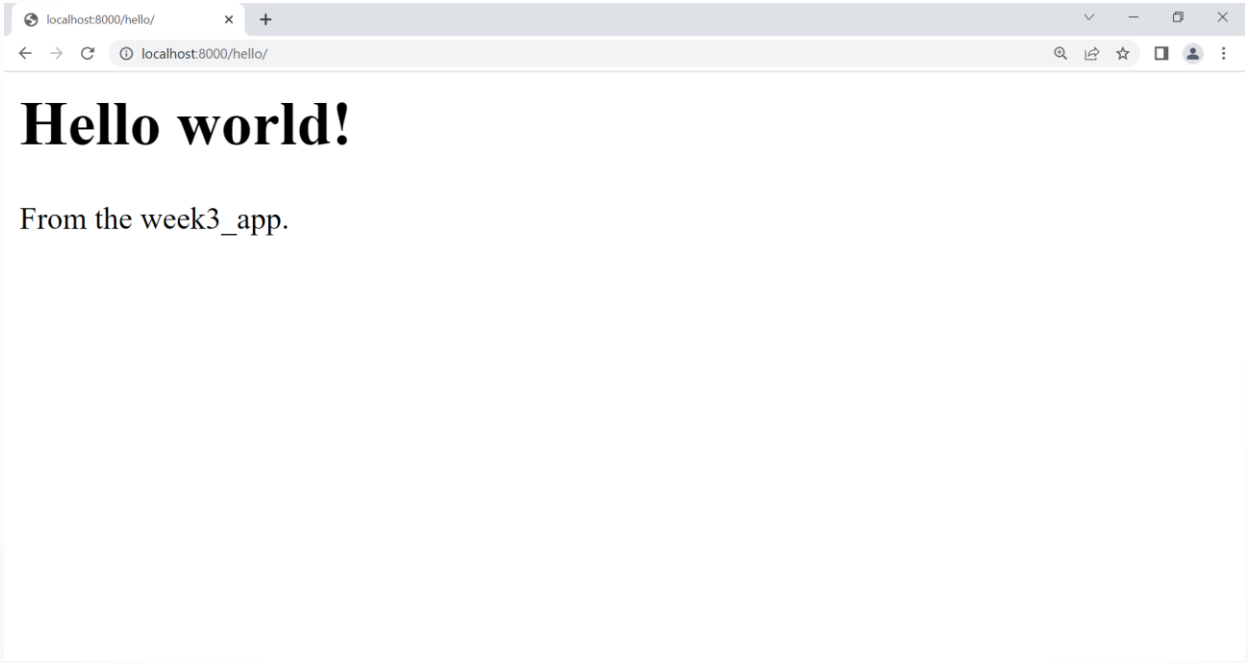
Hello World Page

hello.html

```
<html>
  <body>
    <h1>Hello world!</h1>
    <p>From the week3_app.</p>
  </body>
</html>
```

Hello World Page

Run your Django server and call URL
http://localhost:8000/hello/



Templates & Context

Now let's build the example from the HTML file shown on slide #6 within our new Django app and pass some data through the context to populate the Template's fields.

Copy the HTML from slide #6 into a new HTML file in your template folder. In our example, this path is:

```
…\week3_app\templates\week3_app\order.html
```

Templates & Context

Edit the ***project*** `urls.py` file to include a new URL pattern for our new page.

```
re_path(r'^order/?$', views.order),
```

This will match the URL: http://localhost:8000/order/

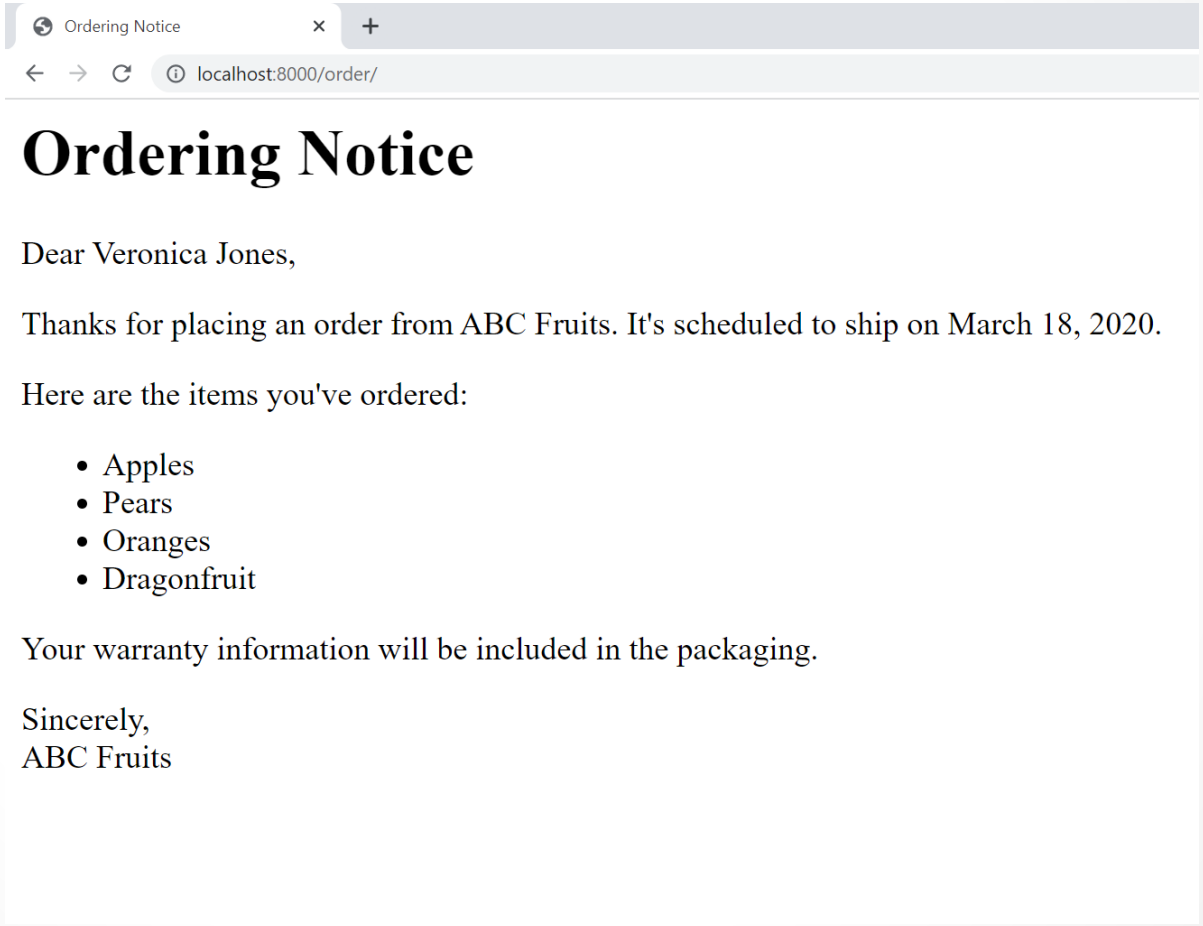
Templates & Context

In the week3_app folder, edit **`views.py`** by adding the new function **order** below the **hello** function:

```
import datetime
--snip--
def order(request):
    company_name = 'ABC Fruits'
    name = 'Veronica Jones'
    ship_date = datetime.date(2020, 3, 18)
    item_list = ['Apples', 'Pears', 'Oranges', 'Dragonfruit']
    context_data = {
        'company': company_name,
        'person_name': name,
        'ship_date': ship_date,
        'item_list': item_list,
        'ordered_warranty': True,
    }
    return render(request, 'week3_app/order.html', context_data)
```

Templates & Context

Output:



What is next?

What Is Next?

Read this short section of the documentation. Stop once you hit the heading "Components".

Experiment with the week3_app example by changing the Context data and altering the values in the Template. Try adding some filters to the variables being rendered.

Constructing a static website with Django

Building Interactive Software

Revision

1. What is a Django Template?
2. The {% if %} tag is true when?
3. Write a valid {% for %} tag statement.
4. List some common Filters.

The include Template Tag

`{% include %}`

This tag allows you to include the contents of another template into the current template.

Anytime you have the same code in multiple templates, consider using an `{% include %}` to remove the duplication.

