Python
Learning Journal

Pre-Work: Before Starting

Reflection questions

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?

   Answer: Prior to this specialization course on Python, I've completed a frontend and a full-stack development course at CF. Throughout these courses, I built a website, several web apps and a mobile app. In order to do so, I've used different technologies / tools such as HTML, CSS, JavaScript, Bootstrap, Node.js, MongoDB, PostgreSQL, React, React Native, Angular, and more. I also learned how to develop different coding technics such as the test-driven development (TDD) or the behavior-driven development (BDD) methods.

   Before enrolling into CF, I also had different working experiences that were not related to coding, but related to data and technologies, which could help me in this course and my coding learning in general.

   For example, I worked as a Senior analyst in data and technology policies, where my main focus was oriented towards data governance, data lifecycle and R&D on sensitive data sharing across organizations. All the work carried on while I was occupying this position was done in an Agile environment. Before that, I also worked as a Professional researcher on a project related to supporting decision-making in a municipal infrastructure development context, using GIS and non-GIS data. During this project, I've been exposed to some Python codes, but only as an observer, not a developer.

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

   Answer: I don't know much about Python so far, except that it's a widely used language across different domains, and that it's one of the most popular coding technology among developers.

   I am very enthusiastic about learning Python. I would like to learn everything useful about this language, from how to set a Python coding environment, to how Python codes are generally structured, what are Python specificities and what makes it a better tool than another language on different types of projects. In general, I would like to acquire a good Python knowledge, both on theory and practice, allowing me to use Python whenever it would be the best tool for X project.

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.

   Answer: As for the challenges that may arise, the main one I am thinking of is the ability to distance myself from the coding concepts, patterns, and structures of other coding languages

I've learned so far (mainly JavaScript), if necessary. I am wondering if, in order to learn Python, it is necessary to distance myself from JavaScript concepts and principles, or if, in the opposite, both languages are close to each other and so knowing one makes it easier to learn the other.

Learning a completely new language also represents a good challenge, since it requires more time and effort in comparison to developing additional knowledge in a language for which the basis is already well-know. This learning from the scratch on Python is however extremely important in order to have a solid foundation on which to build further knowledge.

In order to face these challenges and allow me to successfully acquire a good Python knowledge, I will apply several methods I've developed over my coding learning journey so far. They could be breakdown as follows:

- Take the time to read each text content on Python correctly, and read it back when something isn't clear.
- If something in the course material is still not clear (even after a second read), go online to search for that same topic and see if complementary information / examples are available to help my understanding.
- Always do each task at the best of my capacities – the goal not being to simply get the task accepted, but to understand everything I've done, and why.
- In each task, when something isn't working as expected, always try to troubleshoot the problem by myself first via all the debugging technics I could think of.
    - This not only allow me to develop my debugging skills, it also helps me to remember how to fix that specific problem if I ever encounter it again in the future.
- Compare my Python codes to some other codes I've written in the past (e.g.: in JavaScript) if I think it could be useful to better understand the differences and the functioning of each language respectively.
- Ask my mentor for any questions I have not been able to find the answer by myself.
- Continue learning about Python even after the course end, by reading theory books/articles and by creating other apps by myself.

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?

   Answer:

- **Backend**: Backend is related to all the codes and structures that are not seen by the users on their UI, but that are still essential to ensure that all the elements the users can interact with / see on their screen actually work as expected.

  For example, a backend developer could work on setting up the different API endpoints that allow users to perform different operations in the web app (or other product) database. If a user creates an account, he'll be using a form displayed on the frontend. However, as soon as he submits the form, the API endpoint created to handle user creation account in the backend (POST request) will be used, and the database (which is also part of the backend) will be updated accordingly, by registering the new user info.

  On top of endpoints and database, other elements could also be implemented by a backend developer, such as codes to hash users' passwords, a CORS policy to protect the web app, or codes to define database schemas / models.

  In the MERN and MEAN tech stacks, the backend is represented by Node.js, Express and MongoDB for example.

- **Frontend**: Frontend is related to all the codes and structures that are seen by the users on their UI. The frontend is closely related to the backend, as those two work together to ensure the web app (or other product) is fully functional.

