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?

Other than the CF Full Stack Dev course, I have worked on a variety of projects that involved some degree of web dev frequently over the past 12 years. Mostly I have functioned in the manager role with only an as-needed understanding of the high-level concepts involved. The most involved programming work I have done solo was in C# via Unity to develop an Augmented Reality capable companion application for a textbook.

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

I know very little about Python. I would like to know more of its capabilities for general awareness and as a potential solution resource in the future.

1. 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 do not expect any major challenges other than time availability. This is because my 3rd child was just born very recently, and the early new-born phase can really strain the work schedule.

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

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 web development focuses on the part of a website or web application that users interact with directly, like the layout, design, and user interface elements. Backend web development, on the other hand, deals with the server-side logic, databases, and application functionality that support the frontend.

If hired to work on backend programming for a web application, you would be responsible for operations such as managing databases, ensuring data is securely stored and retrieved, handling server-side logic, implementing APIs, and ensuring efficient server performance and scalability.

1. 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 interpreted languages, meaning they don't need to be compiled before running.
* They support multiple programming paradigms, including object-oriented, procedural, and functional programming.
* Both have large, active communities, providing extensive libraries, frameworks, and tools.

Differences:

* JavaScript is primarily used for frontend development to create interactive web pages, though it can also be used for backend development with Node.js.
* Python is a general-purpose language often used for backend development, data analysis, machine learning, and automation due to its simplicity and readability.
* JavaScript runs natively in web browsers, making it essential for web development, while Python is known for its ease of use and readability, making it popular for rapid development and prototyping.

I would suggest Python for our project because it is easy to learn and understand, making it quicker for new developers to get up to speed. Python also has simple built-in package management with tools like pip, which streamlines the process of installing and managing dependencies. It comes with out-of-the-box essentials, reducing the need for additional configuration. Python's efficient development flow allows for rapid development and iteration, and it has strong community support, providing plenty of resources and libraries to help us solve problems effectively.

1. 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?

Increased breadth of knowledge with regard to development

Enhanced Data Analysis Skills

Explore Automation and Scripting

### 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?

IPython is a more powerful and user-friendly environment for interactive coding and experimentation. It provides features like syntax highlighting, auto-completion, and magic commands, which make it easier and faster to write and test code interactively. There are better tools for debugging, allowing for more efficient problem-solving. Also, It supports displaying plots, images, and other rich media inline, which is useful for data analysis and visualization.

1. 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) | Represents whole numbers without a fractional part (e.g., 5, -3). | Scalar |
| Float (float) | Represents numbers with a fractional part (e.g., 3.14, -0.001). | Scalar |
| String (str) | Represents sequences of characters (e.g., "hello", "123"). | Non-scalar |
| List (list) | Represents ordered collections of items, which can be of different types (e.g., [1, "apple", 3.14]). | Non-scalar |

1. 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.

Tuples are used for fixed collections of items and can be used as keys in dictionaries due to their immutability.  
Lists, due to their mutability, are typically used when you need a collection that may change over time.

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

For a language-learning app with flashcards, a dictionary would be the most suitable data structure. Here's why:

* Flexibility: Dictionaries allow you to store key-value pairs, making it easy to associate vocabulary words (keys) with their definitions and categories (values).
* Ease of Access: You can quickly look up definitions and categories using the vocabulary word as the key.
* Expandability: As you develop the app further, dictionaries can be easily expanded to include additional information (e.g., example sentences, pronunciation).

### 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!)*

|  |
| --- |
| # Travel app script  # Ask the user where they want to travel  destination = input("Where would you like to travel? ")  # Define the travel destinations  destination\_1 = "England"  destination\_2 = "Germany"  destination\_3 = "Italy"  # Check the user's input and respond accordingly  if destination == destination\_1:  print(f"Enjoy your stay in {destination\_1}!")  elif destination == destination\_2:  print(f"Enjoy your stay in {destination\_2}!")  elif destination == destination\_3:  print(f"Enjoy your stay in {destination\_3}!")  else:  print("Oops, that destination is not currently available.") |

1. Imagine you’re at a job interview for a Python developer role. The interviewer says “Explain logical operators in Python”. Draft how you would respond.  
     
   Logical operators in Python are used to combine conditional statements. There are three main logical operators:

* and: Returns True if both statements are true.
* or: Returns True if at least one statement is true.
* not: Reverses the result, returns False if the statement is true.

