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?

Prior to *the entire CF course*, I had limited experience with HTML and CSS thanks to my current role – mostly in WordPress and Adobe Experience Manager. Before beginning *the Python speciality course*, I completed courses in frontend and full-stack development. These covered HTML, CSS, JavaScript, React, Angular, Bootstrap and more.

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

   I know that Python is a very popular language because it's relatively easy to learn. I know that it's used by data journalists to do things like analyze large datasets and scrape websites. I also know that it's a popular language in AI/ML applications. Just getting a working knowledge of Python and a basic comfortability with it will be fantastic and beneficial.

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.

   I think that I have my studying process down pretty well, so I don't foresee any issues with that. I think the main challenge will just be readjusting my thinking to a new language in general. I will have to regularly remind myself that I'm still learning and it's OK to not know everything. I was also very comfortable talking to my tutor from earlier sections of the course, so familiarizing myself with a new tutor will be a process as well.

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 deals with what a user sees and interacts with. It's the text and layout of the web page or app, forms, buttons, animations, etc. It's the visual and interactive elements. Backend development relates to everything that goes on in the background, from API and database development and management, to user authorization, dynamic site updates, processing user requests, communicating with servers and more.

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")*

   Python in similar to JavaScript in several ways, including its line-by-line execution and dynamic typing, which allows for variables to assume any kind of value during runtime. So the switch from JavaScript to Python would not be a difficult one to make. But Python brings with it several values that help it edge out JavaScript. These include built-in package management for things like math, password encryption and JSON formatting. It's also highly readable, which keeps code resistant to errors and easier to debug. Plus, it's strong community support means we'll have access to resources if we run into issues.

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?

   1. I want to understand the types of projects that Python is most suitable for and the types of roles looking for this coding knowledge.

   2. I want to be more comfortable navigating the Command Line Interface on my computer by gaining a better understanding of the Python commands.

   3. I want to really focus on reading the additional resources for each section, adjusting my notes and code as I learn more about Python.

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?

   The iPython shell is a user-friendly, practical shell for reading code. Syntax highlighting makes it much easier to see what is happening and code doesn't need to be manually indented, like in the built-in default python shell. iPython shells also allow you to test small bits of code easily and quickly – each command is executed immediately after typing it. This is a good way to test code before writing out and running entire script files.

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?
Integer (int)	Integers can be whole negative or positive numbers (-20, 23, 178, -182939, etc.) from zero to infinity. They do not include decimal points and can be used to represent things like counts and quantities.	Scalar
Boolean (bool)	Boolean data types store one of two values – True or False – and are useful for checking condition outputs.	Scalar
Dictionary (dict)	A dictionary stores values and objects within itself using key-value pairs. These are essentially a list of items, where each item has a name (the key) and a description (the value). Values have no immutable data type restrictions, they can even be another dictionary.	Non-Scalar
Tuple	Tuples are linear arrays that can store multiple values of any type (integers, strings, etc.). They're useful when you need to save several values, but naming a new variable for every one of them would be too much work. They are similar to lists but are immutable - you can't change, add, or remove items from them once they're created.	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.

Lists and tuples are similar in that they are essentially a sequence of items. They can include a variety of data types, like integers, floats, and strings.

The main difference is that tuples are immutable and you cannot add, change or remove items from them once they are created. In order to “add” a new item to a tuple, you’d have to place the new value in a single-element tuple, and then concatenate (i.e., merge) the two tuples together. Tuples are faster to read and access, especially when large amounts of data are involved.

Lists, on the other hand, are mutable - any of the internal elements of a list can be modified or deleted. You can also rearrange or even insert new elements in lists. They are useful in situations where reordering or modifications may be necessary.

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 a dictionary for the language learning app. This would be ideal because flashcards could be set up with key-value pairs that utilize the word as the key and its definition and category as the values. This structure would allow quick lookups and easy updates to add/delete new words to the dictionary and add new values (synonyms, sentence examples) down the line without altering the entire structure. Since tuples are immutable, I believe they’d be too restrictive for this app. Lists are mutable and could be useful for storing things like synonyms, but the larger set of data for a single word is complex enough that a dictionary would make more sense (with lists making up some of the values).