The include Template Tag

Example File To Include

`now.html`

```
It is currently <em>{% now "jS F Y H:i" %}</em>
```

The include Template Tag

Example Template File

`index.html`

```
<html>
<body>
    <h1>Acme Incorporated</h1>
    <p>Thank you for visiting our website.</p>
    <p>{% include 'now.html' %}</p>
    <p>We expect to return on Friday</p>
</body>
</html>
```

The include Template Tag

The `{% include %}` tag is excellent for these common scenarios:

- Addresses and other contact details.
- Company or division names.
- Navigation menus.
- Copyright messages.

Any common wording that may change over time.

The include Template Tag

Try it yourself:

- Setup your Django Project and App.
- Define the URL Pattern and view function.
- Create a basic HTML Template and use the `{% include %}` tag.
- Be creative.

Template Inheritance

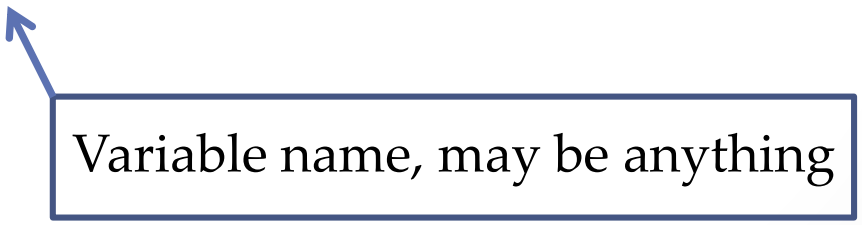
Template inheritance lets you build a base "skeleton" template that contains all the common parts of your site and defines "blocks" that child templates can override.

This allows you to create a general theme across your entire site, without having to recode it for every page on your site.

Template Inheritance

The “blocks” defined within the parent template are named, so that you can reference that section of the page specifically and insert your content there.

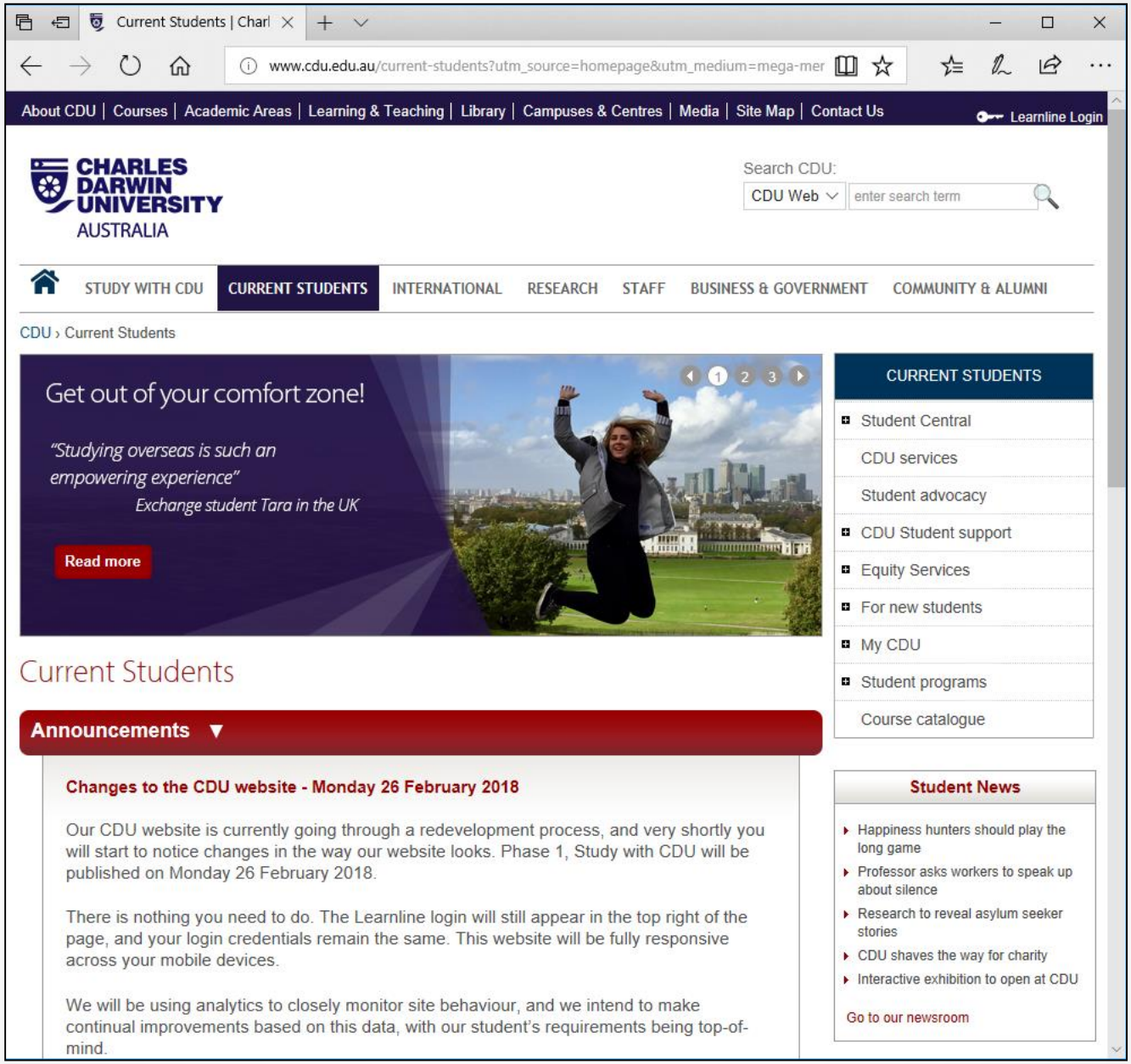
```
{% block my_block %}{% endblock %}
```

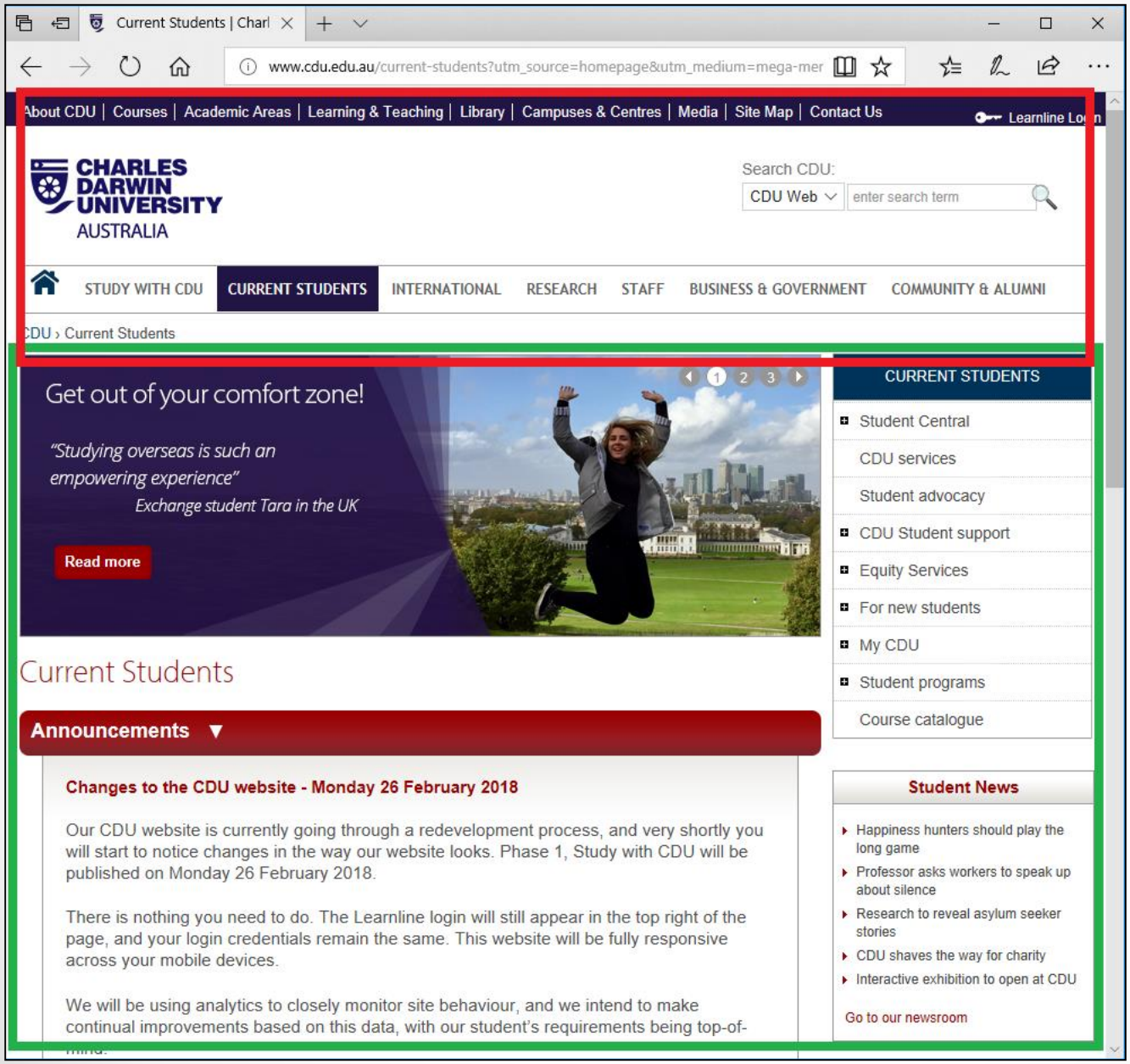


Template Inheritance

A "block" can have a default value supplied if desired. The default content is used if the child template does not supply content for that "block".

```
{% block my_block %}
  <em>There is no news today!</em>
{% endblock %}
```


