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)

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

Remember, you can always refer to [Exercise 1.4](https://careerfoundry.com/en/steps/your-cf-team#receiving-support) 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

* 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: This is what users see and interact with on a website, like buttons and images.
* Backend: This is the behind-the-scenes part where data is processed and managed. If hired as a backend developer, I'd work on things like databases, server logic, and ensuring that data flows correctly to the frontend.
* 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”)*

* Similarities: Both are programming languages used to build software.
* Differences: JavaScript is mainly for web pages, while Python is more versatile, good for web, data analysis, and automation.
* Why Python?: I'd suggest Python for its simplicity and great libraries, especially if our project involves data-heavy tasks or needs quick development.
* 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?
* Learn Python Libraries: I want to get comfortable using Python’s tools for web and data tasks.
* Complete a Project: I aim to build a complete project with Python to apply what I've learned.
* Advance My Career: After learning Python, I plan to tackle bigger projects, possibly combining web development with data analysis.

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

* 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?
* Reasons: The iPython Shell has more features like syntax highlighting, auto-completion, and history, which make it easier and faster to write and test code. It also supports inline plotting and can display rich media like images and videos.
* 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 | Whole numbers, like 1, 42, and -10 | Scalar |
| float | Decimal numbers, like 3.14 or 2.0 | Scalar |
| str | Text or strings of characters, like "hello" | Non-Scalar |
| list | Ordered collection of items, like [1, "apple", 3.14] | Non-Scalar |

* 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: They are mutable, meaning you can change, add, or remove elements after the list is created.
* Tuples: They are immutable, meaning once a tuple is created, its contents cannot be changed, added to, or removed.
* 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.
* Choice: Dictionaries.
* Reason: Dictionaries would allow each flashcard to be a separate entry with keys for the vocabulary word, definition, and category. This makes it easy to add more information, like usage examples, without affecting other entries. They also provide fast look-up and are easy to manipulate and extend, which is beneficial for further development like tracking user progress or adding new features.

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

* 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 would you like to travel? ")  available\_destinations = ["Paris", "Tokyo", "New York"]  if destination in available\_destinations:  print(f"Enjoy your stay in {destination}!")  else:  print("Oops, that destination is not currently available.") |

* 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, help us combine true or false conditions to decide what our program does next.
* What are functions in Python? When and why are they useful?
* Functions are like mini-programs inside your program. They do specific jobs, which helps make your main program easier to handle and stops you from writing the same code over and over.
* 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.
* **Learning Libraries**: I've learned how to use some important Python libraries for my projects.
* **Project Implementation**: I started a project using what I've learned. It's going well, and I'm solving problems as they come.
* **Career Progress**: I'm using new skills in my projects and sharing them with others online. It's helping me build a stronger profile for my career.

Exercise 1.4: File Handling in Python

Learning Goals

* Use files to store and retrieve data in Python

Reflection Questions

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

Exercise 1.5: Object-Oriented Programming in Python

Learning Goals

* Apply object-oriented programming concepts to your Recipe app

Reflection Questions

* In your own words, what is object-oriented programming? What are the benefits of OOP?
* Object-Oriented Programming (OOP) is a way of programming that organizes software design around data, or objects, rather than functions and logic. An object can be defined as a data field that has unique attributes and behavior. The benefits of OOP include:
* **Modularity**: The source code for an object can be written and maintained independently of the source code for other objects.
* **Reusability**: Objects can be reused in different programs.
* **Pluggability and debugging ease**: If a particular object turns out to be problematic, you can simply remove it from your application and plug in a different object as its replacement.
* 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 like a blueprint for creating objects. An object is an instance of a class, with its own set of data and behaviors that are defined by the class.  
  Real-world example: Consider a class Car. The Car class might include general attributes all cars share, like wheels, color, and engine. An object would be a specific car, say a red Toyota. This specific car, or object, has its own specific color (red) and make (Toyota) but follows the general blueprint defined by the class Car.
* In your own words, write brief explanations of the following OOP concepts; 100 to 200 words per method is fine.

|  |  |
| --- | --- |
| **Method** | **Description** |
| Inheritance | This is when a class uses code constructed within another class. If you have a parent class and a child class, the child class can inherit attributes and behaviors (methods) from the parent class. This is useful for code reusability. |
| Polymorphism | This means many forms. It refers to the ability of different objects to respond, each in its own way, to identical messages (methods). This lets you use a shared interface for different underlying forms (data types). |
| Operator Overloading | This refers to the ability to define the same operation (like + or \*) differently for different data types. For example, the + operator can be used to add numbers, concatenate strings, and merge lists. It allows the same operator to have different meanings based on the context. |

