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 studied Mechatronics in the past, for this reason I am not so new to programming.

2. What do you know about Python already? What do you want to know? I know Python is a very flexible programming language, and I would like to learn as much as I can to grow together with AI and Machine Learning.
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.
   Probably some kind of lack of information, since learning a language is a long journey.

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 is what the user sees, Backend is what is behind the scenes. I would be working on databases, server requests architecture, security like authentication & authorization, APIs, etc.

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? Because of its simplicity to deal with data. *(Hint: refer to the Exercise section "The Benefits of Developing with Python")*

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?

My Goals for Learning Python:

1. **Master Basic Python Concepts:**
    - **What I want to learn:** I want to thoroughly understand Python syntax, basic data structures, and control flow mechanisms like loops and conditionals.
    - **Outcome I want:** Gain confidence in writing basic Python scripts and solving simple programming problems.
    - **Future vision:** I see myself working on small projects and solving coding challenges to reinforce my understanding of Python fundamentals.
2. **Develop Problem-Solving Skills:**
    - **What I want to learn:** I want to improve my ability to break down complex problems into smaller, manageable parts using Python.
    - **Outcome I want:** Be able to tackle more challenging tasks and write efficient, clean code.
    - **Future vision:** I aim to contribute to open-source projects or collaborate on coding projects with others to hone my problem-solving skills.
3. **Build Real-World Applications:**
    - **What I want to learn:** I want to learn how to apply Python to real-world scenarios, such as web development, data analysis, or automation.
    - **Outcome I want:** Create projects or applications that demonstrate my skills.
    - **Future vision:** After completing this Achievement, I hope to work on more significant projects, potentially landing an entry-level position as a Python developer.

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? It is convenient to use for easy and quick test of small chunks of code, and better visulization.

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	Represent whole numbers	scalar
float	Represent numbers with decimals	scalar
str	Represent characters, like words	non-scalar
bool	Represent true and false values	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.
   Lists in Python are mutable, meaning they can be changed, while tuples are immutable and cannot be altered once created.
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.
   I would choose dictionaries because they let you store each vocabulary word with its definition and category, making it easy to look up and organize the information.

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!)*

```
destination = input("Where do you want to travel? ")

if destination == "Paris":
    print("Enjoy your stay in Paris!")
elif destination == "New York":
    print("Enjoy your stay in New York!")
elif destination == "Tokyo":
    print("Enjoy your stay in Tokyo!")
else:
    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, like and, or, and not, are used to combine or invert conditions in if statements. They help control the flow of the program based on multiple conditions.

3. What are functions in Python? When and why are they useful? Functions in Python are blocks of reusable code that perform a specific task. They help organize code, make it more readable, and reduce redundancy. You use them when you need to perform the same operation multiple times or break down a complex problem into smaller, manageable parts.

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.

   Progress Towards Goals:

   - **Goal 1:** Understand basic Python syntax and structure.
     - **Progress:** I've completed exercises on conditionals and loops, gaining a solid understanding of how to control program flow.
   - **Goal 2:** Learn to write clean, reusable code.
     - **Progress:** Practiced writing functions to organize code, making it more modular and easier to debug.
   - **Goal 3:** Build small projects to apply what I've learned.
     - **Progress:** Started creating small scripts, like a simple travel app, to apply the concepts of conditionals and functions in real scenarios.

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? File storage is important because it allows you to save data for later use, making it possible to access and manipulate that data even after your program ends. Without storing local files, you'd lose all the data each time the program closes, making it hard to maintain any kind of persistent information.

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 are a way to serialize and save Python objects to a file so you can load them back later. You'd use pickles when you need to save complex data structures, like dictionaries or custom objects, and easily restore them without having to manually parse and reconstruct the data.

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 the current directory, use `os.getcwd()`. To change the current working directory, use `os.chdir('path_to_new_directory')`.

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? Use a `try-except` block to catch and handle errors gracefully. This way, if an error occurs in the block of code, the script can continue running or provide a useful error message instead of just stopping.

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.
   How is it going? It's going well; I'm learning a lot about Python basics and file handling.
   Something you're proud of so far? I'm proud of understanding and applying concepts like loops, conditionals, and file handling.
   Something you're struggling with? I'm finding it a bit challenging to remember all the syntax and functions.
   What do you need more practice with? I need more practice with writing and organizing functions and using file handling effectively.