For example:

* True and False returns False.
* True or False returns True.
* not True returns False.

They help control the flow of the program by allowing more complex conditions.

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

Functions in Python are blocks of reusable code that perform a specific task. They are defined using the def keyword, followed by the function name and parentheses.  
Functions are useful because they:

* Promote Code Reusability: You can call the same function multiple times without rewriting the code.
* Improve Readability: Breaking code into functions makes it easier to read and understand.
* Simplify Maintenance: Bugs are easier to find and fix in smaller, well-defined functions.

You use functions when you have repetitive tasks or when you want to organize your code into logical sections.

1. 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.

* Increased breadth of knowledge with regard to development
  + Learning the fundamentals helps me to gain a basic understanding of Python’s uses compared to other language approaches
* Enhanced Data Analysis Skills
  + Begin to understand the capability of Python in this area due to its ease of data exploration and manipulation
* Explore Automation and Scripting
  + Do not see this yet at such an early stage

### 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?

Data is lost once the script is finished. Need storage for permanence.

1. 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 in Python are serialized representations of objects, allowing you to save complex data structures to a file and load them later using the pickle module. You would use pickles when you need to save and load complex data structures like lists, dictionaries, or custom objects.

1. 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?

os.getcwd() command is used to find out which file directory you’re currently working on  
os.chdir() command lets your Python interpreter change its current working directory

1. 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?

Utilize a try-except-else-finally block.

1. 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.

Going more smoothly than expected. Proud to see the previous course coding experience paying off in facilitating learning this new language more quickly. Do not perceive any particular struggles beyond normal hurdles of gaining familiarity and implementing new information.

### 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?

OOP is a programming style that uses objects to represent data and methods. Objects are instances of classes, which can be thought of as blueprints for creating objects.

The benefits of OOP include:

* Modularity: Code is organized into classes and objects, making it easier to manage and understand.
* Reusability: Classes can be reused across different programs, reducing redundancy.
* Scalability: OOP makes it easier to handle larger and more complex programs by breaking them down into manageable pieces.
* Maintainability: OOP helps keep code organized, making it easier to update and maintain.

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

In Python, classes are blueprints for creating objects. They define the properties (attributes) and behaviors (methods) that the objects created from the class will have. An object is an instance of a class, representing a specific entity with the defined properties and behaviors.

Example:

* ‘Tool’ is a class that defines the attributes ‘name’, ‘tool\_type’, and ‘purpose’, and a method ‘use\_tool’ to describe the action of using the tool.
* An object would be a specific tool created from this class.

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

|  |  |
| --- | --- |
| **Method** | **Description** |
| Inheritance | allows a class (called a child or subclass) to inherit attributes and methods from another class (called a parent or superclass). This helps in reusing code and creating a hierarchy of classes. For example, a ‘Dog’ class can inherit from an ‘Animal’ class, getting all its properties and behaviors while adding its own specific ones. |
| Polymorphism | means "many shapes." In OOP, it allows methods to do different things based on the object they are acting upon. For example, a method called ‘speak()’ could make a ‘Dog’ object bark and a ‘Cat’ object meow, even though the method name is the same. |
| Operator Overloading | allows custom definitions for standard operators (like +, -, \*, etc.) for user-defined classes. For example, you can define how the ‘+’ operator works for a ‘Vector’ class so that adding two ‘Vector’ objects together results in a new ‘Vector’ with summed components. |

### Exercise 1.6: Connecting to Databases in Python

#### Learning Goals

* Create a MySQL database for your Recipe app

#### Reflection Questions

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

Databases are organized collections of data that allow us to store, manage, and retrieve information efficiently. They are used in various applications, such as websites, apps, and other software systems. Advantages of using databases are as follows: organization, efficiency, security, scalability, and data integrity.

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

|  |  |
| --- | --- |
| **Data type** | **Definition** |
| INT (Integer) | Represents whole numbers without decimal points |
| VARCHAR (Variable Character) | Stores variable-length strings |
| DATE | Stores date values in the format ‘YYYY-MM-DD’ |

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

SQLite is best suited for scenarios where simplicity, ease of setup, and lightweight performance are more important than the advanced features and scalability offered by MySQL. (ie. Mobile apps, embedded devices, small web projects)

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

Python feels more accessible and less intimidating from a beginner perspective. It is also more straightforward and readable. JS seems to be more versatile since it ties into the web so well and has asynchronous capabilities.