Exercise 1.6: Connecting to Databases in Python

Learning Goals

* Create a MySQL database for your Recipe app

Reflection Questions

* What are databases and what are the advantages of using them?
* Databases are structured systems for storing, managing, and retrieving data. They provide efficient access to large amounts of information, ensure data consistency through transactions, offer security features to control access, and allow for easy data manipulation and querying.
* List 3 data types that can be used in MySQL and describe them briefly:

|  |  |
| --- | --- |
| **Data type** | **Definition** |
| VARCHAR | A variable-length string data type used to store text. You specify the maximum length when you create the table |
| INT | An integer data type used to store numerical values without decimals. It can be signed or unsigned |
| DATETIME | Used to store date and time information, including year, month, day, hour, minute, and second |

* In what situations would SQLite be a better choice than MySQL?
* SQLite is better suited for applications that require a lightweight database with no setup, minimal configuration, and a small footprint. It’s ideal for small to medium-sized applications, embedded systems, and situations where simplicity and portability are more critical than advanced features and scalability, like desktop or mobile applications that need to work offline.
* 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 is primarily used for web development to add interactivity to web pages, often running directly in the browser (although it can also run on servers with Node.js). Python is a more general-purpose language known for its clear syntax and readability, widely used in data analysis, machine learning, automation, and web applications. Python is considered easier to learn due to its clean and straightforward syntax.
* 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?
* Speed: Python can be slower than compiled languages like C or Java because it is interpreted.
* Memory Consumption: Python's flexibility and ease of use come with a cost of higher memory consumption compared to more tightly controlled languages.
* Concurrency: Python’s Global Interpreter Lock (GIL) can be a hurdle in CPU-bound and multi-threaded code, making it less efficient for tasks requiring heavy parallel processing without using workarounds like multi-processing or other implementations like Jython or IronPython.

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

* What is an Object Relational Mapper and what are the advantages of using one?
* An Object Relational Mapper (ORM) is a tool that lets you manage database data as Python objects instead of using SQL queries directly. Advantages include:
* **Simplification**: It simplifies database interactions by allowing you to write Python code instead of SQL.
* **Productivity**: Boosts productivity by automating database schema creation and operations.
* **Maintainability**: Makes code easier to understand and maintain.
* **Security**: Helps prevent SQL injection attacks by using parameterized queries.
* 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?
* **How did it go?** The development of the Recipe app went well. I successfully implemented features allowing users to create, view, edit, and delete recipes.
* **What did well?** I did well in structuring the application using functions and managing database interactions efficiently using SQLAlchemy.
* **Improvements for next time?** If starting over, I would add more user input validations to ensure robustness and consider implementing additional features like sorting or categorizing recipes. Maybe using dome engine
* 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. This app utilizes SQLAlchemy, an ORM library, which significantly simplified the database operations. I implemented functionalities for users to add, view, edit, and delete recipes, which taught me a lot about Python’s capabilities in web development and database management. This project not only sharpened my coding skills but also improved my understanding of how to structure an application efficiently.

* You’ve finished Achievement 1! Before moving on to Achievement 2, take a moment to reflect on your learning in the course so far:
* What went well during this Achievement?
* Learning to use Python for practical applications like the Recipe app went really well. I became more comfortable with Python’s syntax and third-party libraries like SQLAlchemy.
* What’s something you’re proud of?
* I’m proud of building a fully functional application from scratch that interacts with a real database.
* What was the most challenging aspect of this Achievement?
* The most challenging part was learning how to effectively use the ORM to handle database operations, as it was a new concept for me.
* Did this Achievement meet your expectations? Did it give you the confidence to start working with your new Python skills?
* This Achievement met my expectations by providing a solid foundation in Python. It has definitely boosted my confidence in building and managing Python applications.
* What’s something you want to keep in mind to help you do your best in Achievement 2?
* Moving forward, I want to keep focusing on improving my understanding of more advanced Python functionalities and start exploring web frameworks like Flask or Django to build more interactive applications.

