## Pre-Work: Before You Start the Course

Reflection questions (to complete before your first mentor call)

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

I’ve completed a bachelor’s degree in Computer Science where I studied some Python and Java, completed a small JavaScript course from FreeCodeCamp, and have worked on several personal mini projects in various languages.

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

I am aware of the fundamentals of Python and that it’s one of the most popular and versatile programming languages in the world, with its easy to digest syntax and extensive standard library and an active community contributing to it.

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 am confident that I can tackle any challenges that arise throughout the course with my solid foundation in Python from university and the help of my mentor.

### Exercise 1.1: Getting Started 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 involves creating the visual elements and user interface of a website that users interact with directly, typically using HTML, CSS, and JavaScript. Backend web development, on the other hand, focuses on the server-side operations that power the frontend, such as database management, user authentication, and handling requests. If hired to work on backend programming for a web application, I would be responsible for tasks like designing and maintaining the database structure, implementing server-side logic to process user input, ensuring data security, and integrating with external APIs or services.

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?**

I would explain that both JavaScript and Python are versatile programming languages commonly used in web development, but they have distinct strengths and use cases. While JavaScript excels in frontend development with its browser compatibility and asynchronous capabilities, Python offers a more cohesive ecosystem for backend development, with powerful frameworks like Django, as well as broader applicability in other non-web related fields. Additionally, Python's clean syntax and emphasis on readability can lead to faster development, easier maintenance, and smoother collaboration among team members, making it the better choice for our project's backend requirements.

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?**
2. Master Data Structures and Algorithms: I aim to deepen my understanding of fundamental data structures such as lists, dictionaries, sets, and tuples, as well as algorithms like sorting and searching. This knowledge will enhance my problem-solving skills and enable me to write more efficient and scalable code.
3. Solidify Object-Oriented Programming Concepts: I intend to strengthen my grasp of object-oriented programming principles, including inheritance, encapsulation, and polymorphism. By practicing these concepts extensively, I aim to write cleaner, modular, and more maintainable code for the Recipe app project.
4. Learn Database Interaction: I aspire to become proficient in interacting with databases using Python, including querying, updating, and managing data. Understanding how to integrate databases into my applications will be crucial for the Recipe app project and will prepare me for working with frameworks like Django in the future.

### Exercise 1.2: Data Types in Python

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 offers several advantages over Python's default shell. Firstly, it provides enhanced interactivity and productivity features such as tab completion, syntax highlighting, and inline documentation, which facilitate faster coding and debugging, and iPython supports integration with various data analysis libraries and tools.
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 | An integer represents a whole number without any decimal points. | Scalar |
| Tuple | Tuples are ordered collections of elements, similar to lists, but they are immutable. | Non-scalar |
| List | Lists are ordered collections of elements that can be of any data type. | Non-scalar |
| Dictionary | Dictionaries are unordered collections of key-value pairs. | 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.**

In Python, lists and tuples are both ordered collections of elements, but the main difference lies in their mutability. Lists are mutable, meaning their elements can be modified, added, or removed after creation, while tuples are immutable, meaning once they are created, their elements cannot be changed. Additionally, lists are typically used for collections of items where the order and elements might change, whereas tuples are often employed for fixed collections where immutability is desired, such as coordinates or configurations.

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 like this, where users input vocabulary words, definitions, and categories, a dictionary would be the most suitable data structure because dictionaries allow for storing key-value pairs, where the vocabulary word can be the key and the definition along with the category can be the values. This structure provides fast lookup times based on the vocabulary word, making it efficient for retrieving flashcard information during quizzes.

Using a dictionary also offers flexibility because you can easily update or modify existing flashcards, add new ones, or even incorporate additional information like example sentences or synonyms. This flexibility is valuable for future development if the app expands to include more advanced language-learning features beyond simple vocabulary memorization.

### Exercise 1.3: Functions and Other Operations in Python

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!)***



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 and perform logical operations on boolean values. There are three main logical operators: `and`, `or`, and `not`. The `and` operator returns True if both operands are True, the `or` operator returns True if at least one operand is True, and the `not` operator returns the opposite boolean value of the operand.

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

Functions in Python are reusable blocks of code that perform a specific task, useful for organizing code, improving readability, and promoting reusability by encapsulating logic into modular units.