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 way of designing software by organizing code into "objects." Objects are like little building blocks that combine data (like variables) and functions (like methods) into one unit. These objects are created from "classes," which are like blueprints or templates.
   Benefits of OOP:

   - **Organization**: It helps keep code organized and manageable by grouping related data and functions together.
   - **Reusability**: You can reuse code from one class in other parts of the program or in new programs.
   - **Modularity**: Each object is a self-contained unit, which makes it easier to fix problems and update code.

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. It defines a type of object and its behavior. An **object** is an instance of a class. It is created using the class blueprint and contains data and functions defined in the class.

   Real-World Example:

   - **Class**: Think of a `Car` class as a blueprint for creating cars. It might have attributes like `color`, `model`, and `speed`, and methods like `accelerate()` and `brake()`.
   - **Object**: An individual car, like a red Toyota Corolla, is an object created from the `Car` class. It has specific attributes and can perform actions defined by the `Car` class.

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 way to create a new class that is based on an existing class. The new class, called a "child" or "subclass," inherits attributes and methods from the existing class, known as the "parent" or "superclass." This helps avoid code duplication because the child class can use or override the parent class's features. For example, if we have a `Dessert` class that inherits from a `Recipe` class, the `Dessert` class would get all the attributes and methods of `Recipe` but can also add its own unique features.
Polymorphism	Polymorphism allows different classes to be treated as instances of the same class through a common interface, typically using methods. It means that the same method name can be used in different classes but can work differently depending on which class is calling it. For example, a `print_recipe()` method might show details differently for a `DinnerRecipe` class than for a `DessertRecipe` class. Polymorphism makes code more flexible and easier to extend.
Operator Overloading	Operator overloading lets you define how operators (like +, -, *, etc.) work with objects of a class. By overloading operators, you can make them perform custom operations for your objects. For instance, if you have a `Recipe` class and you want to add two recipes together to combine their ingredients, you can overload the + operator to perform this task. This makes your classes more intuitive and integrates them better with Python's syntax.

Just for “me” to remember well:

Method: A method is a function that is defined inside a class and is meant to operate on the data within that class. Methods can access and modify the object's attributes and can perform actions related to the object. For example, in a `Recipe` class, a method like `add_ingredients()` might add new ingredients to a recipe's list. Methods help objects perform tasks and interact with their data.

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?

   Databases are organized collections of data that are stored and managed electronically. They help keep data structured and easily accessible. Here are some advantages of using databases:

   - **Organization:** Databases organize data in tables, which makes it easier to find and manage.
   - **Efficiency:** They allow for fast retrieval and manipulation of data using queries.
   - **Consistency:** Databases ensure that data is accurate and up-to-date through features like constraints and transactions.

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

Data type	Definition
INT	Stores integer numbers (e.g., 1, 2, 3). Useful for counting or IDs.
VARCHAR	Stores variable-length text strings (e.g., names or descriptions). You specify the maximum length.
DATE	Stores dates (e.g., 2024-08-05). Useful for keeping track of events or deadlines.

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

   SQLite might be a better choice than MySQL in the following situations:

   - **Single-user Applications:** When only one user is interacting with the database, such as in a desktop application.
   - **Small to Medium-Sized Data:** When the application doesn't need to handle large volumes of data or high traffic.
   - **Simplicity:** When you need a simple, self-contained database with minimal setup, such as for a small project or a prototype.

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 are both powerful languages but serve different purposes:

   - **JavaScript:** Primarily used for making web pages interactive. It runs in the browser and can dynamically change content on websites.
   - **Python:** Known for its simplicity and readability, it's used for a variety of tasks like web development, data analysis, automation, and more. Python has straightforward syntax, which makes it easier for beginners to learn.

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?

   Some limitations of Python may include:

   - **Performance:** Python is slower compared to languages like C or Java because it is interpreted rather than compiled.
   - **Memory Usage:** Python can consume more memory, which might be an issue for memory-constrained environments.
   - **Mobile Development:** Python is not as commonly used for mobile app development as languages like Swift (for iOS) or Kotlin (for Android).

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 programming tool that allows developers to interact with a relational database using object-oriented code instead of raw SQL queries. It acts as a bridge between the database and the application, converting data between incompatible type systems in object-oriented programming languages.

The advantages of using an ORM include:

- **Simplified Database Interaction:** ORMs abstract away the complexity of SQL, allowing you to work with objects and methods in your programming language instead of writing raw SQL queries.