Current Students | Charl
www.cdu.edu.au/current-students?utm_source=homepage&utm_medium=mega-mer
About CDU | Courses | Academic Areas | Learning & Teaching | Library | Campuses & Centres | Media | Site Map | Contact Us
Learnline Login
CHARLES DARWIN UNIVERSITY AUSTRALIA
Search CDU:
CDU Web
enter search term
STUDY WITH CDU
CURRENT STUDENTS
INTERNATIONAL
RESEARCH
STAFF
BUSINESS & GOVERNMENT
COMMUNITY & ALUMNI
CDU › Current Students
Get out of your comfort zone!
"Studying overseas is such an empowering experience"
Exchange student Tara in the UK
Read more
1 2 3
CURRENT STUDENTS
Student Central
CDU services
Student advocacy
CDU Student support
Equity Services
For new students
My CDU
Student programs
Course catalogue
Current Students
Announcements
Changes to the CDU website - Monday 26 February 2018
Our CDU website is currently going through a redevelopment process, and very shortly you will start to notice changes in the way our website looks. Phase 1, Study with CDU will be published on Monday 26 February 2018.
There is nothing you need to do. The Learnline login will still appear in the top right of the page, and your login credentials remain the same. This website will be fully responsive across your mobile devices.
We will be using analytics to closely monitor site behaviour, and we intend to make continual improvements based on this data, with our student's requirements being top-of-mind.
Student News
Happiness hunters should play the long game
Professor asks workers to speak up about silence
Research to reveal asylum seeker stories
CDU shaves the way for charity
Interactive exhibition to open at CDU
Go to our newsroom



Inherited Content

Block Content

CDU Website

cdu_base.html

```
<html>
<head>
     <title>
     {% block page_title %}Charles Darwin University{% endblock %}
     </title>
</head>
<body>
     {# Navigation menu html code #}
     --snip: Inherited Navigation Menus --
     {% block main_body %}{% endblock %}
     {# Page footer html code #}
     --snip: Inherited Page Footer --
</body>
</html>
```

CDU Website

current_students.html

Important: Must be on the first line of the file!

```
{% extends "cdu_base.html" %}
{% block page_title %}
    Current Students | Charles Darwin University
{% endblock %}
{% block main_body %}
     <p>Example body content goes here.</p>
{% endblock %}
```

CDU Website

```
def current_students(request):
    page_data = {
        # --snip--
    }
    return render(
            request,
            'current_students.html',
            page_data)
```

Naming URL Paths

It is possible to name the paths you define in the `urls.py` file. This allows us to refer to that URL path in other sections of our code.

```
urlpatterns = [
    path('admin/', admin.site.urls),
    path('home/', views.index, name='home'),
    path('current/', views.current_students),
]
```

Naming URL Paths

Once we have a named path, it is possible to conveniently use the `{% url %}` tag.

The tag returns an absolute path reference (a URL without the domain name) matching a given view function and optional parameters.

Naming URL Paths

```
<p>
 <a href="{% url 'home' %}">Home</a>
</p>
```

--- Renders As ---

```
<p>
 <a href="http://localhost:8000/home/">Home</a>
</p>
```

Naming URL Paths

If at a later point in time, the path for "home" changes, it will automatically use the updated path, because the `{% url %}` tag is looking up the entry in the `urls.py` file and getting the current value.

Example Website

Several example ZIP files are available for download and review.

The Virtual Environment folder is not present in the ZIP. You will need to create that yourself, activate it, and install Django in order to run the example.

Examine the code to determine what the examples currently do. *Hint:* Start with the `URLS.PY` file.

What is next?

What Is Next?

Read this short section of the documentation. Stop once you hit the heading "Automatic HTML escaping".

Search the documentation to learn more about the `{% url %}` tag. Investigate how to make a hyperlink that includes an id number using this tag.

Database concepts

Building Interactive Software

Revision

1. What does the `{% include %}` tag allow?
2. Which 2 tags are used in template inheritance?
3. How do you identify a URL path for later reference?
4. Why is the `{% url %}` tag important/useful?

Data

1. a plural of datum.
2. (used with a plural verb) individual facts, statistics, or items of information:
   These data represent the results of our analyses.
   Data are entered by terminal for immediate processing by the computer.
3. (used with a singular verb) a body of facts; information:
   Additional data is available from the president of the firm.

Model

1. a standard or example for imitation or comparison.
2. a representation, generally in miniature, to show the construction or appearance of something.

(Additional definitions omitted)

Data Model

A representation of data which shows how the information is structured, and the components from which it is derived.

Data Model

Scenario

Company ABC wants a website that will sell customized pet rocks. Their products come in multiple sizes and types of rock. Customers can choose from an array of eye, nose and mouth components for their new pet. ABC also wants to offer discount coupons through their checkout. They offer free home delivery as part of their premium service.

Data Model

- What data can be gathered from the simple scenario statement?
- What information is going to be needed to complete a transaction?
- Is there anything that may be ambiguous or open to interpretation?

Data Model

A data model can help to:

- Visualize the information
- Identify missing information
- Clarify the rules about how the information interacts.

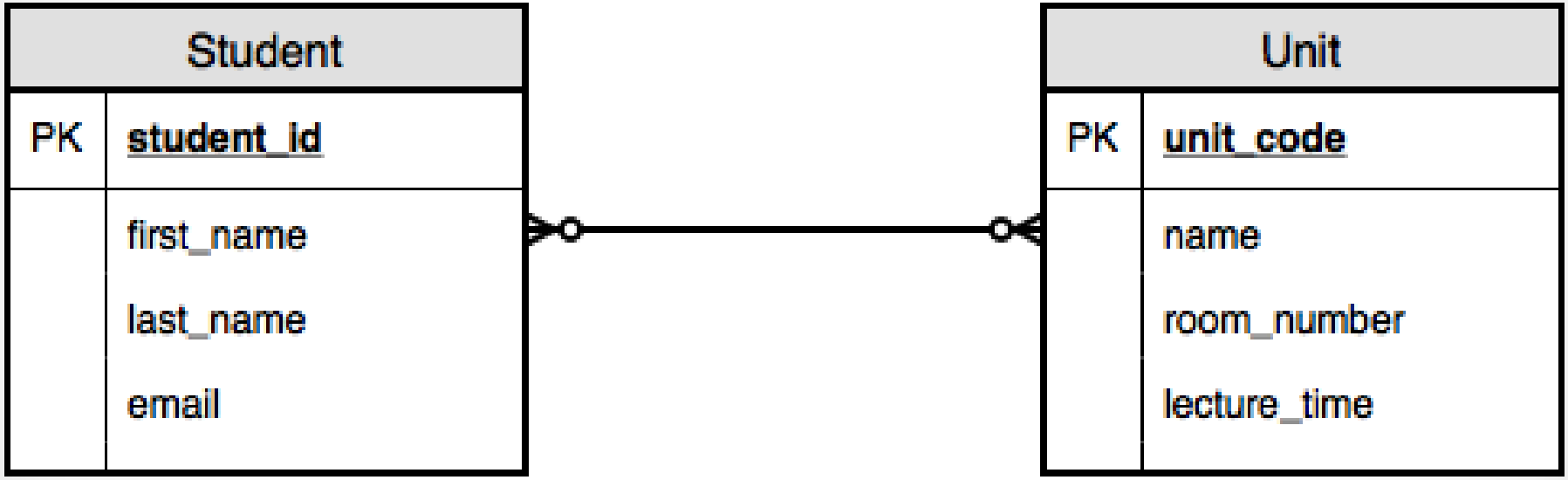
Data Model

A data model can be represented in several ways.