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 == 'New York' or destination == 'Los Angeles' or destination == 'Seattle':
   print('Enjoy your stay in', destination)
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 include things like **and**, **or** and **not**. They return a Boolean value of True or False. They are useful for checking multiple conditions at once and performing an action based on the results. Logical operators are used for controlling the flow of a program, particularly when making decisions with conditional statements like **if**, **while**, or **for** loops.

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

    Functions are sets of instructions that process or manipulate your code. There are built-in functions in Python like like print(), and len(), but you can also create your own functions. They are useful for repeating steps, allowing you to condense the code and save time. Functions also help you keep your code clean and concise.

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'm really happy with my Python progress so far. The command line interface has become much more familiar and I'm getting more comfortable using it to navigate and issue commands. I also feel like I'm gaining a good understanding of Python basics and troubleshooting.

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?

   Files are used to store Python data permanently. While just working with data in the python or iPython shell is useful for learning or testing small bits of code, the data is deleted as soon as the code is finished running. Python uses text and binary files to store variables, functions and data on your machine so that it can be accessed again later.

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 convert complex data into a packaged stream of bytes, known as a "pickle," then write this into a binary file. These files can easily be read by a machine, but are not human-readable. They are useful when the data structure is more complex than a simple list of string or numbers, which work well in a plain text file. Once the complex data structure (like a recipe dictionary) has been converted to a pickle using the dump() method, it can then be converted back to the original, human-readable version via the pickle.load() method.

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 complete either task, you first need to import the os module. Once that's done you can use the command os.getcwd() to determine your current directory. To change directories, you'd use the command os.chdir(). The syntax for the second command is os.chdir("<path to desired folder>").

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?

   If you're worried that a block of code could produce an error, a try-except block can help. You'd place the code you think might cause an error in the try block. Then, in the except block(s), you can write code for what to do if an error is encountered, like a file not found or invalid characters entered by the user. This could include printing an error message for the user or including a return statement to stop the function from going any further. You could then include an else block to include the code to run if the try block does not encounter an error. Lastly, you can use a finally block to write code that should run no matter the result of the try block.

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.

   I'm feeling pretty good so far in the Python Achievement 1. I'm really proud that I was able to write nearly all of the recipe_input.py and recipe_search.py scripts before looking at student submissions and other resources for help. I was also able to troubleshoot issues in the code relatively quickly. I think I need more practice using loops. For example, given the instructions, I want more practice determining how a loop should be structured to get the desired results.

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 is the practice of using classes to organize related objects and their functions in order to keep code clean, efficient, and non-redundant. A class stored object attributes that will apply to all objects in the class, like name and height for a "Human" class. The class also stores functions to extract and manipulate the object values. Defining a class allows you to easily create new objects with the necessary attributes and deploying them for future use.

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

   Classes are the larger group that houses all the objects. For example, a class could be named "Vegetables" and include data attributes like "name," "color," and "calories" that will apply to each object that's initiated. Individual objects would be things like "bell pepper," "zucchini" and "artichoke." When a Vegetable object is initiated, like "brussel sprout," the data attributes are passed as arguments when initiating and any of the class methods, like adding the object to a grocery list or calculating how healthy it is, are called as well.

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 the idea that one class can inherit properties from another class. A parent class (the one being inherited from) is used as an argument when creating a child class (or subclass). For example, class City(State):, in which City is the child class and it's inheriting properties, like data attributes and procedural attributes (methods) from the State class. Inheritance works in only one direction, from parent to child. If a child class has a method or data attribute with the same name as one from the parent class, it will take precedence for the object. For example, if both State and City classes have a data attribute for population, the City attribute will override the State one.
Polymorphism	Polymorphism is where a given data attribute or method has the same name across different classes or data types, but performs different operations depending on where it was defined. For example, if two classes, Car and Airplane, both have attributes and methods called PassengerCount, Speed and TravelTime, the similar names will not interfere with each other since they are defined in separate classes. Another example of polymorphism in Python is the different uses of the len() method, which performs different actions depending on which kind of data it's used on (strings, lists or dictionaries).
Operator Overloading	Special operators like +, -, >=, < and != would result in an error if used on a custom class because they aren't supported there. With operator overloading, you create a custom method within a class that defines what you want to happen when that operator is used. You can use built-in method names from Python, like __add__(), __lt__(), and __sub__() to define the steps that should be taken for that particular operator. Then when the operator is used later (+, < or -, for example) the method you defined is called.

