CAREERFOUNDRY

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?

As a matter of fact, I might call myself a career software developer. I started programming back in school on Fortran IV, having first live implementation in the 10th grade. I continued at the university, solving physics problems. In 1990-s I did industrial coding - simpler things on FoxPro, MS Access, core banking systems on Unisys LINC 4. Later, until 2005 I was implementing SAP R/3 and BW, doing coding in ABAP. Naturally, I got a good understanding of system architecture, databases and data processing. Basic knowledge of deployment and operations.

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

Before starting the course I knew that Python is a popular programming language, especially for data analytics. One of many.

3. 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.

Find practical application for real-life projects? I do not foresee anything dramatic.

Remember, you can always refer to <u>Exercise 1.4</u> 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?

Short: backend is mostly busy handling data and business logic on a server, frontend handles UX on the human facing device. As a backend developer I'd be handling database design and data exchange, business logic, API endpoints.

2. 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")

First of all, I'd make my choice only based on requirements and environment, not a bare "sympathy". Both JavaScript and Python are high-level, dynamically typed languages, but Python's biggest edge is its readability—the clean, indented syntax makes it way easier to debug and share code across the team. It comes with "out-of-the-box" essentials, meaning it handles complex tasks like web security and database connections right out of the box. Plus, with simple package management via `pip` and a massive community to lean on, we can prototype and deploy much faster without reinventing the wheel.

3. 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?

Learning Goals:

1. **Master the Basics: Get comfortable enough with Python syntax and core libraries to write clean code without a cheat sheet.**
2. **Practical Project: Build something functional that actually works.**
3. **Leadership: Learn enough to confidently manage Python projects and the people coding them.**

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 like the "pro" version of the shell because it gives you syntax highlighting and tab-completion, which makes coding much faster and helps catch typos instantly. Plus, being able to use the `%run` command to execute scripts and keep the variables active is useful for 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.

Data type	Definition	Scalar or Non-Scalar?
float	Numbers with a fractional part (decimals)	Scalar
string	A sequence of characters wrapped in quotes	Scalar
dictionary	A collection of key-value pairs	Non-Scalar
tuple	An immutable, ordered sequence of elements	Non-Scalar

3. 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.

Mutability: Lists are mutable (can be changed after creation), while tuples are immutable (fixed).
Data Integrity: Tuples are safer for data that shouldn't change (like constants or coordinates); lists are for data that changes over time.
Performance: Tuples are faster and more memory-efficient because their fixed size allows Python to allocate memory once. Lists have extra overhead to handle resizing.
Usage: Tuples can be used as dictionary keys because they are fixed and thus hashable, lists cannot.

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.) 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.

I chose a List of Dictionaries for the whole flashcard deck to allow for easy shuffling and iteration during quizzes. Each flashcard is a Dictionary that uses a List of Strings for definitions, Strings for grammatical categories ("noun", "verb", etc.) and tags ("regional", "slang", etc. - to provide flexibility for words with multiple nuances or remarks. To ensure data integrity, I would use Tuples for categories and tags; these act as "closed collections," preventing typos and ensuring that every card follows a consistent, valid classification system.

Code example:

```
# Fixed categories that don't change
CATEGORIES = ("noun", "verb", "adjective", "adverb", "preposition")
# Validated remarks/tags
VALID_TAGS = ("slang", "regional", "outdated", "formal", "archaic")
```

```
card_1 = {
    "word": "lassie",
    "definitions": ["A girl", "A young woman"],      # List of strings for nuance
    "category": CATEGORIES[0],                        # Single string from tuple
    "tags": [VALID_TAGS[1], VALID_TAGS[0]]           # List of strings (from tuple) for multiple remarks
}
card_2 = {
    'word': 'bairn',
    'definition': ['A child'],                        # List of one string
    'category': CATEGORIES[0],                        # References "noun"
    'tags': [VALID_TAGS[1]]                           # References "regional" inside a list
}
```

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!)*

```
# Step 1: Ask the user for their destination
destination = input("Where do you want to travel? ")

# Step 2: Check the input against 3 defined destinations
if destination == "Lisbon":
    print("Enjoy your stay in Lisbon!")
```

```
elif destination == "Paris":
    print("Enjoy your stay in Paris!")
elif destination == "New York":
    print("Enjoy your stay in New York!")
else:
    # Step 3: Handle any other input
    print("Oops, that destination is not currently available.")
```

2. 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 - specifically `and`, `or`, and `not` - are the tools we use to combine or invert multiple boolean conditions within a single expression.
In a professional setting, we use them to control the flow of an application, such as verifying if a user has both the correct permissions and an active subscription before granting access. They follow short-circuit logic, meaning Python is efficient enough to stop evaluating an expression as soon as the final result is certain.

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

Functions are reusable blocks of code designed to perform a specific action. You define them using the `def` keyword, and they only run when you "call" them by their name.

When and why they are useful:

- **DRY Principle: They enforce the "Don't Repeat Yourself" (DRY) principle by allowing you to write logic once and use it many times.**
- **Modularity: They help break complex programs into smaller, manageable blocks.**
- **Readability: Descriptive function names (like `calculate_total()`) make code self-explanatory.**
- **Maintenance: If you need to change how a calculation works, you only have to fix it in one place.**

4. 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 have progressed from basic syntax to implementing the core logic of a functional Recipe App, demonstrating the ability to handle nested data structures and complex conditions. This move into practical application shows I am successfully bridging the gap between learning theory and building working software.

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?