  For example, if the backend of a web app has been developed to allow users to create an account and delete it, the frontend will display a user registration form on the users UI, as well as a delete account button. When these elements are used by users, the backend logic is called and perform the required action.

  Important considerations for frontend development usually include easily navigation through the web app, accessibility, great visual, responsiveness, good communication with the backend, and so on.

  In the MERN and MEAN tech stacks, the frontend is represented by React (MERN) and Angular (MEAN) for example.

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?

   Answer: Both languages could be interesting, as both of them share some similarities. For example, both are a high-level scripting language and uses easily understandable keywords to make commands and perform tasks, which is a great plus. Python, as well as JS, uses dynamic typing, an approach that allows variables to assume any kind of value without producing errors, which is another interesting element.

   However, I would suggest to use Python for the following reasons:

   - Python frameworks come with pre-installed common web operations such as URL routing, form handling and validation, template engines, etc. This helps developers to move quickly with application development.
   - Python code is easily readable, which help to understand documentation and performing debugging more easily. Python also comes with its own shell called REPL, which is fairly easy and simple to use, and can be of a great help to test python code syntax and perform debugging quickly.
   - Finally, Python is widely used, so there's a big community of developers around it, a lot of documentation, discussions, GitHub topics and so on. This is very helpful to find different important information over the development process.

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?

   Answer:

   1- By the end of this achievement, I want to have a very good base about core Python principles, in order to be able to discuss on Python with very good level of confidence if someone was to ask me questions such as:

   - What is Python?
   - What are the main characteristics of Python?
   - In which scenarios / projects Python would be the best tool, and why?
   - What are the main differences between Python and JavaScript?

   2- By the end of this achievement, I want to know enough about Python to at least be able to identify what personal projects I could build using Python, and have a good idea of how the different features could be implemented.

3- By the end of this Achievement, I would like to improve the way I developed my documentation. More precisely, I would like to have a professional level README related to my project, to show my Python technical communication skills in the best way possible.

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?

   Answer: In essence, Python's default shell is not that user-friendly, especially when it comes to reading code. It's a good tool for executing and debugging Python commands, but the text in the Python default shell is the same color, making it difficult to distinguish writing, keywords and individual lines of code. Code also needs to be indented manually each time if we are writing out functions or other nested statements, which is not that practical.

   On the other hand, the iPython Shell provide a clearer code visibility, because it displays different features of the code in contrasting fonts and colors. Indenting text for nested statements is also done by iPython shell automatically. Those details, among others, makes it easier to work with iPython than Python default shell.

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	Data type that represents both negative and non-negative numbers (from zero to infinity). Does not include numbers with decimal, since these are **float.**	Scalar
Bool	Data type that represents either of two values : **True** or **False**.	Scalar
List	Data type that represents linear arrays that can store multiple values of different types, such as numbers or string. Lists differ from tuples (another form of similar data type) in that they are mutable - any of the internal elements of a list can be modified or deleted.	Non-scalar
Dictionnary	Data type that represents an unordered set of items,	Non-scalar

	each of them being a key-value pair, where each key is unique. Each item is individually labeled, and does not sequentially related to the other items. A dictionary allows different data types in the key-value pairs such as string, integers and lists.	

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.

   Answer: Tuples are linear arrays that can store multiple values of any type. They're useful for situations where we need to save several values, but naming a new variable for every one of them would be too much hassle. Tuples can't be altered once they're defined, since they're ***immutable***. The only way to change elements in a tuple is to write a new version of the tuple and assign it the original variable name.

   *One small exception to this is when we want to **add** an element in a tuple - it's possible to do so on the original tuple by placing the new value we want to add in a single-element tuple, and then concatenate (merge) the two tuples together.

   List are another type of ordered sequence, similar to tuples. However, lists differ from tuples in that they are ***mutable***, meaning that any of the internal elements of a list can be modified or deleted, which is not the case in a tuple. It's also possible to rearrange or insert new elements in lists.

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.

   Answer: For the Flashcards' creation by users, I would use dictionaries regrouping key-value pairs of strings. The strings would let users write their inputs regarding the vocabulary words, their definitions, and their word category for each flashcard. This structure would allow users to create as many cards as wanted easily. The dictionaries are also flexible, so it would be easy to add potential new key-values pairs / features in the flashcards in the future (with other data types than strings if necessary).