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.**

I’m getting more comfortable using data structures after having some hands-on experience using them while working on the past few optional and main tasks for the achievement.

### Exercise 1.4: File Handling in Python

Reflection questions

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

File storage is crucial in Python because it allows you to save and retrieve data persistently, enabling data persistence across sessions. Without storing local files, any data or progress made within a program would be lost once the program terminates, making it impossible to retrieve or continue from previous states.

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 refer to the serialization process of converting Python objects into byte streams for storage or transmission. Pickles are useful when you need to save complex data structures, such as dictionaries or custom objects, to files, allowing you to easily reload them later without losing their structure or state.

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?**

In Python, you can use the **os.getcwd()** function to find out the current working directory. If you want to change your current working directory, you can use the **os.chdir()** function by providing the path of the directory you want to switch to.

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?**

To prevent the entire script from terminating due to an error in a specific block of code, you can use exception handling with a try-except block. By wrapping the potentially problematic code within a try block, you can catch and handle any exceptions that occur gracefully, allowing the rest of the script to continue executing unaffected by the error.

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.**

It’s going great, I’m getting a lot of practice working with Python and I learned a lot of new things about working with files in Python that I didn’t know before.

### Exercise 1.5: Object-Oriented Programming in Python

Reflection questions

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

Object-oriented programming (OOP) is a programming paradigm centered around the concept of objects, which are instances of classes. In OOP, data and behavior are bundled together within objects, allowing for modular and organized code. The benefits of OOP include improved code reusability, as objects can be reused across different parts of the program, enhanced code maintainability through encapsulation, which hides internal implementation details, and increased productivity due to the ability to model real-world entities more accurately.

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, while objects are instances of classes that encapsulate data and behavior. For instance, consider a **Car** class, where attributes like **make**, **model**, and **year** represent data, and methods like **start()** and **drive()** represent behavior. An object of this class could be **my\_car**, with specific values for the attributes, such as **make="Toyota"**, **model="Corolla"**, and **year=2020**. This **my\_car** object can then call methods like **start()** and **drive()** to perform actions.

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

|  |  |
| --- | --- |
| **Method** | **Description** |
| Inheritance | Inheritance allows a class (subclass) to inherit attributes and methods from another class (superclass). Subclasses can extend or modify the behavior of the superclass, promoting code reuse and establishing a hierarchical relationship between classes. |
| Polymorphism | Polymorphism allows objects of different classes to be treated as objects of a common superclass. It enables methods to be implemented differently in different subclasses while sharing a common interface. This flexibility simplifies code maintenance and supports dynamic method invocation. |
| Operator Overloading | Operator overloading involves defining custom behavior for operators in Python classes. By overloading operators such as **+**, **-**, or **\***, objects of a class can perform operations that are meaningful in the context of that class. This promotes code clarity and allows objects to behave intuitively in mathematical or logical expressions. |

### Exercise 1.6: Connecting to Databases in Python

Reflection questions

1. **What are databases and what are the advantages of using them?**  
   Databases are organized collections of data, allowing for efficient storage, retrieval, and manipulation of information. Advantages include data integrity, scalability, and concurrent access.
2. **List 3 data types that can be used in MySQL and describe them briefly:**

|  |  |
| --- | --- |
| **Data type** | **Definition** |
| VARCHAR | Variable-length character string, used for storing alphanumeric characters of varying lengths. |
| INT | Integer data type, used for storing whole numbers. |
| DATE | Date data type, used for storing dates in the format YYYY-MM-DD. |

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

SQLite might be a better choice than MySQL for small-scale applications or when simplicity and ease of setup are priorities over scalability and multi-user concurrency.

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?**

JavaScript is primarily used for web development and runs in the browser, while Python is a versatile language suitable for a wide range of applications including web development, data analysis, and scripting.

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?**

Python's limitations include slower execution speed compared to languages like C or C++, limited support for mobile development compared to languages like Swift or Java, and the Global Interpreter Lock (GIL) which can hinder multi-threaded performance in CPU-bound applications.

### Exercise 1.7: Finalizing Your Python Program

Reflection questions

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

An Object Relational Mapper (ORM) is a programming technique that allows developers to convert data between incompatible type systems in object-oriented programming languages, effectively bridging the gap between an object-oriented language and a relational database. The advantages of using an ORM include simplifying database interactions by abstracting complex SQL queries, improving code maintainability and readability, and enabling more efficient development through the use of high-level programming constructs.

