Django Web Framework



[Introduction to Django Web Framework | Coursera](https://www.coursera.org/learn/django-web-framework/lecture/X7bGg/introduction-to-django-web-framework) 3-27-2024 0451

**Short Summary:** This video introduces the Django web framework, outlining its purpose and the concepts you'll learn in the course. It also explains the course structure, including modules on views, models, templates, and a final project applying Django skills.

* **Acronyms**
  + HTML (HyperText Markup Language)
  + CSS (Cascading Style Sheets)
  + HTTP (HyperText Transfer Protocol)
  + MVT (Model View Template)
* **Technical Words**
  + Backend (Server-side components of a web application responsible for logic and data storage)
  + Frontend (The user interface of a web application, including HTML, CSS, and JavaScript)
  + Framework (A structured collection of tools and libraries designed to simplify development)
  + Open-source (Software with freely available source code that can be modified and redistributed)
  + Database (A structured collection of data)
  + Web application (Software that runs on a web server and can be accessed through a web browser)
* **Software Packages**
  + Django (A Python-based web framework for building web applications)
  + MySQL (A popular open-source database management system)
* **Commands/Modules**
  + django-admin (A command-line tool for creating Django projects and apps)
  + manage.py (A script for various Django project management tasks)
  + QuerySet API (Django's way of interacting with models to query the database)
  + HTTP methods (GET, POST, PUT, DELETE - actions that can be performed on web resources)
* **Ideas/Techniques**
  + Model-View-Template (MVT) Design Pattern: A software design pattern that separates application concerns into interconnected parts for better organization.
  + Code reusability (The principle of writing code that can be used in multiple places, reducing redundancy)
  + Regular expressions (A pattern matching language used to define search criteria)
  + Data-driven web application (An application whose content and functionality are largely driven by data stored in a database)
* **Mistakes or Misleading Information**
  + While the video is generally a good introduction, it would be beneficial to emphasize that Django is specifically a *backend* framework. It's important to understand that frontend technologies like HTML, CSS, and JavaScript are still required in conjunction with Django.

**Written by Gemini**

[What is Django? | Coursera](https://www.coursera.org/learn/django-web-framework/lecture/rvUTq/what-is-django)

**Short Summary:** This video introduces the Django web framework, explaining its purpose, advantages, and real-world applications. It emphasizes Django's suitability for large-scale, content-heavy, and scalable web projects across diverse industries.

**Detailed Breakdown**

* **Acronyms**
  + API (Application Programming Interface)
  + RPC (Remote Procedure Call)
  + OTT (Over The Top)
  + SaaS (Software as a Service)
* **Technical Words**
  + Web application framework (A software toolkit that provides pre-built components for web development)
  + Open-source (Software with publicly available code that can be modified and shared)
  + Scalability (The ability of a system to handle increased workload and growth)
  + Asynchronous (Processes that run independently without waiting for each other to complete)
  + Fault Tolerance (The ability of a system to continue operating even if some components fail)
  + Django (A Python-based web framework)
* **Mistakes or Misleading Information**
  + While Django is excellent for various applications, it's important to note that it may not always be the optimal choice for smaller projects where a simpler framework might be more suitable.

**Written by Gemini**

[How is Django used in the real world? | Coursera](https://www.coursera.org/learn/django-web-framework/lecture/2Ibnl/how-is-django-used-in-the-real-world) 3-27-2024 0510

[Installing Python (optional for Windows users) | Coursera](https://www.coursera.org/learn/django-web-framework/supplement/NFk5o/installing-python-optional-for-windows-users) 3-27-2024 0510

[Installing Python (optional for Mac users) | Coursera](https://www.coursera.org/learn/django-web-framework/supplement/uQbqk/installing-python-optional-for-mac-users) 3-27-2024 0520

[Setting up a project in VS Code | Coursera](https://www.coursera.org/learn/django-web-framework/supplement/OdgJH/setting-up-a-project-in-vs-code) 3-27-2024 0520

[Working with virtual environments on your local machine | Coursera](https://www.coursera.org/learn/django-web-framework/supplement/rZlSl/working-with-virtual-environments-on-your-local-machine) 3-27-2024 0520

Summary: This lesson provides a comprehensive guide on setting up and managing virtual environments for Python projects, specifically focusing on Django development. It covers the installation of pip, the creation, activation, and deactivation of virtual environments using the venv module, and mentions virtualenv as an alternative for earlier Python versions.

**Acronyms**

* **pip**: Python Package Installer
* **venv**: Virtual Environment
* **virtualenv**: Virtual Environment for older versions of Python

**Technical Words with Definition**

* **Virtual Environment**: A self-contained directory tree that contains a Python installation for a particular version of Python, plus a number of additional packages.
* **Django**: A high-level Python web framework that encourages rapid development and clean, pragmatic design.

**Commands/Modules with Definition**

* **pip install --upgrade pip**: Upgrades pip to the latest version.
* **python3 -m pip install --user virtualenv**: Installs virtualenv for managing virtual environments in Python 2.
* **python3 -m venv env**: Creates a virtual environment named 'env'.
* **.\env\Scripts\activate** (Windows) or **source env/bin/activate** (MacOS): Activates the virtual environment.
* **deactivate**: Exits the virtual environment.

**Ideas/Techniques with Definition**

* **Package Management**: The process of installing, upgrading, configuring, and removing software packages from a computer's operating system in a consistent manner.
* Upgrade pip:
* Windows: py -m pip install --upgrade pip
* MacOS: python3 -m pip install --user --upgrade pip
* Install virtualenv (if needed for earlier versions of Python):
* Windows: py -m pip install --user virtualenv
* MacOS: python3 -m pip install --user virtualenv
* Create a virtual environment in a specific project directory:
* Navigate to your project directory:
* cd mydirectory
* Then create the virtual environment named "env":
* Windows: py -m venv env
* MacOS: python3 -m venv env
* Activate the virtual environment:
* Windows: .\env\Scripts\activate
* MacOS: source env/bin/activate
* Deactivate the virtual environment:
* MacOS and Windows: deactivate
* keywords = ['pip', 'install', '--upgrade', 'python3', '-m', 'venv', 'env', 'activate', 'deactivate', 'virtualenv']

Written by ChatGPT

[Additional resources | Coursera](https://www.coursera.org/learn/django-web-framework/supplement/yqDlw/additional-resources) 3-27-2024 0526

* [Django official website](https://www.djangoproject.com/start/overview/)
* [Django documentation](https://docs.djangoproject.com/en/4.1/)
* [Installing VS Code on Mac - Official](https://code.visualstudio.com/docs/setup/mac)
* [Installing VS Code on Windows - Official](https://code.visualstudio.com/docs/setup/windows)
* [Django installation - Official](https://docs.djangoproject.com/en/4.1/topics/install/)
* [Install Django on Windows](https://docs.djangoproject.com/en/4.1/howto/windows/)
* [Setting up Virtual environment in Python - venv (Windows and MacOS)](https://docs.python.org/3/library/venv.html)

[Projects and Apps overview | Coursera](https://www.coursera.org/learn/django-web-framework/lecture/Q9w8b/projects-and-apps-overview) 3-27-2024 0545

**Short Summary:** This video explains Django's building blocks: projects and apps. Projects represent the entire web application, while apps provide modular, reusable components that implement specific features within the project.

**Detailed Breakdown**

* **Acronyms**
  + HTTP (Hypertext Transfer Protocol)
  + DRY (Don't Repeat Yourself)
* **Technical Words**
  + Web application (Software running on a web server, accessed through a browser)
  + Static website (Website with fixed content, no dynamic elements)
  + Framework (A structured foundation for software development)
  + Database (Structured collection of data)
  + Stateless (A system that doesn't store information between requests)
  + Configuration (Settings and options that control how software works)
  + Self-contained (Independent, not relying on external components)
  + Metadata (Data about data)
* **Software Packages**
  + Django (Python web framework)
* **Commands/Modules**
  + startproject (**django-admin startproject <projectname>)**
  + startapp (**manage.py startapp <appname>**)
* **Ideas/Techniques**
  + Modularity (Breaking down a system into smaller, self-contained parts)
  + Reusability (The ability to use code in multiple places)

**Mistakes or Misleading Information**

* The statement "Django comes with its own development server" could imply it's the only web server option. Django can be deployed on various production-grade web servers as well.

**Written by Gemini**

[Creating your first project | Coursera](https://www.coursera.org/learn/django-web-framework/lecture/BMLIm/creating-your-first-project) 3-27-2024 0920  
Short summary: This video explains how to create a Django project, set up a virtual environment, install Django, and launch the development server. Virtual environments are emphasized to keep project dependencies isolated.

**Acronyms**

* VS Code: Visual Studio Code (a popular code editor)

**Terms**

* Project: An organizational unit in Django, containing settings and database information.
* App: A smaller component within a Django project, focused on a specific feature.
* Terminal: A command-line interface for interacting with the computer.
* Directory: A way to organize files on a computer (also known as a 'folder').

**Technical Words**

* Django: A Python-based web framework for building web applications.
* Virtual environment: An isolated workspace for a project's dependencies.
* Python: A high-level programming language used for web development, data analysis, and more.
* Development server: A web server built into Django for testing applications during development.
* Command-line interface (CLI): A way to interact with a computer by typing commands.
* Dependency: A library or package that a project relies on to function.

**Rarely Used Words**

* Suffix: A group of characters added to the end of a word.

**Software Packages**

* Django: A Python-based web development framework.
* Python: A programming language.

**Commands**

* mkdir: (Make directory) Creates a new directory.
* cd: (Change directory) Navigates to a different directory.
* python3 -m venv [environment\_name]: Creates a new virtual environment.
* source [virtual\_environment\_path]/activate: Activates a virtual environment.
* pip3 install Django: Installs the Django package.
* python3 -m django --version: Checks the installed Django version.
* django-admin startproject [project\_name]: Creates a new Django project.
* ./manage.py runserver: Starts the Django development server.

**Modules**

* manage.py: A Django-specific command-line utility for interacting with a Django project.

**Ideas**

* **Project Structure:** Django applications are built with a clear structure of projects and apps.
* **Virtual Environments:** Best practice for keeping project dependencies separate and avoiding conflicts.
* **Development Workflow:** The process of creating a project, setting up a virtual environment, and using the development server.

**Techniques**

* Using the terminal or command-line interface.
* Creating and activating virtual environments.
* Installing Django.
* Creating a Django project.
* Running the development server.

**Keywords** ["Django", "project", "virtual environment", "development server", "terminal", "VS Code", "Python", "dependencies", "command-line interface", "web framework"]

This analysis was provided by Gemini, a large language model from Google AI.

Notes:

I tried “source tutorial/bin/activate as the video said. This works on Linux maybe or Mac, but not on windows in PowerShell. I used .\Scripts\activate.ps1 from inside the turorial-venv folder.

[Knowledge check: Projects and Apps | Coursera](https://www.coursera.org/learn/django-web-framework/quiz/pGBrD/knowledge-check-projects-and-apps) 3=27-2024 1015

* Easy quiz

[Django-admin & manage.py commands | Coursera](https://www.coursera.org/learn/django-web-framework/lecture/Vg7tL/django-admin-manage-py-commands) 3-27-2024 1027

Short summary: This video explains Django's administrative commands, django-admin and manage.py. Both manage Django projects, but manage.py is project-specific, while django-admin is for broader tasks, especially when working with multiple projects.

**Terms**

* Project: An organizational unit in Django containing settings and database information.
* Virtual environment: An isolated workspace for a project's dependencies.
* Terminal: A command-line interface for interacting with the computer.
* Directory: A way to organize files on a computer (also known as a 'folder').
* Script: A file containing a series of commands.

**Technical Words**

* Django: A Python-based web framework for building web applications.
* Python: A high-level programming language used for web development, data analysis, and more.
* Web server: Software that handles HTTP requests and serves web pages.
* Command-line utility: A program interacted with by typing commands rather than through a graphical interface.
* Environment variable: A dynamic value that affects how processes run on a computer.

**Rarely Used Words**

* Workflow: The sequence of steps involved in a process.
* Denote: To represent or indicate something.
* Explicitly: Stated clearly and in detail.
* Superuser: An administrative account with full system privileges (also known as "root").

**Software Packages**

* Django: A Python-based web development framework.
* Python: A programming language.

**Commands**

* django-admin startproject [project\_name]: Creates a new Django project.
* manage.py runserver: Starts the Django development server.
* django-admin (with various additional subcommands): Django's project-agnostic command-line tool.
* manage.py (with various additional subcommands): Django's project-specific command-line tool

**Modules**

* settings.py: A Django project's configuration file.

**Ideas**

* **Command-line Tools:** Django provides tools to streamline development tasks.
* **Project Management:** django-admin and manage.py offer different levels of project control.
* **Development Server:** A built-in server for testing during development (not suitable for production use).

**Keywords** ["Django", "project", "virtual environment", "manage.py", "django-admin", "development server", "command-line utility", "web framework"]

This analysis was provided by Gemini, a large language model from Google AI.

Note:

In the video it showed these two screens. They implied that you would avoid having two project directories IE a nested directory. But the images are identical. If you imagine the first image having an additional my\_site/ level then you understande.

A close up of a screen

Description automatically generated

A screen shot of a computer

Description automatically generated

Using **django-admin startproject my\_site** (Note there is no ‘.’ At the end)

my\_site/

manage.py

my\_site/

\_\_init\_\_.py

settings.py

urls.py

wsgi.py

Using **django-admin startproject my\_site .** (Note the ‘.’ At the end with a space before.)

manage.py

my\_site/

\_\_init\_\_.py

settings.py

urls.py

wsgi.py

[App structures | Coursera](https://www.coursera.org/learn/django-web-framework/supplement/rDp6w/app-structures)

Summary: This transcript explains how to create and configure a Django app within a project, detailing the file structure and the roles of various components such as views, URLs, and settings.

Acronyms:

* URL: Uniform Resource Locator, the address of a web page on the internet.

Terms:

* Django Project: A collection of configurations and apps for a particular website. A project can contain multiple apps.
* Django App: A sub-module within a Django project that performs a specific task or a set of tasks.
* View: In Django, a view is a Python function that takes a web request and returns a web response.
* URL Dispatcher: A system in Django that matches browser requests with the appropriate view based on the URL pattern.
* Data Model: A Python class that defines the structure of the data stored in a database.

Technical Words:

* manage.py: A command-line utility that lets you interact with your Django project in various ways.
* **init**.py: An empty file that tells Python that the directory should be considered a Python package.
* admin.py: A file within a Django app that is used to register models to include them in the Django admin interface.
* apps.py: A file that is used to configure the settings specific to the app it's contained within.
* models.py: A file that contains the data models of a Django app, representing the database schema.
* tests.py: A file used to write test cases for a Django app.
* views.py: A file that contains the view functions that handle requests and return responses.
* urlpatterns: A list of URL patterns that is used to match browser requests to the correct view function.

Rarely Used Words:

* Serialized: The process of converting an object into a format that can be stored or transmitted and then recreated.
* Migrated: In Django, it refers to the process of applying changes to the database schema based on changes in the data models.

Software Packages:

* Django: A high-level Python web framework that encourages rapid development and clean, pragmatic design.

Commands:

* **startproject**: A command used to create a new Django project with the necessary directory structure. For example, running **django-admin startproject mysite** would create a new Django project named **mysite**.
* **startapp**: A command used to create a new Django app within a project with a default set of files. For example, executing **python manage.py startapp polls** within a project directory would create a new app named **polls**.

Modules:

* django.http: A module in Django that contains classes like HttpResponse to handle HTTP responses.

Ideas:

* Reusability: In the context of Django apps, the idea that an app can be used in multiple projects without modification.

Techniques:

* URL Routing: The process of defining URL patterns that are matched with view functions to handle requests.

Keywords: ['Django', 'Project', 'App', 'View', 'URL Dispatcher', 'Data Model', 'manage.py', 'admin.py', 'apps.py', 'models.py', 'tests.py', 'views.py', 'urlpatterns', 'Serialized', 'Migrated', 'startproject', 'startapp', 'django.http', 'Reusability', 'URL Routing']

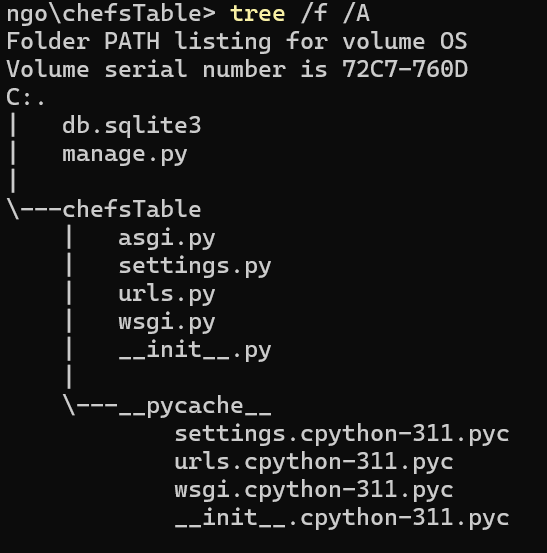
Written by ChatGPT.

A screenshot of a computer program

Description automatically generated

Not in this lesson, but you may find it helpful to view the directory structures. I use the tree command.

**tree /f /A**



I use PowerShell so the tree command is native, but for linux and Mac you will need to install it.

[App structures | Coursera](https://www.coursera.org/learn/django-web-framework/supplement/rDp6w/app-structures) 3-27-2024 1630

Summary: A Django project, structured as a collection of reusable apps, is designed to perform distinct tasks within the overall application. This reading explains the app structure, app creation, and configuration within a Django project.

Acronyms:

* URL: Uniform Resource Locator

Terms with Definitions:

* Django project: A web application that may consist of one or more sub-modules called apps, designed to perform distinct tasks.
* App: A sub-module within a Django project responsible for a single task, designed to be reusable across different projects.
* manage.py: A command-line utility that lets you interact with this Django project.
* Project package: A directory created by Django that contains the project’s configuration and apps.

Technical Words with Definitions:

* Folder structure: The hierarchical arrangement of files and folders.
* Python scripts: Files containing Python code that perform various tasks.
* View: A user-defined Python function in Django that receives web requests and returns web responses.
* URL dispatcher: A component of Django that directs incoming web requests to the appropriate view based on the request URL.
* URL pattern: A specification of URL paths that are matched with views.
* Data model: A Python class that defines the structure of data for database storage.
* Database tables: Structures within a database that store data in rows and columns.

Rarely Used Words with Definitions:

* Module: In the context of Django, a component or a set of features grouped together to perform a specific task within a project.

Software Packages with Definitions:

* Django: A high-level Python web framework that encourages rapid development and clean, pragmatic design.

Commands with Definitions:

* startproject: A Django command used to create a new Django project with the necessary directory structure.
* startapp: A Django command used to create a new application module within a Django project.

Modules with Definitions:

* HttpResponse: A module in Django for constructing HTTP responses in views.
* path: A function in Django used for routing URLs to the appropriate view functions.

Ideas with Definitions:

* Reusability: The ability to use a component or code in different systems or contexts without modification.
* Configuration: The process of setting up a software application based on its requirements and environment.

Techniques Described with Definitions:

* URL routing: The process of defining URL paths that the application responds to and linking them to their respective view functions.

Keywords: ["Django project", "app", "manage.py", "project package", "folder structure", "Python scripts", "view", "URL dispatcher", "URL pattern", "data model", "database tables", "module", "Django", "startproject", "startapp", "HttpResponse", "path", "reusability", "configuration", "URL routing"]

Written by ChatGPT

[Creating an App | Coursera](https://www.coursera.org/learn/django-web-framework/lecture/t5i8t/creating-an-app) 3-38-2024 0450

**Short Summary:** This video explains how to create a Django app, define a view function, and map it to a URL, enabling the display of basic text on a web page. It outlines the core process behind dynamic content generation in Django.

**Detailed Breakdown**

* **Acronyms**
  + HTTP (Hypertext Transfer Protocol)
* **Technical Words**
  + Django (Python-based web framework)
  + App (A self-contained module within a Django project)
  + View (A Python function that handles web requests and generates responses)
  + Route (A URL pattern that maps to a specific view)
  + URL Configuration (File that maps URLs to view functions)
  + Development Server (A server for testing Django projects locally)
* **Rarely Used Words**
  + Sub-module (A module within another module)
  + Intimidating (Causing nervousness or apprehension)
* **Software Packages**
  + Django (Python web framework)
* **Commands/Modules**
  + python manage.py startapp <appname>
  + python manage.py runserver
* **Ideas/Techniques**
  + Model-View-Controller (MVC) Design Pattern (Django loosely follows this pattern)
* **Keywords**
* **Python:** A high-level, general-purpose programming language known for its readability, versatility, and extensive libraries.
* **Django:** A powerful Python-based web framework that simplifies the development of complex web applications. It emphasizes rapid development and clean design.
* **app:** A self-contained module within a Django project that provides specific functionality. Apps help organize your project into manageable, reusable components.
* **view:** A Python function within a Django app that handles web requests. Views process user input, interact with models (if needed), and generate the appropriate HTTP responses to be displayed in the user's browser.
* **URL:** (Uniform Resource Locator) A unique address that identifies a specific web page or resource on the internet.
* **HTTP:** (Hypertext Transfer Protocol) The underlying protocol of the web. It defines how web browsers and web servers communicate to exchange requests and responses.

