## 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. |