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 creation of the app went really well, I think I managed to create a clean code base and user experience using some more advanced techniques that weren’t even mentioned in the course materials. If I were to start over, I would spend more time focusing on catching edge cases that could potentially break the program by the user entering wrong inputs for example.

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.**

In my recent projects, I've gained substantial experience in Python. I’ve developed an application using object-oriented programming principles, allowing for a structured and scalable codebase. I've also integrated database manipulation capabilities, using Python's database access libraries to persistently store and manage recipe data efficiently. This project not only solidified my understanding of Python's capabilities but also provided hands-on experience with database operations, object-oriented design, and practical application development.

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?**

I’ve managed to solidify my foundation of Python’s basics, and also learn more about OOP and databases, and the way ORMs work and how they help with bridging the gap between data structured in an OOP way vs data in relational databases.

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

I’m proud of the clean codebase and some advanced techniques used.

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

Understanding how MySQL works.

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

I would say that it did, yes.

* 1. **What’s something you want to keep in mind to help you do your best in Achievement 2?**

Try to be more efficient in reading and absorbing information.

### Exercise 2.1: Getting Started with Django

Reflection questions

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

Using vanilla Python offers maximum control and simplicity for small or highly specific projects but lacks the built-in features and security optimizations that frameworks like Django provide, which can significantly speed up development for larger applications but may introduce unnecessary complexity for smaller projects.

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 the Model View Template (MVT) architecture over the Model View Controller (MVC) architecture is its clear separation between how the application's user interface is designed and how it interacts with the business logic, making it more streamlined for web development. In MVC, the Controller directly handles both user input and the updating of the View, creating a more intertwined relationship between how data is processed and presented. In contrast, MVT separates these concerns by introducing a Template for defining the UI separately from the Views, which handle the presentation logic based on the Model's data, allowing for a more modular and flexible approach to building web applications.

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?**

I want to learn how to efficiently utilize Django's built-in features to expedite the development of secure and scalable web applications.

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

My goal is to gain a comprehensive understanding of Django's MVT architecture and ORM to enhance my web development skills and efficiency.

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

After completing this achievement, I see myself confidently building and deploying robust web applications with Django, potentially focusing on developing dynamic websites or backend services for start-ups or personal projects.

### Exercise 2.2: Django Project Set Up

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.*)**

For the Spotify website, viewed through the lens of Django, the entire website is a project named "Spotify". Within this project, we could identify several apps that correspond to distinct features of the site: "UserAuth" for user authentication and management, "MusicLibrary" for browsing and searching music, "Playlists" for creating and managing playlists, and "UserProfiles" for user profile customization and social interaction. Each app is designed to handle a specific set of tasks, such as the "UserAuth" app managing login/logout functionality and user registration, thereby modularizing the website's functionality in a maintainable and scalable way.

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

To deploy a basic Django application locally, you would first ensure Python is installed on your system. Then, you would create a virtual environment using the virtualenvwrapper extension for better dependency management by executing `**mkvirtualenv myprojectenv**`. After activating the virtual environment, install Django using pip with `**pip install django**`. Finally, create a new Django project with `**django-admin.exe startproject <name-of-project >**`, navigate into your project directory, and start the development server with `**python manage.py runserver**`, making your application accessible on your local machine.

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

During web application development, I’d use the Django admin site to quickly create, read, update, and delete the database records managed by my application's models, providing an interface for administrative tasks. This powerful feature allows for an efficient way to manage content and users, streamlining the process of testing and interacting with the application's data layer without needing to manually interact with the database or write additional code for these common operations.

### Exercise 2.3: Django Models

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 act as the structured layer that interacts with the database, allowing developers to define the database schema (tables, fields, and relationships) in Python code, which Django then translates into database-specific language, streamlining database operations and ensuring data integrity. Their primary benefit is abstracting complex database queries into simple, readable Python code, making web application development faster and less prone to errors.
2. **In your own words, explain why it is crucial to write test cases from the beginning of a project. You can take an example project to explain your answer.**

Writing test cases from the beginning of a project, such as a web application, is crucial because it ensures each part of the application works as expected from the start, helps catch bugs early, and facilitates easier maintenance and updates by automatically verifying that changes do not break existing functionality, leading to more reliable and robust software.

