Introduction to Python: Core Concepts for Remote Sensing Applications

Prepared by
Mohamed Abdelkader[1], Jorge Bravo[1], and Jibin Joseph[2]
[1]Civil, Environmental, and Ocean Engineering Department, Stevens Institute of Technology
[2]School of Civil Engineering, Purdue University
mabdelka@stevens.edu

FAIR Science in Climate

Objective

- Understand the basic syntax and structure of Python programming.
- Learn to work with different Python data types, including integers, decimals, strings, lists, tuples, and dictionaries.
- Develop the ability to perform arithmetic operations and data manipulations in Python.
- Master the foundational concepts of string formatting for better output presentation.
- Gain familiarity with Python's control structures, including loops and conditional statements.
- Acquire the skills to manage and iterate over collections of data, which is crucial for processing remote sensing datasets.

Overview of steps

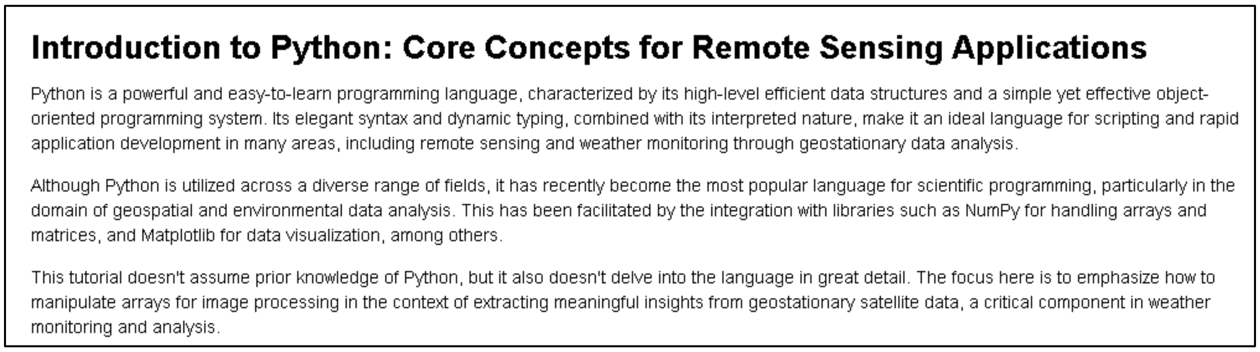
1. Introduction to Python's interactive environment and how it can be used in remote sensing.
2. Exploration of Python's basic data types and their relevance to data representation in remote sensing.
3. Performing and understanding operations between integers and decimals, simulating real-world measurements and calibrations.
4. Handling and formatting strings to annotate and label remote sensing data.
5. Creating and manipulating lists, tuples, and dictionaries, essential for organizing and accessing sensor data.
6. Iterative processes in Python, learning to loop through data sequences which is common in satellite data processing.

Resources

Remote Sensing Tutorials
Guide to GOES-R Series Data
GOES ABI (Advanced Baseline Imager) Realtime Imagery
GOES Image Viewer

Instructions

1. Access Lecture_1 folder and initiate "Introduction to Python_Core Concepts for RS Applications" notebook.
2. Review the introductory overview of Python's capabilities in remote sensing applications to understand the context and relevance of programming in this field.