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?

   A relational database stores blocks of information in columns and rows (each row is an individual item and each column is a data attribute for that item). In terms of advantages over local storage, databases allow you to store data in a standardized format and access it easily. They can be made secure through password access. Since they have a widely acceptable format, you can access them using applications other than Python as well.

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

Data type	Definition
INT (integer)	This is a standard integer. It could be used to represent something like a person's age, the quantity of an item, etc.
VARCHAR	This is a variable-length non-binary string. You can set a character limit by adding the max length in parentheses (like VARCHAR(25)). This type could be used for things like a person or item's name.
DATETIME	This is a date and time value in CCYY-MM-DD hh:mm:ssformat.

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

   SQLite is best for smaller projects, since with SQLite only one process can make changes to the database at any given time, it has no form of user management and less security than MySQL. SQLite can be a better choice than MySQL when testing – when it can be overkill to full DBMS that runs a server process (like MySQL). It can also be a good choice for apps that won't need future expansion.

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?

   I find Python to be more intuitive and simple, which explains why many people can learn it quickly and choose to adopt it for their apps. Python is very readable, making it easy to look at it and have an idea of what the code is doing. JavaScript's syntax is more complicated, making it more difficult for beginners to read and understand. I think both languages are versatile and could be used for a variety of projects, including web development and back-end programming. JavaScript can also be used for front-end applications.

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?

   As far as I can tell, Python is best suited for back-end operations, but maybe this outlook with change with future tasks. Since it's executed on a server or local machine and not typically run directly in web browsers, its execution time can be slower.

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, like SQLAlchemy, allows for easy transferring and converting of databases. It converts the contents and structure of your database into classes and objects that can be interacted with directly. You define the class as you normally would in Python, and then the ORM translates it into SQL queries for the table in your database. This removes the need to work with SQL queries, which can vary from one DBMS to another.

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?

   Overall, I think the app creation process went pretty well. I figured out a solid process for getting started with each step of the app (creating the table, defining functions, etc.). I'd get as far as I could based on my notes and the practice tasks before resorting to help from student submissions and CHATGPT. If I encountered errors in the terminal, I was quickly able to figure out what the issue was a fix it. I think I need more practice with loops, just figuring out how to structure them given the ask, but I definitely felt more comfortable with them by the end of the app creation.

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.

   During my Python specialization course with CareerFoundry, I built a recipe app from the ground up using Python. I started by writing simple scripts that would handle individual operations and testing them in an ipython shell. Results were stored locally. Eventually, I worked up to rebuilding the code using object-oriented programming methods and building/managing a database with the MySQL database management system and then using SQLAlchemy to make the management process simpler. The app allows presents users with a main menu of options and based on their input, they can create recipes, view all recipes, edit individual recipes, search recipes by ingredient or delete entire recipes from the database.

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? I think I've developed a good base knowledge of Python and am able to read it quite easily to figure out what's going on. The ideas behind SQL database creation and management are also pretty intuitive to me and I was able to quickly pick up the syntax for reading and updating database entries.
    b. What's something you're proud of? I am proud of my continued growing confidence in using the terminal – something that always intimidated me. I also think the final task for 1.7 went relatively smoothly and I encountered little to no errors with my code. Those that I did get, I was able to spot the issue and fix it relatively quickly.
    c. What was the most challenging aspect of this Achievement? Figuring out how to structure input validations and loops continues to be a struggle point for me but after putting together quite a few of them for the 1.7 task, I definitely feel more comfortable.
    d. Did this Achievement meet your expectations? Did it give you the confidence to start working with your new Python skills? Yes, definitely. I feel like it gave me a great start on Python and a solid base of knowledge to move forward with.
    e. What's something you want to keep in mind to help you do your best in Achievement 2? Thinking through the different requirements of each method and picturing each step that needs to be included is really helpful for me. Also, moving slowly and writing as much code as I can before turning to resources for help has been a great way to solidify my understanding.

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? I believe that it was effective. I took detailed notes and screenshots, worked through all the practice tasks and read the extra resources thoroughly.
- 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'm most proud of writing a user-friendly and intuitive app that is error-free and thoroughly commented. In Achievement 2, I'm excited to build the app on a more visually-appealing platform that moves the app outside the ipython shell.
- 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? It was a new way of thinking about code, which was intimidating at first. But it's actually quite simple to read and understand and I have to remind myself of that. Keeping all my notes and previous practice scripts handy will be a great help as I move forward and expand my Python knowledge base.

