CAREERFOUNDRY

Python for Web Developers Learning Journal

Objective

We find that the students who do particularly well in our courses are those who practice metacognition. Metacognition is the art of thinking about thinking; developing a deeper understanding of your own thought processes. With the help of this Learning Journal, you'll broaden your metacognitive knowledge and skills by reflecting on what you learn in this course.

Thanks to this Learning Journal, when you finish the course you'll have a complete and detailed record of your learning journey and progress over time. We really recommend that you take the time to complete this Journal; students do better in CF courses and in the working world as a result!

Directions

First complete the pre-work section before you start your course. Then, once you've begun learning, take time after each Exercise to return to this Journal and respond to the prompts.

There will be 3 to 5 prompts per Exercise, and we recommend spending about 10 to 15 minutes in total answering them. Don't overthink it—just write whatever comes to mind!

Also make sure that, once you've started filling this document in, you upload it as a deliverable on the platform. This is so that your mentor can also see your Journal and how you're progressing over time. Don't worry though—what you write here won't affect how you're graded for the Exercise tasks. The learning journal is mostly for you and your self-evaluation!

Pre-Work: Before You Start the Course

Reflection questions (to complete before your first mentor call)

1. What experiences have you had with coding and/or programming so far? What other experiences (programming-related or not) have you had that may help you as you progress through this course?

I started learning to code about 4 years ago learning Python Fundamentals. I then switched to HTML, CSS and JavaScript and did a career change. I now work as a Social Science Survey Programmer where (besides in a Scripting Language called *GESS*) I use those languages.

2. What do you know about Python already? What do you want to know?

I know the fundamentals and played a bit with Flask and Pygame. I'd like to learn the Fundamentals of Django and play with LLMs.

3. What challenges do you think may come up while you take this course? What will help you face them? Think of specific spaces, people, and times of day of week that might be favorable to your facing challenges and growing. Plan for how to solve challenges that arise.

There might be unforeseen technical issued. I am pretty confident I can solve them by gooling 😀

Remember, you can always refer to Exercise 1.4 of the Orientation course if you're not sure whom to reach out to for help and support.

Exercise 1.1: Getting Started with Python

Learning Goals

- Summarize the uses and benefits of Python for web development
- Prepare your developer environment for programming with Python

Reflection Questions

1. In your own words, what is the difference between frontend and backend web development? If you were hired to work on backend programming for a web application, what kinds of operations would you be working on?

Frontend Development: Focuses on the part of the web application that users interact with directly. This includes the design, layout, and interactivity of web pages, typically using HTML, CSS, and JavaScript.
Backend Development: Involves the server-side logic and operations that power the web application. This includes handling database interactions, user authentication and creating APIs. Common languages used are: Python, Ruby, C#, Rust and Java.

2. Imagine you're working as a full-stack developer in the near future. Your team is asking for your advice on whether to use JavaScript or Python for a project, and you think Python would be the better choice. How would you explain the similarities and differences between the two languages

to your team? Drawing from what you learned in this Exercise, what reasons would you give to convince your team that Python is the better option?
(Hint: refer to the Exercise section "The Benefits of Developing with Python")

Ease of Learning and Use: *Python's straightforward syntax makes it easy to learn and quick to develop with, which can speed up the development process and reduce errors.*
Readability: *Python's code readability enhances collaboration and maintenance, allowing team members to understand and debug code more easily.*
Robust Libraries and Frameworks: *Python's extensive libraries and frameworks (e.g., Django, Flask) support rapid development and simplify complex tasks, like database interactions and web security.*
Community Support: *Python has strong community support, providing access to a wealth of resources, documentation, and assistance.*

3. Now that you've had an introduction to Python, write down 3 goals you have for yourself and your learning during this Achievement. You can reflect on the following questions if it helps you. What do you want to learn about Python? What do you want to get out of this Achievement? Where or what do you see yourself working on after you complete this Achievement?

During the achievement I'd like to develop a **Web Application** and learn how to build robust and scalable web applications using Django, focusing on backend development and integrating with frontend technologies.
After the course I'd like to explore **Machine Learning and play with** libraries such as NumPy, Pandas, and TensorFlow.