**Written by Gemini**

Notes by me:

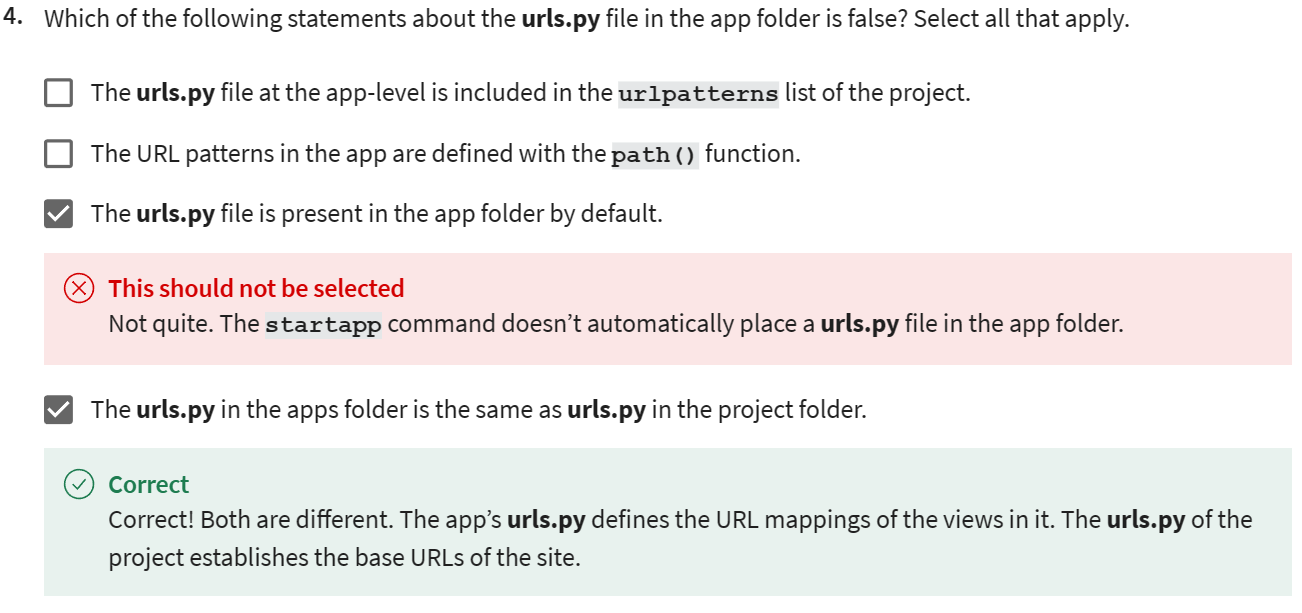
This lesson asked the learner to use **python -m Django startapp myapp** to start an app in the project. You will later transition to **python manage.py startapp myapp**, which is the correct way to do this. The instructor may be old school or it may just be an old video. Stick with manage.py method.

[Exercise: Creating your first project and app | Coursera](https://www.coursera.org/learn/django-web-framework/ungradedLab/a3cL9/exercise-creating-your-first-project-and-app) 3-28-2024 0503

* As you noticed in this exercise the course transitioned to using **python manage.py** **startapp name\_of\_app** to start an app.

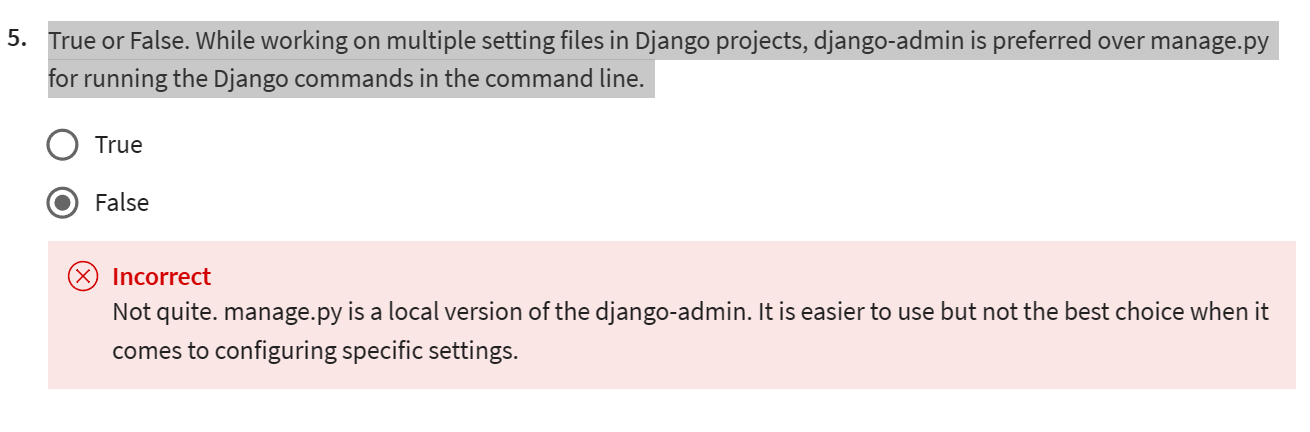
[Knowledge check: Applications | Coursera](https://www.coursera.org/learn/django-web-framework/quiz/4YoJl/knowledge-check-applications) 3-28-2024 0527

I had 80% the first time I did this test. Lets see what I get after running through the course a second time.



From your description, it sounds like there's a discrepancy between the correct answer and the feedback provided in the quiz system. In Django, when you create a new app using the **startapp** command, a **urls.py** file is not created in the app folder by default. You typically need to create this file manually if you want to define URL patterns specific to that app.<ChatGPT4>

You were absolutely correct in identifying that the statement "The urls.py file is present in the app folder by default" is false.<Gemini>



[What is a web framework? | Coursera](https://www.coursera.org/learn/django-web-framework/lecture/CNKkm/what-is-a-web-framework)

**Summary**: This video discusses the importance of web frameworks, with a particular focus on Django, a popular Python framework. It explains the concept of three-tier architecture in web application development, emphasizing Django's benefits, such as speed, security, scalability, and built-in features.

**Acronyms**

* **UI**: User Interface
* **RSS**: Really Simple Syndication

**Technical Words with Definition**

* **Web Framework**: A software framework that is designed to support the development of web applications, including web services, web resources, and web APIs.
* **Client-Server Architecture**: A computing model in which the server hosts, delivers, and manages most of the resources and services to be consumed by the client.

**Rarely Used Words with Definition**

* **Pragmatic**: Dealing with things sensibly and realistically in a way that is based on practical rather than theoretical considerations.

**Software Packages with Definition**

* **Django**: A high-level Python web framework that encourages rapid development and clean, pragmatic design.

**Commands/Modules with Definition**

* **django-admin**: A command-line utility that lets you perform administrative tasks in Django. Example: **django-admin startproject myproject**

**Ideas/Techniques with Definition**

* **Three-Tier Architecture**: A software architecture model that separates applications into three logical layers: presentation, application, and data storage.

keywords = ['Django', 'web framework', 'three-tier architecture', 'client-server', 'scalability', 'security']

Written by ChatGPT

[MVT Overview | Coursera](https://www.coursera.org/learn/django-web-framework/supplement/hpwXW/mvt-overview) 3-28-2024 0608

**Summary**: This reading explains Django's Model-View-Template (MVT) architecture, which is a variation of the Model-View-Controller (MVC) pattern, and is central to developing web applications with Django. It outlines the roles of each component within this framework, highlighting how it simplifies the web development process.

**Acronyms**

* **MVC**: Model-View-Controller
* **MVT**: Model-View-Template
* **ORM**: Object Relational Mapper
* **CRUD**: Create, Read, Update, Delete
* **HTML**: HyperText Markup Language

**Technical Words with Definition**

* **Web Framework**: A platform for developing web applications; it provides standard ways to build and deploy web applications on the internet.
* **API**: Application Programming Interface, a set of functions and procedures for creating software applications.

**Rarely Used Words with Definition**

* **Dispatcher**: In web frameworks, a component that routes web requests to the appropriate handler based on the request's URL.

**Software Packages with Definition**

* **Django**: A high-level Python web framework that encourages rapid development and clean, pragmatic design.

**Commands/Modules with Definition**

* **django.urls**: Django module for URL pattern declarations. Example usage in a Django **urls.py** file: **path('articles/', views.article\_list)**
* **django.http**: Django module that contains classes like **HttpResponse**. Example usage: **return HttpResponse("Hello, world.")**

**Ideas/Techniques with Definition**

* **Three-Tier Architecture**: A design framework that separates an application into three logical layers: presentation, logic, and data storage.

**keywords = ['Django', 'MVT', 'MVC', 'web framework', 'URL dispatcher', 'view', 'model', 'template', 'CRUD', 'ORM']**

**Mistakes or Misleading Information**

* The information provided appears to be accurate and aligns with Django's official documentation regarding its MVT architecture. However, the text should mention that while Django refers to its pattern as MVT, it's analogous to the MVC framework used in other languages, with the template in Django acting like the view in typical MVC frameworks, and Django's view more closely resembling the controller.

Written by ChatGPT

[MVT Example | Coursera](https://www.coursera.org/learn/django-web-framework/lecture/3HCub/mvt-example) 3-28-2024 1051

**Summary: This video introduces the MVT (Model-View-Template) architecture used in Django web development. It explains how Django separates data logic (models), presentation (views), and display styling (templates) for efficient development of dynamic web applications.**

**Acronyms**

* **MVT:** Model-View-Template

**Terms**

* **Django:** A Python-based web framework that simplifies the creation of complex web applications.
* **Models:** Python classes that represent database tables and their data structure.
* **Views:** Python functions that handle web requests, process data, and render responses using templates.
* **Templates:** HTML files with special Django syntax for dynamically inserting data and applying styling.

**Technical Words**

* **Web framework:** A collection of tools and libraries that provide a structure for building web applications.
* **Back-end development:** The behind-the-scenes logic, data management, and server-side functionality of a website.
* **Relational database:** A database organized into tables with defined relationships between them.
* **Dynamic web applications:** Websites that display different content based on user input, data updates, or other triggers.

**Rarely Used Words (for a 12-year-old)**

* **Intimidated:** Feeling nervous or scared due to complexity or scale.
* **Architecture:** The underlying design and organization of a system.
* **Framework:** A structured set of tools and components to aid development.
* **Dynamically:** In a way that changes based on conditions or interactions.

**Software Packages**

* **Django:** A high-level Python web framework.

**Commands**

* None explicitly mentioned beyond the possible use of Python commands within code examples.

**Modules**

* None explicitly mentioned, though the organization of code into files and functions is implicitly a form of modularity.

**Ideas**

* **MVT Architecture:** The core organizational pattern in Django that separates data, logic, and presentation logic.
* **Data-driven applications:** Applications that center around managing, processing, and displaying data.

**Techniques**

* **Separation of concerns:** The principle of dividing a program into distinct sections, each with a specific responsibility (in this case, handling data, logic, and presentation separately).

**By Gemini**

Notes:

I used VSC. Steps I took:

From PowerShell:

1. mkdir mvt-example
2. cd mvt-example
3. code . 🡨 opens Visual Studio Code(VSC)

Once inside VSC I opened a terminal

1. pipenv install --python 3.11 (check your version with python –version)
2. pipenv shell
3. pipenv install django
4. django-admin startproject chefsTable
5. cd chefsTable 🡨 move into project file unless you used ‘django-admin startproject chefsTable .’
6. python manage.py startapp littleLemon

*+---chefsTable*

*| asgi.py*

*| settings.py*

*| urls.py*

*| wsgi.py*

*| \_\_init\_\_.py*

*\-------littleLemon*

*| admin.py*

*| apps.py*

*| models.py*

*| tests.py*

*| urls.py*

*| views.py*

*| \_\_init\_\_.py*

*+-------migrations*

*| 0001\_initial.py*

*| \_\_init\_\_.py*

*\-------templates*

*menu\_card.html*

[Knowledge check: Web Frameworks and MVT | Coursera](https://www.coursera.org/learn/django-web-framework/quiz/S3YA7/knowledge-check-web-frameworks-and-mvt) 3-28-2024 1056

* Woot got 100%. It’s the small victories.

[Module summary: Introduction to Django | Coursera](https://www.coursera.org/learn/django-web-framework/lecture/Qesi2/module-summary-introduction-to-django) 3-28-2024 1130

**Short summary:** This transcript summarizes a Django introduction module. It covers core Django concepts, project/app creation, Django admin tools, web frameworks, and the MVT architecture.

**Acronyms**

* **DRY:** Don't Repeat Yourself
* **MVT:** Model-View-Template
* **API:** Application Programming Interface

**Terms**

* **Django:** A Python web framework simplifying the creation of complex, database-driven web applications.
* **Project:** In Django, a collection of configurations and apps that together form a website.
* **App:** A self-contained component within a Django project, handling a specific set of features.
* **VS Code:** A popular source code editor from Microsoft.
* **Command Line Utility:** A text-based interface for interacting with a computer.

**Technical Words**

* **Web Application:** Software that runs on a web server and can be accessed through a web browser.
* **Framework:** A reusable software structure providing a foundation for building applications.
* **Three-Tier Architecture:** A design pattern that divides applications into presentation, application logic, and data storage layers. MVC-(Model-View-Controller) or in Django MVT-(Model-View-Template)
* **Database:** An organized collection of data.
* **Web Server:** A software that handles web requests and delivers content to clients.

**Rarely Used Words**

* **Scalability:** The ability of a system to handle increased workloads.
* **Architecture:** The overall design and structure of a system.
* **Component:** A self-contained part of a larger system.

**Software Packages**

* **Django:** A Python-based web framework.
* **VS Code:** Visual Studio Code(VSC) A source code editor.

**Commands**

* **django-admin:** A Django command-line utility for administrative tasks.
* **manage.py:** A Django-specific command-line tool for interacting with your Django project.

**Ideas**

* **MVT Architecture:** The core organizational pattern in Django that separates data (models), logic (views), and presentation (templates).
* **Web development frameworks:** The concept of using pre-built frameworks to simplify and streamline web application development.
* **Code reusability:** Writing code in a way that allows it to be used in multiple parts of an application or in different projects.

**Techniques**

* **DRY (Don't Repeat Yourself):** A software development principle emphasizing code reusability and avoiding redundancy.

**Keywords** =["Django", "project", "app", "VS Code", "web application", "framework", "MVT", "database", "scalability", "DRY"]

By Gemini

[Module Quiz: Introduction to Django | Coursera](https://www.coursera.org/learn/django-web-framework/exam/OkPsr/module-quiz-introduction-to-django) 3-28-2024 1237

* I missed several. I studied more and will retake in a week or so.

A close-up of a box

Description automatically generated

* Very misleading. By default it is not, but in general everyone creates one. I answered based on default.

[Additional resources | Coursera](https://www.coursera.org/learn/django-web-framework/supplement/0AF6Z/additional-resources)

* [Writing your first Django app – official documentation](https://docs.djangoproject.com/en/4.1/)
* [MVT Framework - Django](https://docs.djangoproject.com/en/4.1/faq/general/#django-appears-to-be-a-mvc-framework-but-you-call-the-controller-the-view-and-the-view-the-template-how-come-you-don-t-use-the-standard-names)
* [How to structure your Django project](https://docs.djangoproject.com/en/4.1/intro/tutorial01/)

Week 2

[Views | Coursera](https://www.coursera.org/learn/django-web-framework/lecture/dnq4B/views) 3-28-2024 1330

Summary: This transcript describes Django views, which are Python functions that handle web requests and generate responses. To use views, you'll create view functions in views.py, import them in urls.py, and associate them with specific URL patterns.

Short summary: Django views are like special functions that help websites respond to user requests. You write these functions and connect them to website addresses.

**Acronyms:**

* **HTTP:** Hypertext Transfer Protocol. The basic language for communication between web browsers and servers.
* **SDLC:** Software Development Life Cycle. A framework outlining stages involved in building software.

**Terms:**

* **Web Server:** A computer system that hosts websites and delivers them to users over the internet.
* **Framework:** A structure of pre-written code that provides a foundation and common functionalities, making development easier.
* **Logic:** The reasoning and decision-making within code that dictates how it behaves.
* **Rendering:** The process of turning code (like HTML) into a visual webpage on a user's screen.

**Technical Words:**

* **Present:** To display data to the user in a meaningful way.
* **Function:** A block of code that performs a specific task.
* **Static:** Content that doesn't change without manual updates.
* **Dynamic:** Content generated or changed based on user input or other factors.

**Software Packages:**

* **Django:** A popular Python-based web framework for building websites and applications.

**Modules:**

* **django.http:** A module within the Django framework that handles HTTP requests and responses.

**Commands:**  
None explicitly mentioned, but the concept of writing code in a programming language is inherent to the transcript's explanations.

**Ideas:**

* **HTTP Request-Response Cycle:** The fundamental communication model where a browser asks for a web page and the server provides it.
* **View Functions:** The core building blocks for handling requests and delivering responses in Django.
* **Routing:** The process of mapping URLs to specific view functions.

**Techniques:**

* **Creating Python functions:** The basic way to define view functions within Django.
* **Returning HTTP responses:** How Django views send rendered content back to the browser.
* **Keywords=**['HTTP', 'request', 'response', 'Django', 'view', 'function', 'HTML', 'data', 'URL', 'browser']

This output was generated by Gemini.

Notes by me:

For practice I did the same views and urls files that the instructor did. I then reverse engineered the rest. Below is my steps.

* mkdir views
* cd views
* pipenv install –python 3.11
* pipenv shell
* pipenv install django
* django-admin startproject views\_practice .
* python manage.py startapp views\_app
* code . 🡨 to start Visual Studio Code(VSC)

[Creating views and mapping to URLs | Coursera](https://www.coursera.org/learn/django-web-framework/lecture/Y4Z1f/creating-views-and-mapping-to-urls) 3-28-2024 1430

Summary: This transcript explains how to configure URL patterns within a Django project to connect specific URLs with their corresponding views. You'll learn about URLconf files at both the project and app levels and use the 'include' function to link them.

Short summary: This transcript teaches you how to set up website addresses in Django by connecting them to the right functions within your code.

**Terms:**

* **URL:** Uniform Resource Locator. In simple terms, a website address.
* **HTTP Request:** A message sent by a web browser asking for a specific webpage.
* **HTTP Response:** The webpage content sent back by the web server in answer to a request.
* **View Function:** A piece of code in Django that handles a web request and creates the appropriate response.
* **URLconf:** A Django configuration file (urls.py) that maps URLs to view functions.

**Technical Words:**

* **Configuration:** The arrangement of settings in a system.
* **Application:** (often shortened to "app") A self-contained set of functionalities within a larger software project.
* **Module:** A reusable unit of code, generally contained within a single file.
* **Project:** In Django, the top-level container for your website or web application.
* **Deployment:** Making a website or application live on a server so users can access it.
* **Localhost:** A special name that designates your own computer as a server during development.

**Software Packages:**

* **Django:** A Python-based web framework for building websites and applications.

**Commands:**

* **python3 or python manage.py runserver:** A Django command to start the development server.

**Modules:**

* **django.http:** A part of Django that deals with web requests and responses.
* **django.urls:** The part of Django responsible for URL configuration.

**Ideas:**

* **URL Mapping:** Connecting specific URLs to the view functions that should handle them.
* **Project-Level vs. App-Level Configuration:** Django lets you organize URL patterns at both the overall project level and within specific apps.
* **Using the 'include' Function:** A way to combine URL patterns from multiple urls.py files.

**Techniques:**

* **Creating a Django project and app:** Basic steps for setting up a new Django project.
* **Writing a View Function:** Defining a Python function to handle a web request.
* **Updating urls.py:** Configuring the relationship between URLs and view functions.

**Keywords**=['URL', 'HTTP', 'request', 'response', 'Django', 'view', 'function', 'URLconf', 'include', 'project', 'app', 'module']

This output was generated by Gemini.

Note by me

Doing this for muscle memory…

mkdir creating-views-and-mapping-to-urls 🡨 base directory

cd mkdir

pipenv install –python 3.11 🡨 Use your version of python. python –version

pipenv shell 🡨 to step into the virtual environment

pipenv install django

code .

django-admin startproject myproject .

python manage.py startapp myapp

[View logic | Coursera](https://www.coursera.org/learn/django-web-framework/supplement/o2Enm/view-logic) 3-28-2024 1450

Summary: This transcript explains Django views, which handle incoming web requests and send back responses. It discusses how views interact with models (database representations) and templates (HTML structures) to create dynamic web pages.

Short summary: This text describes how Django views process user requests, interact with data, and use templates to generate website content.

**Acronyms:**

* **MVT:** Model-View-Template. A design pattern used in Django.

**Terms:**

* **URL dispatcher:** A component in Django responsible for matching incoming URLs to the appropriate view functions.
* **Web Template:** An HTML file with placeholders for displaying dynamic data.
* **HTTP Request:** A message sent by a web browser to a server, asking for a webpage or to perform an action.
* **HTTP Response:** The content the server sends back to the browser, typically including the requested webpage.

**Technical Words:**

* **Client:** The software requesting something (here, the web browser asking for web pages).
* **Server:** The software fulfilling requests (here, the Django web application).
* **HTTP GET:** A request method used to retrieve data from the server.
* **HTTP POST:** A request method used to send data to the server for processing (e.g., form submission).
* **Instance:** A single object or record within a database model.
* **Rendering:** Creating a final webpage by merging a template with dynamic data.

**Modules:**

* **django.shortcuts:** Provides shortcuts for common Django functions, like rendering templates.
* **django.views:** Contains the base 'View' class and other view-related functionality.
* **django.views.generic:** Offers pre-built generic views for common tasks (e.g., displaying lists, creating objects).

**Ideas**

* **View Functions:** The core logic in Django that handles requests and generates responses.
* **Model Interactions:** Views can fetch data from models (database), modify model data, and control how the data is presented.
* **Template Rendering:** Views use templates to structure the HTML content sent to the user.
* **Function-based views:** Define views as regular Python functions.
* **Class-based views:** Define views as classes inheriting from Django's 'View' class.
* **Generic views:** Pre-built classes in Django for common web tasks, simplifying development.

**Techniques:**

* **Handling GET and POST requests:** Using conditional logic (if statements) to process different request types.
* **Rendering templates:** Using the 'render' shortcut to combine HTML templates with data.

**Theories:**

* **Model-View-Template (MVT):** The architectural pattern Django uses to separate data (model), presentation (template), and request processing logic (view).

**Keywords=**['Django', 'view', 'model', 'template', 'GET', 'POST', 'request', 'response', 'render', 'class-based view', 'generic view', 'MVT']

This output was generated by Gemini

[Creating views and view logic | Coursera](https://www.coursera.org/learn/django-web-framework/lecture/8GRvs/creating-views-and-view-logic) 3-28-2024 1530

Summary: This transcript teaches how to create basic web pages in Django. You'll learn to write view functions in Python that return text or HTML and how to connect these views to URLs for display in a browser.

Short summary: This text explains how to make simple web pages in Django by writing Python functions that send text/HTML and linking them to website addresses.

**Terms:**

* **View function:** A Python function within Django that processes a web request and generates a response.
* **HTTP Response:** The content (often HTML) sent back from the web server to the web browser.
* **URL:** Uniform Resource Locator – the address of a webpage.
* **URL Configuration:** The mapping of URLs to specific view functions within a Django project.

**Technical Words:**

* **Suffix:** Something added to the end of something else (e.g., added to the end of a URL).
* **Markup:** Code, like HTML, used to define the structure and formatting of a web page.
* **Styling:** Applying visual styles to elements on a web page, often using CSS.
* **Web Server:** A system that hosts websites and responds to browser requests

**Software Packages:**

* **Django:** A Python-based web framework for building websites and applications.
* **Python:** A general-purpose programming language that Django is built on.

**Commands:**

* **python manage.py runserver:** A Django-specific command used to start the development server.

**Modules:**

* **views.py:** The standard file in Django apps where view functions are defined.
* **urls.py:** The file where you configure URL patterns in Django.
* **datetime:** A Python module for working with dates and times.