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?

   Vanilla Python can be a good option for smaller projects as it's lightweight and would allow you to keep total control over the project architecture. However, it can be more time consuming as you'd have to handle everything, like routing and database management, manually.

   Django would be the better option for a web application that might require multiple users or database connections. Django's built-in tools would allow you to focus on the business logic while it takes care of the heavy lifting. It would enable fast deployment and scalability. On the other hand, Django's rules can feel restrictive if your project has highly specific needs, and the framework can be overkill for simple applications.

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

   MVT architecture is easy to modify and keeps the application loosely coupled. It eliminates the need to write controller logic – for example to fetch information from the database – as it's taken care of by the framework. An MVT framework like Django provides you with the basics to get your application up and running so you can focus on other important things like writing the models and templates. MVT is more beginner-friendly and allows for faster development because it reduces complexity and handles many of the repetitive tasks.

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 learn how it can be used to reduce the amount of coding for a developer so that they can focus on the logic and create clean, readable code.
    - What do you want to get out of this Achievement? I'm excited to take my existing knowledge of Python and see how it and Django can be used to build a clean, visually-appealing app for users.
    - Where or what do you see yourself working on after you complete this Achievement? Aside from finishing the career prep portion of the CareerFoundy course, I'd like to take my knowledge from these two Achievements and trying my hand at some of the sample beginner Python projects listed here.

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'll use the AV Club website as an example. I'd start by breaking down the different pieces of the site into their relevant Django terms. The website itself is a project, while individual features and sections (News, TV, Film, Search, newsletter, etc.) would be considered apps that have their own functionalities. The site would include configuration files that handle project functions like displaying entire news pages and individual articles, newsletter signup, creating links between them all, etc. A database would likely house content for the articles on the site.

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

   - Create a virtual environment (mkvirtualenv) and install Django in the environment
   - Create a Django project from the desired working environment (like a "Python" folder, for example) with the django-admin.exe startproject command
   - Rename the root directory file name to something like src if need to be help avoid confusion
   - Run migrations to create the database (py manage.py migrate)
   - Run the server to determine the project has been set up successfully (py manage.py runserver)
   - Create a superuser so that you can set yourself up with admin access from the start (py manage.py createsuperuser) and test by repeating the runserver command and logging in via the local host URL + "/admin".

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

   The admin site reads metadata from your models to provide a quick, model-centric interface and is a great tool to use for internal management (by admins and not for building the front-end). The admin site can be used to create, view, update, and delete records (perform CRUD operations), which can save development time. So it can be used to create/update/read/delete recipe or bookstore entries to their relevant databases. It can be used to easily see if data is structured as you intended and customized to your needs. I'd use it as a simple way to manage the model instances of the project without the need to spend extra time developing a custom UI to be able to do so.

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.

   Models in Django define the data in a single database table and act as a single source of truth for the structure of that data, including essential fields and behaviors of the data being stored. Each attribute defined in the table acts as a column in the database table.

Once the model has been created, you don't need to talk to it directly at all — you just write your model structure and other code, and Django handles the work of communicating with the database for you. Models allow developers to store data in the database conveniently and use the Django admin panel to perform CRUD operations on the database. Django models provide simplicity, consistency, version control, and advanced metadata handling.

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 automated tests can help you catch errors and prevent bugs in your code. Being proactive about writing tests can save you time down the line - you won't have to manually check everything as you make changes and the tests will handle low-level details of the app's functionality. Tests also ensure the code is trustworthy (it's been tested) and prevent you or others from inadvertently introducing bugs to the code.

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.

   A view is the logic that Django runs when a user accesses a URL. It's a Python function or class that receives web requests and returns web responses. It can be basic and return a static request (like a welcome HTML page), or it can include more complex code that interacts with the database, accepts user requests, and dynamically displays information to the user based on their request and the function results. Based on the URL sent from the user in the browser, Django will load the view which then performs work like fetching data or processing a form. Once the view's work is done, it returns a response.

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?