Exercise 1.2: Data Types in Python

Learning Goals

- Explain variables and data types in Python
- Summarize the use of objects in Python
- Create a data structure for your Recipe app

Reflection Questions

1. Imagine you're having a conversation with a future colleague about whether to use the iPython Shell instead of Python's default shell. What reasons would you give to explain the benefits of using the iPython Shell over the default one?

Enhanced Interactivity: iPython Shell offers features like auto-completion, syntax highlighting, and better tracebacks, making coding faster and debugging easier.
Improved History: iPython provides a robust history of commands and results, allowing for easy recall and reuse of previous commands.

2. Python has a host of different data types that allow you to store and organize information. List 4 examples of data types that Python recognizes, briefly define them, and indicate whether they are scalar or non-scalar.

Data type	Definition	Scalar or Non-Scalar?
int	A number with no decimals	Scalar
Boolean	True or False	Scalar
String	A sequence of letters	Non-Scalar
Dictionary	Collection of key-value-pairs	Non-Scalar

3. A frequent question at job interviews for Python developers is: what is the difference between lists and tuples in Python? Write down how you would respond.

In Python, lists and tuples serve as collections for storing multiple items, but they differ primarily in mutability and syntax. Lists are mutable, allowing modification of elements after creation using square brackets. Tuples, denoted by parentheses, are immutable, meaning their elements cannot be changed once set. This immutability typically makes tuples faster in terms of iteration and access compared to lists, which manage dynamic resizing and additional memory overhead. Generally, use lists for collections that may change and tuples for fixed collections that remain constant throughout program execution.

4. In the task for this Exercise, you decided what you thought was the most suitable data structure for storing all the information for a recipe. Now, imagine you're creating a language-learning app that helps users memorize vocabulary through flashcards. Users can input vocabulary words, definitions, and their category (noun, verb, etc.) into the flashcards. They can then quiz themselves by flipping through the flashcards. Think about the necessary data types and what would be the most suitable data structure for this language-learning app. Between tuples, lists, and dictionaries, which would you choose? Think about their respective advantages and limitations, and where flexibility might be useful if you were to continue developing the language-learning app beyond vocabulary memorization.

For a language-learning app focused on flashcards where users input vocabulary words, definitions, and categories (like noun, verb, etc.), a dictionary would be the most suitable data structure. This choice allows each flashcard to be represented as a dictionary entry withkeys. Dictionaries provide structured storage, fast access through key-based retrieval, and flexibility for future app features such as adding example sentences or audio pronunciations. This makes dictionaries ideal for managing and expanding the content of the flashcards dynamically as users interact with the app. The dictionaries can be organized in different lists making them sortable and randomizable.

Exercise 1.3: Functions and Other Operations in Python

Learning Goals

- Implement conditional statements in Python to determine program flow
- Use loops to reduce time and effort in Python programming
- Write functions to organize Python code

Reflection Questions

1. In this Exercise, you learned how to use **if-elif-else** statements to run different tasks based on conditions that you define. Now practice that skill by writing a script for a simple travel app using an **if-elif-else** statement for the following situation:

    - The script should ask the user where they want to travel.
    - The user's input should be checked for 3 different travel destinations that you define.
    - If the user's input is one of those 3 destinations, the following statement should be printed: "Enjoy your stay in _____!"
    - If the user's input is something other than the defined destinations, the following statement should be printed: "Oops, that destination is not currently available."

Write your script here. *(Hint: remember what you learned about indents!)*

```
input_locations = input("Enter where you want to travel separated by commas: ")
travel_locations = [location.strip().lower() for location in
input_locations.split(",")]
valid_locations = ["berlin", "london", "mexico"]

found_valid_location = False

for travel_location in travel_locations:
   if travel_location in valid_locations:
       print("Enjoy your stay in " + travel_location.title())
       found_valid_location = True
       break

if not found_valid_location:
   print("Oops, that destination is not currently available")
```

2. Imagine you're at a job interview for a Python developer role. The interviewer says "Explain logical operators in Python". Draft how you would respond.