- **Increased Productivity:** By reducing the need for boilerplate SQL code, ORMs can speed up development time and reduce errors.
- **Better Code Organization:** ORMs help in organizing database-related code in a structured and manageable way.
- **Cross-Database Compatibility:** ORMs often support multiple database systems, making it easier to switch databases without changing your application code.

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 was a rewarding experience. I learned a lot about integrating a database with Python using SQLAlchemy. One aspect I did well with was setting up the basic CRUD (Create, Read, Update, Delete) functionalities. The app successfully performs these operations and interacts with the database as intended.

   If I were to start over, I would focus on improving the user interface. While the command-line interface is functional, a graphical user interface (GUI) could make the app more user-friendly and engaging. Additionally, I would add more validation and error handling to make the application more robust and user-friendly.

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 Recipe management application using Python and SQLAlchemy. This application allows users to create, view, update, and delete recipes through a command-line interface. I utilized SQLAlchemy as an Object Relational Mapper to handle database interactions, which simplified the process of managing data. Through this project, I gained hands-on experience in database design, CRUD operations, and integrating Python with a database system. This experience has enhanced my skills in Python programming and provided me with practical knowledge in building and managing data-driven 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?
   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 Python skills?
   e. What's something you want to keep in mind to help you do your best in Achievement 2?

During this Achievement, I was pleased with how well I grasped the concept of Object-Relational Mapping and how it was implemented in my Recipe app. I'm particularly proud of successfully integrating SQLAlchemy and ensuring that the CRUD operations worked seamlessly.

The most challenging aspect was understanding the intricacies of SQLAlchemy and how it maps Python objects to database tables. Balancing this with the need to manage and debug the command-line interface was also a bit confusing.

Overall, this Achievement met my expectations by providing practical experience with ORMs and database management in Python. It has definitely boosted my confidence in working with databases and Python. For Achievement 2, I want to focus on enhancing the user experience and diving deeper into advanced features of ORMs and Python frameworks.

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? My study routine was effective, but I found that breaking down tasks into smaller chunks and setting specific goals for each study session worked best. For Achievement 2, I plan to continue this approach and possibly incorporate more practice with real-world projects.
- 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? I am most proud of successfully implementing the CRUD operations and understanding ORM concepts. I will build on this by exploring more advanced ORM features and integrating additional functionalities into future projects.
- 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? The primary difficulty was understanding SQLAlchemy's syntax and features. I dealt with this by reviewing documentation, checking google examples. This experience has taught me to approach challenges methodically and seek resources when needed, which will be valuable for tackling 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
    - **Model-View-Template (MVT) Architecture:**
        - **Model:** Manages the data and business logic. In Django, this is represented by database models.
        - **View:** Manages what data is presented to the user and how it is displayed. In Django, views retrieve data from models and pass it to templates.
        - **Template:** Defines how the data sent by the view is presented to the user. It is a mix of HTML and Django Template Language.
    - **Model-View-Controller (MVC) Architecture:**
        - **Model:** Manages the data and business logic.
        - **View:** Displays the data to the user.
        - **Controller:** Handles the user input, processes it, and sends it to the model, and updates the view.

    Comparison:

    - **Similarity:** Both architectures aim to separate concerns in web applications, making them more organized and maintainable.
    - **Difference:** In Django, what is traditionally the "Controller" in MVC is handled by the framework itself, which maps URLs to views, simplifying the developer's task. Thus, Django replaces "Controller" with "View" and "Template" to handle display logic separately.
- Summarize Django's benefits and drawbacks
    - **Benefits:**
        - **Rapid Development:** Django provides many built-in features like authentication, ORM, and an admin interface, which speeds up development.
        - **Security:** Django includes protection against common security threats like SQL injection, cross-site scripting, and cross-site request forgery.
        - **Scalability:** Django is designed to help developers build scalable web applications.
    - **Drawbacks:**
        - **Monolithic:** Django is a full-stack framework, which can be overkill for small projects.
        - **Steep Learning Curve:** Due to its many built-in features and the "Django way" of doing things, it might take some time to learn.
        - **Heavyweight:** For simple applications, the overhead of using Django can be more than necessary.

- 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?

   Vanilla Python:

   - **Advantages:**
     - **Flexibility:** You have complete control over how you design your application.
     - **Lightweight:** No unnecessary features, only what you need.