**Ideas:**

* **Django Views:** The core mechanism in Django for handling web requests and generating responses.
* **Mapping URLs to Views:** The process of connecting specific URLs to the view functions responsible for them.
* **Rendering basic HTML:** Sending simple HTML content directly from a view function.

**Techniques:**

* **Writing Python view functions:** Creating functions in views.py to handle web requests.
* **Returning HTTP responses:** Using the 'HttpResponse' object to send content back to the browser.
* **Configuring URLs:** Editing the urls.py file to map URLs to the correct view functions.

**Keywords=**['Django', 'view', 'HTTP', 'response', 'URL', 'Python', 'module', 'HTML', 'CSS']

This output was generated by Gemini.

Note: Doing this for muscle memory

Create dir, start python in a virtual environment, enter environment, install django in a virtual environment, star project and app.

1. mkdir creating-views-and-view-logic
2. cd creating-views-and-view-logic
3. pipenv install –python 3.11
4. pipenv shell
5. pipenv install django
6. code .

[Exercise: Creating a view and URL configuration | Coursera](https://www.coursera.org/learn/django-web-framework/ungradedLab/M97Kt/exercise-creating-a-view-and-url-configuration) 3-28-2024 1550

* This is a super easy lab. Im skipping it this time through the course.

[Knowledge check: Views | Coursera](https://www.coursera.org/learn/django-web-framework/quiz/5CcPu/knowledge-check-views)

* It probably is me, but dang if they word questions weird in this course. “The include() function includes, in the URL patterns, the URL mapping to a view function. true or false. “I answered true thinking its being included in the project level urls. Oh well.
* The render function doesn't create the template. The template is already created as an HTML file. Instead, the render function loads the template, **combines** it with any provided context data (like variables), and returns an HTTP response object containing the fully rendered HTML. This rendered output is what the client's web browser ultimately displays.
* Ok I retook the test on 3-30-2024 and got it wrong again. I did all the previous videos and readings and still got it wrong. So I

asked Gemini to create a lecture and I think I got it.

* **Q.** The **include()** function includes, in the URL patterns, the URL mapping to a view function. True or False?
* **A.** <False>The include() function doesn't directly include specific URLs. Instead, it references an app's urls.py file, which is where the actual URL-to-view mappings are defined.
* **<Course>** Not quite. The mapping of a URL pattern with a view function is done by the path() function.
* **<Project>** path('blog/', include('blog.urls')), # Include blog app's URLs<includes map created at app url>
* **<APP>** path('article/<int:article\_id>/', views.article\_detail, name='article\_detail'), <Creates map of url>

[HTTP requests | Coursera](https://www.coursera.org/learn/django-web-framework/lecture/Wxiy1/http-requests) 3-28-2024 1626

Summary: This transcript explains HTTP and HTTPS – the protocols used for web communication. It details how browsers send requests, servers send responses, the role of HTTP methods, status codes, and the importance of HTTPS for security.

Short summary: This transcript teaches about the protocols that make the web work, focusing on how websites and browsers talk to each other and the difference between HTTP and HTTPS.

**Acronyms:**

* **HTTP:** Hypertext Transfer Protocol
* **HTTPS:** Hypertext Transfer Protocol Secure

**Terms:**

* **Web Server:** A computer that stores web pages and responds to requests from browsers.
* **Web Browser:** Software (e.g., Chrome, Firefox) used to view websites.
* **Protocol:** A set of rules defining how computers communicate.
* **HTML:** Hypertext Markup Language – the language used to structure web pages.

**Technical Words**

* **Resource:** A file or piece of data on a web server (e.g., an image, a webpage).
* **Client:** The software requesting something (here, the web browser)
* **Request:** A message the client sends to the server to ask for something.
* **Response:** The server's reply to the client's request.
* **Status Code:** A number indicating the result of the request (e.g., 200 for success, 404 for 'Not Found').
* **Encryption:** Scrambling data so only authorized parties can read it.

**Ideas:**

* **HTTP Request-Response Cycle:** The fundamental communication model of the web.
* **HTTP Methods:** Verbs (GET, POST, etc.) that tell the server what action the client wants.
* **HTTP Status Codes:** Numbers that categorize the result of a server handling a request.
* **HTTPS:** The secure version of HTTP that protects data in transit.

**Techniques:**

* **Sending HTTP requests:** How browsers ask servers for resources.
* **Generating HTTP responses:** How servers reply to browser requests.

**Keywords=**['HTTP', 'HTTPS', 'protocol', 'web browser', 'web server', 'request', 'response', 'GET', 'POST', 'status code', 'encryption']

This output was generated by Gemini.

[Request and Response Objects | Coursera](https://www.coursera.org/learn/django-web-framework/supplement/DMTe5/request-and-response-objects)

Summary: This reading details how Django, a Python web framework, handles web requests and responses. It explains the roles of Django's HttpRequest and HttpResponse objects in processing requests and generating responses.

Detailed Breakdown:

* **Acronyms**
  + HTTP: Hypertext Transfer Protocol
  + REST: Representational State Transfer
* **Technical Words**
  + Client-server architecture: A network structure where a client (e.g., web browser) requests resources/services from a server.
  + URL: Uniform Resource Locator, the address of a web resource.
  + View: A Django function that handles web requests, processes data, and generates responses.
  + Metadata: Data that provides information about other data.
* **Rarely Used Words**
  + Instantiate: To create an instance of a class.
* **Software Packages**
  + Django: A high-level Python web framework facilitating quick and clean web development.
* **Python Commands/Modules**
  + django.http: Contains classes for handling requests (HttpRequest) and responses (HttpResponse)
  + loader.get\_template: Loads an HTML template for rendering with data.
  + template.render: Renders the template with a given context (data dictionary)
* **Ideas/Techniques**
  + Request-response cycle: The fundamental communication model for web interactions.
  + HTTP methods: Indicate the action requested by the client (e.g., GET, POST, PUT, DELETE)
* **Mistakes or Misleading Information**
  + The reading could clarify that the Django URL dispatcher *attempts* to match a URL pattern. A match might not always be found.

keywords = ['HTTP', 'REST', 'GET', 'POST', 'PUT', 'DELETE', 'HttpRequest', 'HttpResponse', 'django.http', 'loader.get\_template', 'template.render']

Written by Gemini

[Creating Requests and Responses | Coursera](https://www.coursera.org/learn/django-web-framework/lecture/ryFR1/creating-requests-and-responses) 3-29-2024 0542

**Summary:** This video teaches how Django utilizes HTTP request and response objects to handle the communication cycle between client and server. View functions receive HTTP requests, process them, and generate appropriate HTTP responses.

* **Acronyms**
  + HTTP (Hypertext Transfer Protocol)
* **Technical Words**
  + **URL** (Uniform Resource Locator): A web address that specifies the location of a resource on the internet.
  + **Web Server:** A computer system that hosts websites and responds to web requests over HTTP.
  + **View Function:** A Python function within Django that handles the logic of a web request, generating a response.
  + **HTTP Request:** A message sent from a client (like a web browser) to a server, initiating a web action.
  + **HTTP Response:** A message sent from a server back to the client in response to a request.
* **Rare Words (for a 12-year-old)**
  + **API** (Application Programming Interface): Rules governing how software components communicate with each other.
  + **Character Set** - A defined mapping between numbers and text characters (e.g., UTF-8).
* **Software Packages**
  + **Django:** A Python web framework that simplifies the creation of web applications.
* **Modules/Commands**
  + **django.http:** A Django module containing classes and functions for HTTP requests and responses.
* **Ideas/Techniques**
  + **Request-Response Cycle:** The fundamental communication pattern of the web - a client makes a request, the server processes it, and returns a response.
* keywords = ['HTTP', 'URL', 'Web Server', 'Django', 'View Function', 'HTTP Request', 'HTTP Response', 'API', 'Character Set', 'django.http', 'Request-Response Cycle']

**Mistakes or Misleading Information**

* **Minor Oversimplification:** While the example demonstrates modification of the HttpResponse object, the video correctly notes this isn't a common real-world practice. Direct modification of the object is rarely needed.

**Written by Gemini**

Notes

The instructors in this course are human and make errors, but geesh if you don’t have to look carefully. I copied all the code he had before running the script and got errors. I then looked at the code later in the video and he removed the error sometime off video. “<br> {response.headers}” was removed. No big deal, just funny.

[Understanding URLs | Coursera](https://www.coursera.org/learn/django-web-framework/lecture/flumK/understanding-urls) 3-24-2024 0551  
**Short Summary:** This video explains URLs (Uniform Resource Locators), which are web addresses used to locate specific resources online. The lesson covers the components of a URL and how they work together to direct web requests.

**Detailed Breakdown**

* **Acronyms**
  + URL (Uniform Resource Locator)
  + HTTP (Hypertext Transfer Protocol)
  + HTTPS (Hypertext Transfer Protocol Secure)
* **Technical Words**
  + **Scheme/Protocol:** Rules for data exchange (e.g., HTTP, HTTPS)
  + **Domain Name:** Identifies the website (e.g., [www.example.com](https://www.example.com/))
  + **Subdomain:** Optional part of the domain (e.g., 'www' in [www.example.com](https://www.example.com/))
  + **Second-Level Domain:** Specific organization or company name
  + **Top-Level Domain (TLD):** Category or country indicator (.com, .org, .uk)
  + **File Path/Page Path:** Location of the resource on the server
  + **URL Parameters/Query Strings:** Carry additional data
  + **Web Server:** Computer that hosts a website
* **Rare Words**
  + **Metadata:** Data about data
* **Software Packages**
  + **Django:** (mentioned, but not explained) Python web framework
* **Ideas/Techniques**
  + **URL Design:** Intentional creation of URLs to match website structure
* keywords = ['URL', 'HTTP', 'HTTPS', 'Scheme', 'Protocol', 'Domain Name', 'Subdomain', 'Second-Level Domain', 'Top-Level Domain', 'TLD', 'File Path', 'Page Path','URL Parameters', 'Query Strings', 'Web Server', 'Resource', 'Encrypt', 'Metadata', 'Django', 'URL Design']

**Mistakes or Misleading Information**

* **Conflating Subdomain and Scheme:** The transcript incorrectly suggests that some browsers substitute the scheme (HTTP/HTTPS) with a symbol like a lock. Browsers usually display this lock icon in addition to the scheme.
* **Overemphasis on URL Parameters:** While Django can process both, the framework primarily works with path-like URL parameters rather than query strings.

**Written by Gemini**

[Parameters | Coursera](https://www.coursera.org/learn/django-web-framework/supplement/OhSHQ/parameters) 3-29-2024 0927

Summary: This text explains the use of path, query, and body parameters in web applications and their association with HTTP methods like GET, PUT, POST, and DELETE, focusing on how they are implemented in Django.

Acronyms:

* URL: Uniform Resource Locator
* HTTP: Hypertext Transfer Protocol
* POST: A request method supported by HTTP used by the World Wide Web.
* GET: A HTTP method used to request data from a specified resource.
* PUT: A HTTP method that replaces all current representations of the target resource with the request payload.
* DELETE: A HTTP method that deletes the specified resource.
* HTML: HyperText Markup Language
* CSRF: Cross-Site Request Forgery

Terms:

* Parameters: Instructions or data passed to a function or program.
* Web Application: A software application that runs on a web server and can be accessed through a web browser.
* View Function: In Django, a view function is a Python function that takes a web request and returns a web response.
* Path Parameter: A parameter included in the URL path, used to pass unique identifiers of resources.
* Query Parameter: A key-value pair in the URL after the '?' symbol, used to provide additional instructions to the web server. http://localhost:8000/myapp/?name=John&id=1
* Body Parameter: Data sent through the HTTP request body, typically used with POST and PUT requests.
* URL Dispatcher: A system in Django that directs incoming web requests to the appropriate view based on the request URL.
* Query String: The part of a URL where data is passed in key=value pairs.
* URL Patterns: A list of URL-to-view mappings used by Django to route web requests.
* Path Converter: A path converter is a special pattern within a Django URL that captures a segment of the URL as a variable and passes it to the corresponding view function. Ex. path('articles/<int:article\_id>/', views.article\_detail, name='article\_detail')

Technical Words:

* Endpoint: The part of the URL which comes after the domain name, specifying the web resource being accessed.
* HTTP Methods: The set of request methods supported by HTTP (e.g., GET, POST, PUT, DELETE).
* Slug: A part of a URL that identifies a particular page on a website in a form readable by both users and search engines.
* UUID: Universally Unique Identifier, a 128-bit number used to uniquely identify information in computer systems.

Rarely Used Words:

* Dispatcher: In web development, a component that handles the routing of requests to the appropriate handler based on the URL.
* CSRF Token: A token used to prevent cross-site request forgery attacks, ensuring that a request made to a web application is coming from a trusted source.

Software Packages:

* Django: A high-level Python web framework that encourages rapid development and clean, pragmatic design.

Commands:

* **path**: A function in Django used to define a URL pattern.
* **HttpResponse**: A class in Django used to return a response to a web request.

Modules:

* **django.http**: A module in Django that contains classes like HttpResponse to handle HTTP specific response and request.

Ideas:

* Modularization: The practice of dividing a software application into separate modules that can be developed, tested, and maintained more easily.

Techniques:

* URL Mapping: The process of defining routes and URLs in a web application to direct requests to the correct view functions.

Keywords: ["URL", "HTTP", "POST", "GET", "PUT", "DELETE", "HTML", "CSRF", "parameters", "web application", "view function", "path parameter", "query parameter", "body parameter", "URL dispatcher", "query string", "URL patterns", "endpoint", "HTTP methods", "slug", "UUID", "dispatcher", "CSRF token", "Django", "path", "HttpResponse", "django.http", "modularization", "URL mapping"]

ChatGPT

Notes

This is how a directory tree looks when using ‘**django-admin startproject myproject’** and python manage.py startapp myapp(On Left) And on the right is how it looks when using **‘django-admin startproject myproject .**’ (note the dot is added) and python manage.py startapp myapp.

A screen shot of a computer program

Description automatically generatedA screen shot of a computer program

Description automatically generated

I learned that Django expects the templates to be in the app folder not the project folder by default. This has caused me headaches in the past, to which Bard and ChatGPT solved by working around the issue, but never telling me Im a moron and to put the folder in the app by default. There really are no good reasons to not follow this best practice that I know of. I learned this by using Gemini while driving to work. I asked it “Pretend you’re a college professor giving a lecture on Django forms and tell me everything about the focusing on what might be causing my app not to find the templates.” It did a fantastic job, but the most immediately helpful was it said Django expects the templates in the app folder by default.

[Mapping URLs with Params | Coursera](https://www.coursera.org/learn/django-web-framework/lecture/crrYS/mapping-urls-with-params) 3-29-2024 1122

Summary: This transcript explains how to use URL parameters in Django to pass data from the URL to a view function. It covers capturing values with angle brackets, path converters, and how to use these values within your view logic.

Short summary: This transcript teaches how to include dynamic information within URLs in your Django project and how to make your code respond to those values.

**Acronyms:** None in this transcript.

**Terms:**

* **URL Parameter:** A variable piece of data embedded within a URL.
* **View Function:** A function in Django that handles web requests and generates responses.
* **Path Converter:** A type specifier within a Django URL pattern that determines how to extract and interpret a URL parameter.

**Technical Words:**

* **URL Mapping:** The process of connecting specific URL patterns to view functions.
* **Angle Brackets (**< >**)** Used in Django URL patterns to denote a URL parameter.
* **Dictionary:** A Python data structure that stores key-value pairs.
* **Primary Key:** A unique identifier for a row in a database table.

**Rarely Used Words:**

* **Denote:** To indicate or represent something.

**Modules:**

* **views.py:** The file in Django apps where view functions are defined.
* **urls.py:** The file in Django where URL patterns are configured.

**Ideas:**

* **URL Parameters:** A way to make URLs dynamic and pass information to view functions.**Example: /blog/2023/10/15/my-coding-journey/ to fetch a specific blog post.**
* **Path Converters:** Used to specify the data type expected in a URL parameter.**Example: /shop/product/<int:product\_id>/ to ensure a product ID is an integer.**
* **Data Fetching:** Using URL parameters to potentially retrieve information from a database.**Example: /search/?category=electronics&keywords=smartphones for filtering database search results.**
* **Content Grouping:** Creating structured URLs that logically categorize content.**Example: /news/sports/baseball/article-title/ for organizing news articles.**

**Techniques**

* **Defining URL patterns with parameters:** Using the path function and angle brackets in urls.py.
* **Capturing URL parameter values:** Adding matching arguments to your view function.
* **Using URL parameters in view logic:** Processing the captured value within your view function.

**Keywords=**['Django', 'URL', 'parameter', 'view function', 'path', 'converter', 'dictionary']

This output was generated by Gemini.

[Exercise: Mapping URLs with Params | Coursera](https://www.coursera.org/learn/django-web-framework/ungradedLab/XZlEd/exercise-mapping-urls-with-params) 3-29-2024 1145

* A lab. Wrote it into my project folder for the course for the muscle memory.

[Knowledge check: Requests and URLs | Coursera](https://www.coursera.org/learn/django-web-framework/quiz/0vEN3/knowledge-check-requests-and-urls) 3-29-2024 1153

* I struggled on this one. Retake a few times.

[Additional resources | Coursera](https://www.coursera.org/learn/django-web-framework/supplement/S9v2H/additional-resources) 3-29-2024 1155

* [Creating views– official documentation](https://docs.djangoproject.com/en/4.1/topics/http/views/)
* [Class-based views – Official](https://docs.djangoproject.com/en/4.1/topics/class-based-views/)
* [The render() function in Django](https://docs.djangoproject.com/en/4.1/topics/http/shortcuts/#render)
* [Getting query parameters from a request in Django](https://docs.djangoproject.com/en/4.1/topics/http/urls/#path-converters)
* [The HTTP server responses](https://developer.mozilla.org/en-US/docs/Web/HTTP/Status)

[Regular expressions in URLs | Coursera](https://www.coursera.org/learn/django-web-framework/lecture/sVWFw/regular-expressions-in-urls) 3-29-2024 1216

Summary: This transcript explains how to use regular expressions (RegEx) in Django to create flexible URL patterns. It discusses why RegEx is useful for dynamic URLs and provides an introduction to common RegEx symbols.

Short summary: This text teaches how to use regular expressions for creating more powerful and adaptable URL matching in Django projects.

**Acronyms:**

* **RegEx:** Regular Expressions

**Terms**

* **Dynamic URL:** A URL that includes variable elements.
* **Static URL:** A URL with a fixed, unchanging structure.
* **String:** A sequence of characters (text) in programming.

**Technical Words**

* **Pattern:** A set of rules or characteristics used to identify something.
* **Matching:** Finding instances within a text that fit a specific pattern.
* **Validation:** Checking if something conforms to a required format or standard.
* **Anchor:** A character in RegEx that specifies the position of a pattern relative to the beginning (^) or end ($) of a string.

**Rarely Used Words**

* **Immutable:** Unchangeable.
* **Negation:** Expressing the opposite or absence of something.
* **Literal:** Taking something at its face value (in RegEx, not treating special characters as special).

**Software Packages**

* **Django:** A Python-based web framework for building websites and applications.

**Commands:** None explicitly mentioned in the transcript.

**Modules:**

* **django.urls**: The module in Django responsible for URL configuration.

**Ideas**

* **Regular Expressions (RegEx):** A way to define flexible patterns for text matching and manipulation.
* **Using RegEx in Django URLs:** Employing RegEx to create dynamic URLs that can handle a wider range of values.

**Techniques:**

* **Defining URL patterns with regular expressions:** Using the re\_path() function in Django's urls.py.
* **Using RegEx symbols:** Employing characters like ^, $, [], {}, () to create patterns within URLs.

**Keywords=**['Django', 'URL', 'regular expression', 'RegEx', 'pattern', 'string']

This output was generated by Gemini.

[URL Namespacing and Views | Coursera](https://www.coursera.org/learn/django-web-framework/supplement/9Bl6F/url-namespacing-and-views)

Summary: This text explains how to use namespaces in Django to organize URLs, avoid conflicts when view names overlap, and make your project more maintainable.

Short summary: This reading teaches how to give URLs unique names within a Django project to keep things organized, especially when multiple apps have views with the same name.

**Acronyms:** None in this transcript.

**Terms**

* **URLConf:** The configuration file (urls.py) where you define URL patterns in a Django app.
* **Namespace:** A way to give URLs unique identifiers within a Django project.

**Technical Words:**

* **Hardcoded:** Writing something directly into the code instead of making it flexible.
* **Scalable:** Able to grow or change easily.
* **Reverse Lookup:** Finding a URL based on its view name (and namespace).

**Rarely Used Words**

* **Prepend:** To add something to the beginning.

**Software Packages:**

* **Django:** A Python-based web framework for building websites and applications.

**Commands**

* **python manage.py shell:** Opens an interactive Django shell for working with your project.

**Modules:**

* **django.urls:** The module in Django responsible for URL handling.

**Ideas:**

* **URL Namespaces:** A way to avoid naming conflicts between URLs in different apps of a Django project.
* **Application Namespace:** Created by setting the app\_name variable in an app's urls.py.
* **Instance Namespace** An optional namespace you can add when using the include() function in the project's urls.py

**Techniques**

* **Setting an app namespace:** Defining the app\_name variable in an app's urls.py.
* **Using**reverse()**to get URLs:** Looking up a URL path based on its namespace and view name.
* **Using the**{% url %}**tag in templates:** Generating URLs dynamically within HTML templates.

Keywords = ['Django', 'URL', 'namespace', 'view', 'reverse', 'include']

By Gemini

[Exercise: Creating URLs and Mapping to Views | Coursera](https://www.coursera.org/learn/django-web-framework/ungradedLab/FWVAw/exercise-creating-urls-and-mapping-to-views) 3-29-2024 1326

* This lab was completed first time through. I did it again on my laptop for muscle memory.

[handling-errors-in-views](https://www.coursera.org/learn/django-web-framework/supplement/QpaMm/handling-errors-in-views) 3-29-2024 1401

Summary: This transcript explains how Django handles errors that occur within web applications. It details common HTTP status codes for both client-side (400s) and server-side (500s) errors and how Django uses exception handling to manage them.

Short summary: This text teaches about error types in web applications, how HTTP status codes signal them, and how Django provides mechanisms to deal with errors.

**Acronyms:**

* **HTTP:** Hypertext Transfer Protocol

**Terms:**

* **Status Code:** Number indicating the result of a web server handling a request.
* **Client Error:** Problem caused by the user's request (e.g., bad URL).
* **Server Error:** Problem on the web server side preventing it from fulfilling the request.