Logical operators in Python are used to perform logical operations on expressions, returning either True or False. The three main logical operators are *and*, `or`, and `not`. The *and* operator returns *True* only if both operands are *True*, the `or` operator returns *True* if at least one of the operands is *True*, and the `not` operator inverts the Boolean value. These operators are fundamental in controlling the flow of a program through conditional statements and loops.

3. What are functions in Python? When and why are they useful?

Functions in Python are reusable blocks of code designed to perform specific tasks. Functions are useful because they allow you to organize code, make it more readable and they avoid repetition. They are especially helpful when you need to perform the same operation multiple times within a program.

4. In the section for Exercise 1 in this Learning Journal, you were asked in question 3 to set some goals for yourself while you complete this course. In preparation for your next mentor call, make some notes on how you've progressed towards your goals so far.

I got a refresh of the Python syntax and its built-in comfort features like list comprehensions. I also set up the tools I needed for programming in Python on my Mac, which I didn't have the last time I played with it.

Exercise 1.4: File Handling in Python

Learning Goals

- Use files to store and retrieve data in Python

Reflection Questions

1. Why is file storage important when you're using Python? What would happen if you didn't store local files?

Effective file storage is crucial for preserving data when you exit a script, allowing you to seamlessly continue with previously gathered information. Without this storage, the data would be lost, and you would not have access to it the next time you run the program.

2. In this Exercise you learned about the pickling process with the **`pickle.dump()`** method. What are pickles? In which situations would you choose to use pickles and why?

Pickles enable the conversion of complex data into files for easy storage. I would use pickles whenever I need to maintain complex data, such as dictionaries, ensuring they are readily available when I rerun the program.

3. In Python, what function do you use to find out which directory you're currently in? What if you wanted to change your current working directory?

To find out which is the current directory in Python, the os.getcwd()-Function is used. Forn changing the directory, the os.chdir()-Function is used.

4. Imagine you're working on a Python script and are worried there may be an error in a block of code. How would you approach the situation to prevent the entire script from terminating due to an error?

I would place the code inside a try block and handle potential errors with catch blocks, providing customized error messages.

5. You're now more than halfway through Achievement 1! Take a moment to reflect on your learning in the course so far. How is it going? What's something you're proud of so far? Is there something you're struggling with? What do you need more practice with? Feel free to use these notes to guide your next mentor call.

It is going great. The task has the perfect level of difficulty for me, requiring thoughtful effort and experimentation. I feel a sense of accomplishment when the various parts function as expected. I am proud to have learned different programming concepts and to be able to apply them to real-world problems. I am not really struggling; I can solve any issues I encounter by researching them online.

Exercise 1.5: Object-Oriented Programming in Python

Learning Goals

- Apply object-oriented programming concepts to your Recipe app

Reflection Questions

1. In your own words, what is object-oriented programming? What are the benefits of OOP?

Object-oriented programming (OOP) is a programming paradigm based on the concept of objects, which contain data and methods. The benefits of OOP include code reusability, modularity, easier troubleshooting, and the ability to model real-world scenarios.

2. What are objects and classes in Python? Come up with a real-world example to illustrate how objects and classes work.

In Python, a class is a blueprint for creating objects, defining their attributes and behaviors. An object is an instance of a class, containing specific data and the ability to perform actions. For example, a "Car" class might define attributes like make, model, and color, and methods like start and stop. An object would be a specific car, such as a red Toyota Corolla, which can perform actions like starting and stopping.

3. In your own words, write brief explanations of the following OOP concepts; 100 to 200 words per method is fine.

Method	Description
Inheritance	Inheritance is a concept in object-oriented programming where a new class, called a subclass, is created based on an existing class, known as the superclass. The subclass inherits the properties and methods of the superclass, allowing for code reuse. For example, if you have a class called Vehicle with attributes like speed and methods like move, you can create a Car class that inherits from Vehicle but also adds features specific to cars, such as the number of doors. Inheritance helps organize code by creating a natural relationship between classes.
Polymorphism	Polymorphism allows objects of different classes to be treated as objects of a common superclass. It means "many forms" and lets the same method work in different ways for different classes. For example, if you have a Shape class with a method called draw, subclasses like Circle and Square can each have their own version of the draw method. When you call draw on a shape object, the correct method for that specific shape is executed. Polymorphism makes code more flexible and easier to extend.
Operator Overloading	Operator overloading allows you to define how operators like +, -, *, and / work with objects of a class. For example, if you have a Vector class to represent mathematical vectors, you can overload the + operator to add two Vector objects together using the same syntax as adding numbers. This makes your code easier to read and write, as custom objects can be manipulated using familiar operators. However, it's important to use operator overloading carefully to keep the behavior of your operators intuitive and clear.