1. 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?

Honestly, I don’t think I know enough about it to be able to accurately anticipate its limitations. Know it is not particularly used for mobile dev and that the GUI libraries are considered weaker than other languages is all that I can really say in confidence.

### Exercise 1.7: Finalizing Your Python Program

#### Learning Goals

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

#### Reflection Questions

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

An Object-Relational Mapper (ORM) is a tool that allows developers to interact with a relational database using an object-oriented paradigm. Instead of writing raw SQL queries, an ORM lets you manipulate database tables as if they were regular Python objects.

1. 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?

The project went well overall. One aspect of the app that I did particularly well was the user interface design for the terminal. I made sure it was user-friendly and intuitive, with clear prompts and error handling to guide users through creating, viewing, searching, updating, and deleting recipes. This helped ensure a smooth user experience.

If I were to start over, I would focus on adding an option to exit from functions such as updating a recipe. Currently, once you start a function, you must complete it or restart the app to return to the main menu. Implementing an exit option within these functions would provide users with more flexibility and control, allowing them to easily return to the main menu without completing an action they no longer wish to perform.

1. 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.

I've gained experience in creating a recipe management application using Python. This project involved building a database-driven application with SQLAlchemy to manage recipes. I designed a user interface in the terminal to create, view, search, update, and delete recipes. The app uses a MySQL database to store recipe data, including ingredients, cooking time, and difficulty levels, which are calculated based on specific criteria. Through this project, I developed skills in database interactions, user input validation, and modular programming in Python.

1. You’ve finished Achievement 1! Before moving on to Achievement 2, take a moment to reflect on your learning in the course so far:
   1. What went well during this Achievement? No major conceptual walls
   2. What’s something you’re proud of? Timely and Consistent Progress
   3. What was the most challenging aspect of this Achievement? Understanding Environment setup and linking
   4. Did this Achievement meet your expectations? Did it give you the confidence to start working with your new Python skills? yes
   5. What’s something you want to keep in mind to help you do your best in Achievement 2? Take it one piece at a time. No rush.

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?

It has proved effective.

* 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?

Useful comments and clean formatting which helped facilitate trouble-shooting issues.

* 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?

Biggest frustration was environment setup and connection between IDE, Python, and MySQL. Found good resources for troubleshooting and clearer explanations of what is happening.

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

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?

When deciding between vanilla Python and Django, there are key advantages and drawbacks. Vanilla Python offers maximum flexibility and is lightweight, but it can be time-consuming and requires manual handling of features like routing and authentication. Django, however, promotes rapid development with built-in features and enhanced security, and it has strong community support. The drawbacks of Django include less flexibility and potential overhead for simple projects where a full framework might be unnecessary.

1. 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 Model View Template (MVT) architecture over Model View Controller (MVC) architecture is that MVT simplifies web development by integrating the template (view logic) directly into the framework, making it easier to manage and render web pages dynamically, whereas MVC requires separate handling of the view and the controller.

1. 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?

A good understanding of its benefits and capabilities vs other languages.

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

Improve the ability to effectively apply Python in future projects when appropriate.

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

Since this is a continuing education course for my business, it will depend on what clients come down the pipe as to when and how this Python training will be applied. I am currently in the late stages of a TypeScript, Next.js, AWS job that will be my sole focus for the next couple of months.

### 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.*)

To convert a company's website into Django terms, I would proceed as follows:

* Models: Define the database schema by creating Django models for the main entities of the website, such as Users, Products, BlogPosts, and Comments. These models represent the data structure and relationships.
* Views: Implement the logic for handling requests and returning responses. For example, create views for displaying product listings, user profiles, and blog posts. Views will process data using models and render the appropriate templates.
* Templates: Design HTML templates to define the layout and structure of the web pages. Templates will be used to display the data passed from views, ensuring separation of presentation from business logic.
* URLs: Set up URL routing to map different URLs to their corresponding views. Define URL patterns for different parts of the website, such as the homepage, product details, and blog sections.
* Forms: Use Django forms to handle user input for tasks like registration, login, and posting comments. Forms will manage validation and data processing.
* Admin: Utilize Django's built-in admin interface to manage website content easily. Configure the admin for managing models like Users, Products, and BlogPosts.