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

Pre-Work: Before You Start Achievement 2

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

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

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

Remember that can always refer to [Exercise 1.4](https://careerfoundry.com/en/steps/your-cf-team#receiving-support) 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

* 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?
* Python
* **Advantages**: Full control over the architecture and setup, ideal for small or highly customized projects. No additional overhead from unused components
* **Drawbacks**: More code to write for common backend functions like routing, form handling, and database operations, which can slow down development significantly
* Django
* **Advantages**: Speeds up development by providing ready-made components for common tasks (authentication, administration, sessions). Great for applications that fit its structure well
* **Drawbacks**: Can be too bulky for very small projects. Sometimes less flexible, as it encourages certain ways of doing things
* 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, as used in Django, over the traditional MVC architecture is that it explicitly separates the "logic" layer (the Model and the View) from the "presentation" layer (the Template). This separation makes the MVT architecture simpler and more intuitive for managing the user interface and the data flow, especially when the focus is on how the data is displayed rather than on the interactions within the data flow
* 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 Django's MVT architecture works in comparison to other frameworks and how to effectively use it for building robust web applications
* What do you want to get out of this Achievement?
* By the end of this Achievement, I aim to build a complete web application using Django, which includes user authentication, data management, and responsive templates to understand how Django manages all these aspects
* Where or what do you see yourself working on after you complete this Achievement?
* I see myself using Django for larger, more complex projects in the future. I want to be comfortable enough with the framework to decide when it’s the right tool for future projects and to confidently use it for a potential job

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

* 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.*)

* **Identify Components**: First, I would break down the website into functional components, such as user authentication, product listings, content pages, etc
* **Django Projects and Apps**: Each major function would be a Django app within the project. For instance, user management could be one app, blog management another, and product management a third
* **Models**: Each piece of data the website needs to store, like user profiles or product details, would correspond to a Django model in the appropriate app
* **Views and Templates**: The interactive parts of the website, like forms or dynamic pages, would be handled by Django views, with HTML templates displaying the data
* In your own words, describe the steps you would take to deploy a basic Django application locally on your system.
* **Set up a Python environment**: Create a virtual environment and activate it to manage dependencies
* **Install Django**: Install Django within this environment using pip install django
* **Create a Django project**: Use django-admin startproject myproject to create a new Django project
* **Develop your application**: Add apps to your project with python manage.py startapp myapp and develop your models, views, and templates
* **Configure settings**: Adjust settings in settings.py, such as DATABASES for database setup
* **Run migrations**: Apply database migrations with python manage.py migrate to set up your database schema
* **Run the server**: Start the local development server with python manage.py runserver and access the app via a web browser at <http://127.0.0.1:8000/>
* Do some research about the Django admin site and write down how you’d use it during your web application development.
* **Register models**: First, I would register all my application models with the admin site to make them accessible via a graphical interface
* **Customize interface**: I'd customize the admin interface to display relevant information about each model and to make it easy to navigate and manage the data
* **Manage data**: Use the admin site to add, delete, and edit records. It's very helpful for tasks like managing user accounts, updating content, or checking orders
* **Troubleshoot and test**: Early in development, I'd use the admin to quickly test model changes and data relationships before creating more complex front-end interfaces

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

* 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 your database in Python code. They work as a blueprint for your data, specifying what kind of data you want to store (like text, numbers, dates) and any rules about that data (like whether it's required or unique). Each model corresponds to a single database table, and each attribute of the model represents a field in that table. The benefits of Django models include:
* Abstraction: You don't need to write raw SQL; Django handles the database operations for you.
* Ease of Use: Models make it easy to create, retrieve, update, and delete records in your database without needing to directly interact with the database.
* Maintainability: Changes to the database schema are handled through migrations, making it easy to evolve your database alongside your application.
* In your own words, explain why it is crucial to write test cases from the beginning of a project. You can take an example project to explain your answer.
* Writing test cases from the beginning of a project is crucial because it helps ensure that your application works as expected and continues to do so as you make changes and add new features. For example, if you're building a Django web application like a blog, writing tests for creating, updating, and displaying blog posts can help catch errors early in the development process. This approach provides several advantages:
* Catch Bugs Early: Early testing helps identify and fix bugs before they become embedded in the codebase, reducing the cost and effort needed to resolve them later.
* Refactor Confidently: Tests serve as a safety net that allows developers to refactor or improve the code with confidence, knowing that they'll be alerted if something breaks.
* Improve Code Quality: Regular testing encourages better code structure and design, making the code more robust and easier to maintain.