Exercise 1.6: Connecting to Databases in Python

Learning Goals

- Create a MySQL database for your Recipe app

Reflection Questions

1. What are databases and what are the advantages of using them?

In programming, databases are crucial for storing and managing data used by applications. They enable programmers to efficiently retrieve and update data, ensure consistency and integrity, and handle large datasets. Databases also allow multiple applications and users to access and manipulate the data concurrently, which is essential for dynamic and scalable software development. This makes data handling more reliable and simplifies complex data-related tasks in programming.

2. List 3 data types that can be used in MySQL and describe them briefly:

Data type	Definition
INT	Used to store whole numbers without decimals
VARCHAR	A String. The maximum length has to be specified
Date	Used to store date values in the format 'YYYY-MM-DD'.

3. In what situations would SQLite be a better choice than MySQL?

SQLite is a better choice than MySQL in situations where simplicity, low resource usage, and minimal setup are needed.

4. Think back to what you learned in the Immersion course. What do you think about the differences between JavaScript and Python as programming languages?

JavaScript and Python have distinct strengths that suit different needs. I find JavaScript great for making web pages interactive and dynamic. It runs directly in the browser, making it a go-to for front-end development and handling events smoothly.

On the other hand, Python feels like a more versatile tool in my programming toolkit. Its code readability and simplicity are refreshing, especially when working on diverse projects like data analysis, machine learning, or scripting. However, Python requires proper indentation to work correctly, which helps maintain clean and organized code.

In essence, I turn to JavaScript when building web interfaces and Python when I need a powerful, easy-to-read language for a variety of tasks.

5. Now that you're nearly at the end of Achievement 1, consider what you know about Python so far. What would you say are the limitations of Python as a programming language?

ChatGPT

Python, despite its many strengths, has a few limitations. Its execution speed can be slower compared to compiled languages like C or Java, which might be an issue for performance-critical applications. Additionally, Python's dynamic typing can lead to runtime errors that are harder to catch compared to statically-typed languages.

Exercise 1.7: Finalizing Your Python Program

Learning Goals

- Interact with a database using an object-relational mapper
- Build your final command-line Recipe application

Reflection Questions

1. What is an Object Relational Mapper and what are the advantages of using one?

An Object Relational Mapper (ORM) is a tool that allows developers to interact with a database using Python objects instead of writing SQL queries. It translates Python code into SQL and manages database records as if they were Python objects. The advantages of using an ORM include simplicity, as you can interact with the database using Python code, which is easier than writing SQL; consistency, as ORMs provide a consistent way to interact with different types of databases; security, since ORMs help prevent SQL injection attacks by handling query parameters safely; productivity, as they reduce the amount of boilerplate code and speed up development; and portability, making it easier to switch between different database systems since the ORM abstracts the database-specific details.

2. By this point, you've finished creating your Recipe app. How did it go? What's something in the app that you did well with? If you were to start over, what's something about your app that you would change or improve?

Creating the Recipe app went well overall. One aspect that turned out particularly well was the seamless integration of SQLAlchemy for database communication, making data handling efficient and straightforward. If I were to start over, I would focus on improving the user interface to make it more intuitive and user-friendly, perhaps by adding more interactive features or better navigation options.

3. Imagine you're at a job interview. You're asked what experience you have creating an app using Python. Taking your work for this Achievement as an example, draft how you would respond to this question.

In my recent project, I developed a command-line app for managing cooking recipes using Python. The app leverages SQLAlchemy to handle database interactions, making it robust and efficient. Throughout the development process, I focused on creating a seamless experience for users to add, view, and manage their recipes. This project improved my skills in database management, Python programming, and creating practical applications.