**Technical Words**

* **Exception:** A signal in programming that an error or unexpected event has occurred.
* **Override:** To replace default behavior with custom behavior.
* **Subclasses:** In programming, classes that inherit properties from a parent class.

**Rarely Used Words**

* **Abort:** To end a process prematurely.

**Software Packages**

* **Django:** A Python-based web framework for building websites and applications.

**Ideas**

* **Error Handling:** Strategies to manage unexpected issues in applications gracefully.
* **HTTP Status Codes:** Standardized codes for communicating web request results.
* **Exception Handling in Django:** Django's way of dealing with errors using specific views.
* **Custom Error Views:** Overriding default error pages for a tailored user experience.

**Techniques**

* **Raising Exceptions:** How Django signals that an error has occurred.
* **Customizing Error Handling:** Using handler400, handler403, handler404, handler500 to define how different error types are presented to the user.

**Keywords**=['Django', 'error', 'HTTP', 'status code', 'client error', 'server error', 'exception']

Gemini

[Handling Errors in Views | Coursera](https://www.coursera.org/learn/django-web-framework/supplement/QpaMm/handling-errors-in-views)

[Demo: Handle errors in views | Coursera](https://www.coursera.org/learn/django-web-framework/lecture/bYSvR/demo-handle-errors-in-views) 3-29-2024 1610

Summary: This transcript teaches how to handle 404 "Page Not Found" errors in Django. It explains how to create custom error pages, set debug mode, and use the HttpResponseNotFound class for correct error signaling.

**Acronyms:**

* **HTTP:** Hypertext Transfer Protocol
* **URL:** Uniform Resource Locator (the address of a web page)

**Terms**

* **Debug Mode:** A development setting in Django that provides detailed error messages.

**Technical Words**

* **View:** A function in Django that handles web requests and generates responses.
* **Status Code:** Number indicating the result of a server handling a web request.
* **Branding:** The visual style and messaging that represents your website or application.

**Rarely Used Words**

* **Render:** In web development, the process of generating an HTML page from code and data.

**Software Packages**

* **Django:** A Python-based web framework for building websites and applications.

**Commands**

* **python manage.py runserver:** Starts the Django development server.

**Modules**

* **views.py:** The file in Django apps where you define view functions.
* **urls.py:** The file where you define URL patterns.

**Ideas**

* **Error Handling:** Managing unexpected issues gracefully in your application.
* **404 Error:** The standard "Page Not Found" error.
* **Custom Error Pages:** Tailoring error pages to match your website's style.
* **HTTP Status Codes:** Standardized codes that communicate the result of web requests.

**Techniques**

* **Creating a**handler404**view:** Defining a function to handle 404 errors.
* **Using**HttpResponseNotFound**:** Returning the correct 404 status code.
* **Switching Debug Mode:** Controlling whether detailed error messages are displayed to users.

**Keywords=**['Django', 'error', 'debug', 'view', 'URL', 'HTTP', '404', 'HttpResponseNotFound' ]

By Gemini

[Class-based views | Coursera](https://www.coursera.org/learn/django-web-framework/lecture/BxGIU/class-based-views) 3-29-2024 1627

Summary: This video explains the concept of class-based views in Django, highlighting how they simplify the management of HTTP request methods through object-oriented techniques and inheritance, and discussing mixins for reusable component creation.

Acronyms:

* HTTP: HyperText Transfer Protocol
* CRUD: Create, Read, Update, Delete
* OOP: Object-Oriented Programming

Terms:

* Framework: A set of tools, guidelines, and structure to help developers build and manage software projects efficiently.
* Design Patterns: Standard solutions to common software design problems.
* MVT: Model-View-Template, a software architectural pattern used in Django.
* View: In web development, a view processes requests and returns responses.
* Application Logic: The part of the program that performs the data processing. It's where the main functionality of the application is implemented.
* Object-Oriented Techniques: Programming methods based on objects and classes to create reusable and modular code.
* Inheritance: A principle of object-oriented programming where a class can inherit properties and methods from another class.
* Mixins: A class that provides a certain functionality to be inherited by a subclass, without being the primary parent class.
* Query Set: In Django, a collection of database queries to retrieve objects.

Technical Words:

* Class-Based Views: A feature in Django that allows developers to structure views as classes for better organization and reuse.
* Function-Based Views: Traditional views in Django defined as functions.
* Instance Methods: Methods of a class that can access and modify the object's attributes.
* Conditional Logic: Programming constructs used to perform different actions based on different conditions.
* Multiple Inheritance: A feature of some object-oriented programming languages where a class can inherit behaviors and attributes from more than one parent class.

Rarely Used Words:

* Robust: Strong and effective in all or most situations and conditions.

Software Packages:

* Django: A high-level Python web framework that encourages rapid development and clean, pragmatic design.

Commands:

* Get, Post, Put, Delete: HTTP methods for requesting data from, or submitting data to, a server.

Modules:

* No specific modules are mentioned beyond Django and its components.

Ideas:

* Simplification of code through class-based views.
* Separation of logic using object-oriented principles.

Techniques:

* Using class-based views to handle different HTTP methods with specific instance methods.
* Employing mixins to factor code into reusable components.

Keywords= ["Django", "class-based views", "function-based views", "HTTP", "CRUD", "OOP", "inheritance", "mixins", "view", "application logic", "object-oriented techniques", "instance methods", "conditional logic", "multiple inheritance", "robust", "framework", "design patterns", "MVT", "query set"]

ChatGPT

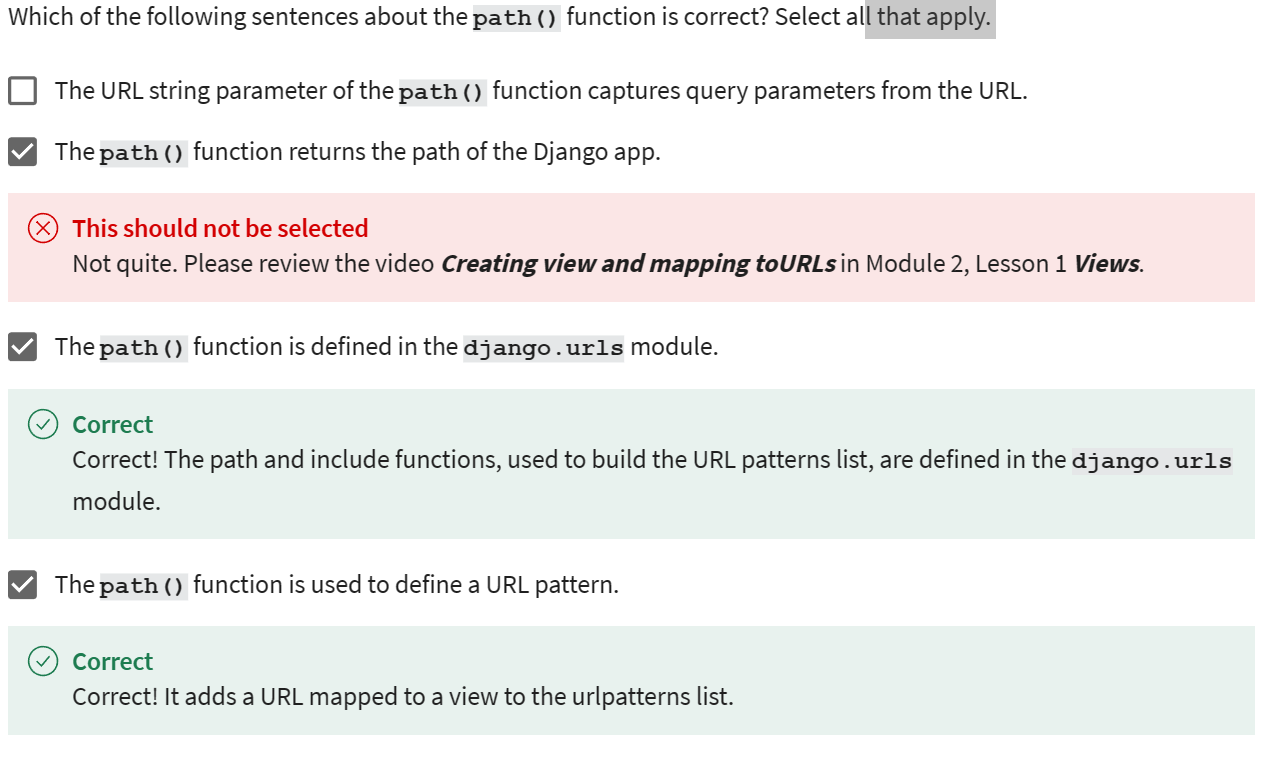
[Knowledge check: Handle errors in Views | Coursera](https://www.coursera.org/learn/django-web-framework/quiz/bNrDG/knowledge-check-handle-errors-in-views) 3-29-2024 1631

* Missed 1

[Module summary: Views | Coursera](https://www.coursera.org/learn/django-web-framework/lecture/D0o12/module-summary-views) 3-29-2024 1640

* Just a summary

[Module Quiz: Views | Coursera](https://www.coursera.org/learn/django-web-framework/exam/MmkTf/module-quiz-views) 3-30-2024 0825



This test is eating me up. I clearly need to study it more. I keep getting the above answer wrong. I have retaken test many times. I can retake again after 3-30-2024 5pm.

Gemini

Its annoying that the test suggests I review the video about this, yet nowhere in the video does the instructor talk about this exact answer and why its wrong. Gemini was able to tell me.  
You're absolutely right! The transcript contains a misstatement regarding the path() function. Let me clarify the key points:

**What the path() function does:**

* **Defines URL patterns:** It maps incoming URL patterns to specific views within your Django app.
* **Can capture path parameters:** Using path converters like <int:year>, it can extract values from the URL and pass them to your view functions.

**What the path() function does NOT do:**

* **Return the file system path of your Django app:** That's not its purpose. Django has other mechanisms to get the project's file system location if needed.

**Why the transcript might be misleading:**

The transcript likely aims to emphasize the distinction between URL routing within your app (what path() handles) and the physical location of your Django project on your computer's file system. However, the wording could lead to confusion.

**Let me know if there's anything else from the transcript you'd like clarified!**

Week 3

[Models | Coursera](https://www.coursera.org/learn/django-web-framework/lecture/SNEvS/models)

**Short Summary:** This video introduces the concept of models in Django, a web framework that uses models as a way to interact with databases. Models streamline database operations (creating, reading, updating, deleting data) within a Python environment.

**Detailed Breakdown:**

* **Acronyms**
  + CRUD (Create, Read, Update, Delete)
  + MVT (Model-View-Template)
  + SQL (Structured Query Language)
* **Technical Words**
  + Model: A Python class representing a database table, defining data fields and behaviors.
  + Database: Structured collection of data.
  + Framework: A software platform providing basic structure and tools for application development.
  + Object: An instance of a class in programming, containing data and methods.
  + Migration: A way to track and execute changes to your Django models (and resulting database schema).
* **Rarely Used Words**
  + Modular: Built with self-contained components.
  + Definitive: Authoritative, providing the final representation.
  + Subclass: A class derived from another class, inheriting its properties.
  + Persist: To store data permanently.
* **Software Packages**
  + Django: A high-level Python web framework.
* **Commands/Modules**
  + django.db.models.Model: The base class for Django models.
  + get(): Retrieves a single object from a database.
  + save(): Saves changes to an object in the database.
  + delete(): Deletes an object from the database.
* **Ideas/Techniques**
  + Object-Relational Mapping (ORM): Technique of mapping database tables to objects in code. Django's models provide an ORM layer.
* **Mistakes or Misleading Information**
  + The transcript implies that defining a model class directly creates a database table. **Clarification:** Model definitions need to be applied through migrations to make the actual database changes.

**Keywords**

* model
* database
* table
* CRUD
* Django
* SQL
* object
* class
* subclass
* attribute

**Written by Gemini**

[Model relationships | Coursera](https://www.coursera.org/learn/django-web-framework/supplement/3gkfi/model-relationships)

Short Summary: This lesson describes how Django models represent database tables and the types of relationships that can exist between models (one-to-one, one-to-many, many-to-many). It explains how Django implements these relationships for streamlined database interactions.

Detailed Breakdown:

* **Acronyms**
  + ORM (Object-Relational Mapping)
* **Technical Words**
  + Model: A Python class representing a database table.
  + Primary Key: A unique identifier for each row in a database table.
  + Foreign Key: A field in one table that references the primary key of another table, establishing a relationship.
  + Migration: A way to track and apply changes to Django models, affecting the database structure.
* **Rarely Used Words**
  + Redundancy: Unnecessary repetition of data.
  + Integrity: Accuracy and consistency of data.
  + Deferrable: Postponing an action (like constraint checking) until later in a database transaction.
* **Software Packages**
  + Django: A high-level Python web framework.
* **Commands/Modules**
  + models.Model: Base class for Django models.
  + models.IntegerField: Represents an integer field in a model.
  + models.CharField: Represents a string field in a model.
  + models.EmailField: Represents an email field in a model.
  + models.URLField: Represents a URL field in a model.
  + models.OneToOneField: Defines a one-to-one relationship between models.
  + models.ForeignKey: Defines a one-to-many relationship between models.
  + models.ManyToManyField: Defines a many-to-many relationship between models.
  + on\_delete: Specifies behavior when a related object is deleted (options include CASCADE, PROTECT, RESTRICT).
* **Ideas/Techniques**
  + Database Relationships: How database tables are connected to model database operations efficiently.
  + Object-Relational Mapping (ORM): The technique of mapping database tables to objects in code, which Django's models provide.
* keywords = ['model', 'database', 'table', 'CRUD', 'Django', 'SQL', 'object', 'class', 'subclass', 'attribute']
* **Mistakes or Misleading Information**
  + The lesson could be clearer about the fact that migrations are necessary to make model changes take effect in the actual database.

**Written by Gemini**

[Creating models | Coursera](https://www.coursera.org/learn/django-web-framework/lecture/yAFex/creating-models) 3-30-2024 1230

Short Summary: This video demonstrates how to create basic models in Django's models.py file. It covers defining model attributes, interacting with the model via the Django shell, and making simple updates to model data.

Detailed Breakdown:

* **Acronyms**
  + ORM (Object-Relational Mapping)
* **Technical Words**
  + Model: Python class representing a database table.
  + Attribute: A property or field within a model.
  + Field Type: Specifies the kind of data a model attribute holds (e.g., CharField for text, IntegerField for numbers).
  + Migration: Django's way of tracking and applying model changes to the database.
  + Django Shell: Interactive command-line interface for working with Django models.
  + Custom Method: Function defined within a model to customize its behavior.
* **Rarely Used Words**
  + Dunder method: Special method in Python (often with "\_\_" underscores) with predefined behavior.
* **Software Packages**
  + Django: High-level Python web framework.
* **Commands/Modules**
* python manage.py shell: Opens the Django shell. Ex. python manage.py shell
* models.CharField: Represents a text field in a Django model. Ex. name = models.CharField(max\_length=50)
* models.IntegerField: Represents an integer field in a Django model. Ex. age = models.IntegerField()
* from ... import ...: Python statement to import classes/modules. Ex. from myapp.models import Product
* Menu.objects.all(): Retrieves all objects from the Menu model. Ex. all\_menu\_items = Menu.objects.all()
* Menu.objects.create(): Creates a new object in the Menu model. Ex. new\_item = Menu.objects.create(name="Salad", price=8)
* Menu.objects.get(): Retrieves a single object, often by its primary key. Ex. pasta = Menu.objects.get(pk=1)
* **Ideas/Techniques**
  + Object-Relational Mapping (ORM): Technique of mapping database entities to objects in code.
  + Using the Django shell to interact with models.
* keywords = ['model', 'Django', 'database', 'Python', 'shell', 'import', 'create', 'attribute', 'object', 'migration']
* **Mistakes or Misleading Information**
  + The video suggests that models are directly usable in the shell without first adding them to 'INSTALLED\_APPS' in settings.py and running migrations. **Clarification:** These steps are essential before working with models.

**Written by Gemini**

[Migrations | Coursera](https://www.coursera.org/learn/django-web-framework/lecture/tMqae/migrations) 3-30-2024 1300

Short Summary: This video introduces migrations in Django, explaining how they're used to track and implement changes to database schemas that are reflected in Django models. It emphasizes the benefits of migrations for syncing, version control, and database maintenance within development teams.

Detailed Breakdown:

* **Acronyms**
  + ORM (Object-Relational Mapping)
  + RDBMS (Relational Database Management System)
  + SQL (Structured Query Language)
  + CLI (Command-Line Interface)
* **Technical Words**
  + Migration: A way to track and apply changes to Django models (and their corresponding database tables).
  + Schema: The structure of a database, defining tables, columns, and their relationships.
  + Version Control: System for tracking changes to code and files over time.
* **Rarely Used Words**
  + Alter: Modify or change something.
  + Repository: Central storage location for code and related files.
* **Software Packages**
  + Django: High-level Python web framework.
* **Commands/Modules**
  + python manage.py (Django management command): Provides various utility commands, including those for migrations (e.g., makemigrations, migrate). *Note: No specific usage example given in the transcript.*
* **Ideas/Techniques**
  + Database Migrations: The concept of using migrations to manage database changes in sync with code.
  + Object-Relational Mapping (ORM): Technique of mapping database tables to objects within code for easier interaction.
* keywords = ['Django', 'migration', 'model', 'database', 'schema', 'attribute', 'ORM']
* **Mistakes or Misleading Information**
  + The transcript implies that SQL knowledge is not needed at all when using Django migrations. **Clarification:** While migrations streamline the process, understanding basic database concepts and SQL can be helpful for troubleshooting and more complex changes.

**Written by Gemini**

[How to use migrations | Coursera](https://www.coursera.org/learn/django-web-framework/supplement/wcsZg/how-to-use-migrations) 3-30-2024 1329

Short Summary: This lesson explains how to use migrations within Django to manage changes to database schemas in tandem with model updates. It covers essential migration commands, the concept of migrations as version control, and the process of applying migrations.

Detailed Breakdown:

* **Acronyms**
  + RDBMS (Relational Database Management System)
  + SQL (Structured Query Language)
* **Technical Words**
  + Migration: Mechanism for tracking and applying model changes to the database.
  + Propagate: To spread or transmit (in this case, changes from models to the database).
  + Version Control: System for tracking changes to code and files over time.
* **Rarely Used Words**
  + Mechanism: A system of parts or processes working together.
* **Software Packages**
  + Django: High-level Python web framework.
  + SQLite: A type of database often used as the default in Django.
* **Commands/Modules**
  + makemigrations: Generates migration scripts based on model changes. Ex: python manage.py makemigrations
  + migrate: Applies pending migration scripts to the database. Ex: python manage.py migrate
  + showmigrations: Displays the status (applied or unapplied) of migrations. Ex: python manage.py showmigrations
  + sqlmigrate: Shows the SQL queries a migration script will execute. Ex: python manage.py sqlmigrate app\_name migration\_number
* **Ideas/Techniques**
  + Database Migrations: The concept of using migrations to manage database changes in sync with application code changes
  + Django Migrations as Version Control: How migrations help track database schema history.
  + keywords = ['Django', 'migration', 'database', 'schema', 'model', 'SQLite', 'makemigrations', 'migrate', 'showmigrations', 'sqlmigrate']
* **Mistakes or Misleading Information**
  + While the lesson does a good job explaining migrations, it would benefit from further clarification that SQL knowledge, while not strictly essential, can be helpful for complex migrations or troubleshooting.

**Written by Gemini**

Notes:

The lesson skips over adding ‘myapp’ to the installed apps. Its very possible they assume you do this automatically. Maybe its good to make the error and figure out what the problem is.

To avoid running into issues migrating add a default age.

[Working with Migrations | Coursera](https://www.coursera.org/learn/django-web-framework/lecture/V0P1d/working-with-migrations) 3-30-2024 1640

**Short Summary:** This video tutorial explains how to use Django migrations to manage changes to database schemas. It demonstrates the use of makemigrations, migrate, and sqlmigrate commands, and emphasizes the importance of migrations for version control within Django development.

**Detailed Breakdown**

* **Acronyms**
  + SQL (Structured Query Language)
* **Technical Words**
  + Migration: The mechanism for tracking and applying changes to models (and their associated database tables).
  + Version Control: System for tracking changes to code and files over time.
  + Attribute: A property or characteristic of a model (equivalent to a database column).
* **Rarely Used Words**
  + Commit: The act of saving changes in a version control system.
* **Software Packages**
  + Django: High-level Python web framework.
* **Commands/Modules**
  + makemigrations: Generates migration scripts based on model changes. Ex: python manage.py makemigrations myapp
  + migrate: Applies pending migration scripts to the database. Ex: python manage.py migrate
  + showmigrations: Displays the status (applied or unapplied) of migrations. Ex: python manage.py showmigrations
  + sqlmigrate: Shows the SQL queries that a migration would execute. Ex: python manage.py sqlmigrate app\_name migration\_number
* **Ideas/Techniques**
  + Django Migrations: The concept of using migrations to manage database changes in sync with code.
  + Database Version Control: The use of migrations to track schema changes over time.
* Keywords = ["Django", "migrations", "database", "schema", "version control", "model", "attribute", "SQL", "makemigrations", "migrate", "showmigrations", "sqlmigrate"]

**Mistakes or Misleading Information**

* **Imprecise Language:** The transcript sometimes uses phrases like "Django displays a message..." instead of accurately stating that the makemigrations or migrate commands produce output.
* **Potential for Confusion:** The explanation of makemigrations stating "...it creates a database such as this one named db.sqlite3" could be clearer. While migrations *drive* the database creation process, it's important to understand that Django handles database setup as part of the initial project creation.

**Written by Gemini**

[A history of changes | Coursera](https://www.coursera.org/learn/django-web-framework/lecture/hfFTH/a-history-of-changes) 3-30-2024 1655

Short Summary: This video explains how Django's migration syste

m tracks model changes and provides version control for database schemas. It details the migration process, the Django migrations table, and the structure of migration files.

Detailed Breakdown:

* **Acronyms**
  + SQL (Structured Query Language)
* **Technical Words**
  + Migration: The mechanism for tracking and applying model changes to a database.
  + Schema: The structure of a database, defining tables, columns, and relationships.
  + Version Control: System for tracking changes to code and files.
  + Attribute: A property or characteristic of a model.
  + Index: A data structure that improves the speed of data retrieval operations on a database table.
* **Rarely Used Words**
  + Incremental: Increasing by small degrees or regular amounts.
* **Software Packages**
  + Django: High-level Python web framework.
* **Commands/Modules**
  + makemigrations: Generates migration scripts based on model changes. Ex: python manage.py makemigrations myapp
  + migrate: Applies pending migration scripts to the database. Ex: python manage.py migrate
  + showmigrations: Displays the status (applied or unapplied) of migrations. Ex: python manage.py showmigrations
* **Ideas/Techniques**
  + Django Migrations: Using migrations to manage database changes in sync with application code.
  + Migration File Structure: How migration files store instructions for database schema changes.
  + Django migrations table: How Django tracks applied migrations.