Entity Relationship Diagrams (ERD) allow high level visualization of the data.

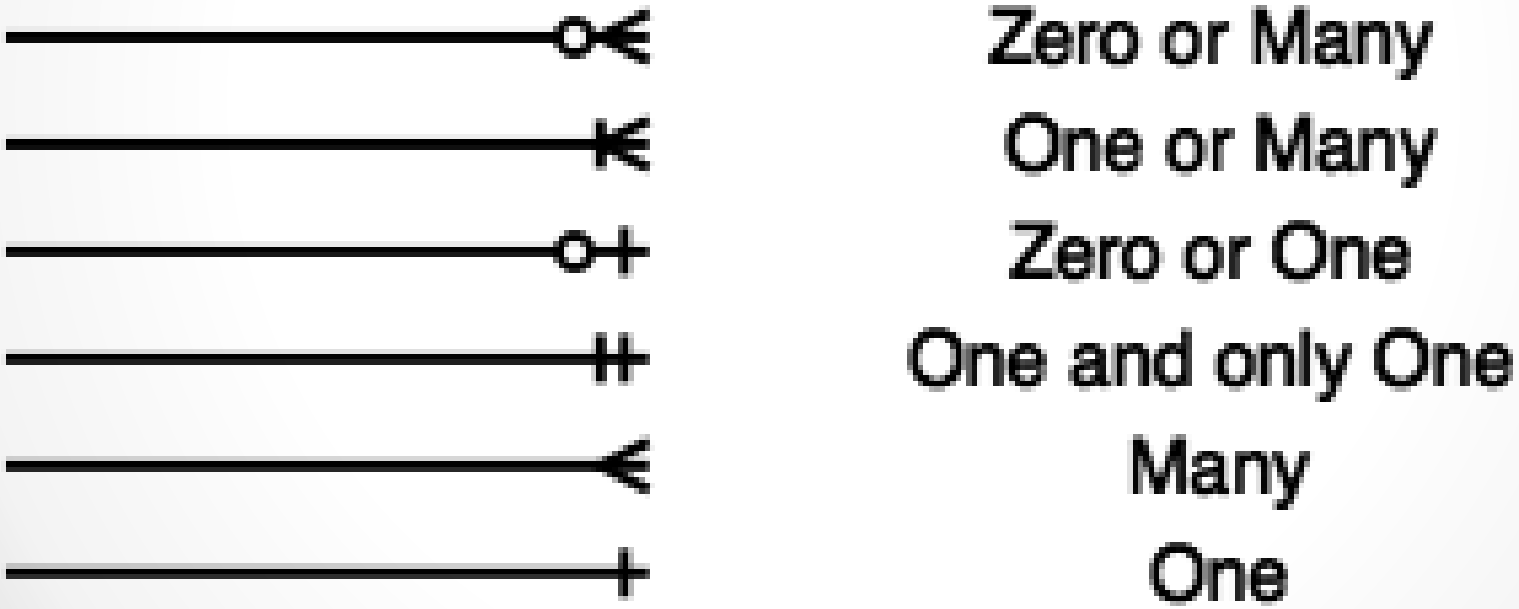
Data dictionary allows a granular description of the data.

Entity Relationship Diagram



Entity Relationship Diagram

Cardinality



Data Dictionary

Name	Type	Size	Constraints	Description	Example
student_id	Numeric	6	Primary Key	Unique identifier for each student	123456
first_name	String	25	Not Null	First name that the student goes by	Dharin
last_name	String	25	Not Null	Last name that the student goes by	Moi
email	String	100	Validated	Contact email address for student	abc@efg.com

Relational Database

In simplest terms, a relational database is one that presents information in tables with rows and columns.

A table is referred to as a relation in the sense that it is a collection of objects of the same type (rows).

Relational Database

Employees Table

Employee_Number	First_name	Last_Name	Date_of_Birth	Car_Number
10001	John	Washington	28-Aug-43	5
10083	Arvid	Sharma	24-Nov-54	null
10120	Jonas	Ginsberg	01-Jan-69	null
10005	Florence	Wojokowski	04-Jul-71	12
10099	Sean	Washington	21-Sep-66	null
10035	Elizabeth	Yamaguchi	24-Dec-59	null

Each row represents a unique employee record, collected into a table of Employees.

Relational Database

Primary Key

The Primary Key constraint uniquely identifies each record in a database table.

Primary keys must contain UNIQUE values, and cannot contain NULL values.

A table can have only one primary key, which may consist of single or multiple fields.

C.R.U.D.

Common database actions can be summarized as:

Create – Add new records

Read – Access existing records

Update – Edit existing records

Delete – Remove records

Data models

Building Interactive Software

Revision

1. What is a data model?
2. Describe cardinality between entities. Use examples.
3. What common fields should be included in a data dictionary?
4. What does C.R.U.D. stand for?

Data Model

`models.py`

This file is located in the App's folder, and is created when the Django App is first initialized.

In this file we define the Class (or classes) that will represent our data model.

These are then used by Django when interfacing with the database to access our information.

Data Model

```python
from django.db import models

# Create your models here.
class Widget(models.Model):

    ### The id field is automatically added if we do not specify a primary_key.
    # id = models.AutoField(primary_key=True)
    part_number = models.CharField(max_length=6)
    description = models.CharField(max_length=200)
    price = models.DecimalField(max_digits=9, decimal_places=2)
    date_added = models.DateTimeField(auto_now_add=True)

    def __str__(self):
        """Return a string representation of the model."""
        return str('%s: %s ~~~ $%.2f'
                   % (self.part_number, self.description[0:35], self.price)
                   )

```

Data Model

Django Field **Types**

https://docs.djangoproject.com/en/4.1/ref/models/fields/#field-types

BooleanField

CharField

DateField

DecimalField

EmailField

IntegerField

TextField

URLField

Data Model

Django Field **Options**

https://docs.djangoproject.com/en/4.1/ref/models/fields/#field-options

`null` – Allows the database to store a null value in the field.

`blank` – Allows for a blank or empty value to be stored in the database.

Data Model

```
6       ### The id field is automatically added if we do not specify a primary_key.
7       # id = models.AutoField(primary_key=True)
8       part_number = models.CharField(max_length=6,
9           unique=True, help_text='Unique part number for reordering.')
10
11      description = models.CharField(max_length=200,
12          help_text='Description for this product.')
13
14      price = models.DecimalField(max_digits=9, decimal_places=2,
15          default=9999.99, help_text='Tax inclusive price per each.')
16
17      date_added = models.DateTimeField(auto_now_add=True,
18          help_text='Date item was first created.')
19
```

Data Model

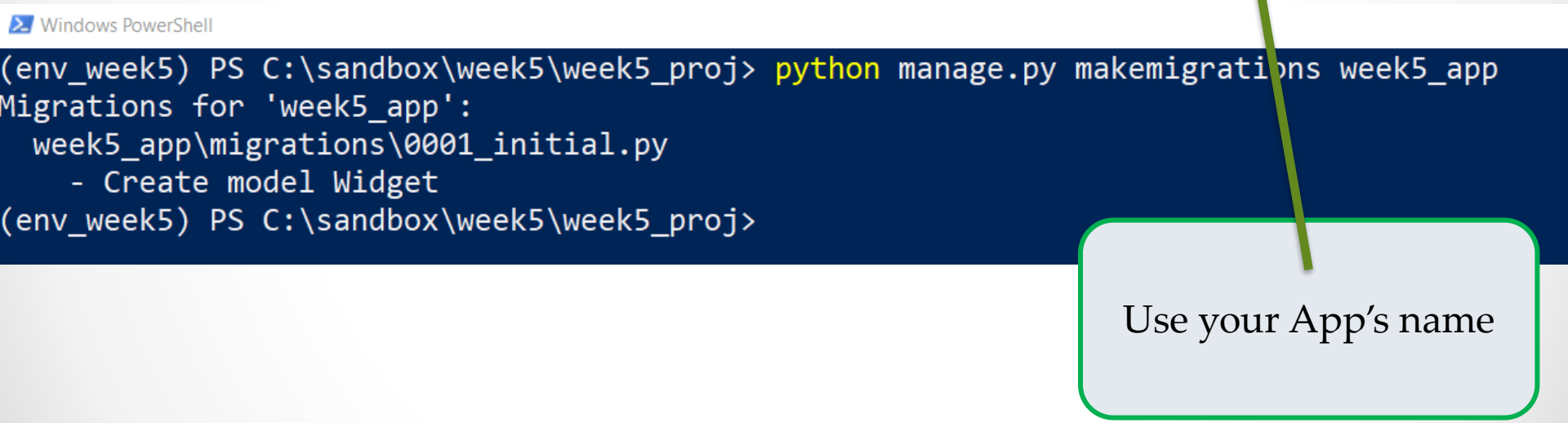
After we have defined a valid Django model, we need to let the database know about it.