Example:

```
flashcard_1 = {
            'word_name' = 'word',
            'word_definition': ‘definition’,
            'word_category": ‘category’
           }
```

To allow users to quiz themselves, I would create an outer list, which would contain all flashcards created previously by the user. I would then set up the UI so that this list would be used to show each card in a way that allows users to quiz themselves / do some revision.

The list would be more useful than a tuple, since lists are mutable, meaning that any of the internal elements of in it can be modified or deleted. They can also be sorted, which could be great if users want to organize their cards in certain ways. All of these characteristics would make the list a great choice.

Example:

```
flashcard_list = [‘flashcard_1', ‘flashcard_2’, ‘flashcard_3']
```

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.

```
travel_destination = str(input("Where would you like to travel? "))
travel_destination = travel_destination.capitalize()
```

```
if travel_destination == "Canada":
    print("Enjoy your stay in " + travel_destination + ' !')

elif travel_destination == "Mexico":
    print("Enjoy your stay in " + travel_destination + ' !')

elif travel_destination == "Bolivia":
    print("Enjoy your stay in " + travel_destination + ' !')

else:
    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.

   Answer: In Python, logical operators are OR, AND and NOT. The OR and AND are used to check more than one condition at the time and return a boolean value accordingly (**true** or **false**).

   The OR operator checks if at least one condition is **true** in a statement. If it's the case, the code using the OR operator will return a **true** value. The only way a code using the OR operator will return a **false** value is if all conditions are **false**.

   The AND operator checks if at all conditions are **true** in a statement. If it's the case, the code using the AND operator will return a **true** value. In any other situation (if one of the condition is **false** or all conditions are **false**), the code using the AND operator will return a **false** value.

   The NOT operator is used to reverse the result of a logical expression that comes after it. For example, if a condition statement would normally return a **true** value, but the NOT operator is placed before the condition statement, the result will be reversed to **false**.

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

   Answer: In Python, there are built-in functions (e.g.: print( ) and append( )) and on-purpose created functions (functions created by developers to answer certain needs). In essence, functions are sets of instructions that process the code in order to achieve certain things. They can have argument(s), which ensure that the functions are running with specific value(s) to process.

   Functions are particularly useful, because they can be defined one time, and then called whenever necessary elsewhere in the code. This allows to reduce significantly the amount of code / repetition that is required to run some actions, since a simple call to an already defined function doing these same actions do the thing.

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.

Answer: So far, I think I mostly progressed on my goal #1, which was to have a good base about core Python principles, in order to be able to discuss about Python with very good level of confidence if someone was to ask me questions such as:

- What is Python?
- What are the main characteristics of Python?
- In which scenarios / projects Python would be the best tool, and why?
- What are the main differences between Python and JavaScript?

Throughout the exercises, I acquired a good knowledge of the different data types in Python, as well as the logical operators, the if-elif-else statements, loops, functions and so on. I also have a better idea of what are the benefits of developing with Python, and how to use effectively iPython Shell, which are all pieces of knowledge that could be useful in any discussion regarding Python, whether on a more technical level, or more general level.

As for my two other goals, I feel like I am progressing, but that more knowledge / practice / reflection / revision would still be necessary / useful to reach them more.

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?

   Answer: File storage is important because when we use variables in scripts to assign and keep track of values, these data no longer exists when the script stops running. Data can't be retrieved for later use if they're not stored, which is not practical at all. Data therefore need to be stored somewhere in files in order to be read, updated or modified later, even after the script, the program, or the application is closed.

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 useful when we need to store more complex data structures than just strings or numbers (dictionaries for example). Simpler data structures can be stored, for example, in txt. files, but with more complex data, pickles are the best option

   Pickles convert complex data into a packaged stream of bytes, known as a 'pickle', then write this into a binary file. A binary file can store complex information in a way that can be read by a machine but not by humans. Then, when necessary, it is possible to retrieve and read these data

stored as binary file using **pickle.load()** (and some logic to indicate how we want to display specifically the data retrieved).

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 know in which directory you are in, we can run the command: **os.getcwd()**