1. In your own words, describe the steps you would take to deploy a basic Django application locally on your system.
   1. Set Up Environment: Install Python and Django on your system. Use virtualenv to create an isolated environment for your project.
   2. Create Django Project: Run django-admin startproject projectname to create a new Django project.
   3. Run Migrations: Navigate to the project directory and run python manage.py migrate to apply database migrations.
   4. Create Superuser: Create an admin user by running python manage.py createsuperuser and follow the prompts.
   5. Start Development Server: Run python manage.py runserver to start the local development server.
   6. Access Application: Open your web browser and go to http://127.0.0.1:8000/ to see your Django application running locally.
2. Do some research about the Django admin site and write down how you’d use it during your web application development.

Use it to manage and oversee my web application's data without having to write custom interfaces. First, I would enable the admin interface by creating a superuser and registering my models in the admin.py file. Then, I could log into the admin site to add, edit, and delete entries in my database, like users, posts, or products. This tool would help me quickly manage the content and verify that my models and database interactions are working correctly during development.

### Exercise 2.3: Django Models

#### Learning Goals

* Discuss Django models, the “M” part of Django’s MVT architecture
* Create apps and models representing different parts of your web application
* Write and run automated tests

#### Reflection Questions

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

Django models work by defining the structure of your database tables using Python classes. Each model represents a table, and each attribute of the model represents a column in that table. You create these models in your models.py file. The benefits of using Django models include simplicity, as they make it easy to manage database tables using Python code instead of complex SQL queries. Django handles database creation, updates, and schema changes automatically, which simplifies database management. Additionally, you can add validation and constraints directly in the model, ensuring data integrity. Models are also tightly integrated with the rest of Django, making it easy to use them in views, forms, and templates.

1. 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 the reliability and correctness of your code. Early test cases catch bugs and issues before they become more complex and harder to fix. They provide a safety net, allowing you to make changes and refactor your code with confidence, knowing that any regressions or new bugs will be caught early. Test cases also improve the design and quality of your code by encouraging you to think about edge cases and requirements from the start. Additionally, they save time and resources in the long run by reducing the need for extensive debugging and manual testing.

### Exercise 2.4: Django Views and Templates

#### Learning Goals

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

#### Reflection Questions

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

Django views work by handling the logic for your web application. For example, if you have a blog and want to display a list of posts, you create a view that queries the database for all blog posts and sends that data to a template. The template then generates the HTML to display the posts. When a user visits the blog page, the view gets the request, fetches the data, and returns the rendered HTML to the user's browser. This separation of logic (views) and presentation (templates) makes it easier to manage and organize your code.

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

In this scenario, I would use Django class-based views because they allow for better code reuse and organization. Class-based views provide a way to group related logic into reusable components, making it easier to manage and extend the project as it grows. They also come with built-in generic views that handle common tasks, reducing the amount of code I need to write.

1. 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.

Overview:

Purpose: Django's Template Language is used to create dynamic HTML pages by embedding Python-like syntax within HTML.

Basic Syntax:

Variables: Denoted by double curly braces ({{ variable\_name }}). Used to display data. Example:

<p>{{ username }}</p>

Filters: Modify the display of variables. Example:

{{ date|date:"Y-m-d" }} Example: <p>{{ name|upper }}</p>

Tags:

Control Flow: Use {% tag %} for logic like loops and conditionals.

For Loop: Iterates over a list. Example:

{% for item in item\_list %}

<p>{{ item }}</p>

{% endfor %}

If Statement: Conditional rendering. Example:

{% if user.is\_authenticated %}

<p>Welcome, {{ user.username }}!</p>

{% else %}

<p>Please log in.</p>

{% endif %}

Template Inheritance:

Extending Templates: Create a base template and extend it in child templates.

Base Template: Example:

<!DOCTYPE html>

<html>

<head>

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

</head>

<body>

{% block content %}{% endblock %}

</body>

</html>

Child Template: Example:

{% extends "base.html" %}

{% block title %}Home{% endblock %}

{% block content %}

<h1>Welcome to my site!</h1>

{% endblock %}

Comments:

Template Comments: Ignored during rendering. Example: {# This is a comment #}

Template Variables and Context:

Context: Data passed to templates from views. Example:

def my\_view(request):

context = {'username': 'John'}

return render(request, 'my\_template.html', context)

### 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.

Django static files are files like CSS, JavaScript, and images used in your web app. Django handles them by using the STATIC\_URL setting to define the URL path for these files and the STATICFILES\_DIRS setting to specify directories where these files are stored. During development, Django serves static files automatically. In production, you need to collect all static files into a single directory using the collectstatic command and configure your web server to serve them. This ensures that your web app's appearance and behavior are consistent and efficient.