Whenever we create or update a data model in our project, we must repeat the follow process. This allows the database to remain in sync with our data model.

Data Model

From the command prompt, run the following:

```
python manage.py makemigrations week5_app
```



Data Model

The `makemigrations` command tells Django to examine the App name provided, and prepare the required database commands for any changes in the data model.

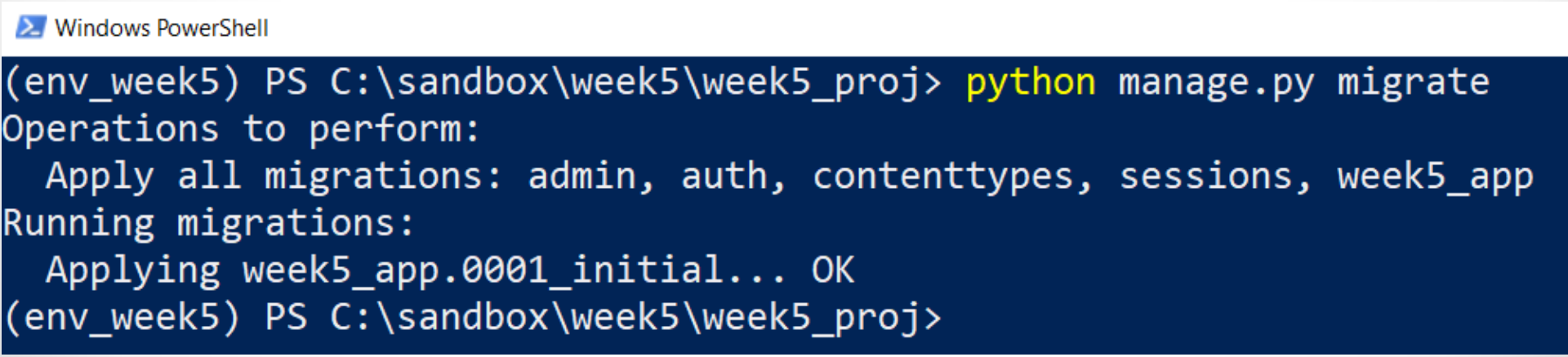
That includes adding or removing tables, or adjusting existing tables.

However, the changes are not automatically applied to the database by this command.

Data Model

To apply the prepared changes to the database, run the following command:

```
python manage.py migrate
```



Data Model

Practice

Perform the `makemigrations` and `migrate` commands on your App.

Once done, your App's data model will be setup in the database ready for use. However, we don't have any data yet.

Admin App

The Admin App is created by Django when we first establish our project. We can use the Admin App to view and edit the entries in the database for our newly created data model.

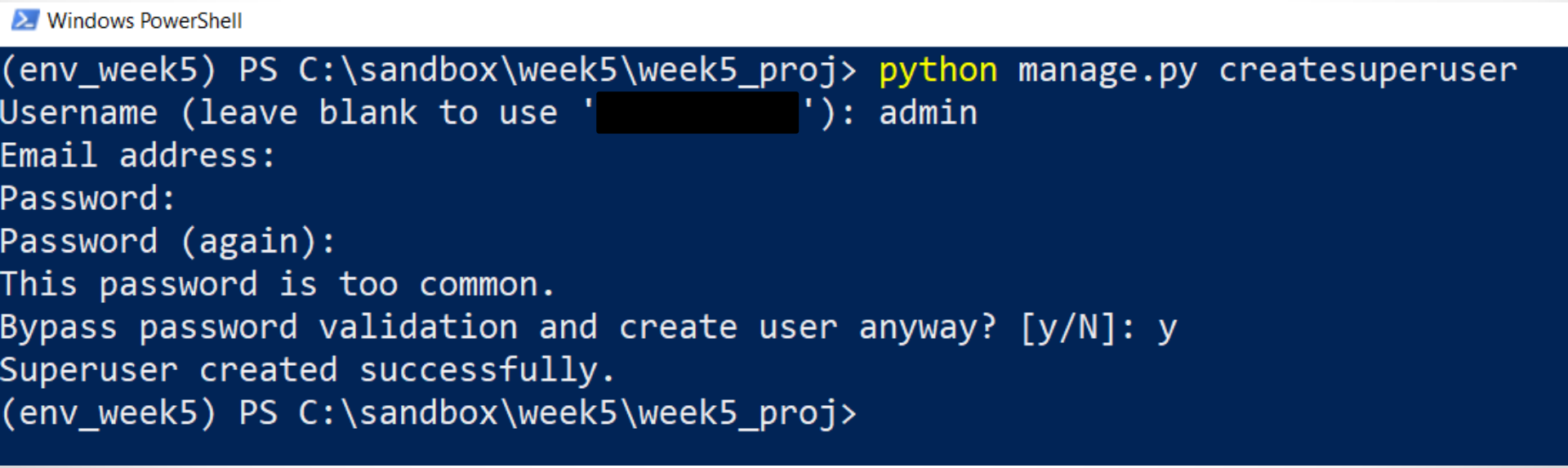
To do this, we need to:

1. Create a superuser to access the Admin App.
2. Register our data model with the Admin App.

Admin App

To create the superuser, run the following command:

```
python manage.py createsuperuser
```



Admin App

In your App's directory locate and edit the file called "`admin.py`". Apply the changes below.

Your App's name

```
from django.contrib import admin
from week5_app.models import Widget

# Register your models here.
admin.site.register(Widget)
```

Your class name

Admin App

Practice

Start your Django server and navigate your web browser to:

http://localhost:8000/admin

Log in with the newly created superuser account, and add some new customer data to your database.

Data Model

```python
from django.db import models

# Create your models here.
class Widget(models.Model):

    ### The id field is automatically added if we do not specify a primary_key.
    # id = models.AutoField(primary_key=True)
    part_number = models.CharField(max_length=6)
    description = models.CharField(max_length=200)
    price = models.DecimalField(max_digits=9, decimal_places=2)
    date_added = models.DateTimeField(auto_now_add=True)

    def __str__(self):
        """Return a string representation of the model."""
        return str('%s: %s ~~~ $%.2f'
                   % (self.part_number, self.description[0:35], self.price)
                   )

```

We will re-use this simplified version of the Widget class for the following slides.

Retrieving Data

Get a single record:

```
def widget_details(request, get_id):
    result = Widget.objects.get(id=get_id)
    page_data = {'widget': result}
    return render(request,
                  'widget_details.html',
                  page_data)
```

Retrieving Data

Get all records, sorted by column:

```
def all_widgets(request):
    results = Widget.objects.order_by('price')
    page_data = {'list_widgets': results}
    return render(request,
                  'all_widgets.html',
                  page_data)
```

Retrieving Data

Filter records by column:

```
Widget.objects.filter(
    part_number='POP079')
```

```
Widget.objects.filter(
    part_number__contains='POP')
```

Readings

Be sure to read through this entire page of the documentation. You don't need to understand it all just yet, but you will be coming back here.

https://docs.djangoproject.com/en/4.1/ref/models/fields/

Get a general understanding of what is available within Django's Data Models.