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

* Do some research on Django views. In your own words, use an example to explain how Django views work.
* Django views are Python functions or classes that take a web request and return a web response. Views access data through models and delegate formatting to templates. For example, a view can query a database for articles and pass them to a template that formats them for display. Essentially, views handle the logic of your application: they receive input from users, process that input using business logic and models, and return the resulting output to users.
* 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?
* For a project where you anticipate reusing a lot of code, class-based views (CBVs) are generally more suitable than function-based views. CBVs are more modular and reusable because they encapsulate behavior into classes. You can extend base views and override methods to add custom behavior, making them ideal for creating a maintainable codebase where similar patterns repeat across different parts of the application. CBVs also make it easier to maintain and extend complex views by breaking down the functionality into mixins that can be mixed and matched across different views.
* Read Django’s documentation on the [Django template language](https://docs.djangoproject.com/en/3.2/ref/templates/language/#templates) and make some notes on its basics.
* Django’s template language allows you to build dynamic HTML easily. Here are some basics:
* **Variables**: You can use curly braces {{ variable\_name }} to insert the contents of a variable into your HTML
* **Tags**: For logic and control flow, you use tags that are enclosed in {% tag %}. Common tags include for and if for looping and conditional rendering, respectively
* **Filters**: Filters modify variables and can be applied in the template itself. They look like this: {{ variable|filter:"argument" }}. For example, {{ text|lower }} would convert a text string to lowercase
* **Inheritance**: Templates can extend a base template, allowing you to define a common structure and fill in blocks with specific content. This is done with {% extends "base.html" %} and {% block content %}

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

* In your own words, explain Django static files and how Django handles them.
* Django handles static files like CSS, JavaScript, and images, which are necessary for web pages but don't change often. These files are stored in one place in your project and aren't modified by Django. During development, Django can serve these files for you, but in production, you typically configure your web server to handle them, making them faster to load and reducing the load on your Django server
* 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 | The ListView is a Django class-based view that simplifies the process of displaying a list of objects from the database. It handles fetching the data and sending it to a template, reducing the amount of code you need to write. Essentially, it’s great for pages where you need to show a list of items, such as a list of recipes or blog posts |
| DetailView | The DetailView is another class-based view in Django designed to display detailed information about a specific object. When you need to show more detailed information about a single item, like a single recipe or a blog post, the DetailView automatically handles fetching the object based on its ID or slug and passing it to the template |

* 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 well! I’m learning a lot about Django’s capabilities, especially how to efficiently create web pages using class-based views. I’m proud of how I’ve managed to use Django’s built-in views to simplify much of the coding required for displaying lists and detailed pages of items. It’s satisfying to see how quickly functional pages can be set up with these tools.I need more practice with Django’s form system, particularly with complex validation scenarios and handling various types of user input safely and efficiently. I want to also explore more about deploying Django applications to production environments.

**Frontend inspirations**

1. AllRecipes (<https://www.allrecipes.com/>)

AllRecipes is known for its huge collection of user-contributed recipes. It features:

* Community Contributions: Users submit and review recipes.
* Robust Search: Filters help users find recipes by ingredients, meal type, and more.
* Clean Layout: The website has clear, large photos and well-organized recipe information.

2. Food Network Recipes (<https://www.foodnetwork.com/recipes>)

Food Network Recipes connects with the Food Network TV channel, offering:

* Celebrity Chefs: Recipes from famous chefs and TV shows.
* Rich Media: Includes lots of videos, like cooking demonstrations and tips.
* Vibrant Design: Colorful and engaging, with easy navigation highlighting popular categories.

3. Simply Recipes (<https://www.simplyrecipes.com/>)

Simply Recipes focuses on home cooking with a simple and friendly approach:

* Ease of Use: The website is straightforward and user-friendly, reflecting the simplicity of the recipes.
* Home Cooking: It features practical recipes for daily meals with a personal touch.
* Meal Planning: Provides helpful sections like meal plans and cooking tips.

