# **Exercise 1.1: Getting Started with Python**

## **Learning Goals:**

* Understand how Python can be used for web development and its benefits.
* Set up a development environment to work with Python.

## **Reflection Questions:**

1. **What's the difference between frontend and backend web development? If you were hired to work on backend programming, what tasks would you be working on?  
   Answer:** Frontend development focuses on the user interface and experience—everything the user interacts with, such as buttons, forms, and layouts. This is typically built using technologies like HTML, CSS, and JavaScript. Backend development, on the other hand, deals with the server-side operations, including database management, authentication, and data processing. If I were hired for backend development, I’d work on tasks like managing user accounts, interacting with databases, securing sensitive information, and building APIs that connect the frontend with the backend.
2. **If your team was deciding between using JavaScript or Python for a project, and you thought Python was the better choice, how would you explain the differences and why Python might be better?  
   Answer:** Both Python and JavaScript are strong, versatile programming languages, but they serve different purposes. JavaScript is essential for frontend development and is widely used in building interactive web elements. However, if we're working primarily on backend development, Python might be a better choice due to its clean, readable syntax, making it easier for teams to collaborate and maintain code over time. Python also has a vast collection of libraries and frameworks (like Django and Flask) that are well-suited for backend work, offering robust tools for data management, security, and API development. Additionally, Python’s community support and regular updates make it a reliable option for long-term projects.
3. **Now that you’ve had an introduction to Python, what are three goals you have for your learning during this course?  
   Answer:**
   * Improve my understanding of object-oriented programming (OOP) in Python.
   * Learn how to write Python scripts to automate repetitive tasks.
   * Gain deeper knowledge of backend development, focusing on Python’s handling of databases, APIs, and server-side tasks.

# **Exercise 1.2**

## **Benefits of Using the IPython Shell Over Python’s Default Shell**

I would suggest using the IPython Shell over the default one because it has a lot of useful features that make coding easier and faster. For example, it has tab completion, which means I don’t have to type out full variable names or commands, saving me a lot of time. It also has magic commands like %run, %timeit, and %debug, which are great for running scripts, measuring performance, or jumping into debugging when something goes wrong. Plus, IPython has better command history management and formatting, which makes it easier to go back and check what I’ve run before.

### **Four Data Types in Python:**

* **int:** This represents whole numbers, like 10, -5, or 1000. It’s scalar because it holds a single value.
* **float:** This represents decimal numbers, like 3.14 or -0.01. It’s also scalar.
* **str:** This represents strings of characters, like "hello" or "Python". It’s non-scalar because it can hold multiple characters as a sequence.
* **list:** This represents a collection of items, like [1, 2, 3] or ["apple", "banana"]. It’s non-scalar because it can hold multiple values.

## **Differences Between Lists and Tuples in Python**

The main difference between lists and tuples is that lists are mutable, meaning you can change, add, or remove elements after the list is created. On the other hand, tuples are immutable, so once they’re created, you can’t change them. Lists use square brackets [], while tuples use parentheses (). If I need a collection of items that won’t change, I would use a tuple for better performance. But if I need flexibility, like adding or removing items, I would use a list.

## **Data Structure for the Recipe App**

I used dictionaries to store individual recipes, with each recipe’s attributes (name, cooking time, ingredients) as key-value pairs. I used a list to store all the recipes, allowing for easy storage, retrieval, and modification.

## **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?  
   Answer:** I’d recommend using IPython because it offers a more interactive and efficient coding environment. IPython includes features like tab completion, which makes writing and editing code faster, and magic commands like %run for executing scripts and %timeit for measuring the performance of code. IPython also makes debugging easier with commands like %debug, allowing you to interact with your code right after an error. Overall, IPython provides a smoother and more feature-rich development experience, particularly when testing and debugging.
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.**
   * **Integer (int):** A whole number (e.g., 5). Scalar.
   * **Float:** A number with a decimal point (e.g., 3.14). Scalar.
   * **List:** A mutable, ordered collection of items (e.g., [1, 2, 3]). Non-scalar.
   * **Dictionary:** A collection of key-value pairs (e.g., {'name': 'Joud'}). Non-scalar.
3. **Difference Between Lists and Tuples in Python:  
   Answer:** The key difference between lists and tuples is mutability. Lists are mutable, meaning you can add, remove, or change elements, while tuples are immutable and cannot be changed after creation. Lists are defined with square brackets [], while tuples use parentheses (). Tuples are generally faster and use less memory than lists because they are immutable, making them ideal for storing fixed data like coordinates. For dynamic data that may change, such as a shopping list, lists are a more appropriate choice.
4. **In the task for this Exercise, you decided what you thought was the most suitable data structure for storing all the information for a recipe. Now, imagine you’re creating a language-learning app that helps users memorize vocabulary through flashcards. Users can input vocabulary words, definitions, and their category (noun, verb, etc.). Between tuples, lists, and dictionaries, which would you choose?  
   Answer:** If I were building a vocabulary-learning app, I would choose dictionaries. That’s because dictionaries allow me to store key-value pairs, which is perfect for associating each vocabulary word (the key) with its definition and category (the values).  
   Example:

vocab = { 'run': {'definition': 'move quickly', 'category': 'verb'}, 'apple': {'definition': 'a fruit', 'category': 'noun'} }  
  
 Dictionaries allow for quick lookups and are flexible enough to expand in the future, like adding example sentences or difficulty levels for each word.  
  