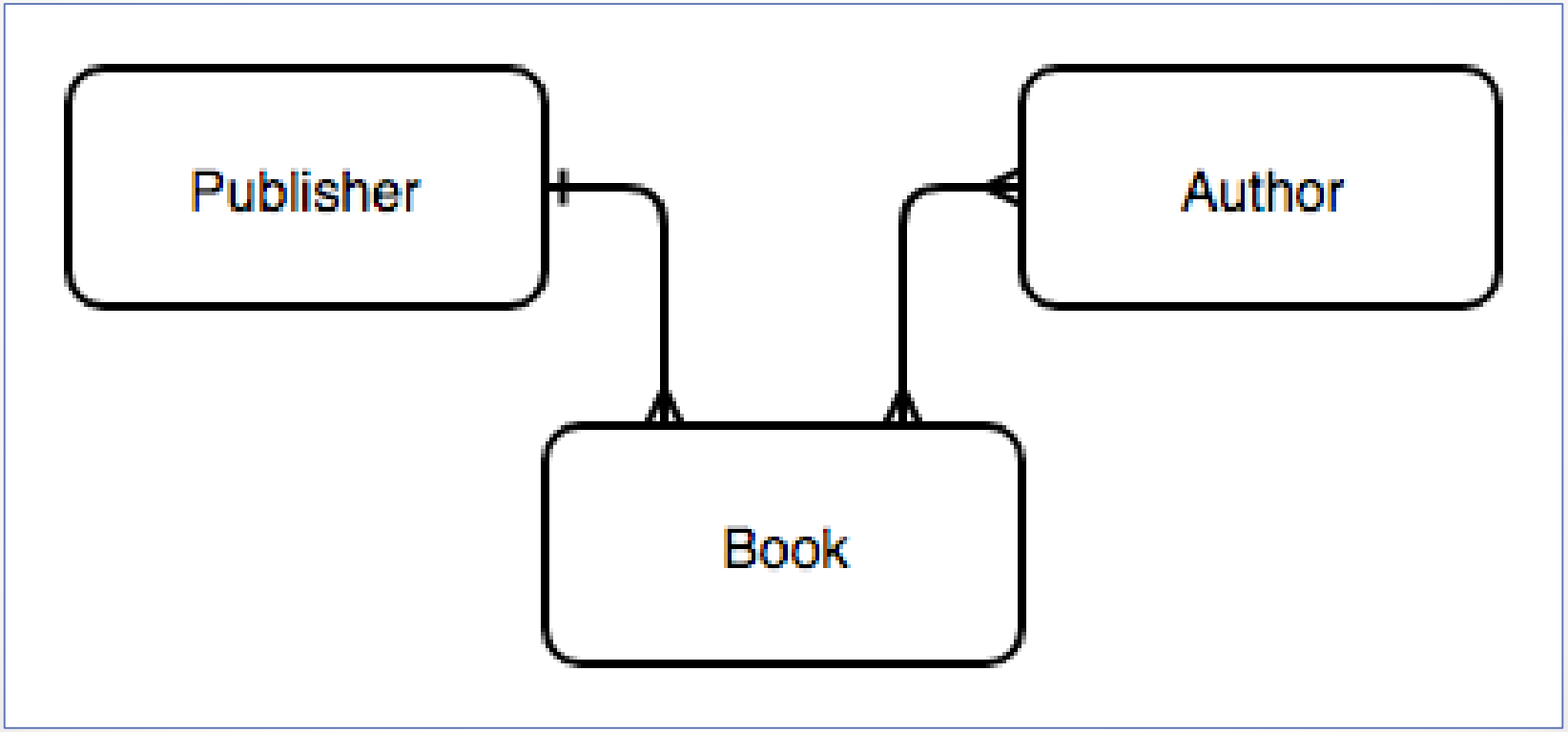
Data models continued

Building Interactive Software

Revision

1. Where do we define the class(es) for our data model?
2. What is the difference between CharField and TextField?
3. What are the commands to prepare changes to the data model and implement them to the database?

Data Model



Data Model

models.py

```
class Publisher(models.Model):
    name = models.CharField(max_length=30)
    address = models.CharField(max_length=50)
    city = models.CharField(max_length=60)
    state_province = models.CharField(max_length=30)
    country = models.CharField(max_length=50)
    website = models.URLField()

    def __str__(self):
        return self.name
```

Data Model

models.py (continued)

```
class Author(models.Model):
    first_name = models.CharField(max_length=30)
    last_name = models.CharField(max_length=40)
    email = models.EmailField()

    def __str__(self):
        return u'%s %s'
                % (self.first_name, self.last_name)
```

Data Model

models.py (continued)

```
class Book(models.Model):
    title = models.CharField(max_length=100)
    authors = models.ManyToManyField(Author)
    publisher = models.ForeignKey(
                        Publisher,
                        on_delete=models.CASCADE
                )
    publication_date = models.DateField()

    def __str__(self):
        return str(self.id) + " : " + self.title
```

Data Model

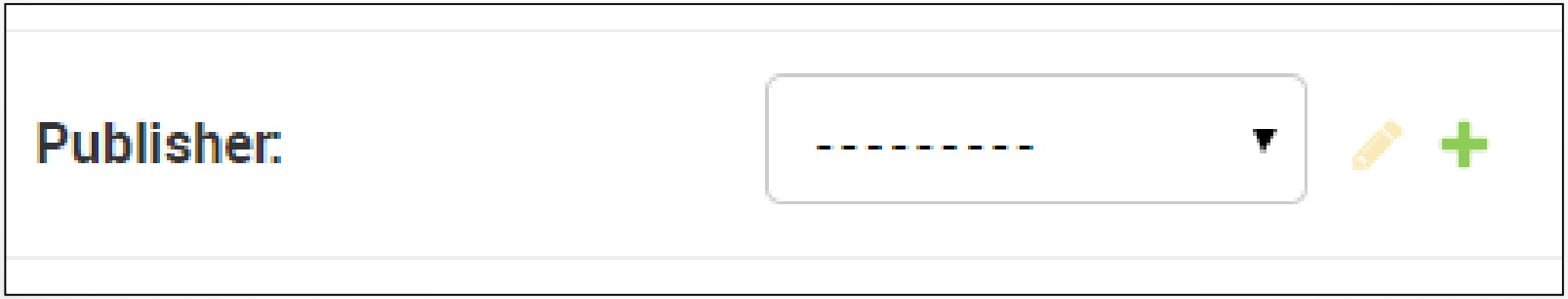
Practice

Setup your project and the data model from the previous slide.

Remember to implement the data model changes to the database, and register your classes with the Admin App.

Admin App

```
publisher = models.ForeignKey(
                Publisher,
                on_delete=models.CASCADE
            )
```



Admin App

```
authors = models.ManyToManyField(Author)
```

Adding Data

views.py

```
def add_publisher(request):
    pub = Publisher(
        name = 'XYZ Inc.',
        address = '101 There Way',
        city = 'Small Town',
        state_province = 'VIC',
        country = 'Australia',
        website = 'N/A'
    )
# At this point, pub is not saved to the database yet!
    pub.save()
# Now it is.
    return render(request, 'done.html')
```

Adding Data

views.py

```
def add_author(request):
    auth = Author(
        first_name = 'John',
        last_name = 'Smith',
        email = 'smith@yahoo.com'
    )
# At this point, auth is not saved to the database yet!
    auth.save()
# Now it is.
    return render(request, 'done.html')
```

Adding Data

Practice

Create some urls.py entries to link to your view functions. Then add some publishers and authors to your database using your views functions.

You will need to manually change the contents of your strings in the view functions for now.

Adding Data

views.py

```
def add_book(request):
    pub = Publisher.objects.get(name='XYZ Inc.')
    new_book = Book(
        title = 'Beyond the Rear Lines',
        publisher = pub,
        publication_date = datetime.datetime.now()
    )
    new_book.save()
# Next section links the many-to-many relationship
    list_authors = Author.objects.all()
    for auth in list_authors:
        new_book.authors.add(auth)
    return render(request, 'done.html')
```

Adding Data

Practice

Create some urls.py entries to link to your view functions. Then add a Book (instead of linking all available Authors to the Book as I did, try just one Author by last_name).

You will need to manually change the contents of your strings in the view functions for now.

Updating Data

There are 2 ways to update data on an existing database record. The first is to modify the desired attribute and then save() the object.

```
pub = Publisher.objects.get(id=1)
pub.city = 'Snailsville'
pub.save()
```

This way of updating the data causes every field to be re-saved to the database. This may cause problems in a multi-user environment.

Updating Data

The second way is preferred. Perform the selection of the record and the update of the column all as one action.

```
Publisher.objects.filter(id=1).update(city='Snailsville')
```

This way of updating the data causes only the specific field to be modified, and is more efficient. This will also not be prone to issues in multi-user environments.

Deleting Data

To delete a record from the database, select the record as usual and call the delete() method on it.

```
pub = Publisher.objects.filter(id=1)
pub.delete()
```

You can also delete ALL records in a table by:

```
Publisher.objects.all().delete()
```

Manipulating Data

Practice

Create some view functions to practice updating and deleting your Publishers, Authors and/or Books.