### Exercise 2.4: Django Views and Templates

Reflection questions

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

Django views act as the logic layer between the user interface (template) and the database (model), handling HTTP requests from users and returning HTTP responses. For instance, a view might retrieve information on published articles from the database using Django's ORM, and then employ a template to render this data into HTML format for the user's browser. Essentially, views are where the "business logic" of a Django application is implemented, dictating how user inputs are processed and what is returned to them.

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?**

For a Django web development project that anticipates significant code reuse throughout various components, class-based views would be the preferred choice due to their enhanced reusability and organization. Class-based views facilitate the creation of reusable behaviors and common patterns through inheritance and mixins, allowing developers to extend and customize standard views more efficiently than with function-based views. This architectural approach helps manage complexity and reduces duplication by encapsulating common functionality in classes that can be extended or overridden as the project evolves.

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.**

The Django template language (DTL) is designed to strike a balance between power and ease, allowing you to build dynamic web pages by separating the presentation layer from the logic. Key basics include:

* **Variables**: Use {{ variable\_name }} to display the value of a variable.
* **Tags**: Enclosed in {% tag %}, tags are more complex operations that can perform loops, conditionals, and other logic within templates.
* **Filters**: Applied to variables, filters transform their values. Syntax: {{ variable | filter:"argument" }}.
* Inheritance: Template inheritance lets you build a base "skeleton" template that contains the common structure of your site, on which other templates can build by overriding blocks of content.
* **Auto-escaping**: For security, DTL automatically escapes variables to prevent injection attacks, but this can be manually disabled for individual variables or blocks when necessary.
* **Comments**: You can comment out code in templates using {# comment #} for single lines or {% comment %} and {% endcomment %} tags for block comments.

DTL's design philosophy emphasizes the ease of use for template authors (designers), aiming for an intuitive and easy-to-learn syntax while providing developers with the power to extend template functionality as needed.

### Exercise 2.5: Django MVT Revisited

Reflection questions

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

Django static files are files such as images, CSS, and JavaScript that aren't dynamically generated and are served unchanged to the user's browser. Django handles them by allowing developers to store these files in one or more directories that Django can retrieve and serve efficiently when requested by a web page.

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 Django package that provides a way to display a list of objects from the database, making it easier to manage the display of collections, such as articles in a blog. |
| DetailView | Compared to ListView, DetailView is designed to present detailed information about a specific instance of an object from the database, such as the details of a single blog post, by handling the fetching of the object based on a provided identifier (like a primary key). |

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.**

I can definitely use more practice with Django, as the workflow is quite different compared to frameworks such as React and Angular. That, combined with the sudden change from JS to Python made it a big learning experience, but it’s going well so far.

### Exercise 2.6: User Authentication in Django

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, such as a financial management app, is crucial for verifying the identity of users to ensure that sensitive information and user-specific functionalities are securely accessed only by authorized individuals, thereby protecting both user privacy and data integrity.

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

To create a login for your Django web application, start by configuring your **urls.py** to include a path for the login, often utilizing Django's built-in **auth.views.LoginView** or a custom view that you define. Next, implement the login view in your **views.py**, which can involve creating a form that handles user authentication, using Django's **authenticate()** function to verify credentials, and **login()** function to log the user in. Finally, create a login template, referred to by your login view, that presents a form for username and password input, ensuring a user-friendly interface for logging in.

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

|  |  |
| --- | --- |
| **Function** | **Description** |
| authenticate() | This function is used in Django's authentication system to verify a set of credentials. It takes username and password as arguments and returns a user object if the credentials are valid and correspond to an existing user, otherwise it returns **None**. |
| redirect() | This function is a convenient way to make HTTP redirects. It takes a model, view, or URL as an argument and returns an **HttpResponseRedirect** or **HttpResponsePermanentRedirect** to the appropriate URL for that argument. |
| include() | Used in URLconf, **include()** allows you to include other URLconfs. It's a way to reference the URLs defined in other modules, making it easier to organize and maintain the URL structure of larger Django projects by breaking it into smaller, manageable pieces. |