If code will likely be reused, class-based views would be the most appropriate. They might be more difficult to initially write, but they reduce the effort spent duplicating or rewriting code. Class-based views have pre-configured functionality that you can reuse and extend if need be. Modules like logins, lists or forms that may show up several times across an application would not need to be recreated each time with CBVs, making them the ideal choice for this kind of project.

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

   - It's designed to feel comfortable to those used to working with HTML.
   - It's meant to express presentation, not program logic.
   - A template is a text file. It can generate any text-based format (HTML, XML, CSV, etc.).
   - A template contains **variables** (looks like {{ variable }}), which get replaced with values when the template is evaluated. Variable names cannot have spaces or punctuation characters. You can use dot notation to access variable attributes.
   - Filters (marked by a | pipe character) can modify the display of a variable. For example, { name|lower }} will lowercase the name value. Other built-in filters are available to set a default value, determine the length of the value, format dates, and more.
   - **Tags** (look like {% tag %}), which control the logic of the template. They can create text in the output, control flow by performing loops or logic, and load external information into the template to be used by later variables.
   - To comment-out part of a line in a template, use the comment syntax: {# *comment goes here* #}.
   - Template inheritance is a powerful feature that allows you to build a base "skeleton" template that contains all the common elements of your site and defines **blocks** that child templates can override. The feature maximizes code reuse and helps to add items to shared content areas, such as section-wide navigation.

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 those that do not change dynamically throughout the app and user's interactions with it. They can include things like images (developer-provided), CSS and JavaScript. With images, backend images tend to be stored at the project level, while frontend images are stored at the app level. So images that users will see as the background and ones on the homepage are stored in the recipes app in my project. Images uploaded with recipes and stored in the database (backend) are stored in a media folder in the root directory. Static files are accessed via templates with the {% load static %} command.

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 class-based view that allows you to access the records from the model. When the view executes, the model's records are stored as a list in the variable called object_list. To access individual records within the object_list, you use a for loop in the template like {% for object in object_list %}.
DetailView	A generic class-based view that returns the object that the view is operating upon. It will help you display data attributes for the object. In the case the recipe app, these data attributes include recipe name, cooking time, and ingredients.

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.

   I'm proud of the clean design and ease of use of the app so far. I'm excited to add more features like a navigation, log in and options for adding/updating/searching recipes that will make it more user friendly and engaging. I think the URL-View mapping has been the toughest part for me to grasp so far – just going back and forth between several files and making sure it's all set up correctly – and look forward to getting more experience with that.

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.

   Authentications allows you provide user-specific functions and content only to those that are logged into the application. You can also set different permissions for users based on their role, group, etc. to only allow certain users to perform certain actions. It also allows you to protect certain pages, perhaps with sensitive information, so that only valid users can view it. For example, if you were a registered user of a news site, like the New York Times, once logged in, you'd have access to more articles, bonus features and podcasts. If you are not logged in, certain pages would be off limits and you'd be directed to a login or registration page instead. Authentication provides the developer with more control over who can see and do what in the application.

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

   First, you'd create a views.py file under the root folder (recipe_project, for example). In the views.py file, you'd define the view, which incorporates authenticate, login, and AuthenticationForm packages from Django. The view would generate a POST request with the user-provided username and password and check them against the backend. If valid, the view would redirect the user to a page of the developer's choosing. If not valid, an error message would be displayed. The form and error message would be passed to the template. The template would be created in a templates/auth folder path that lives ideally under the src folder. You'd then update the DIRS list under TEMPLATES in the main settings.py file to tell Django where to look for templates ('DIRS': [BASE_DIR / 'templates']). You could skip normal URL mapping since the view was set up outside an app. You'd then move directly to registering the URL in the main urls.py file (don't forget to import the view!).

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()	Verify a set of credentials. It takes credentials as keyword arguments, username and password for

	the default case, and checks them against the backend, and returns a User object if the credentials are valid for a backend. Otherwise, it returns None. Example: user = authenticate(username=username, password=password)
redirect()	Returns an HttpResponseRedirect to the appropriate URL for the arguments passed. Arguments can be a model, view name or an absolute or relative URL. Returns a temporary redirect by default but will accept a permanent argument as well. Example: def logout_view(request): logout(request) return redirect('login')
include()	A function that takes a full Python import path to another URLconf module that should be "included" in this place. Optionally, the application namespace and instance namespace where the entries will be included into can also be specified. Example: urlpatterns = [path(' ', include('recipes.urls')), *# Include recipe app URLs as the root URLconf*]

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.