4. You've finished Achievement 1! Before moving on to Achievement 2, take a moment to reflect on your learning in the course so far:
    a. What went well during this Achievement?

During Achievement 1, I found that the integration of SQLAlchemy with the Python app went particularly well. This allowed for efficient and straightforward management of the database, making data handling much easier. Additionally, my understanding of database interactions and Python programming improved significantly, and I successfully created a functional and user-friendly app.

    b. What's something you're proud of?

That the App works as suggested 😀

    c. What was the most challenging aspect of this Achievement?

Dealing with user input and passing the data in the right format.

    d. Did this Achievement meet your expectations? Did it give you the confidence to start working with your new Python skills?

Absolutely!

    e. What's something you want to keep in mind to help you do your best in Achievement 2?

Break complex tasks into small steps.

Well done—you've now completed the Learning Journal for Achievement 1. As you'll have seen, a little metacognition can go a long way!

Pre-Work: Before You Start Achievement 2

In the final part of the learning journal for Achievement 1, you were asked if there's anything—on reflection—that you'd keep in mind and do similarly or differently during Achievement 2. Think about these questions again:

- Was your study routine effective during Achievement 1? If not, what will you do differently during Achievement 2?
- Reflect on your learning and project work for Achievement 1. What were you most proud of? How will you repeat or build on this in Achievement 2?
- What difficulties did you encounter in the last Achievement? How did you deal with them? How could this experience prepare you for difficulties in Achievement 2?

Note down your answers and discuss them with your mentor in a call if you like.

Remember that can always refer to Exercise 1.4 of the Orientation course if you're not sure whom to reach out to for help and support.

Exercise 2.1: Getting Started with Django

Learning Goals

- Explain MVT architecture and compare it with MVC
- Summarize Django's benefits and drawbacks
- Install and get started with Django

Reflection Questions

1. Suppose you're a web developer in a company and need to decide if you'll use vanilla (plain) Python for a project, or a framework like Django instead. What are the advantages and drawbacks of each?

Using vanilla Python for a web development project offers the advantage of complete control and flexibility. You can build everything from scratch, which means the application can be highly customized to meet specific needs. This approach can be lightweight and efficient since you only include the features you need. However, it also requires more time and effort because you need to handle everything yourself, from routing and templating to security and database interactions.

On the other hand, using a framework like Django simplifies the development process significantly. Django provides a robust set of tools and features out of the box, such as an ORM for database management, an admin interface, user authentication, and routing. This allows for rapid development and ensures that best practices and security measures are followed. The drawback is that it can be more rigid and may include features you don't need, potentially making the application heavier.

2. In your own words, what is the most significant advantage of Model View Template (MVT) architecture over Model View Controller (MVC) architecture?

The most significant advantage of the Model View Template (MVT) architecture over the Model View Controller (MVC) architecture is simplicity in development and maintenance. In MVT, the framework handles much of the controller logic for you, allowing you to focus more on the actual application logic and the design of the user interface. This reduces the amount of code you need to write and maintain, making it easier and faster to develop web applications. Essentially, MVT abstracts away some of the complexities, allowing developers to work more efficiently.

3. Now that you've had an introduction to the Django framework, write down three goals you have for yourself and your learning process during this Achievement. You can reflect on the following questions if it helps:
    - What do you want to learn about Django?

I want to gain a solid understanding of the fundamental components of Django, such as models, views, and templates. This includes learning how they interact with each other to build a web application. By mastering these core concepts, I aim to feel confident in setting up a basic Django project from scratch.

- What do you want to get out of this Achievement?

My goal is to develop a cooking Recipe application using Django. This will involve creating a database and and deploying the application. Through this hands-on experience, I want to ensure that I can apply what I've learned in real-world scenarios.

- Where or what do you see yourself working on after you complete this Achievement?

I want to work on an Application called *Sunhunt*, which tells you where to drive depending on weather conditions. I planned to do it in React. Depending on my experiences in this exercise I might consider if Django might be a fit.