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

* 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 essential for ensuring that users are who they say they are. This is crucial for any application that needs to secure user data or provide personalized experiences. For example, in a banking app, authentication protects accounts and financial information, allowing only the account owners to access and manage their funds.
* In your own words, explain the steps you should take to create a login for your Django web application.
* Import Authentication Views: Use Django's built-in views for authentication, such as LoginView.
* Create a Login Template: Design an HTML form where users can enter their username and password.
* Configure URLs: Add a URL pattern in your project’s urls.py to link to the login view.
* Link Template with View: Ensure the login template is connected with Django’s authentication system by using the {% url %} template tag.
* Test the Login: Run your server and check the login functionality by entering credentials.
* Look up the following three Django functions on Django’s official documentation and/or other trusted sources and write a brief description of each.

|  |  |
| --- | --- |
| **Function** | **Description** |
| authenticate() | This function checks a user's credentials (like username and password) and returns a User object if the credentials are correct. If the credentials are not valid, it returns None. It's used in the login process to verify if a user's login information matches the records in the database |
| redirect() | This function is used to redirect a user to a different URL. For example, after successfully logging in, you might use redirect() to take the user to their profile page or the homepage. It simplifies the process of sending HTTP responses to the client that instruct the client to request a different page |
| include() | This function is used in URL configurations to include other URL configurations from different apps within a project. It helps to keep URL configurations clean and organized by separating them into different files, which are then included in the project’s main urls.py file. This is particularly useful for larger projects with multiple components, allowing each app to manage its own URLs |

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

* 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.
* Websites and apps, like CareerFoundry, collect various types of data such as user interactions, course progress, forum posts, and feedback. Analyzing this data can help improve user experience by identifying popular courses or areas needing more resources. It can also tailor learning paths based on user performance and preferences, and enhance marketing strategies by understanding user demographics and behaviors.
* Read the <https://docs.djangoproject.com/en/3.2/ref/models/querysets/>. Note down the different ways in which you can evaluate a QuerySet.
* Retrieving a single object with methods like .get() and .first().
* List conversion using list(), which evaluates the entire QuerySet.
* Looping through a QuerySet, which evaluates items as needed (lazy loading).
* Using slicing (QuerySet[:5]), which retrieves a specified number of items.
* Checking the existence of items in a QuerySet with .exists().
* Using aggregations like .count(), .aggregate(), and .annotate() to compute values directly in the database.
* 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.
* QuerySet:
* Advantages: Tightly integrated with Django’s ORM; efficient for database operations like filtering, updating, and deleting directly in the database; lazy loading which optimizes memory usage.
* Disadvantages: Limited to database functions; complex computations or transformations require pulling large datasets into memory, which can be inefficient.
* DataFrame (Pandas):
* Advantages: Powerful for complex data manipulations and calculations; supports a wide range of operations like pivoting, merging, and sophisticated statistical analysis that are not possible with QuerySets. Better for handling large datasets and integrating data from multiple sources.
* Disadvantages: Requires loading data into memory, which can be a constraint with very large datasets. Less efficient for simple database queries and updates compared to using QuerySets directly.

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

* Explain how you can use CSS and JavaScript in your Django web application.
* CSS
* Include CSS files in the static directory.
* Link CSS files in your HTML templates using the {% static %} tag.
* JS
* Place JavaScript files in the static directory.
* Include JavaScript files in your HTML templates using the {% static %} tag.
* Use Django templates to pass data to JavaScript if needed.
* In your own words, explain the steps you’d need to take to deploy your Django web application.
* Prepare Your Project:
* Set DEBUG to False.
* Configure allowed hosts.
* Collect static files using collectstatic.
* Set Up Heroku:
* Install the Heroku CLI.
* Log in to Heroku using the CLI.
* Create a Heroku App:
* Run heroku create to create a new Heroku app.
* Configure Your Project for Heroku:
* Add a Procfile to specify the commands that run your app.
* Add gunicorn to your requirements.txt to handle WSGI requests.
* Set up whitenoise for serving static files.
* Set Up the Database:
* Add the dj-database-url package to handle the Heroku database.
* Configure the DATABASES setting in your settings.py to use the Heroku database URL.
* Push Your Code to Heroku:
* Initialize a Git repository if you haven't already.
* Add and commit your changes.
* Push your code to Heroku with git push heroku main.
* Run Migrations and Collect Static Files:
* Run heroku run python manage.py migrate to apply database migrations.
* Run heroku run python manage.py collectstatic to collect static files.
* (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.
* You’ve now finished Achievement 2 and, with it, the whole course! Take a moment to reflect on your learning:
* What went well during this Achievement?
* Understanding Django's structure.
* Building and testing the application.
* What’s something you’re proud of?
* Successfully creating a functional web app.
* Learning to deploy a Django project.
* What was the most challenging aspect of this Achievement?
* Debugging and fixing errors.
* Configuring deployment settings.
* Did this Achievement meet your expectations? Did it give you the confidence to start working with your new Django skills?
* Yes, it met my expectations.
* Yes, I feel confident in my Django skills now.

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