   The New York Times website collects data on subscribers (names, billing info) and subscriptions (start date, payment type), which allows the site to provide functionality for users to manage their accounts as well as access paid content. NYT also gathers data on what articles users are reading, how long they stay on pages, the links they click and what they search for on the site. This kind of data helps the site understand how readers are engaging with content so that it can better optimize articles and improve search functionalities. Cookies and tracking pixels can help the NYT site provide more targeted content and advertisements to users. All of these data types can help improve content personalization, user experience, content development and advertising revenue.

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

   - **Iteration:** Loop through QuerySet to access each object individually and perform an action on it
   - **Slicing:** Slice a QuertySet to return another unevaluated QuerySet
   - **Pickling/caching:** Force all the results to be loaded into memory prior to pickling, usually as a precursor to caching
   - **repr():** Evaluate a QuerySet and immediately see your results when using the API interactively
   - **len():** Returns the length of the result list
   - **list():** Force a QuerySet's evaluation
   - **bool():** Test QuerySet in a boolean context. If there is at least one result, it'll be True. Otherwise, it will be False.

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.

   QuerySets make it easy to retrieve, filter, and manipulate data without writing raw SQL. They also integrate seamlessly with Django views, forms, and templates, which makes them ideal for use within Django projects where models are tied to relational database tables. They aren't executed until explicity used and handle queries efficiently, which leads to performance optimizations. However, they aren't ideal for complex data analysis or transformation or for working with external data sources.

   Pandas DataFrames are incredibly flexible and powerful for data manipulation, allowing for slicing, filtering, and transforming data in various ways. They integrate well with Python and can be used for advanced data analysis, machine learning, and visualization. They allow in-memory data processing and can be exported in various formats such as CSV, Excel, or JSON, making it

easy to share or store processed data. However, they can be resource intensive and can cause performance bottlenecks for huge datasets.

DataFrame is ideal when you need to perform complex data analysis, data transformation, or visualizations, especially on larger datasets or datasets that come from multiple sources.

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.

   CSS and JavaScript can be used in a Django web application to create a professional-looking app with interactive elements that a user can engage with. This can include consistently-designed pages with similar fonts/backgrounds/colors/buttons, a navigation menu that runs across the site's pages, search bars and embedded videos.

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

   The first step is preparing a GitHub repository that can store changes to the code and that a web server like Heroku will use later to pull the code from. Generally, you should create a backup file in case anything goes wrong in later steps.

   Next, the Django application itself needs to be updated for hosting. This includes creating a Procfile in the root directory of the application, installing an HTTP server like Gunicorn and configuring the database so that it converts to Django's desired configuration format. You'll also make sure static files can be hosted, remove any sensitive info like a SECRET_KEY and set DEBUG to False to prevent security threats. Lastly in this stage, you'll create a requirements.txt file and push all the changes to the GitHub repo.

   Finally you'll create a Heroku (or other web server) application that syncs to the GitHub repo. This step will include adding data to the database, updating the SECRET_KEY and testing the webpage.

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 feel like the majority of the Python course went really well. I was able to quickly pick up on the concepts and put together a user-friendly, easy-to-navigate app. I also feel like the forms were a much smoother process than what I'd experienced in the past, which was confidence-boosting.
    b. What's something you're proud of? I'm proud of myself for getting through the majority of this Achievement without needing much help from my mentor. I was able to work through the test bookstore app and apply those steps to the Recipe application relatively easily. When I got stuck, I was able to reference student submissions, the error log in the terminal and ChatGPT to quickly resolve issues.
    c. What was the most challenging aspect of this Achievement? The visualization and deployment were the most challenging parts for me. Figuring out how to implement different attributes into the charts and make sure that the right data was being passed from the view to the template required quite a bit of trouble shooting. I also ran into unforeseen issues with the deployment that took time to sort out.
    d. Did this Achievement meet your expectations? Did it give you the confidence to start working with your new Django skills? Yes, I really enjoyed this Achievement. I feel like for the most part, the reading and direction were clear and helpful, allowing me to feel confident in setting up the majority of the application. I'm really proud of how well the app works and how it looks, so I'm excited to keep using these new skills.

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