**Keywords**=[Django, migrations, schema, database, version control, model, SQL, attribute, index, makemigrations, migrate, showmigrations]

* **Mistakes or Misleading Information**
  + The phrase "Django directly provides a history of changes to the code base" might be slightly misleading. While migrations reflect model changes, they don't inherently capture all codebase modifications outside the models.

**Written by Gemini**

[Exercise: Models and migrations | Coursera](https://www.coursera.org/learn/django-web-framework/ungradedLab/8lLeM/exercise-models-and-migrations) 3-31-2024 0641

**Short Summary**

This lab teaches you how to create a Django model (representing a drinks menu) and use migrations to update your database structure. You'll also learn how to inspect the generated database.

**Detailed Breakdown**

* **Concepts**
  + Django Models
  + Database Migrations
  + MVT Pattern (Model View Template)
* **Technical Terms**
  + Model: A Python class representing a database table.
  + Field: An attribute within a model, corresponding to a database column.
  + Migration: A file describing changes to be made to the database schema.
  + SQLite: A lightweight database engine often used for Django development.
* **Commands**
  + python manage.py makemigrations: Creates migration files based on model changes. Ex: python manage.py makemigrations
  + python manage.py migrate: Applies changes described in migration files to the database. Ex: python manage.py migrate
  + python manage.py shell: Opens the Django shell for interacting with models. Ex: python manage.py shell
* **Software**
  + Django: A high-level Python web framework.

Keywords = ['Django', 'models', 'migrations', 'database', 'SQLite', 'makemigrations', 'migrate', 'shell']

**Mistakes or Misleading Information**

The instructions seem clear and accurate. However, here are a few points to clarify:

* **Database Choice:** Emphasize that while SQLite is convenient for development, larger projects might use more robust databases like PostgreSQL or MySQL.
* **MVT in Detail:** Briefly explain that models handle data, views handle the logic of what's displayed, and templates control the presentation (HTML).

**Written by Gemini**

[Models using Foreign Keys | Coursera](https://www.coursera.org/learn/django-web-framework/lecture/G8ZS9/models-using-foreign-keys) 3-31-2024 0921

**Short Summary:** This video demonstrates how to implement a one-to-many relationship in Django using foreign keys. It teaches how to connect models for representing menu items and menu categories, enabling organized data within a restaurant website.

**Detailed Breakdown**

* **Acronyms**
  + ORM (Object-Relational Mapping)
* **Technical Words**
  + **Database:** A structured collection of data.
  + **Table:** A way of organizing data within a database, with rows and columns.
  + **Foreign Key:** A database column that references the primary key of another table, creating a relationship.
  + **One-to-Many Relationship**: A database relationship where one record in a table can be associated with multiple records in another table.
  + **Migration:** A change to a database schema managed by Django.
* **Software Packages**
  + **Django:** A Python web framework for building web applications.
* **Commands/Modules**
  + **python manage.py shell:** Opens the interactive Django shell for working with models.
  + **python manage.py migrate:** Applies migrations to synchronize database changes with Django models.
* **Ideas/Techniques**
  + **Model Relationships:** Using foreign keys to establish relationships between different models in your Django application.

Keywords = ['Django', 'model', 'foreign key', 'one-to-many', 'database', 'table', 'relationship', 'menu', 'category', 'migration', 'python manage.py shell', 'python manage.py migrate']

**Mistakes or Misleading Information**

* **Minor Omission:** The video mentions 'subcategories' but doesn't demonstrate implementing them. This could be clarified by explaining that subcategories would involve another model and foreign key relationship.

**Written by Gemini**

Notes:

I added in values from the lesson. This was a learning experience since my sqlite installation seemed to be nonstandard. I finally installed a new installation at c:\sqlite and changed my environment values to reference it. Here is a short list of my learning:

Locating previous installations with PowerShell: **Get-ChildItem -Recurse -Filter "sqlite3.exe"**

* Checking drive space in PS because I was concerned about space left after so many virtual environments and installations of sqlite, python, Django ect. **Get-PSDrive**
* **For some mote info on the disks. Get-Volume**

The end use of related\_name was very confusing to me. I spent an hour researching why it was not changing “category\_id\_id” to Category\_name or why the index number displayed was not changing from1-3 to the actual string values. I could not see any use. I learned that ‘related\_name=’ is used as a reerence in the ORM code. Examples:

* path('menu/<str:category\_name>/', menu\_items\_by\_category, name='menu-items-by-category'),
* ]
* def menu\_items\_by\_category(request, category\_name):
* if category:
* menu\_items = category.category\_name.all()
* return render(request, 'menus/category\_menu\_items.html', context)
* <h2>Menu Items for {{ category.name\_category }}</h2>
* <ul>
* {% for menu\_item in category.category\_name.all %}
* <li>{{ menu\_item.menu\_item }} - ${{ menu\_item.price }}</li>
* {% endfor %}
* </ul>

I practiced adding values to a db in sqlite.

All in all this lesson took a long time, but I learned and I’m smurter.

[Exercise: Models using Foreign Keys | Coursera](https://www.coursera.org/learn/django-web-framework/ungradedLab/MtTPZ/exercise-models-using-foreign-keys)

**Summary:** This lab teaches how to create models in Django and establish a one-to-many relationship using foreign keys. The learner creates models for drink categories and individual drinks, linking them to simulate a real-world menu structure.

**Detailed Breakdown:**

* **Objectives**
  + Create a DrinksCategory model
  + Create a Drinks model with a foreign key referencing DrinksCategory
* **Technical Terms**
  + **Foreign Key:** A field in a database table that references the primary key of another table, establishing a parent-child relationship.
  + **One-to-Many Relationship:** A database relationship where one record in a table can be associated with multiple records in another table. (Example: One drink category can have many drinks)
  + **Models (in Django):** Python classes that represent database tables, defining their structure.
  + **MVT pattern:** Model-View-Template, the architectural pattern used by Django.
* **Software/Libraries**
  + **Django:** Python web framework for building web applications. Ex. from myapp.models import DrinksCategory
  + **SQLite:** Embeddable database engine often used with Django.
* **Commands**
  + **python manage.py shell:** Opens the Django shell for interacting with project models.
  + **python manage.py makemigrations:** Creates instructions for updating the database based on model changes.
  + **python manage.py migrate:** Applies the changes from 'makemigrations' to the database.
* **Important Ideas**
  + **Model relationships:** How to connect data models to represent real-world scenarios.
  + **Database interaction (Django):** Creating models, accessing them from the shell, and updating the database.

**Keywords = ['Django', 'models', 'foreign key', 'one-to-many relationship', 'SQLite', 'python manage.py shell', 'python manage.py makemigrations', 'python manage.py migrate']**

**Notes:**

* The instructions could be more concise (less conversational).
* Some basic knowledge of databases is assumed.

Let me know if you'd like any of the explanations expanded or have further analysis requirements!

Written by Gemini

Notes:

I ran into a weird issue after getting my sqlite3 path setup so I could run sqlite form the PS command line I lost the ability to use sqlite viewer by right clicking on the db and open database. I get this error:  
A screenshot of a computer program

Description automatically generated

Heres the full output:  
[3:34:09 PM][vscode-sqlite][ERROR] Failed to open database 'c:\Users\<myfoldername>\<myfoldername>\meta-backend-developer-professional-certificate\django-web-framework\week#3\excercise-models-using-foreign-keys\db.sqlite3': Parse error near line 4: no such column: table

  aster                                 WHERE (type="table" OR type="view")

                                      error here ---^

I give up because eventually Im switching to MySQL.

For the past 5 or more lessons, I’ve been closing all windows and starting to make directories and loading environments from a blank slate for muscle memory.

[Object Relationship Mapping - ORM | Coursera](https://www.coursera.org/learn/django-web-framework/supplement/S6VCq/object-relationship-mapping-orm) 4-1-2024 1055

Short Summary: This text explains Object-Relational Mapping (ORM), a technique that lets you interact with databases using object-oriented programming concepts instead of raw SQL queries.

**Acronyms**

* **ORM:** Object-Relational Mapping
* **SQL:** Structured Query Language
* **CRUD:** Create, Read, Update, Delete (basic database operations)

**Terms**

* **Object-Oriented Programming (OOP):** A programming paradigm where data and code are organized into objects.
* **Relational Database:** A database that stores data in tables with defined relationships.
* **Primary Key:** A unique identifier for a record in a database table.

**Technical Words**

* **Class:** A blueprint for objects in object-oriented programming.
* **Object:** An instance of a class, representing a real-world entity.
* **Attribute:** A property or characteristic of an object.
* **Field:** A column in a database table.
* **Manager:** In Django, a class that handles database interactions for a model.
* **QuerySet:** A collection of objects retrieved from the database.
* **Migration:** A system for tracking and applying changes to database schemas.

**Rarely Used Words**

* **Propagate:** To spread or transmit across a structure.

**Software Packages**

* **Django:** A Python-based web framework.

**Commands**

* **python manage.py shell:** Opens a Django interactive shell.

**Modules**

* **django.db.models:** Contains the core components of Django's ORM.
* **models.Model:** The base class for Django models.

**Ideas**

* **Object-Relational Mapping (ORM):** The concept of bridging the gap between object-oriented programming and relational databases.
* **Django Models:** Python classes representing database tables.
* **QuerySets:** Representing database queries in an object-oriented way.

**Techniques**

* **Creating Models:** Defining Python classes to map to database tables.
* **Generating Migrations:** Creating instructions to modify the database schema.
* **Manipulating Data with QuerySets:** Using the ORM layer for creating, reading, updating, and deleting database records.

**Keywords**=['ORM', 'SQL', 'Django', 'model', 'QuerySet', 'object', 'database', 'CRUD']

[Using ORM | Coursera](https://www.coursera.org/learn/django-web-framework/lecture/d43y4/using-orm)

**Short Summary:** This video transcript explains how Django, a web development framework, uses Object-Relational Mapping (ORM) to simplify database interactions for developers. It introduces the concept of QuerySets for retrieving and manipulating data.

**Acronyms**

* **ORM:** Object-Relational Mapping
* **API:** Application Programming Interface
* **SDLC:** Software Development Life Cycle

**Terms**

* **Database:** A structured collection of data.
* **SQL (Structured Query Language):** A language for interacting with databases.
* **Web application:** Software that runs on a web server and can be accessed through a web browser.
* **Django:** A Python-based web development framework.
* **QuerySet** A collection of objects from a Django model, representing the results of a database query.

**Technical Words**

* **Object-oriented programming:** A programming paradigm where data and code are organized into objects.
* **Data objects:** Objects in code that represent entities as data (e.g., a customer object might store name, reservation).
* **Shell:** Command-line interface for interacting with an operating system or program.
* **Relational database:** A database that organizes data in tables with relationships between them.
* **Abstraction layer:** A layer of code that hides complex details, providing a simpler interface.
* **Structured map:** Represents how objects in code relate to database tables and fields.
* **Prefixed query set:** The format Django displays when representing database results in the shell.

**Rarely Used Words (for a 12-year-old)**

* **Paradigm:** A model or pattern of how something works.
* **Prefixed:** Having something attached to the beginning.
* **Entries:** Individual records or pieces of data in a collection.
* **Corresponding:** Matching or relating to.

**Software Packages**

* **Django:** Python-based web framework for building web applications.
* **VS Code:** A popular code editor from Microsoft.

**Commands**

* **python3 manage.py shell:** Starts the Django interactive shell.
* **from myapp.models import Customer:** Imports the Customer model.
* **customer.objects.get():** Retrieves a single Customer object by its ID.
* **customer.objects.filter():** Filters Customer objects based on criteria.
* **customer.objects.all():** Retrieves all Customer objects.

**Modules**

* **models.py:** A Django file where data models are defined.

**Ideas**

* **Object-Relational Mapping (ORM):** A technique for bridging the gap between object-oriented code and relational databases.
* **Lazy Evaluation:** A strategy where computation (in this case, database queries) is delayed until the result is actually needed. (FYI not in the video)

**Techniques**

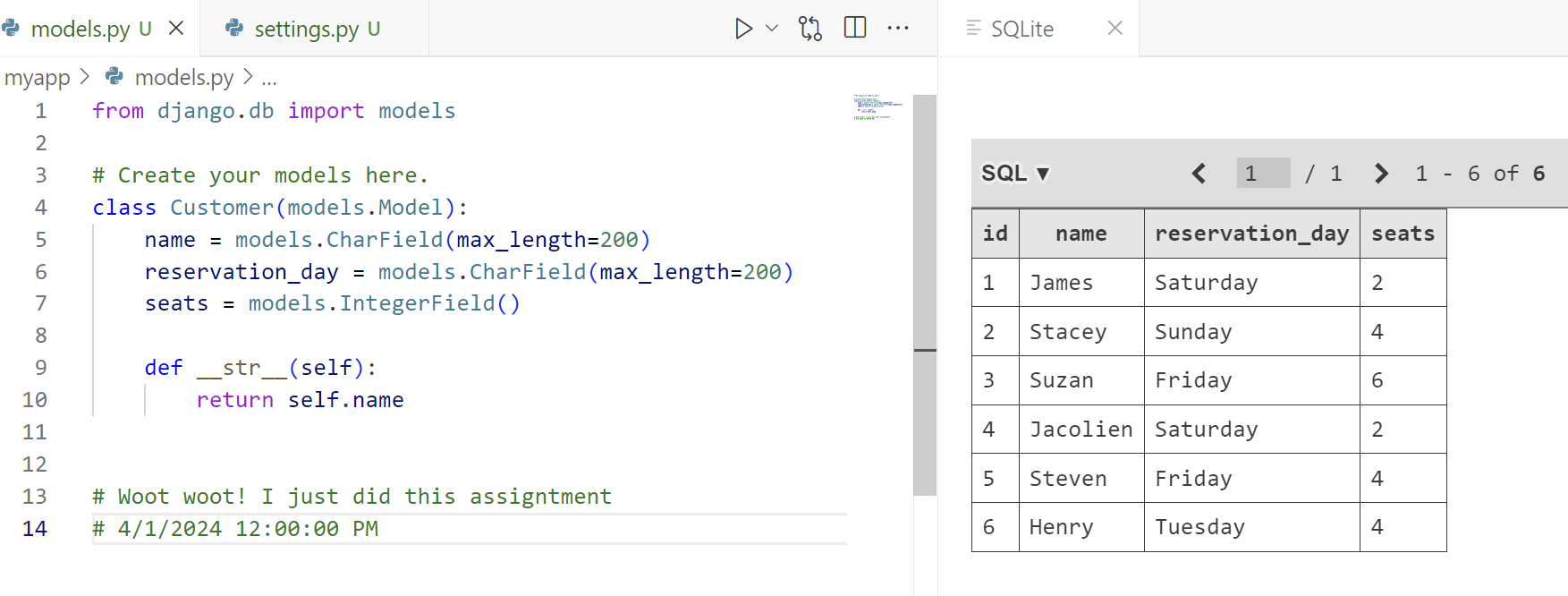
* **Django QuerySet API:** The set of methods provided by Django to interact with the database through model objects.

**Keywords**=["Django", "ORM", "SQL", "database", "web application", "object-oriented programming", "QuerySet", "shell", "VS Code"]

**Note:** This breakdown was generated by an AI language model(Gemini) and may have some imperfections.

Note:

I honestly do not know how many times I am going to forget to add 'myapp.apps.MyappConfig’ to the settings.py installed apps section. I’m constantly forgetting, getting an error and then thinking oh ya!



[Knowledge Check: Models & Migrations | Coursera](https://www.coursera.org/learn/django-web-framework/quiz/4Qeoz/knowledge-check-models-migrations) 4-1-2024 1200

* Easy

[Forms | Coursera](https://www.coursera.org/learn/django-web-framework/lecture/jUmuw/forms) 4-1-2024 1245

Summary: This transcript describes how Django, a web application framework, simplifies form creation and processing. Developers use form classes to represent expected form data and automatically generate corresponding HTML elements, improving efficiency and reducing errors.

Short summary: This transcript explains Django's approach to forms, allowing developers to define form structures in classes for automatic HTML generation and data handling.

**Acronyms:**

* **SDLC:** Software Development Life Cycle: A framework outlining stages involved in building software.

**Terms**

* **Web application:** Software that runs on a web server and can be accessed by users through a web browser.
* **HTML:** Hypertext Markup Language. The standard markup language for structuring web pages.
* **Django:** A Python-based web framework that simplifies complex website development.
* **Server-side code:** Code that executes on the web server, as opposed to in the user's browser (client-side).
* **Backend:** The part of a web application that handles data processing, logic, and interaction with databases.
* **Templating:** A technique where dynamic data is inserted into placeholder elements within a basic HTML structure.

**Technical Words**

* **Authentication:** Verifying a user's identity.
* **Input:** Data entered by the user into a form.
* **Checkbox:** An interactive element allowing users to select multiple options.
* **Post request:** An HTTP method used to send data to the server.
* **Processing:** Handling and manipulating data.
* **Label:** Text associated with a form element to describe its purpose.
* **Action:** The URL the form data is sent to upon submission.
* **Method:** The HTTP method (e.g., POST, GET) used to submit the form.
* **Attribute:** Properties or characteristics of HTML elements.
* **Validation:** Checking if the data entered by the user meets specific criteria.
* **Class:** A blueprint for creating objects in object-oriented programming.
* **Object-oriented programming:** A programming paradigm based on the concept of objects, which encapsulate data and behavior.
* **Subclass:** A class that inherits properties and methods from another class (its parent class).
* **Persist:** To store data permanently.
* **Model:** A representation of a database table in Django.

**Rarely Used Words**

* **End-user:** The person who interacts with the final software product.
* **Conditional flow:** Different execution paths in a program based on specific conditions.
* **Templating:** A technique where dynamic data is inserted into placeholder elements within a basic HTML structure.

**Software Packages**

* **Django:** A Python-based web framework that simplifies complex website development.

**Modules**

* **forms.py:** A Python file in Django applications where form classes are typically defined.

**Ideas**

* **Form abstraction:** Using classes to define form structure and behavior, making management and processing easier.
* **Model-based form generation:** Automatically creating forms directly from Django models to ensure consistency and reduce errors.

**Techniques**

* **Form rendering:** The process of converting a Django form class into HTML elements.
* **Form validation:** Checking if user-entered data is correct and meets specific requirements.
* **Form processing:** Handling the data submitted from a form on the server-side.

**Keywords**=["web application", "user details", "login", "authentication", "registration details", "order details", "HTML", "form elements", "inputs", "checkboxes", "server", "processing", "Django", "form class", "POST request", "backend", "input", "type text", "type submit", "form action", "form method", "errors", "conditional flows", "Django models", "persisting data", "model forms"]

Gemini

Note:

This instruction in the learning was worded awkwardly to me.

The **Customer.objects** gives the Manager of the model. It handles all the CRUD operations on the database table. For example, an object can also be created with the **create()** method as follows:  
I think its saying this:

The **Customer.objects** attribute provides access to a Django model manager. This manager offers methods to interact with the database for your 'Customer' model, including creating new records (create()), retrieving data (get(), filter(), etc.), updating, and deleting records.

[Working with Django form fields and data types | Coursera](https://www.coursera.org/learn/django-web-framework/lecture/rvnXj/working-with-django-form-fields-and-data-types) 4-2-2024 1034

**Short Summary**

This video introduces Django form fields for collecting user data in web applications. It covers field types like CharField, EmailField, IntegerField, and others, along with customization using widgets and arguments.

**Detailed Breakdown**

* **Acronyms**
  + HTML (HyperText Markup Language)
* **Technical Words**
  + **Form:** A structure on a web page that allows users to input data.
  + **Field:** An individual input element within a form.
  + **Django:** A high-level Python web framework used to build web applications.
  + **Validation:** The process of checking if user input meets specific criteria.
  + **Widget:** A visual component that renders a form field in a specific way.
  + **Argument:** A value passed to a field to modify its behavior.
* **Rarely Used Words**
  + **Render:** To generate or display something on-screen.
* **Software Packages**
  + **Django:** (See previous definition)
* **Commands/Modules**
  + **forms.py:** A conventional Django file for storing form definitions.
  + **forms:** A Django module providing form-related classes and functions.
  + **forms.CharField:** A Django form field for handling strings. Ex. forms.CharField(max\_length=50)
  + **forms.EmailField:** A Django form field to handle email addresses. Ex. forms.EmailField(required=True)
  + **forms.IntegerField:** A Django form field for handling whole numbers. Ex. forms.IntegerField(min\_value=1)
  + **forms.Textarea:** A Django widget to render a multi-line text input area. Ex. forms.Textarea(attrs={'rows': 5})
  + **forms.NumberInput:** A Django widget to render a number input field. Ex. forms.NumberInput()
  + **forms.RadioSelect:** A Django widget to render choices as radio buttons. Ex. forms.RadioSelect()
* **Ideas/Techniques**
  + **Form Design:** The process of structuring and arranging input elements for user data collection.
  + **Form Validation:** Ensuring that user-submitted data is in the correct format and meets requirements.

**Keywords** Keywords = ['Django', 'forms', 'form fields', 'CharField', 'EmailField', 'IntegerField', 'FileField', 'Textarea', 'widget', 'validation', 'HTML', 'RadioSelect', 'NumberInput']

**Mistakes/Misleading Information**

* **MultipleChoiceField:** The transcript seems to imply it corresponds to <select> and <option>. It can, but it's more flexible and often used with radio buttons or checkboxes.
* **FileField:** Incorrectly states it's equivalent to the <input type="file"> element. While related, FileField brings more functionality for file handling and validation.

**Written by Gemini**

[Django fields | Coursera](https://www.coursera.org/learn/django-web-framework/supplement/n2FLA/django-fields) 4-2-2024 1126

**Short Summary**

This lesson provides an introduction to Django models. It explains how models define database tables, field types (e.g., CharField, IntegerField), and how to establish relationships between models (e.g., one-to-one, many-to-many).

**Detailed Breakdown**

* **Acronyms**
  + ORM (Object-Relational Mapping)
* **Technical Words**
  + **Model:** A Python class representing a database table.
  + **Field:** An attribute of a model, mapping to a database column.
  + **Primary Key:** A unique identifier for each record in a table.
  + **Django ORM:** The layer in Django that handles database interactions.
  + **Relationship (Database):** A connection between database tables.
* **Rarely Used Words**
  + **Auto-incrementing:** A value that automatically increases with each new record.
* **Commands/Modules**
  + **django.db:** Django module for database access.
  + **models.py** Standard file in Django apps for model definitions.
  + **django.models.Model:** Base class for Django models.
* **Ideas/Techniques**
  + **Object-Relational Mapping (ORM):** Concept of bridging object-oriented programming with relational databases.
  + **Database Relationships:** How different database tables connect.