   - **Drawbacks:**
     - **Slower Development:** You have to build everything from scratch, which takes more time.
     - **Less Security:** You have to implement security features manually, which can lead to vulnerabilities if not done correctly.
     - **Maintenance:** More effort is required to maintain and update the code.

   Django:

   - **Advantages:**
     - **Built-in Features:** Comes with many features like ORM, admin interface, and authentication, which saves development time.
     - **Security:** Provides built-in protection against common security threats.
     - **Community and Documentation:** Large community and extensive documentation make it easier to find help and resources.
   - **Drawbacks:**
     - **Learning Curve:** Takes time to learn and understand the framework fully.
     - **Overhead:** Can be overkill for simple projects, adding unnecessary complexity.
     - **Less Flexibility:** You have to follow the "Django way" of doing things, which might limit some customizations.
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 MVT architecture over MVC is that MVT simplifies development by handling the controller part internally. This means that developers can focus on defining how data is presented (views) and how it looks (templates), making development faster and easier.

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?
      Goal 1: Understand Django's Core Features I want to learn about Django's core features like models, views, templates, and the admin interface to build complete web applications.
    - What do you want to get out of this Achievement?
      Goal 2: Build a Functional Web Application By the end of this Achievement, I aim to create a functional web application that includes user authentication, data handling, and a user-friendly interface.
    - Where or what do you see yourself working on after you complete this Achievement?
      Goal 3: Gain Practical Experience I want to gain practical experience by working on real-world projects or assignments using Django, so I can apply what I've learned in future job roles or personal projects.

    Reflection:

    - **What I Want to Learn:** I want to learn how to effectively use Django to build and manage web applications, focusing on its powerful features and best practices.
    - **What I Want to Get Out of This Achievement:** I aim to have a solid understanding of Django, enabling me to develop web applications confidently and efficiently.
    - **Future Aspirations:** After completing this Achievement, I see myself working on more complex web projects, possibly contributing to open-source Django projects, or using my skills in a professional setting.

Exercise 2.2: Django Project Set Up

Learning Goals

- Describe the basic structure of a Django project

Describe the basic structure of a Django project:

- A Django project contains settings and configuration for your entire web application. It consists of several key files and folders:
    - `manage.py`: A command-line utility that lets you interact with your project.
    - `project_name/`: A folder with settings and configuration specific to your project.
    - `project_name/settings.py`: Settings and configurations for the project.

    - `project_name/urls.py`: URL declarations for the project.
    - `project_name/wsgi.py`: An entry-point for WSGI-compatible web servers.
    - `project_name/asgi.py`: An entry-point for ASGI-compatible web servers.

- Summarize the difference between projects and apps
  A **project** is the overall container for your web application, containing settings and configurations. An **app** is a component within the project that does a specific function. For example, if you have a blog, the blog itself is an app within the larger project.

- Create a Django project and run it locally
  Install Django in a virtual environment.
  Use `django-admin startproject project_name` to create a new project.
  Navigate to the project directory and run `python manage.py runserver` to start the development server.

- Create a superuser for a Django web application
  Run `python manage.py createsuperuser` and follow the prompts to set up a username, email(optional), and password.

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.*)

   The dream company has a website with different sections like Home, About, Services, Blog, and Contact.
   In Django terms, the **project** would be the overall website.
   Each section (Home, About, Services, Blog, Contact) could be an **app**.

   - **Home app**: Manages the homepage content.
   - **About app**: Manages the about page content.
   - **Services app**: Manages the services offered.
   - **Blog app**: Manages blog posts and comments.
   - **Contact app**: Manages the contact form and submissions.

   The project's `urls.py` would include routes that direct to each app's URLs.

2. In your own words, describe the steps you would take to deploy a basic Django application locally on your system.
   Set up your environment: Create a new project folder, then create and activate a virtual environment.
   Install Django: Run `pip install django`.
   Start a new project: Use `django-admin startproject project_name`.
   Navigate to the project: `cd project_name`.
   Run the server: Use `python manage.py runserver` and visit `http://127.0.0.1:8000/` in your browser.
   Create an app: Run `python manage.py startapp app_name` and add it to the project's settings.
   Define models and views: In the app's files, create your database models and views.
   Create templates and URLs: Define how your pages will look and how they can be accessed.

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 interface for managing your web application's data.