Readings

Read the documentation section for
Relationship fields

Go To Documentation – URL is in the slide notes if the hyperlink does not work for you.

The section ends once you reach Field API reference

HttpRequest and Django Forms

Building Interactive Software

Revision

1. How do we create a new record and add it to the database?
2. How do we update an existing record in the database?
3. How do we delete a record from the database?

HttpRequest Object

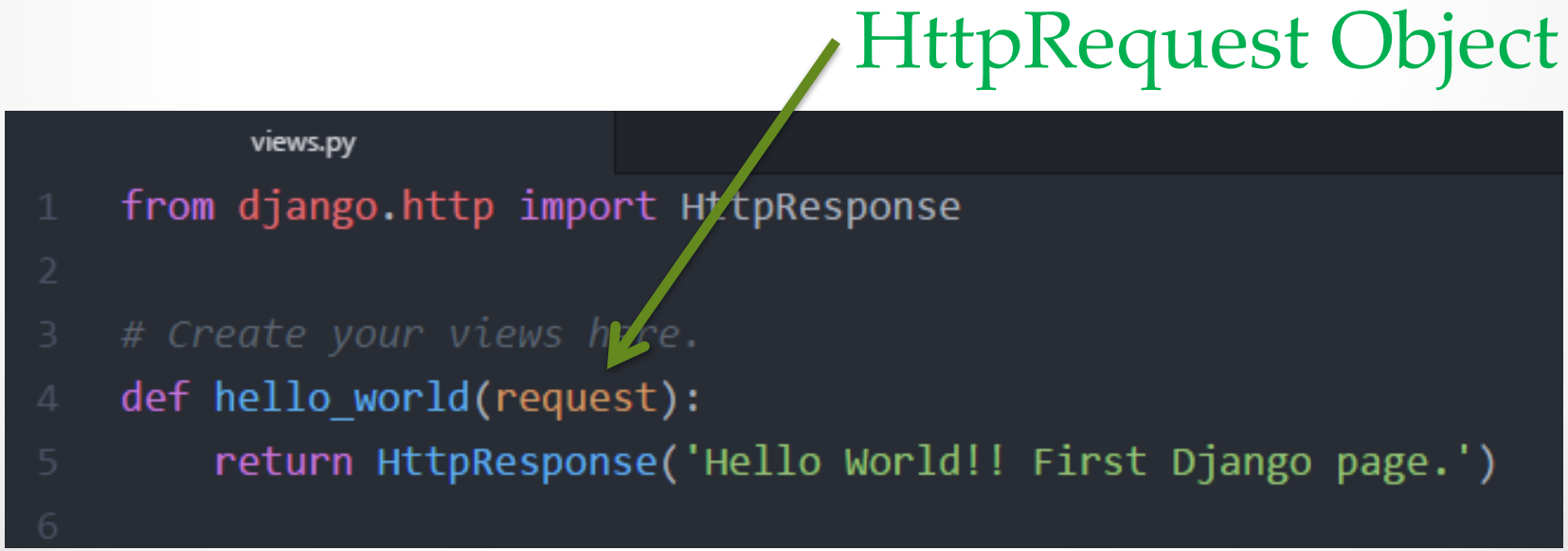
The HttpRequest object is provided automatically by Django to any view function you create.

This object contains information relevant to the current request being made to your Django application.

Each time the browser is directed to a web page; this is a distinct request to the server.

HttpRequest Object

Using the basic HelloWorld example, we highlight the request object being supplied by Django.



HttpRequest Object

Django passes the HttpRequest object as the first parameter in the view function. This variable can be called any valid variable name.

By convention we use the name 'request' so we know what the variable holds.

HttpRequest Object

The HttpRequest Object contains useful data that can be accessed by your application.

This data is sent from the web client that is making the request of the server, and can be thought of as the client's way of suppling information to the server.

HttpRequest Object

Attribute/method	Description	Example
`request.path`	The full path, not including the domain but including the leading slash.	`"/hello/"`
`request.get_host()`	The host (i.e., the "domain," in common parlance).	`"127.0.0.1:8000"` or `"www.example.com"`
`request.get_full_path()`	The path, plus a query string (if available).	`"/hello/?print=true"`
`request.is_secure()`	`True` if the request was made via HTTPS. Otherwise, `False`.	`True` or `False`

HttpRequest Object

The HttpRequest object permits the identification of the exact URL that was requested by the client. Consider the URL paths below.

```
urlpatterns = [
    path('admin/', admin.site.urls),
    path('', views.home, name='homepage'),
    path('sales/', views.list_sales, name='list_sales'),
    path('megasales/', views.list_sales, name='super_sales'),
    path('bargainsales/', views.list_sales, name='bargain_sales'),
]
```

Three paths lead to the same view function.

HttpRequest Object

Practice

Create a project which has multiple URL Paths that point to the same view function.

In the view function, use the HttpRequest object to determine which path was used to access the page and display the result.

HttpRequest Object

The HttpRequest object also holds information about any data that the client may have submitted for the server to act upon.

```
request.GET
request.POST
```

For the more information read:

https://code.djangoproject.com/wiki/HttpRequest

HttpRequest.GET

Data submitted through the GET method, could be data from a HTML form or from the query string parameters in a URL like the example below.

/library/books/?search=IoT

The GET method is normally used to make requests for information, not to supply information.

HttpRequest.POST

Data submitted through the POST method comes from HTML forms. The POST method is used when the process is intended to change a resource on the server, such as add a new record or update a record.

The POST method is generally thought of as a little more secure, as the data is not visible within the URL path.

HTML Form

A HTML Template containing a simple form for adding a new Author, using the POST method.

```
<body>
    <form action="/add/author/done/" method="post">
        {% csrf_token %}
        First name:<br>
        <input type="text" name="first_name"><br>
        Last name:<br>
        <input type="text" name="last_name"><br>
        Email:<br>
        <input type="email" name="email"><br><br>
        <input type="submit" value="Add Author">
    </form>
</body>
```

The csrf_token Tag

Django's built in security includes the CSRF middleware, which is enabled by default. It is designed to prevent Cross Site Request Forgery.

The csrf_token tag allows Django to verify that the form being submitted was generated from your own website.

```
<body>
    <form action="/add/author/done/" method="post">
        {% csrf_token %}
```

HttpRequest Data

To access the data submitted by the client we can access the Dictionary-like objects for GET and POST. Both GET and POST are accessed in the same manner.

These objects are not Python Dictionary objects but behave similarly, with similar functionality. As such we can treat them like a Dictionary.

HttpRequest Data

Getting data from the request object

```
49  def add_author_submit(request):
50      fname = lname = e_mail = ''
51      if 'first_name' in request.POST:
52          fname = request.POST['first_name']
53      if 'last_name' in request.POST:
54          lname = request.POST['last_name']
55      if 'email' in request.POST:
56          e_mail = request.POST['email']
57      page_data = {
58          'given_name' : fname,
59          'surname' : lname,
60          'address' : e_mail
61      }
62      return render(request, 'done.html', page_data)
```

HttpRequest Data

Because the data within the GET and POST objects are being supplied by the client, we have no control over what is being provided. Always check to make sure the value exists before using it.

```
51          if 'first_name' in request.POST:
52              fname = request.POST['first_name']
```

HttpRequest Data

Practice

Create a HTML Template that contains a basic HTML form. Make the associated URL Paths and view functions required to display the form, and the path for the submit action.

When the form is submitted, collect the data and display it back to the user on a webpage.

Django Forms

Django provides built in functionality for supporting the processing of forms and form validation.

To access this functionality, we first import the library and then create our own custom form class by extending from the `forms.Form` class.

Django Forms

Create a new file for your forms, called:

`forms.py`

```
1  from django import forms
2
3  class PublisherForm(forms.Form):
4      name = forms.CharField(max_length=30)
5      address = forms.CharField(max_length=50)
6      city = forms.CharField(max_length=60)
7      state_province = forms.CharField(max_length=30)
8      country = forms.CharField(max_length=50)
9      website = forms.URLField()
```

Django Forms

In `views.py`: import `forms.py` and pass an instance of your new custom form class to the HTML Template using the page context.

```
33  def add_publisher(request):
34      page_data = {'my_form' : PublisherForm(), }
35      return render(request, 'add_publisher.html', page_data)
```

Django Forms

In the HTML Template, use the `as_table` attribute of the Form class to get suitably formatted HTML output of the custom form you designed.

```
<body>
    <form action="/add/publisher/done/" method="post" novalidate>
        {% csrf_token %}
        <table>
            {{ my_form.as_table }}
        </table>
        <input type="submit" value="Add Publisher">
    </form>
</body>
```

Django Forms

Practice

Create your own custom form and get it to display in a HTML Template.