Exercise 2.2: Django Project Set Up

Learning Goals

- Describe the basic structure of a Django project
- Summarize the difference between projects and apps
- Create a Django project and run it locally
- Create a superuser for a Django web application

Reflection Questions

1. Suppose you're in an interview. The interviewer gives you their company's website as an example, asking you to convert the website and its different parts into Django terms. How would you proceed? For this question, you can think about your dream company and look at their website for reference.
   (*Hint: In the Exercise, you saw the example of the CareerFoundry website in the Project and Apps section.*)

I pick the site of the New York Times (https://www.nytimes.com/). In Django-Terms the whole website is a Project, whereas the different functionalities (Login, Articles, Videos, Contact etc.) are Apps.

2. In your own words, describe the steps you would take to deploy a basic Django application locally on your system.

To deploy a basic Django application locally, start by ensuring you have Python installed on your system, which you can download from the official Python website or via Homebrew.
Next create a Virtual Enviroment.
With the virtual environment activated, install Django by running *pip install django*. After installing Django, create a new Django project by running *django-admin startproject myproject*.. Change into your project directory by running cd myproject.
Next initialize the database by running *python manage.py migrate*. Create a superuser account for the admin interface by running *python manage.py createsuperuser* and following the prompts to set up a username, email, and password.

Finally, start the development server to view your project locally by running *python manage.py runserver*. Open a web browser and go to http://127.0.0.1:8000/ to see your Django application in action. You can access the admin interface at http://127.0.0.1:8000/admin/ using the superuser credentials you created.

3. Do some research about the Django admin site and write down how you'd use it during your web application development.

The Django admin site is a built-in tool that allows you to manage the data in your web application easily. It provides a user-friendly interface where you can add, edit, and delete data records without writing any code.
During development, the admin site is particularly useful for testing and managing your application's data without having to interact directly with the database. This makes it a powerful tool for efficiently managing and reviewing the content of your web application.

Exercise 2.3: Django Models

Learning Goals

- Discuss Django models, the "M" part of Django's MVT architecture
- Create apps and models representing different parts of your web application
- Write and run automated tests

Reflection Questions

1. Do some research on Django models. In your own words, write down how Django models work and what their benefits are.

Django models are a fundamental part of the Django framework, used to define and manage the data structure of your web application. They represent the data in your database, making it easier to interact with the database using Python code rather than writing raw SQL queries.

When you define a Django model, you are essentially creating a blueprint for how data should be stored in the database. Each model corresponds to a table in the database, and each attribute of the model represents a column in that table. Django then takes care of creating the necessary database schema, managing the data, and providing an interface to perform operations like creating, reading, updating, and deleting records.

The benefits of using Django models are:

- **Simplified database management**: Django automatically handles the creation and modification of database tables based on your model definitions, reducing the need for manual SQL scripting.
- **Consistency and reusability**: Models ensure that data structure is consistent across your application, and they can be reused in different parts of your project.

- **Scalability**: Django models are designed to work well with large databases and can be easily adapted as your application grows.

2. In your own words, explain why it is crucial to write test cases from the beginning of a project. You can take an example project to explain your answer.

Writing test cases from the beginning of a project is crucial because it ensures that your application works correctly as you develop it and makes it easier to catch and fix errors early on. Think of it like testing a recipe while you're cooking—if you taste the dish at different stages, you can catch mistakes like too much salt or missing ingredients before the meal is finished.

In the context of a cooking recipe app, imagine you're building a feature that lets users add new recipes. If you write test cases from the start, you can verify that the "add recipe" function works as expected, such as ensuring that the recipe is saved correctly, all required fields are filled out, and that the app behaves properly if something goes wrong (like entering too many characters for a field).

If you wait until the end of the project to write tests, you might find that there are bugs hidden deep in your code, which can be much harder to fix later. Early testing helps prevent these issues and saves time in the long run. It also gives you confidence that your app is reliable and works as intended, making it easier to build new features without breaking existing ones.

Exercise 2.4: Django Views and Templates

Learning Goals

- Summarize the process of creating views, templates, and URLs
- Explain how the “V” and “T” parts of MVT architecture work
- Create a frontend page for your web application

Reflection Questions

1. Do some research on Django views. In your own words, use an example to explain how Django views work.

Django views are functions or classes in Python that handle requests from users and return responses. When a user visits a URL on your website, Django uses a view to determine what content should be shown.