   - You can use it to:
     - **Add, update, and delete** entries in your database.
     - **Create and manage user accounts**.
     - **Monitor site activities**: View logs and manage site content efficiently.
   - To use it:
     - Register your models in the admin site by adding them to `admin.py` in your app.
     - Access it at `http://127.0.0.1:8000/admin/` and log in with your superuser credentials.
   - This tool is especially useful during development for quickly testing and managing data without needing to write complex SQL queries.

   By following these steps and using the Django admin site effectively, you can manage and develop your Django web application more efficiently.

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 way to define the structure of our database tables in code. Think of a model as a blueprint for creating database records. Each model is a class in Python, and each attribute of that class represents a field in the database.

   For example, if we're building a recipe app, we might have a `Recipe` model with attributes like `title`, `ingredients`, and `instructions`. Django takes care of creating the database table based on this model when you run migration commands.

   Benefits of Django models include:

   - **Ease of Use**: You don't need to write raw SQL to create or modify database tables. You just define your models in Python.
   - **Automatic Admin Interface**: Django automatically creates an admin interface for managing your models, which makes it easy to add, edit, or delete records.
   - **Consistency**: Models help ensure that your database structure is consistent with your application's needs because you define it in one place.
   - **Validation**: You can add validation rules directly to your models, ensuring that only valid data gets saved to the database.
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 really important because it helps catch bugs early and makes sure that your code is working as expected.

   For example, imagine you're building a recipe app and you write a test case to check if recipes can be added correctly. If you write this test case when you first build the recipe model, you can quickly see if any changes you make later break the functionality.

   Testing from the start:

   - **Catches Bugs Early**: If something is wrong, you'll find out sooner rather than later.
   - **Ensures Reliability**: You can be more confident that new changes won't break existing features.
   - **Saves Time**: Fixing bugs early is usually quicker than fixing them later, especially when your codebase gets larger.

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 like the brains of your web application. They handle the request from a user, interact with the database if needed, and then send back a response, which is usually an HTML page.

   For example, if someone visits the home page of your recipe app, the view function for the home page would:

   - Get the data needed (like a list of recipes).
   - Prepare the data to be displayed.
   - Return an HTML page that shows the list of recipes.

   In code, a view function might look like this:

   ```
   from django.shortcuts import render
   from .models import Recipe
   def home(request):
       recipes = Recipe.objects.all()
       return render(request, 'home.html', {'recipes': recipes})
   ```

   This function fetches all recipes from the database and sends them to `home.html` to be displayed.

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 this scenario, I would use class-based views because they help organize and reuse code more efficiently.

Class-based views (CBVs) allow you to create reusable components by inheriting from built-in views and customizing them. For example, if you need to handle different types of views (like displaying a list of items, showing details of an item, etc.), you can use CBVs to keep your code DRY (Don't Repeat Yourself).

CBVs are useful because:

- **Reuse and Extensibility**: You can easily extend and reuse common functionality by inheriting from built-in views.
- **Organization**: CBVs help organize your code into methods, making it easier to understand and maintain.
- **Modularity**: They allow you to break down complex views into smaller, manageable pieces.

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

Django's template language is used to create dynamic HTML pages by inserting data into HTML templates. Here are some basics:

- **Variables**: You can insert variables into your HTML using `{{ variable_name }}`. For example, `{{ recipe.title }}` will display the title of a recipe.
- **Filters**: Filters are used to modify how variables are displayed. For example, `{{ recipe.title|lower }}` will display the title in lowercase.
- **Tags**: Tags are used to control the flow of the template. For example, `{% if recipe %}...{% endif %}` lets you include content conditionally.
- **Loops**: You can loop through lists with `{% for item in list %}...{% endfor %}`. For example, `{% for recipe in recipes %}{{ recipe.title }}{% endfor %}` will display a list of recipe titles.

Templates help separate the design from the logic of your web application, making it easier to update the look and feel without changing the underlying code.

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.
   Static files in Django are things like images, CSS, and JavaScript that don't change and are the same for every user. Django handles them by collecting them in one place when you run `collectstatic`, making it easier to serve them in production. You use the `static` directory in your app and the `{% static %}` template tag to link to them.

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	`ListView` is a Django view that displays a list of objects from a model. It automatically handles pagination and context for you, making it easier to show a bunch of records on one page.
DetailView	`DetailView` is used to display a single object from a model. It's great when you want to show detailed information about one specific item in your database, like a single blog post or product.