  
**Exercise 1.3: Functions and Other Operations in Python**

## **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:**

**Answer** 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.”

Here’s how I would write the travel app script using an if-elif-else statement:

destination = input("Where do you want to travel? ")

if destination == "Paris": print("Enjoy your stay in Paris!") elif destination == "Tokyo": print("Enjoy your stay in Tokyo!") elif destination == "New York": print("Enjoy your stay in New York!") else: print("Oops, that destination is not currently available.")

## **Imagine you’re at a job interview for a Python developer role. The interviewer says “Explain logical operators in Python”. Draft how you would respond.**

**Answer:** Logical operators in Python are used to combine multiple conditions. There are three main ones:

* and: This returns True if both conditions are true. For example, True and False gives False.
* or: This returns True if at least one of the conditions is true. For example, True or False gives True.
* not: This reverses the Boolean value, so if something is True, not will make it False, and vice versa.

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

**Answer:** Functions in Python are reusable blocks of code that perform specific tasks. They are useful because they make the code more organized and easier to maintain. Instead of writing the same code multiple times, I can write a function once and call it whenever needed. This makes my code shorter and more efficient. Functions can also accept parameters and return values, which allows them to be flexible.

Example:

def greet(name): return f"Hello, {name}!"

print(greet('John')) # Output: Hello, John!

Functions help break complex problems into smaller, manageable pieces.

## **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’ve been making progress toward my goals, especially in understanding conditional statements and loops. I’ve learned to use if-elif-else statements to control the flow of my program, and I’m getting better at writing functions to make my code more modular. Using loops has helped me reduce repetitive code, and I feel more confident about organizing my programs.

Moving forward, I’d like to focus on improving my ability to handle edge cases and work with larger datasets more efficiently.

# **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?  
   Answer:** File storage is essential because it allows data to persist beyond the runtime of a program. When data is stored in a file, it can be retrieved later, even after the program has stopped running, which is critical for saving user data, logs, or configurations. Without file storage, all data would be lost once the program terminates, meaning you'd have to re-enter all the data every time you run the program. This would severely limit the functionality of applications, especially those that rely on maintaining records, like databases, logs, or user sessions.
2. **In this Exercise you learned about the pickling process with the pickle.dump() method. What are pickles? In which situations would you choose to use pickles and why?  
   Answer:** Pickles are serialized representations of Python objects, meaning they convert Python objects into a byte stream that can be saved to a file or sent over a network. The pickle module allows you to store complex data types like lists, dictionaries, or even custom objects in a file and retrieve them later in the same form. You would use pickling when you need to save the state of a Python object to reuse it later, such as saving machine learning models, configurations, or data structures between program runs. However, pickles should be used cautiously because they are specific to Python and could pose security risks if the source is untrusted.
3. **In Python, what function do you use to find out which directory you’re currently in? What if you wanted to change your current working directory?  
   Answer:** To find out which directory you are currently in, you use the os.getcwd() function. This function retrieves the current working directory of the script. If you want to change the current working directory, you can use the os.chdir() function to specify a new path.

**4. Imagine you’re working on a Python script and are worried there may be an error in a block of code. How would you approach the situation to prevent the entire script from terminating due to an error?**

**Answer:** To prevent the entire script from terminating due to an error, I would use a try-except block. This technique allows me to catch and handle exceptions that may occur during the execution of the code. If an error occurs within the try block, the program will jump to the except block instead of terminating.

**5. You’re now more than halfway through Achievement 1! Take a moment to reflect on your learning in the course so far. How is it going? What’s something you’re proud of so far? Is there something you’re struggling with? What do you need more practice with? Feel free to use these notes to guide your next mentor call.**

**Answer:** The course has been going well so far, and I’ve made significant progress. I'm particularly proud of my growing confidence with Python’s conditional statements, loops, and writing functions to make my code more modular. I’ve also improved in using file handling techniques, which has been exciting.

However, I’m still struggling with understanding more complex data structures, especially when they become nested. I aim to focus on gaining a deeper understanding of these areas moving forward. I’m confident that with additional practice, I will improve in working with larger datasets and nested data structures.