Exercise 1.3: Operators and Functions in Python

Reflection Questions :

- 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!)*

```
def travel_app():
    destination = input("Where do you want to travel? ")

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

travel_app()
```

- *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 are coding expressions that perform logical operations on boolean ("true" or "false") values, allowing you to combine and/or modify them respectively. They often use conditional if-elif-else statements to generate decisions based on multiple layered conditions.

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

  - Functions are re-usable blocks of code that perform tasks. In python they are defined with the "def [function name]():" syntax, then indented to start writing the behavior for it.
  - Reasons why functions are useful :
    - The fact that they are re-usable
    - They organize the code by breaking the different tasks into separate pieces
    - Values can take parameters, allowing values to be passed in to them

- *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 already gotten my hands dirty on many small aspects (mostly syntax) of Python that are different than what I have learned in my previous modules (HTML, JS, React, and Angular)*
  - *I have dove into the scripting logic head first. As I had hoped, the very direct nature of this Python logic (or perhaps due to the nature of this course's exercises so far), I feel like I am getting good learning reps with programmatic logic in general. I really enjoy the "mathy" feel to it – a sort-of math/linguistic synergy. That is the kind of logic I expected to see more of when signing up with CareerFoundry, to learn programming. I am returning to an inspired stat of mind,*

where I look for better/best ways to code things that may be beyond what is necessary approval of the exercises' tasks. Being curious and willing to go down those "rabbit holes" is the ideal state of mind to be in for productive learning.

- *The "dependent/causal action trees" I had mentioned being an idea I hope to explore more (and perhaps find the appropriate industry terms for this idea), or definitely present here. e.g. With if-elif-else functions, I am already starting to imagine scenarios where the layering of those conditions could be infinitely complex, and are likely the origins of many simple (and even perhaps some highly complex) games. While I am not particularly targeting game development as a personal ambition, my knowledge of gaming logic makes it a great scenario for imagining the potential of complexity in programmatic logic such that Python could be capable of.*