1. 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 class-based view that displays a list of objects from a model. It's useful for showing multiple records from the database in a single page, like a list of blog posts or products. By using ListView, you don't need to write a lot of boilerplate code to fetch and display the data. |
| DetailView | a class-based view that displays a single object from a model. It's perfect for showing detailed information about a specific record, like the details of a single blog post or product. DetailView simplifies the process by handling the retrieval and display of the object's data with minimal code. |

1. 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.

Going smoothly still. Proud to see the cumulative benefit of the previous courses showing now in how much more easily I can pick up these concepts for a new language. No particular struggle, but definitely need to look more into current design concepts and how to implement them.

### 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.

Incorporating authentication into an application is crucial because it ensures that only authorized users can access certain parts of the application, protecting sensitive data and functionality. Authentication helps verify the identity of users, allowing the application to provide personalized experiences and maintain user privacy. It also enhances security by preventing unauthorized access, which can protect against data breaches and misuse of the application. Additionally, authentication is essential for implementing other security measures, such as authorization and auditing, which are necessary for maintaining the integrity and reliability of the application.

1. In your own words, explain the steps you should take to create a login for your Django web application.
   1. Add Authentication URLs:
      * Include Django's built-in authentication URLs in your project's URL configuration.
   2. Create Login Template:
      * Create a login.html template in your app's templates directory.
   3. Configure URL Redirects:
      * Set up URL redirects in your settings to specify where users should go after logging in.
   4. Protect Views with Login Required:
      * Use the login required decorator to restrict access to certain views.
2. Look up the following three Django functions on Django’s official documentation and/or other trusted sources and write a brief description of each.

|  |  |
| --- | --- |
| **Function** | **Description** |
| authenticate() | checks if a user's credentials (like username and password) are correct. You provide it with the credentials, and it returns a user object if they are valid, or None if they are not. This is commonly used during the login process to verify user identities. |
| redirect() | sends a user to a different URL. Instead of rendering a template, it tells the user's browser to visit a new URL. You can use it to redirect users after they log in, log out, or submit a form. It's useful for guiding users to appropriate pages based on their actions. |
| include() | is used in URL configuration. It allows you to include other URL patterns from different apps or modules. This helps keep your URL configurations organized and modular, making it easier to manage URLs across large projects. |

### Exercise 2.7: Data Analysis and Visualization in Django

#### Learning Goals

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

#### Reflection Questions

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

Analyzing collected data can significantly enhance a website or application by providing valuable insights into user behavior and preferences. By understanding which features are most used, where users drop off, and what content is most engaging, developers can make informed decisions to improve user experience. Data analysis can also help identify trends and patterns, enabling personalized content and targeted recommendations, which can increase user satisfaction and retention. Additionally, insights from data can guide strategic decisions, optimize marketing efforts, and ultimately drive growth and efficiency for the website or application.

1. Read the [Django official documentation on QuerySet API](https://docs.djangoproject.com/en/3.2/ref/models/querysets/). Note down the different ways in which you can evaluate a QuerySet.

 **Iteration**:

* When you iterate over a QuerySet, Django fetches the data from the database.
* Example: for obj in queryset:

 **Slicing**:

* Slicing a QuerySet, using Python’s array-slicing syntax, will also evaluate the query.
* Example: queryset[:10]

 **Conversion to List**:

* Converting a QuerySet to a list will evaluate it.
* Example: list(queryset)

 **Boolean Evaluation**:

* When a QuerySet is evaluated in a Boolean context, such as in an if statement, it will execute the query.
* Example: if queryset:

 **len()**:

* Using the len() function on a QuerySet evaluates it and returns the number of records.
* Example: len(queryset)

 **repr() or str()**:

* Calling repr() or str() on a QuerySet evaluates it.
* Example: str(queryset)

 **Pickling**:

* When you pickle a QuerySet, it will be evaluated.
* Example: import pickle; pickle.dumps(queryset)