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.
   It's going pretty well! I'm proud of getting a better grasp on how Django's MVT structure works, especially connecting models to views and templates. Struggling a bit with more complex views and understanding when to use different class-based views, like `ListView` and `DetailView`. I think I need more practice with debugging errors in Django and understanding the flow of data through 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 important because it ensures that only authorized users can access certain parts of your application. For example, in a recipe app, you might want to let only logged-in users manage their own recipes. Without authentication, anyone could edit or delete recipes, which could lead to security issues or data loss.

2. In your own words, explain the steps you should take to create a login for your Django web application.
   1. Create a Login View: Use Django's built-in `LoginView` to handle login requests.
   2. Create a Login Template: Design a simple HTML form where users can enter their username and password.
   3. Update URLs: Add a URL pattern for the login page in your `urls.py`.
   4. Add Login Links: Place links or buttons on your site that lead to the login page.
   5. Redirect Users: Set up redirection so users are sent to a specific page after logging in.

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()	Checks a username and password against the user database and returns the user object if the credentials are valid.
redirect()	Sends the user to a different URL. It is often used after form submissions to send users to a new page.
include()	Includes another URLconf module. It helps organize URLs by including other URL patterns from different modules.

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.
   Analyzing the data collected by your favorite website or application, such as Instagram, Netflix, or Amazon, can significantly enhance the user experience and the platform's effectiveness. By analyzing user behavior, preferences, and demographics, the website can personalize recommendations, improve user navigation, and provide targeted advertising, which ultimately leads to a more engaging and satisfying experience for users. For instance, Netflix suggests shows based on your viewing history, and Amazon recommends products based on your past purchases, all made possible through data analysis.

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 a powerful tool for fetching data from the database, and it can be evaluated in several ways. You can iterate through it using a loop, slice it to retrieve specific records, or use methods like `.count()` and `.exists()` to get quick insights. Additionally, converting a QuerySet to a list forces the retrieval of all data. These methods allow developers to interact with and manipulate database records efficiently while maintaining performance.

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.
   When comparing QuerySet and DataFrame, each has its strengths and weaknesses. QuerySets are database-specific, optimized for performance, and use lazy evaluation, making them efficient for simple data retrieval and queries. However, they lack advanced data manipulation capabilities. On the other hand, DataFrames, particularly from Pandas, excel in complex data analysis, allowing for sophisticated calculations, transformations, and integrations with libraries like Matplotlib for visualization. The trade-off is that DataFrames consume more memory and may be slower for simple queries. Therefore, while QuerySets are ideal for straightforward database operations, DataFrames are better suited for intensive data processing tasks that require flexibility and power.

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, I can use CSS to style my web pages and make them look better, like changing colors, fonts, and layouts. I can include CSS files in my templates by linking them in the `<head>` section of my HTML files. JavaScript is used to add interactive features like buttons that change when clicked or forms that show errors before submitting. I can include JavaScript files in my templates just like CSS, usually at the bottom of my HTML files.

2. In your own words, explain the steps you'd need to take to deploy your Django web application.
   To deploy my Django app, first, I would set up a server, like using a platform such as Heroku or PythonAnywhere. Then, I would configure my Django settings for production, including setting `DEBUG = False` and adding the server's domain to `ALLOWED_HOSTS`. After that, I would install and configure a web server like Gunicorn or Nginx to serve my app. Finally, I would upload my code to the server, apply any necessary migrations to the database, and start the server to make my app live.

3. (Optional) Connect with a few Django web developers through LinkedIn or any other network. Ask them for their tips on creating a portfolio to showcase Python programming and Django skills. Think about which tips could help you improve your portfolio.

4. 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?
      I think I did well in understanding the basics of Django and how to create a web application from scratch. I was able to follow the instructions and complete the exercises successfully.
   b. What's something you're proud of?
      I'm proud that I built my first web application with Django and styled it using plain CSS to make me remember how good it feels . It feels great to see my project come to life on a web page!
   c. What was the most challenging aspect of this Achievement?
      The most challenging part was understanding how to deploy the Django project. There were a lot of new terms and steps, but I learned a lot by doing it.
   d. Did this Achievement meet your expectations? Did it give you the confidence to start working with your new Django skills?
      Yes, this Achievement met my expectations. I feel more confident now in using Django, and I'm excited to keep practicing and building more projects.

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