Django Forms

Django forms include built-in validation, since we defined the field types and some optional parameters when the class was specified.

When the form is submit by the user, we can test the `is_valid()` method of the form to confirm if the validation was successful or not.

Django Forms

If the request method is not POST then the form was not submitted by the user and we got here by mistake.

```
def add_publisher_submit(request):
    if request.method != 'POST':
        return HttpResponseRedirect('/add/publisher/')
    else:
        page_data = {}
        form = PublisherForm(request.POST)
        if form.is_valid():
            cd = form.cleaned_data
            # do something with the data
            page_data = {'form_input': cd, }
        else:
            page_data = {'val_errors': form.errors, }

    return render(request, app_name + 'done.html', page_data)
```

Django Forms

One method of viewing the validation errors is to iterate over the key/value pairs in the Form class' `errors` attribute.

```
<!DOCTYPE html>
<html lang="en" dir="ltr">
  <head>
    <meta charset="utf-8">
    <title>Add data</title>
  </head>
  <body>
    {% if val_errors %}
      <h1>Validation errors</h1>
      <ul>
        {% for key, value in val_errors.items %}
        <li>{{ key }} : {{ value }}</li>
        {% endfor %}
      </ul>
    {% elif form_input %}
      <h1>Done!</h1>
      {% for key, value in form_input.items %}
        <li>{{ key }} : {{ value }}</li>
        {% endfor %}
    {% endif %}
  </body>
</html>
```

Manipulating Data

Practice

Create the URL Path and view function for the HTML form's action path. Test the Django form and display the form values.

Try out some of the other built-in field types from:
https://docs.djangoproject.com/en/4.1/ref/forms/fields/#built-in-field-classes

Django Forms with Models

Building Interactive Software

Revision

1. Consider GET and POST; when would you choose one method over the other?
2. If a form is submit using the GET method and contains a 'name' variable, how would your view function access that data?
3. What steps should be taken to implement your own custom Django form?

Forms & Models

In addition to the `forms.Form` class, Django also provides the `forms.ModelForm` class.

The `ModelForm` class allows us to readily construct a form that is matched to our model class that we use to interact with the database. That include the field types and validation criteria.

Forms & Models

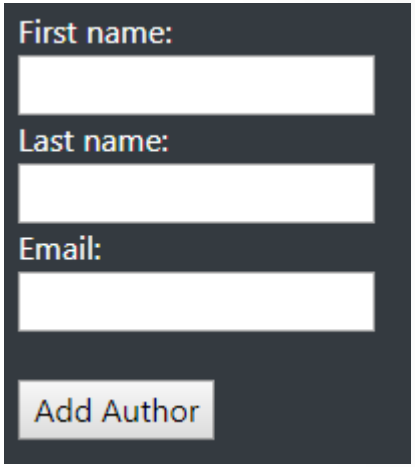
```python
from django import forms
from .models import *

class PublisherForm(forms.ModelForm):
    class Meta:
        model = Publisher
        fields = ['name', 'address', 'city', 'state_province', 'country', 'website']
        labels = {
            'address' : 'Postal Address',
            'state_province' : 'State or Province'
            }
```

Notice the label names on the form.

Forms & Models

```
13  class AuthorForm(forms.ModelForm):
14      class Meta:
15          model = Author
16          exclude = []
```



You must specify <u>either</u> the `fields` to include in the form, or the fields to `exclude` from the form. Either option is acceptable. Here I am excluding an empty array, so fields are included on the form.

Forms & Models

`ModelForm` is passed to the context the same as a normal `Form` (refer to last weeks notes).

```
def add_publisher(request):

    page_data = {'myform': PublisherForm(), }

    return render(request, app_name + 'add_publisher.html', page_data)
```

```
def add_publisher_submit(request):
    if request.method != 'POST':
        return HttpResponseRedirect('/add/publisher/')
    else:
        page_data = {}
        form = PublisherForm(request.POST)
        if form.is_valid():
            save_new_publisher(form)
            return HttpResponseRedirect(reverse('homepage'))
        else:
            page_data = {'val_errors': form.errors, }

    return render(request, app_name + 'done.html', page_data)
```

Forms & Models

The HTML Template can display a regular `forms.Form` class and a `forms.ModelForm` class the same way.

```
<body>
  <form action="/add/publisher/done/" method="post" novalidate>
    {% csrf_token %}
    <table>
      {{ myform.as_table }}
    </table>
    <input type="submit" value="Add Publisher">
  </form>
```

HTML Forms

W3Schools has a very good set of resources about HTML, including a page dedicated to the HTML <form> tag.

If you are unsure about what is happening with the HTML forms, I recommend you read:

https://www.w3schools.com/html/html_forms.asp

Forms & Models

Practice

Create a ModelForm class which uses one of the models you have associated with your database.

Display the form to the screen and test it.

Forms & Models

Invalid form messages can be handled by the template as follows:

```
<body>
  <h1>Done!</h1>
  {% if val_errors %}
    <h3>Validation Errors</h3>
    <ul>
      {% for key, value in val_errors.items %}
      <li>{{ key }} : {{ value }}</li>
      {% endfor %}
    </ul>
  {% endif %}
</body>
```

Forms & Models

We can also redirect the client's web browser to a new URL using the `reverse` function on the name associated with a URL Path in the `urls.py` file.

```
3   from django.http import HttpResponseRedirect
4   from django.urls import reverse
```

```
45              # ... doing something with the data.
46              return HttpResponseRedirect(reverse('homepage'))
```

Forms & Models

Let's save the data now.We could treat the form as ifit were a regular `forms.Form`

```
48  def save_new_publisher(form):
49      data = form.cleaned_data
50      if 'name' in data:
51          f_name = data['name']
52      if 'address' in data:
53          f_address = data['address']
54      if 'city' in data:
55          f_city = data['city']
56      if 'state_province' in data:
57          f_state_province = data['state_province']
58      if 'country' in data:
59          f_country = data['country']
60      if 'website' in data:
61          f_website = data['website']
62      pub = Publisher(
63          name = f_name,
64          address = f_address,
65          city = f_city,
66          state_province = f_state_province,
67          country = f_country,
68          website = f_website
69      )
70      pub.save()
```

Although, there is an easier way as shown on the next page.

Forms & Models

We can use `forms.ModelForm.save()`

```
else:
    # Validation passed, use the data.
    save_new_publisher(form)
    return HttpResponseRedirect(reverse('homepage'))
```

```
48  def save_new_publisher(form):
49      # This will create a NEW RECORD in the database.
50      new_pub_object = form.save()
```

This creates a *NEW* record in the database.

Forms & Models

Practice

Finish your view function so that when the form is submitted, a new record is saved into your database.

Experiment with a 'success' page that shows the newly created record.

Forms & Models

To update an existing record in the database we must associate the record with the form when we create it from the POST data.

Then the `save()` method will update the existing record, not create another.

Use this Publisher object for the form.

```
def edit_publisher(request, key=1):
    pub = Publisher.objects.get(id=int(key))

    if request.method == 'POST':
        # do something
        form = PublisherForm(request.POST, instance=pub)
        if form.is_valid() != True:
```

Forms & Models

To display an existing record to the screen, we also use the same syntax, and associate the instance of the Publisher object when we create the form.

This variable was defined on the previous slide.

```
else:
    # This request is not generated by a form's Submit button.
    # Display the Form containing the queried record.
    form = PublisherForm(instance=pub)
    page_data = {'myform': form,}
```

Forms & Models

Updating a record.

```
def edit_publisher(request, key=1):
    pub = Publisher.objects.get(id=int(key))

    if request.method == 'POST':
        # do something
        form = PublisherForm(request.POST, instance=pub)
        if form.is_valid() != True:
            # Validation failed. Redisplay the form.
            page_data = {'myform': form, }
        else:
            # Validation passed. Save the data.
            form.save()
            return HttpResponseRedirect(reverse('homepage'))
    else:
        # This request is not generated by a form's Submit button.
        # Display the Form containing the queried record.
        form = PublisherForm(instance=pub)
        page_data = {'myform': form,}

    return render(request, app_name + 'edit_publisher.html', page_data)
```

Forms & Models

Practice

Create a new view function that allows you to edit a record from your database and save the changes.