**Keywords** Keywords = ['Django', 'models', 'ORM', 'database', 'fields', 'CharField', 'IntegerField', 'ForeignKey', 'OneToOneField', 'ManyToManyField', 'relationships']

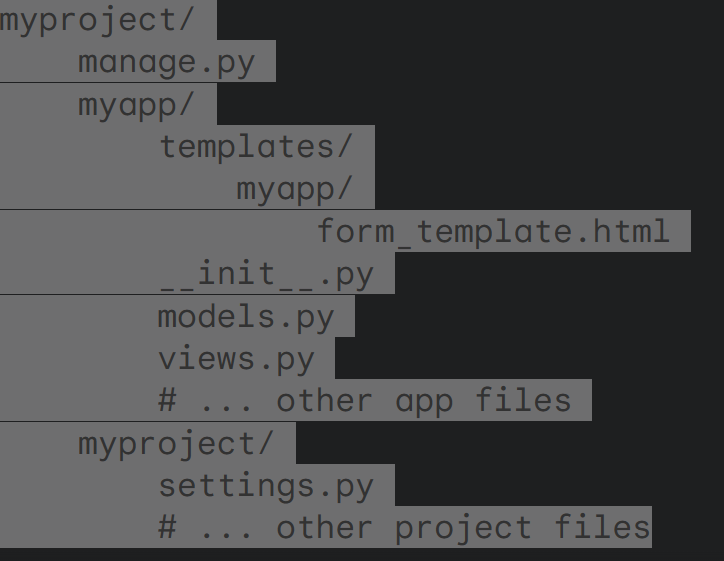
**Mistakes/Misleading Information**

* **Minor Imprecision:** The statement "The Django ORM enables storing and retrieving data in tables, not by executing raw SQL queries..." is generally true but slightly oversimplified. While the ORM *primarily* handles interactions without raw SQL, Django lets you execute SQL directly when needed.

**Written by Gemini**

Notes:

I followed the course in on my computer in VSC. I made mistakes in where I placed the original templates. I fixed it, but still had misconfigured the views render line “return render(request, 'myapp/form\_template.html', {'form': form})” I had “return render(request, 'form\_template.html', {'form': form})” With apps set to true in thesettings.py file Django will look in the app folder for ‘templates/myapp/form\_template.html’.



[Form API | Coursera](https://www.coursera.org/learn/django-web-framework/supplement/xphM6/form-api)

**Short Summary**

This lesson introduces the Django Form API, which simplifies creating and processing web forms. It explains how Django forms map to HTML, including fields (e.g., CharField, IntegerField), validation, and how to handle submitted form data.

**Detailed Breakdown**

* **Acronyms**
  + API (Application Programming Interface)
  + ORM (Object-Relational Mapping)
* **Technical Words**
  + **Form:** A structured way to collect user input on a web page.
  + **Field:** An individual input element within a form.
  + **Validation:** The process of ensuring user-submitted data meets specific criteria.
  + **HTML:** HyperText Markup Language – the language used to structure web pages.
  + **Django Form:** A Django class used to define and render forms within the framework.
* **Rarely Used Words**
  + **Render:** To generate or display something on-screen.
* **Software Packages**
  + **Django:** A Python-based web framework for creating web applications.
* **Commands/Modules**
  + **django.forms:** Module containing Django's forms-related classes and functions.
  + **forms.py:** A conventional file in Django apps to store form definitions.
* **Ideas/Techniques**
  + **Object-Relational Mapping (ORM):** The concept of bridging object-oriented programming with relational databases.
  + **Form Design:** Structuring forms with appropriate input elements for effective user interaction.
  + **Form Validation:** Enforcing rules to ensure the quality and correctness of user-submitted data.

Keywords = ['Django', 'forms', 'Form API', 'HTML', 'fields', 'CharField', 'IntegerField', 'validation', 'rendering']

**Mistakes/Misleading Information**

While generally accurate, the lesson could be improved in a couple of areas:

* **JavaScript for Validation:** The statement that HTML lacks robust validation implies JavaScript cannot be used. It's essential to note that while Django Forms provide server-side validation, JavaScript is often used for immediate client-side validation.
* **Pillow Library**: The lesson states that the Pillow library is a hard requirement for ImageField. While recommended, it's technically optional; Django can do basic image validation on its own.

**Written by Gemini**

[Creating Forms | Coursera](https://www.coursera.org/learn/django-web-framework/lecture/EDCAg/creating-forms) 4-1-2024 1247

Short Summary: This transcript explains how to build basic web forms in Django. It covers creating forms using Django's Form class, rendering forms within templates, and adding basic form validation.

**Acronyms**

* **CSRF:** Cross-Site Request Forgery

**Terms**

* **Form:** A structured way to collect user input on a website.
* **Form Element:** Individual input fields within a form (e.g., text box, dropdown).
* **Validation:** Checking if the data a user enters into a form is correct and in the expected format.

**Technical Words**

* **Class:** Blueprint for objects in object-oriented programming.
* **Import:** A statement to bring in code from another file or module.
* **Attribute:** A property or characteristic of an object.
* **Character Field:** A field in a database or form that stores text.
* **Choice Field:** A form element that presents the user with a list of options to select from.
* **View:** A function in Django that handles web requests and generates responses.
* **Template:** An HTML file with placeholders for displaying dynamic content.
* **Render:** The process of combining a template with data to generate an HTML page
* **Parameter:** An argument passed to a function to modify its behavior.
* **Inline Styling:** Adding CSS styles directly within HTML elements.

**Rarely Used Words**

* **Iterable:** In Python, something that can be looped over (e.g., a list or tuple).

**Software Packages**

* **Django:** A Python-based web framework.

**Commands**

* **python manage.py runserver:** Starts the Django development server.

**Modules**

* **django.forms:** Contains classes and tools for working with forms in Django.

**Ideas**

* **Form Creation:** Defining the structure of a form using Django's Form class.
* **Form Rendering:** Displaying a form on a web page using Django Templates.
* **Form Validation:** Ensuring that user-submitted data is valid.
* **Cross-Site Request Forgery (CSRF) Protection:** A security mechanism to prevent unauthorized form submissions.

**Techniques**

* **Using**forms.Form**class:** Creating a form object in forms.py.
* **Rendering Forms in Templates:** Using {{ form }} or {{ form.as\_p}} to display the form in an HTML template.
* **Basic Validation:** Setting the required attribute and using help\_text for form fields.
* **Inline CSS:** Adding simple styles directly within HTML elements.

**Keywords=**['Django', 'form', 'class', 'template', 'render', 'validation', 'CSRF']

By Gemini

Note:

I mistakenly used {(crsf\_token)} instead of {{ crsf\_token }}. My bad eyes saw ().

I am learning many people structure the templates folders differently

[Model Form | Coursera](https://www.coursera.org/learn/django-web-framework/lecture/4jY5c/model-form) 4-2-2024 1400

Short Summary: This transcript explains how to use Django's ModelForms to create forms that directly save user-entered data into a database table.

**Acronyms**

* **SQLITE:** A lightweight database engine often used with Django

**Terms**

* **Database Table:** A structured way to store data in a database.
* **POST Method:** An HTTP method used to submit data to a server.

**Technical Words**

* **ModelForm:** A Django class that combines the functionality of forms and models for easy form creation and data saving.
* **Meta Class:** A class within a class that provides configuration information.
* **Conditional Statement:** Code that executes different sections based on whether a condition is true or false (e.g., if...else).
* **Migration:** Changes made to the database schema that are tracked by Django.

**Rarely Used Words**

* **Implementation:** The process of putting a plan or design into action.

**Software Packages**

* **Django:** A Python-based web framework.

**Modules**

* **django.forms:** Contains classes for working with forms in Django.
* **.models** (Custom): The module within a Django app where database models are defined.

**Ideas**

* **ModelForm Integration:** Combining forms and models for efficient data handling.
* **Data Validation:** Checking if data submitted in a form is in the correct format.
* **Database Interaction:** Saving form data into a database.

**Techniques**

* **Creating a ModelForm Class:** Defining a class that inherits from ModelForm and specifies the associated model.
* **Handling Form Submission (POST request):** Writing view code to process form data submitted via the POST method.
* **Saving Form Data:** Using the form.save() method to save data to the database.

**Keywords=** ['Django', 'ModelForm', 'model', 'form', 'database', 'POST', 'save' ]

By Gemini

Note:  
I am improving. I'm getting quicker at identifying and resolving runtime errors, beyond the straightforward syntax errors caught during compilation.

[Exercise: Working with Forms | Coursera](https://www.coursera.org/learn/django-web-framework/ungradedLab/oRIG2/exercise-working-with-forms) 4-2-2024 1506

Short summary: This lab teaches how to create Django ModelForms, connect them to database models, and use them to collect and store user-submitted data.

**Acronyms:**

* **SDLC:** Software Development Life Cycle: A framework outlining stages involved in building software.

**Terms**

* **Django:** Python-based web framework for rapid development.
* **ModelForm:** Django class for generating forms directly from database models.
* **Database Table:** Organized structure for storing data within a database.

**Technical Words**

* **Model:** Representation of a database table in Django.
* **Field:** Individual data element within a model.
* **Arguments:** Values passed to modify field behavior (e.g., 'max\_length').
* **Import:** Bringing external code modules into your current file.
* **Class:** Blueprint for creating objects in object-oriented programming.
* **Attribute:** Characteristic or property of an object.
* **View Function:** Function in Django that handles web requests and generates responses.
* **Template:** HTML file with placeholders for dynamically inserted data.
* **Migration:** Changes to database structure.
* **URL Configuration:** Mapping of URL patterns to view functions.

**Rarely Used Words**

* **Proof of concept:** Demonstration of a project's feasibility.

**Software Packages**

* **Django:** Python-based web framework.
* **SQLite:** Lightweight database engine often used for Django development.

**Commands**

* **makemigrations:** Django command to generate migration files.
* **migrate:** Django command to apply migrations to the database.
* **runserver:** Django command to start the development server.

**Modules**

* **models.py:** File for defining Django models.
* **forms.py:** File for defining Django forms.
* **admin.py:** File for customizing the Django admin interface.
* **views.py:** File for defining Django view functions.
* **urls.py:** File for configuring URL patterns.

**Ideas**

* **Form-based Data Collection:** Using forms as a structured interaction for users to submit data.
* **Data Persistence:** Storing data in a database for later use.
* **Model-Driven Development:** Forms automatically derived from models reduce manual code and errors.

**Techniques**

* **ModelForm Creation:** Defining a Django ModelForm to map to a database model.
* **Field Customization:** Using arguments to control field types, lengths, etc.
* **Data Validation:** Enforcing rules on user-submitted data (not explicitly shown in this lab).
* **Database Interaction:** Saving form data to the database.

**Keywords**

["Django", "ModelForm", "model", "field", "import", "class", "view", "template", "migration", "URL", "database"]

These notes written by Gemini

Note:

Yet again I had difficulty getting the template in the correct folder.

This is my configuration :

* myapp\templates\myapp\booking.html
* return render(request, "myapp/booking.html", context)
* app ----- path('booking/', views.form\_view),
* project - path('', include('myapp.urls'))

[Knowledge Check: Model and Forms | Coursera](https://www.coursera.org/learn/django-web-framework/quiz/MIZX1/knowledge-check-model-and-forms) 4-2-2024 1525

100%

[Django Admin | Coursera](https://www.coursera.org/learn/django-web-framework/lecture/G9EVk/django-admin) 4-2-2024 1538

Short summary: This transcript discusses Django's built-in admin interface, which provides a convenient way for site managers to add, edit, and delete content stored in a web application's database. It outlines how to create an admin user and navigate the interface's basic features.

**Acronyms:**

* **SDLC:** Software Development Life Cycle: A framework outlining stages involved in building software.

**Terms**

* **Web Application:** Software that runs on a web server, accessed by users through a web browser.
* **Database:** An organized collection of data, often stored and accessed electronically.
* **Admin/Administration Site:** A backend interface within a web application, allowing designated users to manage data and settings.

**Technical Words**

* **Metadata:** Data providing information about other data.
* **Interface:** A point of interaction between components, such as a graphical user interface (GUI) or an application programming interface (API).
* **Project:** In Django, a collection of configuration and code for a web application
* **Model:** A representation of a database table within a Django application.
* **Terminal:** A text-based interface for interacting with a computer's operating system.
* **Utility**: A software tool designed to perform a specific task.
* **Credentials**: Authentication details (e.g., username and password)
* **Prompt:** A message or symbol presented by a computer program, requesting user input.
* **Form:** An interface element for collecting user input within a web application.

**Rarely Used Words**

* **Designated:** Selected or assigned for a specific role.

**Software Packages**

* **Django:** A Python-based web framework for rapid development.

**Commands**

* **python3 manage.py createsuperuser:** Django command to create an administrative user for the admin interface.
* **python3 manage.py runserver:** Django command to start the development server.

**Modules**

* **django.contrib.admin:** Django module providing the admin interface.

**Ideas**

* **Data Management:** Using the admin interface to add, edit, and delete data stored within the application's database.
* **User Management:** Creating and managing user accounts with varying levels of permissions within the admin interface.
* **Automation:** Django's admin interface automates the creation of common administrative tasks.

**Techniques**

* **Command Line Interaction:** Using the terminal to execute Django administrative commands.

**Keywords**

["Django", "admin", "interface", "web application", "database", "user", "manage", "data", "credentials", "project", "model"]

These notes written by Gemini

[Managing users in Django Admin | Coursera](https://www.coursera.org/learn/django-web-framework/supplement/9tW6t/managing-users-in-django-admin)

Short Summary: This text explains how to use the Django Admin interface to manage users, including creating user profiles and groups. It also covers customizing the Admin interface to control field editing and improve object display.

**Acronyms**

* **Admin:** Short for administrator or administration.

**Terms**

* **User Profile:** A collection of information associated with a specific user in a system.
* **Group:** In an administrative context, a way to assign permissions to multiple users at once.
* **Super User:** A user account with full administrative privileges.
* **Permissions:** Settings that control what actions a user can take within a system.

**Technical Words**

* **Django:** A Python-based web framework.
* **Interface:** The way a user interacts with a system, often referring to the visual elements and controls.
* **Decorator (**@admin.register()**):** A way to modify the behavior of a class in Python.
* **Subclass:** A class that inherits properties from another class.
* **Method:** A function within a class.

**Rarely Used Words**

* **Judiciously:** With careful judgment or discretion.
* **Warranted:** Justified or necessary.

**Software Packages**

* **Django:** A Python-based web framework.

**Modules**

* **django.contrib.auth:** Django's built-in authentication and authorization system.
* **django.contrib.admin:** Provides the Django admin interface.
* **myapp.models** (Custom): Where you define data models within your Django project.

**Ideas**

* **User Management:** The process of creating, modifying, and controlling user accounts.
* **Admin Interface Customization:** Tailoring the Django admin to specific needs and security practices.
* **Display Customization:** Changing how data is shown in the admin interface for better usability.

**Techniques**

* **Preventing Unauthorized Field Changes:** Using readonly\_fields to restrict editing.
* **Selective Field Restriction** Using conditional logic with get\_form() and the disabled attribute to limit editing based on user type.
* **Displaying Data:** Using list\_display to control column display.
* **Adding Search Functionality:** Using search\_fields to filter objects in the admin.

**Keywords**

['Django', 'admin', 'user', 'group', 'permission', 'field', 'customize', 'display']

These notes written by Gemini

[Adding groups and users | Coursera](https://www.coursera.org/learn/django-web-framework/lecture/HKod6/adding-groups-and-users) 4-2-2024 1628

Short Summary: This transcript explains how to build a basic reservation system using the Django admin interface. It covers creating a reservation model, registering it for admin access, and adding users with permissions to manage reservations.

**Terms**

* **Reservation System:** A system for booking appointments or resources in advance.
* **Django Admin Interface:** The built-in administrative control panel provided by Django.
* **Model:** A representation of a data table within a Django application.

**Technical Words**

* **Python:** A programming language.
* **SQLite:** A lightweight database engine often used with Django.
* **Import Statement:** A way to bring in code from other files or modules in Python.
* **String Representation:** How an object is displayed as text.
* **Version History:** A record of changes made to an object over time.

**Rarely Used Words**

* **Prompt:** A message or symbol from a computer program asking for user input.

**Software Packages**

* **Django:** A Python-based web framework.
* **VS Code:** A popular code editor (although not strictly required for Django development).

**Commands**

* **python manage.py createsuperuser:** Django command to create administrative users.
* **python manage.py runserver:** Django command to start the development server.

**Modules**

* **admin.py**: File within a Django app to control how models are displayed in the admin interface.
* **models.py**: File within a Django app where data models are defined.
* **settings.py:** Django project's configuration file.

**Ideas**

* **Database Integration:** Connecting a web application to a database to store data.
* **User Authentication:** Verifying a user's identity to grant access and permissions.
* **Admin Customization:** Adjusting the Django admin interface for specific use cases.

**Techniques**

* **Creating a Django Model:** Defining the structure of a database table.
* **Registering a Model for Admin Access:** Making the model manageable through the Django admin interface.
* **Creating Users and Assigning Permissions:** Controlling access to the reservation system.

**Keywords**

['Django', 'admin', 'reservation', 'model', 'register', 'user', 'group', 'permission', 'database']

These notes written by Gemini

[Permissions | Coursera](https://www.coursera.org/learn/django-web-framework/lecture/KSTSa/permissions) 4-2-2024 1636

Short Summary: This transcript explains Django's user permissions system, including how permissions are granted, different user types (superuser, staff, regular user), and the use of groups for efficient permission management.

**Terms**

* **Permissions:** Control what actions a user is allowed to perform within a system.
* **Authentication:** Verifying a user's identity (who they are).
* **Authorization:** Determining what a user is allowed to do (their permissions).
* **User:** An entity that interacts with the system, represented as a Python object in Django.

**Technical Words**

* **Django Shell:** An interactive command-line environment for interacting with Django projects.
* **HTTP Response:** The data sent back from a web server in response to a request.
* **Decorator:** A way to modify a function's behavior in Python (e.g., @permission\_required)

**Rarely Used Words**

* **Tedious:** Boring, repetitive task.

**Software Packages**

* **Django:** A Python-based web framework.

**Commands**

* **python manage.py createsuperuser:** Django command for creating administrative users.

**Modules**

* **django.contrib.auth**: Django's module for authentication and authorization.

**Ideas**

* **User Types:** Differentiating users with varying levels of access (superuser, staff, regular).
* **Model Permissions:** Permissions associated with specific data models (add, change, delete, view).
* **Groups:** A way to manage permissions for multiple users at once.

**Techniques**

* **Creating Users:** Using the admin interface or the create\_user function.
* **Granting Staff Status:** Setting the is\_staff property on a user object.
* **Checking Permissions:** Using the has\_perm function.
* **Assigning Users to Groups:** Managing permissions efficiently.

**Keywords**

['Django', 'user', 'permission', 'authentication', 'authorization', 'superuser', 'staff', 'group', 'model']

These notes written by Gemini

[Enforcing Permissions | Coursera](https://www.coursera.org/learn/django-web-framework/supplement/LGEV4/enforcing-permissions) 4-2-2024 1641

Short summary: This transcript discusses Django's built-in admin interface, which provides a convenient way for site managers to add, edit, and delete content stored in a web application's database. It outlines how to create an admin user and navigate the interface's basic features.

**Acronyms:**

* **SDLC:** Software Development Life Cycle: A framework outlining stages involved in building software.

**Terms**

* **Web Application:** Software that runs on a web server, accessed by users through a web browser.
* **Database:** An organized collection of data, often stored and accessed electronically.
* **Admin/Administration Site:** A backend interface within a web application, allowing designated users to manage data and settings.

**Technical Words**

* **Metadata:** Data providing information about other data.
* **Interface:** A point of interaction between components, such as a graphical user interface (GUI) or an application programming interface (API).
* **Project:** In Django, a collection of configuration and code for a web application
* **Model:** A representation of a database table within a Django application.
* **Terminal:** A text-based interface for interacting with a computer's operating system.
* **Utility**: A software tool designed to perform a specific task.
* **Credentials**: Authentication details (e.g., username and password)
* **Prompt:** A message or symbol presented by a computer program, requesting user input.
* **Form:** An interface element for collecting user input within a web application.

**Rarely Used Words**

* **Designated:** Selected or assigned for a specific role.

**Software Packages**

* **Django:** A Python-based web framework for rapid development.

**Commands**

* **python3 manage.py createsuperuser:** Django command to create an administrative user for the admin interface.
* **python3 manage.py runserver:** Django command to start the development server.

**Modules**

* **django.contrib.admin:** Django module providing the admin interface.

**Ideas**

* **Data Management:** Using the admin interface to add, edit, and delete data stored within the application's database.
* **User Management:** Creating and managing user accounts with varying levels of permissions within the admin interface.
* **Automation:** Django's admin interface automates the creation of common administrative tasks.

**Techniques**

* **Command Line Interaction:** Using the terminal to execute Django administrative commands.

**Keywords**

["Django", "admin", "interface", "web application", "database", "user", "manage", "data", "credentials", "project", "model"]

These notes written by Gemini

[Users and Permissions | Coursera](https://www.coursera.org/learn/django-web-framework/lecture/3xgxz/users-and-permissions) 4-3-2024 0440

Short summary: This transcript explains how to manage user permissions in Django. It covers creating users and groups, assigning permissions using both the Django shell and the admin interface, and discusses the concept of default model permissions.

**Acronyms:**

* **SDLC:** Software Development Life Cycle: A framework outlining stages involved in building software.

**Terms**

* **Authentication**: The process of verifying a user's identity.
* **Authorization**: The process of determining what a user is allowed to do.
* **Permissions:** Specific actions allowed for a user or group on a particular object or type of object.
* **Django Shell**: A Python-based interactive command line for interacting with Django projects.

**Technical Words**

* **Django admin:** The built-in administration interface provided by Django.
* **Group:** A collection of users in Django.
* **Migrate (command):** Django command to apply database changes.

**Rarely Used Words**

* **Entitle (to)**: Grant a right or privilege.

**Software Packages**

* **Django:** A Python-based web framework for rapid development.

**Commands**

* **manage.py createsuperuser:** Django command to create an administrative user.
* **manage.py migrate:** Django command to apply database changes.

**Modules**

* **django.contrib.auth:** Django module providing authentication and authorization functionality.