 **count()**:

* The count() method on a QuerySet evaluates it and returns the number of records.
* Example: queryset.count()

 **exists()**:

* The exists() method evaluates a QuerySet and returns a Boolean indicating whether the QuerySet contains any results.
* Example: queryset.exists()

 **aggregate()**:

* Using the aggregate() method triggers evaluation.
* Example: queryset.aggregate(Sum('field'))

 **first() and last()**:

* The first() and last() methods evaluate the QuerySet and return the first or last object.
* Example: queryset.first()

 **Values QuerySet Evaluation**:

* When using values() or values\_list(), the QuerySet is evaluated when the result is consumed.
* Example: list(queryset.values('field'))

 **get()**:

* The get() method evaluates the QuerySet and returns the object matching the query.
* Example: queryset.get(id=1)

1. 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.

DataFrames are often better than QuerySets for data processing because they are designed specifically for handling and analyzing large datasets. They make it easy to perform complex data transformations, aggregations, and statistical analyses. DataFrames also support importing and exporting data in various formats and have many built-in functions for data cleaning and manipulation. This makes DataFrames a powerful tool for tasks that require intensive data processing and analysis, providing more flexibility and efficiency than QuerySets.

### 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 can be utilized to create a more visually appealing interface, ensuring consistency across different pages and devices through responsive design techniques. You can use frameworks like Bootstrap or Tailwind CSS to quickly implement modern design elements and layout structures.  
JavaScript can be leveraged to add interactivity and dynamic behavior to your application. For instance, you can use JavaScript to create real-time form validation, interactive charts, and graphs (using libraries like Chart.js or D3.js), and asynchronous data loading with AJAX to improve page load times and provide a smoother user experience. Additionally, JavaScript frameworks like React or Vue.js can be integrated to build more complex and dynamic user interfaces.

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

* **Install the Heroku CLI**:
* Download and install the Heroku Command Line Interface (CLI) from the Heroku website.
* **Create a Heroku Account**:
* Sign up for a Heroku account if you don't already have one.
* **Prepare Your Django Project**:
* Make sure your Django project is in a Git repository.
* Update your settings.py:
  + Set DEBUG = False.
  + Configure ALLOWED\_HOSTS to include Heroku's domain.
* Add django-heroku, gunicorn, and whitenoise to your project requirements.
* Configure static files with whitenoise.
* **Create a requirements.txt File**:
* Generate this file using pip freeze > requirements.txt to list all dependencies.
* **Create a Procfile**:
* Define a Procfile to specify the commands that are executed by the app on startup. Typically, it includes web: gunicorn <your\_project\_name>.wsgi.
* **Install gunicorn**:
* Add gunicorn to your requirements.txt to handle the application server.
* **Initialize a Git Repository**:
* Ensure your project is initialized with Git and commit all changes.
* **Login to Heroku via CLI**:
* Use heroku login to authenticate your Heroku account.
* **Create a New Heroku App**:
* Run heroku create <your\_app\_name> to create a new application on Heroku.
* **Add Heroku PostgreSQL Add-on**:
* Run heroku addons:create heroku-postgresql:hobby-dev to add a PostgreSQL database to your Heroku app.
* **Configure Django Settings for Heroku**:
* Use django-heroku to configure your Django settings for Heroku.
* **Migrate the Database**:
* Run heroku run python manage.py migrate to apply migrations on the Heroku database.
* **Push Code to Heroku**:
* Deploy your application by pushing your code to Heroku: git push heroku main.
* **Collect Static Files**:
* Run heroku run python manage.py collectstatic to collect static files on Heroku.
* **Open Your App**:
* Use heroku open to open your deployed application in the browser.
* **Monitor Logs**:
* Monitor your application logs using heroku logs --tail to troubleshoot any issues.

1. You’ve now finished Achievement 2 and, with it, the whole course! Take a moment to reflect on your learning:
   1. What went well during this Achievement?

Avoided major roadblocks by using improved methods of troubleshooting and research gained through this full curriculum.

* 1. What’s something you’re proud of?

Adapted well to the a new language while being able to apply previously learned concepts fluidly.

* 1. What was the most challenging aspect of this Achievement?

Running down connection errors between views, templates, forms, utils, and models.

* 1. Did this Achievement meet your expectations? Did it give you the confidence to start working with your new Django skills?

Yes, very much actually. Glad to have done it.

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