For example, imagine a blog website. When a user visits the homepage to see a list of blog posts, Django will use a view to fetch those posts from the database and display them. If the user clicks on a specific post, another view will handle the request, fetch that post's details, and render it on the screen.

2. Imagine you're working on a Django web development project, and you anticipate that you'll have to reuse lots of code in various parts of the project. In this scenario, will you use Django function-based views or class-based views, and why?

In a scenario where you anticipate reusing a lot of code, class-based views (CBVs) in Django would be a better choice. CBVs allow you to structure your code using inheritance, making it easier to reuse and extend functionality across different views. With CBVs, you can create a base class with common behavior and then create specific views by extending that base class, reducing duplication and making your code more maintainable.

3. Read Django's documentation on the Django template language and make some notes on its basics.

- Template Tags: These are special syntax constructs enclosed in *{% %}* that perform various actions, such as loops, conditionals, or including other templates.
- Variables are enclosed in *{{ }}* and are used to insert dynamic content into the template.
- Filters: Filters modify the display of variables. They are applied using the | symbol, like *{{ post.title|upper }}* which would convert the title to uppercase.
- Comments: You can add comments in templates using *{# #}*. These comments won't be rendered in the final HTML.
- Template Inheritance: This allows you to create a base template with common structure (like headers and footers) and extend it in other templates, promoting reuse and consistency.
- Static Files: *{% static %}* is used to link to static files like CSS or JavaScript.

Exercise 2.5: Django MVT Revisited

Learning Goals

- Add images to the model and display them on the frontend of your application
- Create complex views with access to the model
- Display records with views and templates

Reflection Questions

1. In your own words, explain Django static files and how Django handles them.

In Django, "static files" are files like images, CSS, JavaScript, and other resources that are not dynamically generated by the server but are instead served as-is to the user. For example, when a webpage needs an image or a specific style, it pulls these from the static files.

2. Look up the following two Django packages on Django's official documentation and/or other trusted sources. Write a brief description of each.

Package	Description
ListView	A generic view provided by Django that simplifies displaying a list of objects from a database.
DetailView	Another generic view in Django that focuses on displaying a single object or record from the database.

3. You're now more than halfway through Achievement 2! Take a moment to reflect on your learning in the course so far. How is it going? What's something you're proud of so far? Is there something you're struggling with? What do you need more practice with? You can use these notes to guide your next mentor call.

My experience with the course has been going quite well so far. At the start of Exercise 2, I was concerned that it might be challenging to adapt the course material to the custom boilerplate I'm using. However, I'm pleased to say that I've managed to understand how the structure works. One highlight for me was the moment I saw the recipes displayed on the screen for the first time—it was incredibly satisfying. I would like to have more information on how to best implement a navbar into the App.

Exercise 2.6: User Authentication in Django

Learning Goals

- Create authentication for your web application
- Use GET and POST methods
- Password protect your web application's views

Reflection Questions

1. In your own words, write down the importance of incorporating authentication into an application. You can take an example application to explain your answer.

Authentication is crucial for any app because it ensures that only the right users can access their personal data and features. For example, in a cooking recipe app, authentication ensures that users can log in to view and manage their own recipes without others interfering. This keeps their information secure and allows for personalized experiences, like saving favorite recipes or creating meal plans. Without authentication, anyone could access or change another user's data, which would compromise the app's security and user trust.

2. In your own words, explain the steps you should take to create a login for your Django web application.

Set up a login form and a URL route that points to the login view. Use Django's tools to handle authentication, checking user credentials when they submit the form. If successful, redirect them to a specified page.. Finally, add a logout view to allow users to end their session.

3. Look up the following three Django functions on Django's official documentation and/or other trusted sources and write a brief description of each.

Function	Description
authenticate()	This function is used to verify a user's credentials. Typically, you pass in a username and password, and if the credentials are valid, it returns the user object.
redirect()	This function is used to redirect the user to a different URL. It can take various forms, such as a URL string, a view name, or an object.
include()	This function allows you to include other URL configurations in your Django project's urls.py. It's commonly used to reference URL patterns from different apps within the same project, helping to organize and modularize the URL structure.