**Ideas**

* **User Management:** Creating and managing users with different roles.
* **Permission Management:** Controlling access to actions and data based on user roles or groups.
* **Default Permissions:** Automatic permissions (add, change, delete, view) that come with Django models.
* **Custom Permissions:** The capability to define additional permissions beyond the defaults.

**Techniques**

* **Admin Interface:** Using the Django admin panel to manage users and permissions visually.
* **Django Shell:** Managing permissions programmatically through the Django shell.

**Keywords**=["Django", "admin", "user", "permission", "group", "authentication", "authorization", "shell", "migrate"]

These notes written by Gemini

[Exercise: Using Django Admin | Coursera](https://www.coursera.org/learn/django-web-framework/ungradedLab/mxDgE/exercise-using-django-admin) 4-3-2024 1722

**Short Summary**

This lesson explains how to manage user permissions in Django. It covers working with permissions through both the Django shell and the more user-friendly Django admin interface, including adding users, groups, and assigning permissions.

**Detailed Breakdown**

* **Acronyms**
  + ORM (Object-Relational Mapping)
* **Technical Words**
  + **Permissions:** Control what actions a user can perform within the application.
  + **User Groups:** A way to organize users and apply permissions collectively.
  + **Django Shell:** An interactive command-line for working with Django projects.
  + **Django Admin Interface:** A web-based interface for managing Django models and data.
  + **Authentication:** Verifying a user's identity.
  + **Authorization:** Determining what a user is allowed to do.
* **Software Packages**
  + **Django:** The high-level Python web framework used to build the application.
* **Commands/Modules**
  + **manage.py**: A Django command-line utility.
  + **manage.py migrate:** Applies database changes to reflect model updates, including default permissions.
  + **django.contrib.auth:** Django's built-in authentication and authorization system.
* **Ideas/Techniques**
  + **User/Group-based Permissions:** Managing access control efficiently.
  + **Default Permissions:** Permissions automatically assigned to models.
  + **Custom Permissions:** Defining additional permissions in the model's Meta class.

Keywords = ['Django', 'permissions', 'users', 'groups', 'Django shell', 'Django admin', 'authentication', 'authorization']

**Mistakes/Misleading Information**

* **Minor Omission:** The term "superuser" is used without a complete explanation. It's important to clarify that a superuser has full administrative privileges within a Django application.

**Written by Gemini**

[Knowledge Check: Admin | Coursera](https://www.coursera.org/learn/django-web-framework/quiz/UXWwP/knowledge-check-admin) 4-3-2024 0532

Basic

[Database options | Coursera](https://www.coursera.org/learn/django-web-framework/lecture/Fy0uM/database-options) 4-3-2024 0538

**Short Summary**

This lesson explains how to switch a Django project from the default SQLite database to a MySQL database. It covers necessary configuration changes, database creation, and the importance of database security.

**Detailed Breakdown**

* **Acronyms**
  + SQL (Structured Query Language)
* **Technical Words**
  + **Database:** An organized collection of data.
  + **Production Environment:** The live, public-facing version of an application.
  + **Database Driver/Connector:** Software that facilitates communication between an application and a database.
  + **Authentication:** Verifying a user's identity (e.g., with a username and password)
  + **Authorization:** Determining what actions an authenticated user can perform.
* **Software Packages**
  + **Django:** The Python-based web framework.
  + **SQLite:** A lightweight file-based database.
  + **MySQL:** A popular open-source relational database system.
* **Commands/Modules**
  + **settings.py:** A Django project's configuration file.
* **Ideas/Techniques**
  + **Database Configuration:** Specifying connection details (database type, address, credentials) to link the project to a database.
  + **Database Security:** Protecting database credentials and user data.

Keywords = ['Django', 'SQLite', 'MySQL', 'database', 'configuration', 'settings.py', 'security', 'authentication', 'authorization']

**Mistakes/Misleading Information**

* **Security Oversimplification:** While the lesson stresses the importance of database security, it could provide more concrete guidance on best practices (strong passwords, user permissions, etc.).
* **File Locations:** The path "etc/mysql" is mentioned, but might not be standardized across systems. Clarification on potential OS differences would be helpful.

**Written by Gemini**

[Configuring MySQL Connection | Coursera](https://www.coursera.org/learn/django-web-framework/supplement/dF9YT/configuring-mysql-connection) 4-3-2024 1214

**Short Summary**

This lesson explains how to switch a Django project from its default SQLite database to a MySQL database. It outlines the advantages of MySQL, the installation process, configuration steps, and how to verify the connection.

**Detailed Breakdown**

* **Acronyms**
  + SQL (Structured Query Language)
* **Technical Words**
  + **Database Server:** A system that manages and provides access to databases.
  + **Client-Server Architecture:** A model where the database runs on a dedicated server, handling requests from client applications.
  + **Authentication:** Verifying a user's identity (e.g., username and password).
  + **DB API Compliant Driver:** Software that allows a programming language to interact with a specific database.
* **Software Packages**
  + **Django:** Python web framework.
  + **MySQL:** Popular relational database system.
  + **mysqlclient:** Python driver for MySQL connections.
* **Commands/Modules**
  + **pip3:** Python package installer. Ex: pip3 install mysqlclient
  + **mysql (command line):** Command to access the MySQL console.
  + **manage.py (Django):** Command-line utility for Django project management. Ex: python manage.py migrate
* **Ideas/Techniques**
  + **Database Migration:** Switching a project to use a different database.
  + **settings.py:** Django configuration file where database settings are defined.

**Keywords** Keywords = ['Django', 'MySQL', 'SQLite', 'database', 'configuration', 'settings.py', 'mysqlclient', 'pip', 'migrate']

**Mistakes/Misleading Information**

* **Oversimplifying Privileges:** The note about the ALTER USER command lacks context. User privilege management in MySQL is more complex, and best practices should be emphasized for security.

**Written by Gemini**

[Setting up a MySQL connection | Coursera](https://www.coursera.org/learn/django-web-framework/lecture/tFv7N/setting-up-a-mysql-connection)

Short summary: This transcript explains how to install and configure MySQL for use with a Django project. It covers database setup, user creation, connector installation, and the process of modifying Django settings.

**Acronyms:**

* **ORM:** Object-Relational Mapper
* **SDLC:** Software Development Life Cycle

**Terms:**

* **Database:** Organized collection of data, often stored and accessed electronically
* **Database Management System (DBMS):** Software that manages and allows interaction with a database.
* **MySQL:** A popular open-source relational database management system.
* **Django:** A Python-based web framework for rapid development.
* **SQLite:** Lightweight database engine often used for Django development.

**Technical Words:**

* **Install/Installation:** The process of setting up software on a system.
* **Configure:** Modify settings within a software to meet specific requirements.
* **Query:** A request for information from a database.
* **Shell:** Command-line interface for interacting with a system.
* **Library:** Collection of pre-written code providing reusable functionality.

**Rarely Used Words:**

* **Verification:** The process of establishing the truth or accuracy of something.

**Software Packages:**

* **MySQL:** Open-source database management system.
* **Homebrew:** Package manager for macOS.
* **Django:** Python-based web framework.
* **mysqlclient:** Python library for interacting with MySQL databases.

**Commands:**

* **brew install mysql:** Homebrew command to install MySQL on macOS.
* **mysql -u root -p:** Command to access the MySQL shell.
* **show databases;** MySQL command to list available databases.
* **create database feedback;** MySQL command to create a new database.
* **pip3 install mysqlclient:** Command to install the MySQL client library.
* **python3 manage.py makemigrations:** Django command to create database migrations.
* **python3 manage.py migrate:** Django command to apply migrations.

**Modules**

* **settings.py:** Django project file containing configuration settings, including database connection details.

**Ideas**

* **Database Selection:** Choosing an appropriate database type for a Django project based on scale and requirements.
* **Database Connector:** Using libraries to enable interaction between Django and a chosen database.
* **Migrations:** Managing database schema changes in Django.

**Techniques**

* **Command-line Interaction:** Using commands to configure a database system.
* **Package Installation:** Installing necessary software using package managers.

**Keywords**

["Django", "MySQL", "database", "install", "configure", "query", "migrate"]

These notes written by Gemini

[Knowledge Check: Database Configuration | Coursera](https://www.coursera.org/learn/django-web-framework/quiz/AMICd/knowledge-check-database-configuration) 4-3-2024 1259

* basic

[Module summary: Models | Coursera](https://www.coursera.org/learn/django-web-framework/lecture/GMEgF/module-summary-models) 4-3-2024 1300

* Module review-No new knowledge

[Module Quiz: Models | Coursera](https://www.coursera.org/learn/django-web-framework/exam/oDdzY/module-quiz-models) 4-3-2024 1345

* I had some trouble with this test. Retake soon. I got 90% but it was not solid.

-----------------------------------------Week 4---------------------------------------

[Templates | Coursera](https://www.coursera.org/learn/django-web-framework/lecture/Bmp0k/templates) 4-3-2024 1402

Summary: This transcript explains how to use Django templates to create dynamic web content. It covers the Django Template Language (DTL), the template engine, and how templates promote code reusability through inheritance.

**Acronyms**

* **DTL:** Django Template Language
* **MVT:** Model-View-Template (a design pattern used by Django)

**Terms**

* **Template:** A text file with placeholders for dynamic data, used to generate HTML.
* **Context:** A dictionary of variables passed to a template for rendering.

**Technical Words**

* **API:** Application Programming Interface – a way for different software components to interact.
* **Markup Language:** A way to annotate text to control formatting (e.g., HTML).

**Software Packages**

* **Django:** A Python-based web framework.
* **Jinja2:** A popular template engine that can be used with Django.

**Ideas**

* **Dynamic Content:** Content that changes based on data or user input.
* **Template Engine:** Software that processes templates and inserts dynamic data.
* **Code Reusability:** Writing code that can be used multiple times to reduce redundancy.
* **Template Inheritance:** Creating a base template that child templates can extend and modify.

**Keywords**=['Django', 'template', 'DTL', 'dynamic', 'render', 'engine', 'inheritance']

These notes written by Gemini

[Template Examples | Coursera](https://www.coursera.org/learn/django-web-framework/supplement/Pqugv/template-examples) 4-3-2024 1555

Summary: This text introduces how Django uses templates to create dynamic HTML content. It explains variables, filters, and basic examples within the Django Template Language (DTL).

**Acronyms**

* **DTL:** Django Template Language

**Terms**

* **Template:** A text file with placeholders for dynamic data, used to generate HTML.
* **Placeholder:** A spot in a template where dynamic data will be inserted. <h1>Hello, {{ name }}!</h1>
* **Variable:** A named container for data within a template.
* **Filter:** A modifier that transforms the output of a variable within a template.

**Technical Words**

* **Rendering:** The process of combining a template with data to generate the final HTML.
* **Context:** A dictionary containing variables passed to the template during rendering.

**Software Packages**

* **Django:** A Python-based web framework.

**Ideas**

* **Separation of Concerns:** Templates separate presentation logic (HTML) from application logic (Python).
* **Template Syntax:** The special symbols and tags used within Django templates.

**Keywords**=['Django', 'template', 'DTL', 'render', 'variable', 'filter', 'context']

These notes written by Gemini

Notes:

Theres a few missing things, but by now we should be learning to fill in the gaps. For example they move from path('', views.index, name='index') to needing  path('index/<str:name>/', views.index, name='index') because of passing a variable.

[Creating Templates | Coursera](https://www.coursera.org/learn/django-web-framework/lecture/f3UvB/creating-templates) 4-4-2024 0445

**Short Summary**

This video explains how to use Django templates to structure web pages. It covers the basics of creating templates, combining static HTML with dynamic data passed from Python views.

**Detailed Breakdown**

* **Acronyms**
  + HTML (HyperText Markup Language)
* **Technical Words**
  + **Template:** A file defining the structure and placeholders for dynamic content in a web application.
  + **Django Template Language (DTL):** The syntax used within Django templates to insert variables, apply logic, and render content.
  + **View (function):** A Python function in Django that handles web requests, often responsible for preparing data to be rendered by a template.
  + **Dynamic Content** Data that changes based on context, user input, or database interactions.
  + **Render Function:** A Django function (django.shortcuts.render) that combines a template with data and returns an HTTP response object.
* **Software Packages**
  + **Django:** Python-based web framework.
* **Commands/Modules**
  + **manage.py** A Django utility for project management tasks (python3 manage.py runserver)
  + **django.shortcuts.render** Function to streamline the template rendering process (return render(...))
* **Ideas/Techniques**
  + **Code Reusability:** Templates promote separation of presentation logic (HTML) from application logic (Python), allowing reuse of common layouts.
  + **Model-View-Template (MVT) Pattern:** Django's architectural pattern where models handle data, views handle logic, and templates handle presentation.

**Python Keywords** keywords = ['Django', 'templates', 'views', 'render', 'HTML', 'dynamic', 'DTL']

**Mistakes or Misleading Information**

* **Oversimplifying URLs:** The lesson doesn't fully explain the importance of URL patterns and how they map views to specific web addresses.

**Written by Gemini**

[Exercise: Creating Templates | Coursera](https://www.coursera.org/learn/django-web-framework/ungradedLab/I8dhW/exercise-creating-templates) 4-4-2024 0530

**Short Summary**

This lab guides learners through creating basic Django templates. It covers rendering views, using the Django Template Language (DTL) to insert dynamic data, and adding static assets like images.

**Detailed Breakdown**

* **Acronyms**
  + DTL (Django Template Language)
* **Technical Words**
  + **Template:** A file defining a web page's structure with placeholders for dynamic content.
  + **View (function):** A Python function within Django that handles web requests and prepares data for the template to render.
  + **Render Function:** The Django function (render) that processes a template, combines it with data, and generates an HTML response.
  + **Static Files:** Assets like images, CSS, and JavaScript that are not generated dynamically.
* **Software Packages**
  + **Django:** Python-based web framework.
* **Ideas/Techniques**
  + **Templating:** Separating presentation logic (HTML, CSS) from application logic (Python).
  + **Dynamic Content:** Using DTL ({{ }}) to insert variables from Python views into templates.
  + **Static File Management:** Serving images and other static assets from designated folders.
* **Tips**
  + **Word Wrap:** Using ALT+Z in VSCode for long lines of code.
  + {% load static %}**:** This tag in templates is necessary to reference static files.

**Python Keywords** keywords = ['Django', 'render', 'templates', 'views', 'DTL', 'static']

**Mistakes or Misleading Information**

* **Limited Scope:** The lab focuses on very basic templating concepts. It's important to note that Django templates offer more advanced features like inheritance, filters, and custom tags.

**Written by Gemini**

* **Tags:** Tags are used to control the flow of execution in a template. They are enclosed in curly braces ({% %}).
* **Filters:** Filters are used to modify the output of a variable. They are enclosed in double curly braces ({{ %}}).
* **Variables:** Variables are used to store data that can be used in a template. They are enclosed in single curly braces ({% %}).

[Knowledge check: Templates | Coursera](https://www.coursera.org/learn/django-web-framework/quiz/6kPij/knowledge-check-templates) 4-4-2024 0838

**Short Summary:** This quiz reviews Django templates, including template configuration, inheritance for code reuse, filters, and the render function.

**Terms**

* **Templates:** Text files with placeholders for dynamic data, used to generate HTML in Django.
* **Template Inheritance:** Creating reusable base templates that child templates can extend.
* **Template Polymorphism:** While the concept exists broadly in programming, it's not a strongly emphasized feature within the Django Template Language itself.
* **Filters:** Modifiers that transform the output of variables within Django templates.

**Technical Words**

* **Template Engine:** Software component within Django that processes templates and inserts dynamic data.
* **Polymorphism**: It comes from Greek roots "poly" (many) and "morph" (form).
  + Literally, it means "having multiple forms."
  + Programming Context: In object-oriented programming (OOP), polymorphism refers to the ability of objects of different classes to respond to the same method call or operation in their own unique ways. Here's how it works:

**Software Packages**

* **Django:** A Python-based web framework.

**Ideas**

* **Code Reusability:** Writing code that can be used in multiple places to reduce redundancy.

**The** render() **Function**

The render() function in Django is a shortcut that combines several steps to generate an HTTP response containing rendered HTML:

**Required Arguments:**

* **request:** The HTTP request object. It contains information about the incoming request, like user data.
* **template\_name:** The relative path to the template file to be rendered.

**Optional Argument:**

* **context:** A dictionary containing variables that should be made available to the template.

**What Happens Behind the Scenes**

1. **Loading the Template:** Django finds and loads the specified template.
2. **Creating the Context:** Django creates a context dictionary, combining data from the context argument with built-in context processors.
3. **Rendering the Template:** The Django template engine processes the template, substituting placeholders with values from the context.
4. **Returning an HTTP Response:** Django creates an HttpResponse object containing the rendered HTML and returns it.

**Keywords**

['Django', 'template', 'render', 'filter', 'inheritance', 'context']

[Working with template language | Coursera](https://www.coursera.org/learn/django-web-framework/lecture/HcXVD/working-with-template-language)

**Short summary**

Summary: This video teaches how to use the Django Template Language (DTL). DTL helps insert dynamic data into web pages, using variables (data placeholders), tags (logic), filters (data modifiers), and comments (notes).

**Acronyms**

* DTL: Django Template Language

**Terms**

* Template language: A language used to mix static content with dynamic data in a template.
* Django template: A text document with DTL syntax, used to generate HTML in Django.
* DRY (Do Not Repeat Yourself): A software development principle that promotes code reusability to save time.

**Technical Words**

* Boolean: A data type having two possible values: true or false.
* Dictionary: A data structure storing key-value pairs. {key:value}
* Attribute lookup: Accessing a property of an object.
* List index lookup: Accessing an item in a list by its position.
* String: A sequence of text characters.
* Unordered list: An HTML list with bullet points.
* List item: An individual element within a list.
* **extends**: Creates a parent-child relationship between templates, allowing for inheritance. Example: {% extends "base.html" %}
* **Include**: Embeds a reusable template snippet within another template. Example: {% include "footer.html" %}

**Rarely Used Words**

* Render: The process of generating the final output (e.g., HTML) from a template.
* Iteration: The process of repeating a block of code multiple times.

**Software Packages**

* Django: A high-level Python web framework.

**Ideas**

* Separation of presentation and logic: DTL lets you keep application logic (Python) separate from the HTML presentation layer.
* Template rendering: The template engine processes DTL code, replacing variables and executing tags to produce the final HTML.

**Techniques**

* Using variables to insert dynamic data.
* Employing 'if' tags for conditional logic.
* Creating dynamic lists with 'for' loops.
* Applying filters to modify variable output.

**Keywords**

["DTL", "Django", "template", "variable", "tag", "filter", "comment", "if", "for"]

**Let my LLM notes make you chuckle(Gemini)!**

[Template Language and Variable Interpolation | Coursera](https://www.coursera.org/learn/django-web-framework/supplement/zmvUc/template-language-and-variable-interpolation) 4-4-2024 1216

Short summary: This reading explains how web frameworks use template languages to create dynamic web pages. It focuses on Django's Template Language (DTL), introducing variables, tags (like 'if' and 'for'), and filters to modify output.

Acronyms

* DTL: Django Template Language
* CSRF: Cross Site Request Forgeries

Terms

* Template language: A language used to mix static content with placeholders for dynamic data.
* Template Engine: A software component that processes a template, replacing placeholders with data to generate the final output (e.g., HTML).
* Context: A dictionary of data passed from the Django view to the template.

Technical Words

* Variable: A placeholder within the template, replaced with its corresponding value from the context.
* Tag: A keyword in the template language that adds logic (e.g., conditional statements, loops).
* Filter: A modifier applied to a variable to alter its display format.
* Iterable: A data type that can be looped over, such as a list, tuple, or string.
* Boolean expression: An expression that evaluates to True or False.
* **Variable Interpolation (General)** **Definition:** The process of embedding the value of a variable directly into a string or text block. This allows you to create dynamic text content where parts change based on the data stored in the variable.**Example (Simple):**
  + name = "Alice"
  + greeting = "Hello, " + name + "!"
  + print(greeting) # Output: Hello, Alice!
* **Interpolation in Django DTL** **Definition:** In Django templates, variable interpolation is the mechanism for inserting dynamic data from your Python view code into the HTML. DTL uses double curly braces {{ variable\_name }} to mark where variables should be substituted. **Example (Django):**
  + Python **View:**
    - def my\_view(request):
    - context = {'name': 'Bob'}
    - return render(request, 'my\_template.html', context)
  + **Template (**my\_template.html**):**
    - HTML
    - <h2>Welcome, {{ name }}!</h2>
  + **Key points:**
    - Variable interpolation makes templates incredibly flexible.
    - You can interpolate strings, numbers, and even the results of calculations or function calls.

Rarely Used Words

* Rendering: The process of the template engine generating final output by combining the template with data.

Software Packages

* Django: A high-level Python-based web framework.

Ideas

* Separation of concerns: DTL promotes separating presentation logic (HTML) from application logic (Python).
* Dynamic content: DTL allows you to create web pages where the displayed content changes based on the data provided.

Techniques

* Using variables to embed data in templates.
* Conditional rendering with 'if' tags.
* Looping over data with 'for' tags.
* Modifying output presentation with filters.

Keywords ["DTL", "Django", "template", "variable", "tag", "filter", "if", "for", "context", "include", "block" ]

**My LLM notes are electrifying! Created by Gemini**

Notes:

Be careful 'index/<str:name>/' needs url index/string/ where as 'index/<str:name>' needs url index/string. The ‘/’ at the end needs to match your urls.py file. path('index**/<str:name>/**', views.index, name='index'), versus path('index**/<str:name>**', views.index, name='index'),.

Had some fun making all the example code into one page.

[Dynamic Templates in Django | Coursera](https://www.coursera.org/learn/django-web-framework/lecture/rdyE8/dynamic-templates-in-django) 4-4-2024 1316

Short summary: This transcript explains how to use Django templates to display dynamic data on a webpage. It covers creating dictionaries, passing them to templates, and using for loops to display the data in HTML.

**Acronyms:**

* **URL:** Uniform Resource Locator (a web address)

**Terms:**

* **Web Page:** A document viewed in a web browser.
* **HTML:** Hypertext Markup Language, the standard language for structuring web pages.
* **API:** Application Programming Interface, a way for software components to interact with each other.

**Technical Words:**

* **Template:** A file in Django that contains a mix of static HTML and special syntax for displaying dynamic data.
* **Render:** The process of Django combining a template with data to produce a final HTML page.
* **Dynamic Data:** Information that can change, often retrieved from a database or external source.
* **Parameter:** A piece of information passed to a function.
* **Variable:** A named container for storing data within a program.