File storage allows for data persistence. Without it, all data (like our recipes) exists only in the computer's memory and is lost right after the Python script stops running. Storing local files ensures that data is still there when we need them again.

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?

"Pickles" are the result of serialization - converting a complex Python object (like a nested dictionary or a list of lists) into a byte stream for storage. Developers use pickles when they need to save specific Python-only data structures that text files cannot properly handle, as it preserves the exact data types without manual formatting.

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?

To find the current working directory, we use `os.getcwd()`. To change it, we use `os.chdir('/path/to/directory')`. Both commands require importing the `os` module.

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?

I would wrap the risky code in a `try-except` block. This catches the error (exception) if it occurs, allowing the script to handle it smoothly (e.g., by printing a warning or setting a default value) instead of crashing the program.

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.

How it's going: It's going smoothly. The concepts are pretty clear, and I'm finding the logic manageable without it feeling overly complicated.

Proud of: Mastering the complete `try-except-else-finally` suite for robust error handling and successfully implementing data persistence with `pickle`. I'm also proud of incorporating more modern, "Pythonic" tools like f-strings, relative paths, and the `.get()` method to make my code cleaner and more efficient (thanks to my mentor for the advice to use Set type on certain occasions).

Struggling with / Need practice: Dealing with the nuances of file paths - specifically handling user input errors like accidental quotes or incorrect directory levels. I might focus more on input validation and making my code "bulletproof" against unexpected user behavior.

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?

Object-Oriented Programming is a paradigm that organizes software design around data, or objects, rather than functions and logic. An object is defined as a data field that has unique attributes and behavior. Instead of focusing on the logic required to manipulate objects, OOP focuses on the objects themselves.

The Benefits:
- **Modularity: Code is divided into independent units, making it easier to troubleshoot and navigate.**
- **Reusability: Code created for one program can be easily bridged into another through the use of established classes.**
- **Scalability: The structured nature of OOP allows systems to grow in complexity without becoming unmanageable.**

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

A Class serves as a template or a blueprint for creating objects. It defines the structure and the capabilities that any object created from it will possess. An Object is a specific instance of that class, containing actual data.

Real-World Example: Consider a Car. A car manufacturer has a master template (the Class) that specifies a vehicle must have an engine, four wheels, and a steering wheel. It also defines that the vehicle can "accelerate" or "brake." When the factory actually produces a car - a red sedan with a specific VIN number - that physical vehicle is an Object. Multiple objects (cars) can be created from the same template, but each has its own unique color, speed, and location.

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

Method	Description
Inheritance	Inheritance is a mechanism that allows a new class to derive its properties and characteristics from an existing class. This establishes a hierarchy where a "child" class inherits the attributes and methods of a "parent" class. For example, by defining `class ShoppingList(object):`, the `ShoppingList` class inherits fundamental behaviors from Python's base object template. This allows the new class to utilize built-in logic for identity and comparison without manual coding.
Polymorphism	Polymorphism is the ability of different objects to respond to the same method call in a manner specific to their own data type. It allows a single interface to represent a general class of actions. For example, if multiple different classes have a `__str__` method, calling `print()` on any of them will work perfectly, but each will produce a different text output based on its own internal data. The program doesn't need to know the specific class type to execute the display command.
Operator Overloading	Operator Overloading is the practice of defining custom logic for standard operators - such as `+`, `-`, or `==` - when they are applied to user-defined classes. By default, Python does not know how to mathematically interact with custom objects; operator overloading provides the specific instructions for these interactions. For example, in a `Height` class, using `__sub__` allows the use of the `-` operator to calculate the difference between two people. Similarly, in a `ShoppingList` class, `__add__` can be defined to merge two sets of items using the `+` symbol.

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?

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

Data type	Definition

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

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

5. 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?

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?

2. 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?

3. 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.

4. You've finished Achievement 1! Before moving on to Achievement 2, take a moment to reflect on your learning in the course so far:

a. What went well during this Achievement?
   b. What's something you're proud of?
   c. What was the most challenging aspect of this Achievement?
   d. Did this Achievement meet your expectations? Did it give you the confidence to start working with your new Python skills?
   e. What's something you want to keep in mind to help you do your best in Achievement 2?

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

Pre-Work: Before You Start Achievement 2

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

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

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

Remember that can always refer to Exercise 1.4 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?

2. In your own words, what is the most significant advantage of Model View Template (MVT) architecture over Model View Controller (MVC) architecture?

3. 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?
   - What do you want to get out of this Achievement?
   - Where or what do you see yourself working on after you complete this Achievement?

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.*)

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

3. Do some research about the Django admin site and write down how you'd use it during your web application 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.

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.

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.

2. 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?

3. Read Django's documentation on the Django template language and make some notes on its basics.

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.

2. 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	
DetailView	

3. 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.

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

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.

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

3. 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()	
redirect()	
include()	

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.

2. Read the Django official documentation on QuerySet API. Note down the different ways in which you can evaluate a QuerySet.

3. 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.

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.

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

3. (Optional) Connect with a few Django web developers through LinkedIn or any other network. Ask them for their tips on creating a portfolio to showcase Python programming and Django skills. Think about which tips could help you improve your portfolio.

4. You've now finished Achievement 2 and, with it, the whole course! Take a moment to reflect on your learning:
   a. What went well during this Achievement?
   b. What's something you're proud of?
   c. What was the most challenging aspect of this Achievement?
   d. Did this Achievement meet your expectations? Did it give you the confidence to start working with your new Django skills?

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