Exercise 2.7: Data Analysis and Visualization in Django

Learning Goals

- Work on elements of two-way communication like creating forms and buttons
- Implement search and visualization (reports/charts) features
- Use QuerySet API, DataFrames (with pandas), and plotting libraries (with matplotlib)

Reflection Questions

1. Consider your favorite website/application (you can also take CareerFoundry). Think about the various data that your favorite website/application collects. Write down how analyzing the collected data could help the website/application.

Reddit collects various data like user interactions (upvotes, downvotes, comments), browsing habits (subreddits visited, time spent), and user demographics (age, location). By analyzing this data, Reddit can improve user experience by showing more relevant content, customizing recommendations, and suggesting popular subreddits. It can also help Reddit understand what topics are trending, identify spam or harmful content, and make better decisions about ads to show users based on their preferences and behavior.

2. Read the Django official documentation on QuerySet API. Note down the different ways in which you can evaluate a QuerySet.

In Django, a QuerySet is evaluated when certain actions are performed. By default, QuerySets are lazy, meaning they don't fetch data from the database until required. Some actions that trigger evaluation include iterating over the QuerySet, slicing it, or converting it to a list. Additionally, methods like len(), count(), exists(), get(), first(), and last() will cause the QuerySet to hit the database. Similarly, printing or accessing the data in a QuerySet will also force evaluation.

3. In the Exercise, you converted your QuerySet to DataFrame. Now do some research on the advantages and disadvantages of QuerySet and DataFrame, and explain the ways in which DataFrame is better for data processing.

A Django QuerySet is great for efficiently querying and interacting with the database, with lazy evaluation and easy filtering. However, a Pandas DataFrame is more powerful for data processing and manipulation. DataFrames excel in tasks like complex transformations, statistical analysis, and handling large datasets in memory. While QuerySets are limited by their database focus, DataFrames offer flexibility for detailed, in-memory data analysis, making them more suitable for tasks beyond basic querying.

Exercise 2.8: Deploying a Django Project

Learning Goals

- Enhance user experience and look and feel of your web application using CSS and JS
- Deploy your Django web application on a web server
- Curate project deliverables for your portfolio

Reflection Questions

1. Explain how you can use CSS and JavaScript in your Django web application.

In Django, you can use CSS and JavaScript by adding them to the "static" directory of your app. Django serves these files when you reference them in your HTML templates. For CSS, link to the stylesheet in the HTML <head>, and for JavaScript, you can place script tags either in the <head> or before the closing <body> tag to load your JavaScript files. Django's {% static %} template tag is used to correctly point to these static files.

2. In your own words, explain the steps you'd need to take to deploy your Django web application.

To deploy your Django web application, you'd start by pushing your project to GitHub. On Heroku, you would set up a new app and connect it to your GitHub repository. In Heroku, you'd configure environment variables (like SECRET_KEY and API keys) to keep sensitive information safe. Then, you'd install necessary add-ons like Heroku Postgres for your database and Cloudinary for media storage. Finally, you would push your production-ready code to Heroku, making sure your `prod.py` file is used and any production-specific dependencies are installed (like Gunicorn for serving the app).

3. You've now finished Achievement 2 and, with it, the whole course! Take a moment to reflect on your learning:
    a. What went well during this Achievement?
    b. What's something you're proud of?
    c. What was the most challenging aspect of this Achievement?
    d. Did this Achievement meet your expectations? Did it give you the confidence to start working with your new Django skills?

Reflecting on Achievement 2, I'd say that my understanding of deploying Django applications went really well, especially managing different environments and working with external services like Heroku and Cloudinary. I'm proud of how I handled the production settings and learned to manage sensitive information securely with environment variables.

The most challenging aspect was getting everything configured properly for production, particularly the integration of media storage and databases. I expected these challenges, but working through them gave

me confidence in applying my Django skills in real-world scenarios. I definitely feel prepared to build and deploy more complex Django applications now!

Well done—you've now completed the Learning Journal for the whole course.