**Commands:**

* **render()**: A Django function that takes a request, a template, and a data dictionary to generate a web page. **return render(request, 'menu.html', newmenuitem)**
* **path()**: A Django function used to map URLs to views. **path('menu/', views.menu, name='menu'),**

**Modules:**

* **views.py:** A Python file in a Django app where view functions are defined.
* **urls.py:** A Python file in a Django project where URL patterns are defined.
* **Ideas**
* **Dynamic Content:** Using Django templates to make web pages that change based on data, instead of being purely static.
* **Data Sources:** Django's flexibility in using different data sources (dictionaries, databases, APIs) for templates.
* **Django Template Language:** The special syntax with double curly braces ({{ }}) that Django uses to insert data into templates.

**Techniques**

* **Creating Dictionaries:** Using curly braces {} and key-value pairs to store data in Python.
* **Passing Data to Templates:** Including a dictionary as an argument to the render() function.
* **For Loops in Templates:** Using {% for item in items %} syntax to iterate over data and display it within HTML.

**Keywords**=["Django", "template", "render", "dynamic", "data", "dictionary", "view", "HTML"]

This analysis was brought to you by Gemini and the wonders of Large Language Models (LLMs)!

[Mapping model objects to a template | Coursera](https://www.coursera.org/learn/django-web-framework/lecture/jPooL/mapping-model-objects-to-a-template) 4-4-2024 1330

**Summary**: This transcript demonstrates how to fetch data from a database model in Django and display it dynamically on a webpage. It covers retrieving menu items, passing the data to a template, and using for loops within the template to render them in HTML.

**Terms**

* **Database:** A structured collection of data stored electronically.
* **Model:** In Django, a representation of a database table within a Python class.
* **Logic:** The programming that controls how data is manipulated or decisions are made.
* **End User:** The person who interacts with the website.

**Technical Words**

* **Query Set:** A collection of results returned from a database query.
* **Conditional Statement:** A piece of code that executes different sections depending on whether certain conditions are true or false (if).

**Rarely Used Words:**

* **Dunder String Method (**\_\_str\_\_()**)** A special method in Python classes that determines how objects of that class are represented as strings.

**Commands/Modules**

* **models.py:** A Python file in a Django app where database models are defined.
* **views.py:** A Python file in a Django app where view functions are defined.
* **urls.py:** A Python file in a Django project where URL patterns are defined.
* **menu.objects.all()**: Django syntax to retrieve all objects from the Menu model.

**Ideas**

* **MVC Pattern:** Django follows the Model-View-Controller pattern, separating data representation (Model), presentation logic (View), and user interaction control (Controller).
* **ORM:** Django's Object-Relational Mapper allows interaction with databases using Python objects rather than raw SQL.

**Techniques**

* **Model Creation:** Defining models in Django to represent database tables.
* **Accessing Model Data:** Using Django's ORM syntax to query data.
* **Template Rendering:** Using Django template language ({% for %}, {{ }}) to embed dynamic data into HTML.

**Keywords**

["Django", "database", "model", "view", "template", "dynamic", "render", "query"]

This breakdown was generated by Gemini and the magic of Large Language Models (LLMs)!

[Mapping model objects to a template | Coursera](https://www.coursera.org/learn/django-web-framework/lecture/jPooL/mapping-model-objects-to-a-template) 4-4-2024 1415

Short summary: This transcript explains how to fetch menu data from a Django database model and display it dynamically on a webpage using template rendering and for loops.

**Terms**

* **Database:** A structured collection of data stored electronically.
* **Model:** In Django, a representation of a database table within a Python class.
* **Query Set:** A collection of results returned from a database query.
* **End User:** The person who interacts with the website.

**Technical Words**

* **Conditional Statement:** Code that executes different sections depending on whether certain conditions are true or false (if).
* **Dunder String Method (**\_\_str\_\_()**)** A special Python method that determines how objects are represented as strings.
* **Template Rendering:** The process of Django combining data with HTML templates to produce the final webpage.

**Rarely Used Words:**

* **Visualization** : Creating a mental or visual image of something.

**Software Packages**

* **Django:** A Python-based web framework for rapid development.

**Modules**

* **models.py:** A Python file in a Django app where database models are defined.
* **views.py:** A Python file in a Django app where view functions are defined.
* **urls.py:** A Python file in a Django project where URL patterns are defined.

**Ideas**

* **MVC Pattern:** Django follows the Model-View-Controller pattern, separating data representation (Model), presentation logic (View), and user interaction control (Controller).
* **ORM:** Django's Object-Relational Mapper allows interaction with databases using Python objects rather than raw SQL.

**Techniques**

* **Model Creation:** Defining models in Django to represent database tables.
* **Accessing Model Data:** Using Django's ORM syntax to query data (menu.objects.all()).
* **Template Rendering:** Using Django template language ({% for %}, {{ }}) to embed dynamic data into HTML.

**Keywords**=["Django", "database", "model", "view", "template", "dynamic", "render", "query"]

This breakdown was crafted with the help of Large Language Models (LLMs)!

[Exercise: Creating Dynamic Templates | Coursera](https://www.coursera.org/learn/django-web-framework/ungradedLab/Um738/exercise-creating-dynamic-templates) 4-4-2024 1530

**Short Summary:** This transcript explains template inheritance in Django, which allows developers to reuse common website elements (like headers and footers) across multiple pages for efficient development and maintenance.

**Terms**

* **Web Application:** Software that runs on a web server and is accessed through a web browser.
* **Header:** The top section of a webpage, often containing the logo, navigation, and sometimes login functionality.
* **Footer:** The bottom section of a webpage, often containing copyright information, contact details, and additional navigation.
* **DRY Principle:** "Don't Repeat Yourself." A programming best practice emphasizing code reusability.
* **User Interface (UI):** The visual and interactive elements through which a user interacts with a website.
* **User Experience (UX):** The overall experience and satisfaction a user has when interacting with a website or application.

**Technical Words**

* **Template:** A file in Django that combines HTML structure with special syntax for displaying dynamic data.
* **Template Inheritance:** A technique where a "child" template extends a "parent" template, reusing common sections and optionally overriding specific areas.
* **Block:** A named section within a Django template that can be replaced or filled in by child templates.
* **Rendering:** The process of combining a Django template with data to produce the final HTML output.

**Techniques**

* **Include Tag (**{% include %}**):** Used to insert the contents of another template into the current one. {% include 'menu\_items.html' %}
* **Extends Tag (**{% extends %}**):** Used to create a child template that inherits from a parent template, allowing for the reuse of common elements and modification of specific blocks. {% extends 'base.html' %}

**Keywords**=["Django", "template", "inheritance", "include", "extends", "DRY", "header", "footer", "UI", "UX"]

**Important Concepts**

* **Why Template Inheritance Matters:**
  + **Efficiency:** Avoids redundant code for common elements, streamlining development.
  + **Consistency:** Ensures a cohesive look and feel across the entire website.
  + **Maintainability:** Changes to shared elements only need to be made in one place.

This analysis is brought to you by the power of Large Language Models (LLMs) Gemini!

[More on Template inheritance | Coursera](https://www.coursera.org/learn/django-web-framework/supplement/8jY11/more-on-template-inheritance)

**Short Summary:** This transcript explains template inheritance in Django, a technique to create consistent website layouts by defining a base template and having child templates selectively override sections. It also introduces how to manage static files (images, CSS, etc.) in Django.

**Acronyms**

* **URL:** Uniform Resource Locator – a web address.

**Terms**

* **Web Application:** Software that runs on a web server, accessed through a web browser.
* **Django:** A Python-based web framework for rapid development.

**Technical Words**

* **Template:** A file in Django that combines HTML structure with special syntax for displaying dynamic data.
* **Inheritance:** In programming, when a child class/template adopts properties and behaviors from a parent class/template.
* **Block:** A named section within a Django template that can be replaced or extended by child templates.
* **Render:** The process of Django combining a template with data to produce the final HTML output.
* **Static Files:** Files like images, CSS, and JavaScript that are not generated dynamically by Django.

**Modules**

* **django.contrib.staticfiles:** A Django module that helps manage static files.

**Ideas**

* **Web Page Structure:** The concept of organizing a web page into consistent sections like headers, footers, and sidebars.
* **Templating System:** Django's mechanism for combining HTML structures with dynamic data.

**Techniques**

* **Template Inheritance:** Using {% extends %} and {% block %} tags to create reusable website layouts.
* **Static File Management:** Django's conventions for organizing and referencing static files using {% static %} tags.

**Keywords**=["Django", "template", "inheritance", "block", "extends", "static", "render"]

**Let me know if you'd like clarification on any of the concepts or want to focus the analysis differently!**

This analysis was crafted with a dash of humor thanks to the power of Large Language Models (LLMs)!

Gemini

[Working with Template inheritance | Coursera](https://www.coursera.org/learn/django-web-framework/lecture/M9ayL/working-with-template-inheritance)

**Summary:** This video demonstrates how to use Django's Extends and Include tags for template inheritance. Template inheritance allows you to create a base template with common elements and then have child templates inherit and customize specific content areas.

**Short summary:** This transcript teaches how to reuse website layouts in Django using the Extends and Include tags for template inheritance.

**Acronyms:**

* **HTML:** HyperText Markup Language (the standard language for building web pages)
* **SDLC:** Software Development Life Cycle (A framework outlining stages involved in building software)

**Terms:**

* **Django:** A Python-based web framework for rapid development.
* **Template:** A blueprint for generating HTML pages in Django.
* **Block content:** A section within a template that can be overridden by child templates.

**Technical Words:**

* **Web development:** The process of creating and maintaining websites.
* **URL:** Uniform Resource Locator (a web address).
* **Development server:** A local server used for testing website code during development.
* **Render:** The process of a template engine combining a template with data to produce the final HTML output.

**Rarely Used Words:**

* **Stub:** A basic placeholder implementation (in this context, the basic HTML structure).

**Modules**

* **views.py:** A Python file in Django containing functions (views) that handle web requests and generate responses.

**Ideas**

* **Template inheritance:** A design pattern in web development where templates can inherit structure and content from a parent template, promoting reusability and consistency.

**Techniques**

* **Using the Extends tag:** In Django, the {% extends %} tag indicates that a template inherits from another (parent) template.
* **Using the Include tag:** In Django, the {% include %} tag embeds another template within the current one.

**Keywords**=["Django", "Template", "Extends", "Include", "Inheritance", "views.py", "URL", "HTML", "Web development"]

This breakdown was created by Gemini, a helpful LLM.

[Exercise: Working with Templates | Coursera](https://www.coursera.org/learn/django-web-framework/ungradedLab/5eDzx/exercise-working-with-templates) 4-5-2024 0554

**Summary**

This lab teaches Django template inheritance concepts by having learners create reusable header and footer elements that are shared across multiple web pages within the application. The focus is on using include and extends tags for effective code organization.

**Detailed Breakdown**

* **Django Template Concepts**
* **Static Content Management**
* **URL Configuration and Views**

keywords= [ "include", "extends", "static", "render", "block", "url" ]

**Acronyms**

* **DTL:** Django Template Language

**Technical Words**

* **Template Inheritance:** A way to create reusable base templates and have child templates inherit and customize their structure.
* **Static Content:** Files like images, CSS, and JavaScript that aren't dynamically generated by Django.

**Software**

* **Django:** A Python-based web framework for building web applications.

**Important Ideas**

* **Base Template:** Defines the common layout and structure of pages.
* **Child Template:** Extends the base template and customizes specific blocks of content.
* **Content Blocks (**{% block %}**):** Placeholders within base templates that can be overridden by child templates.

**Tips**

* Use descriptive block names to make your templates more readable.
* The {{ block.super }} variable allows you to include parent template content within a child template's overridden block.

**Mistakes or Misleading Information**

* The instructions could be clearer about the relationships between view functions, URL patterns, and the specific HTML templates to render. A brief explanation of how Django's URL routing connects to views would be helpful for beginners.

Written by Gemini, you can call me the template wrangler 😉

Notes: Pretty straightforward. The samples show a reservation section, but there is no mention of a reservation section and no URL that would lead to one. I wanted to get going to the next lesson so I did not bother to make one, but it would be pretty straightforward.

[Knowledge check: Working with Templates | Coursera](https://www.coursera.org/learn/django-web-framework/quiz/sO7IX/knowledge-check-working-with-templates) 4-5-2024 0601

Missed 2, one no big deal, the second was definitions for extend and include tags, which is important. Retake test

**{% extends %}**

* **Definition:** Signals that the current template inherits from a parent (base) template. This allows you to create a common layout and structure that can be reused across multiple pages, promoting consistency and maintainability.
* **Example Code (base.html):**

HTML base.html

<!DOCTYPE **html**>

<html>

<head>

<title>{% block title %}My Website{% endblock title %}</title>

</head>

<body>

<header>

<h1>Website Header</h1>

</header>

<main>

{% block content %}{% endblock content %}

</main>

<footer>

<p>&copy; 2024 My Website</p>

</footer>

</body>

</html>

* **Example Code (child\_page.html):**

HTML **child\_page.html**

{% extends "base.html" %}

{% block title %}About Us{% endblock title %}

{% block content %}

<h2>About Our Company</h2>

<p>We are a passionate team...</p>

{% endblock content %}

**Explanation:**

* base.html defines the basic website structure with placeholders ({% block %}) for specific content.
* child\_page.html extends base.html, overriding the title and content blocks. The resulting rendered page will include the common header and footer from base.html.

**{% include %}**

* **Definition:** Embeds the contents of another template file into the current template. This is useful for reusing smaller snippets of HTML across different templates.
* **Example Code (main\_page.html):**

HTML

{% include "partials/navigation.html" %}

<div class="main-content">

</div>

* **Example Code (partials/navigation.html):**

HTML

<nav>

<ul>

<li><a href="/">Home</a></li>

<li><a href="/about">About</a></li>

</ul>

</nav>

**Explanation:**

* main\_page.html includes the contents of partials/navigation.html, typically containing a navigation menu. This keeps your code organized and modular.

[Debugging Django applications | Coursera](https://www.coursera.org/learn/django-web-framework/lecture/H4Ffy/debugging-django-applications) 4-5-2024 0619

**Summary**

This video introduces debugging techniques for Django web applications. It covers common errors, the Django debug page, logging information, and the importance of a systematic approach to troubleshooting.

**Acronyms**

* **CSRF:** Cross-Site Request Forgery (a security mechanism)

**Technical Words**

* **Debugging:** The process of identifying and fixing errors in code.
* **Traceback:** A report that shows the sequence of function calls leading up to an error.
* **Production Environment:** The live version of a website or application accessible to end-users.

**Ideas/Techniques**

* **Examining the Django error page:** Provides clues about the error's nature and location.
* **Checking the console log:** Displays error messages and other runtime information.
* **Using a systematic troubleshooting approach:** Helps isolate the problem effectively.

**Tips**

* Pay close attention to error messages and the traceback.
* Don't run the Django debugger in a production environment due to security risks.

keywords = ["debug", "error", "logging", "traceback", "Django"]

**Mistakes or Misleading Information**

* The video implies that all Python-specific errors will appear in the console log. This isn't always true – some Python errors might only appear in the Django error page or if you have additional debugging tools set up.
* The statement about the CSRF token missing being helpful for troubleshooting could be clearer. It's helpful because it gives a specific direction to investigate, but the initial error message might be confusing for beginners.

Let me know if you want a more in-depth analysis of any of these debugging concepts! Written by Gemini, you can call me the error exterminator 🤖💥

[Testing in Django | Coursera](https://www.coursera.org/learn/django-web-framework/lecture/lHI6q/testing-in-django) 4-5-2024 0925

**Summary**

This video introduces unit testing in Django applications. It explains how to use Django's TestCase class to create tests, demonstrating how to verify model fields and timestamps.

**Acronyms**

* **CSRF:** Cross-Site Request Forgery (a security mechanism)

**Technical Words**

* **Unit Testing:** A method of testing individual, isolated pieces of code (functions, classes, etc.)
* **Assertion:** A statement that expresses an expected outcome for your code.
* **Granular:** Small and focused in scope.

**Software**

* **Django:** A Python-based web framework for building web applications.
* **Python:** A general-purpose programming language.

**Commands/Modules**

* **python manage.py test:** Executes Django test cases.

**Ideas/Techniques**

* **Test-Driven Development (TDD):** A development practice where tests are written before the code they will test.
* **setUpTestData() method:** Used in TestCase classes to create initial data for tests.

keywords = ["test", "TestCase", "unittest", "assert", "Django", "model"]

**Mistakes or Misleading Information**

* The video suggests that Django tests are fundamentally different from regular Python unit tests. While there are Django-specific classes (TestCase), the underlying concepts and usage of assertions are very similar to standard Python unit testing.

**Tips**

* Consider Test-Driven Development (TDD) for a structured approach.
* Aim to test edge cases and potential error scenarios in addition to basic functionality.

Written by Gemini, you can call me the test technician 🧪

[Sub-classing Generic Views | Coursera](https://www.coursera.org/learn/django-web-framework/supplement/usxgu/sub-classing-generic-views) 4-5-2024 1335

**Summary:** This text introduces class-based generic views in Django, explaining how they streamline web development compared to function-based views. It details views for common CRUD (Create, Read, Update, Delete) operations.

**Acronyms**

* **CRUD:** Create, Read, Update, Delete (fundamental database operations)

**Terms**

* **Class-Based Views (CBVs):** A way to structure views in Django using classes, promoting code reusability.
* **Function-Based Views:** The traditional way to define views in Django using functions.
* **HTTP Methods (GET, POST):** Protocols for how a browser communicates with a web server (e.g., GET to fetch data, POST to submit data).

**Technical Words**

* **View:** A component in Django that handles web requests and generates responses.
* **Template:** An HTML file with Django Template Language (DTL) placeholders for dynamic content.
* **Model:** A Python class representing a database table in Django.
* **URL Pattern:** A mapping between a URL and a view function or class.
* **as\_view() method:** A method used to convert a class-based view into a callable object for URL routing.

**Rarely Used Words**

* **Sub-classing:** Creating a new class that inherits properties and methods from an existing class.
* **Implicit:** Implied, not directly stated.
* **Explicit:** Directly and clearly defined.

**Modules**

* **django.views:** Contains base view classes.
* **django.views.generic:** Contains generic view classes (e.g., TemplateView, ListView).
* **django.views.generic.edit:** Contains classes for editing data (e.g., CreateView, UpdateView, DeleteView).

**Ideas**

* **Code Reusability:** CBVs promote reusability and prevent code duplication.
* **DRY (Don't Repeat Yourself):** A principle emphasizing the importance of avoiding redundancy.
* **Object-Oriented Programming (OOP):** CBVs align with OOP principles.

**Techniques**

* **Using Generic Views:** Employing pre-built Django classes to handle common web development tasks.

**Important Notes**

* **Choose the right view type:** Generic views are efficient, but for highly custom logic, function-based views might be more suitable.
* **Understand inheritance:** To fully customize generic views, grasp how class inheritance works.

**Keywords**=['def', 'from', 'import', 'class', 'model', 'fields', 'success\_url', 'path', 'name']

This analysis was carefully crafted with the assistance of Gemini, your friendly LLM helper! 🤖

Notes:

I am frustrated with the documentation of this course. There is so many things that need to be debugged when I type everything in correctly. For a simple example:

The course says to enter **/employees/1, but the same course had** path('show/<int:pk>', EmployeeDetail.as\_view(), name = 'EmployeeDetail')

As the url setup. This means /employees/show/1 is the url by default. This is just one of many.

[Module summary: Templates | Coursera](https://www.coursera.org/learn/django-web-framework/lecture/9YUxs/module-summary-templates) 4-5-2024 1352

* Course summary

[Module Quiz: Templates | Coursera](https://www.coursera.org/learn/django-web-framework/exam/clss8/module-quiz-templates) 4-5-2024 1420

Missed a few, but nothing serious. Some of my misses were missed by both Gemini and ChatGPT. Some seemed just awkwardly worded.

-------------------------Week 5--------------------------------

Project done and on GitHub

[Final graded quiz: Django Web Framework | Coursera](https://www.coursera.org/learn/django-web-framework/exam/bOtB6/final-graded-quiz-django-web-framework/attempt)

I missed 3 of 10. 3rd try was 80% Lets do this one again!

***--------------Notes unrelated to the course---------------------***

PowerShell

Search all subdirectories: **Get-ChildItem -Path C:\ -Filter sqlite\*.exe -Recurse | Select-Object FullName**

To copy all subs and files from a dir to cwd: **Copy-Item -Path ..\week#3-temp\\* -Destination . -Recurse**

To copy the target directory and all sub-dirs to cwd: **copy-item -Path ..\week#3-temp\ -Destination . -Recurse**

Pipenv

Pipenv graph: list dependencies

Pipenv –venv: find virtual environment and where stored

This is off my notebook

Directory: C:\Users\inouy\.vscode\extensions\alexcvzz.vscode-sqlite-0.14.1\bin

Mode LastWriteTime Length Name

---- ------------- ------ ----

-a---- 12/20/2023 4:13 PM 911360 sqlite-v3.26.0-win32-x86.exe

A screenshot of a message

Description automatically generated

The following commands printed out information needed to configure MySQL.  
mysql> SELECT SUBSTRING\_INDEX(host, ':', 1) AS 'ip',

-> IF(POSITION(':' IN host) > 0, SUBSTRING\_INDEX(host, ':', -1), NULL) AS 'port',

-> COUNT(\*) AS 'connections'

-> FROM information\_schema.processlist

-> WHERE ID=connection\_id()

-> GROUP BY ip, port;

+-----------+-------+-------------+ **Courier new font**

| ip | port | connections |

+-----------+-------+-------------+

| localhost | 52855 | 1 |

+-----------+-------+-------------+

1 row in set, 1 warning (0.01 sec)

Getting mysql hooked up and running was a bit of a learning curve. I was using 127.0.0.1 which was supposed to be ok, but it would not work until I changed to localhost.

Thee following setup worked:

DATABASES = {

    'default': {

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

        'NAME': 'mydatabase',

        'USER': 'root',

        'PASSWORD': 'Coursera', # Note this pw is for school stuff and never used anywhere

        'HOST': 'localhost',  # I was using '127.0.0.1' but it was not working.

        'PORT': '3306',

        'OPTIONS': {

            'init\_command': "SET sql\_mode='STRICT\_TRANS\_TABLES'"

        }

    }

}

Lessons on git  
Somehow, I was not pushing a directory created in a branch. I thought I merged them correctly. I then tried git fetch origin and git merge origin/master and it did it. When having trouble like this try adding a test file. Git add and push and see if it moves to your repo. If not, this may be the issue. Regardless if there is an issue make sure its fixed before moving on or it could cause you and the team problems.  
  
Lessons on VSC

Alt-z for word wrap