   To change from a working directory to another, we can run the command: **os.chdir(<path to desired folder>)**

   *Important to note that to run these commands, it is necessary to import the **os module**:
   import os

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: I would implement a **try-except** block. More precisely, I would place the code where I expect a potential error to occur in the **try** section. That way, if no errors are found, the rest of my code would execute as normal. However, if there's an error, a logic implemented in the **except** section (following the **try** one) would be put in use to notify me. That way, if any errors are found, Python won't terminate the entire program, but will highlight the existence of an error so I can fix it.

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: So far so good. I am particularly happy and proud that in general, I am capable to fix the problems I encounter when I code in Python by myself, without having to rely on external resources such as discussions on Github or Stack Overflow for example. Maybe this is because the way Python is (generally very readable and logic at first sight), or it's because I already had a good base in coding before jumping into learning Python (with JavaScript).

   I can't say that I am struggling with something in particular. Of course, I've encountered some problems and bugs so far while I worked with Python, but I've always been able to fix the issue after a certain time.

   That being said, I would like to get more familiar with the **for loops**, the **try-expect blocks** and how to write data into **binary files**, as these elements are very useful in many ways. They are not necessarily complicated to work with, but I would like to figure out more rapidly in which scenario they should be used and could be helpful, and know right away how I could code it / implement logics accordingly.

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?

   Answer: OOP is a programming concept that organizes the code around data (also called objects). It's different from functional programming.

   Most of the objects in Python can be broken down into the data they contain and into the methods that are possible to use to interact with them. Each object also has a corresponding type or class (template describing an internal structure). In OOP, objects are instances of classes (and classes define the structure and behavior of objects). The process of abstracting data and methods into classes is part of OOP. The advantage of pre-defining a class is that the script can then generate objects containing empty/partially data attributes that are waiting to be filled later.

   Inheritance, which allow to create new classes based on existing ones (inheriting their attributes and methods from other classes), and **Polymorphism**, which allow data attribute or method that has the same name across different classes or data types to perform different operations depending on where it was defined, are some key elements of OOP.

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

   Answer:

   Object: Most of the objects in Python can be broken down into the data they contain and into the methods it's possible to use to interact with the data. Some examples of objects:

   - **Numbers**: 123, 123.45
   - **Text**: "Hello"
   - **Lists**: ['This', 'is', 'a', 'list!']
   - **Tuples**: ('And', 'this', 'is', 'a', 'tuple.')
   - **Dictionaries**: { 'A+': 100, 'A': 90, 'A-': 80, 'B+': 70, 'B': 60, 'B-': 50 }
   - **Instances of custom classes**

   Class: Each object also has a corresponding type or class (a template describing an internal structure). One class may contain multiple corresponding objects. It's possible to define specific classes based on different needs (rather than relying on only in pre-built elements), in a similar way to how it's possible to define your own functions.

When creating a class, some specific elements are required:

- The **class** keyword
- The **class name** (to be chosen - normally in CamelCase)
- The **class object** (the class inheriting from - which is **object** by default)
- The class body, which contain data attribute and procedural methods

A real-world example of objects and classes could be a province and the cities within the province. The province would be the class and each city inside of it would be an object that belongs to this class.

Each city (or object) would hold data attribute such as the city name, date of foundation, square kilometers, population… (store information about the object), as well as methods for interacting with them and perform operations. Such methods could be used to calculate the population median age, the number of green areas compared to the total surface of the city etc.

Using the **class province** would make it easier to create any new **object city**, or update/modify an existing object city.

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 describes to possibility to inherit methods from one class to another, when the properties or methods of a particular class could be useful to another class (instead of making copies of the code).
Polymorphism	Polymorphism is when a given data attribute or method has the same name across different classes or data types, but performs different operations depending on where it was defined.
Operator Overloading	When using operators such as + and - on a custom class, a *TypeError* is thrown, since the operators aren't supported. To use such operators in a custom class, it is necessary to define methods for them, and this process is known as operator overloading. Most of Python's built-in classes however support special operators, so operator overloading is not necessary in those cases.