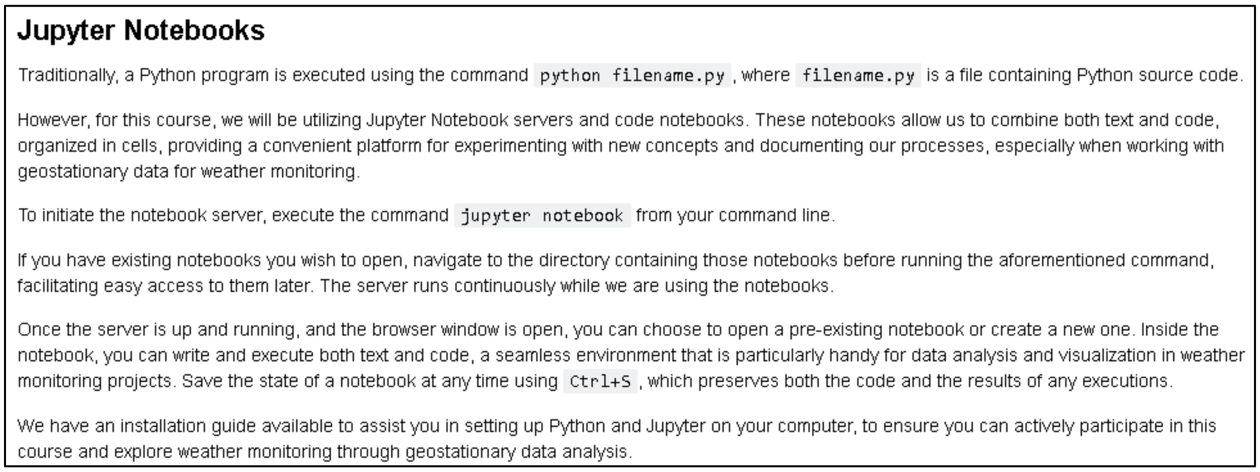
Introduction to Python: Core Concepts for Remote Sensing Applications

Python is a powerful and easy-to-learn programming language, characterized by its high-level efficient data structures and a simple yet effective object-oriented programming system. Its elegant syntax and dynamic typing, combined with its interpreted nature, make it an ideal language for scripting and rapid application development in many areas, including remote sensing and weather monitoring through geostationary data analysis.

Although Python is utilized across a diverse range of fields, it has recently become the most popular language for scientific programming, particularly in the domain of geospatial and environmental data analysis. This has been facilitated by the integration with libraries such as NumPy for handling arrays and matrices, and Matplotlib for data visualization, among others.

This tutorial doesn't assume prior knowledge of Python, but it also doesn't delve into the language in great detail. The focus here is to emphasize how to manipulate arrays for image processing in the context of extracting meaningful insights from geostationary satellite data, a critical component in weather monitoring and analysis.

3. Review the instructions for using Jupyter Notebooks as our interactive computing environment.



Jupyter Notebooks

Traditionally, a Python program is executed using the command `python filename.py`, where `filename.py` is a file containing Python source code.

However, for this course, we will be utilizing Jupyter Notebook servers and code notebooks. These notebooks allow us to combine both text and code, organized in cells, providing a convenient platform for experimenting with new concepts and documenting our processes, especially when working with geostationary data for weather monitoring.

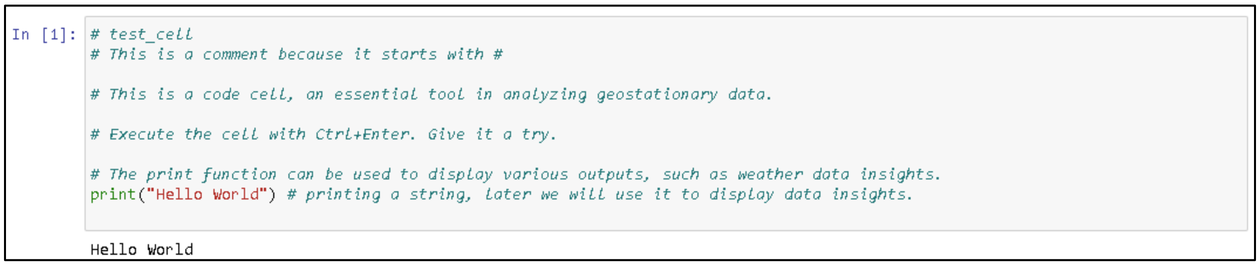
To initiate the notebook server, execute the command `jupyter notebook` from your command line.

If you have existing notebooks you wish to open, navigate to the directory containing those notebooks before running the aforementioned command, facilitating easy access to them later. The server runs continuously while we are using the notebooks.

Once the server is up and running, and the browser window is open, you can choose to open a pre-existing notebook or create a new one. Inside the notebook, you can write and execute both text and code, a seamless environment that is particularly handy for data analysis and visualization in weather monitoring projects. Save the state of a notebook at any time using `Ctrl+S`, which preserves both the code and the results of any executions.

We have an installation guide available to assist you in setting up Python and Jupyter on your computer, to ensure you can actively participate in this course and explore weather monitoring through geostationary data analysis.

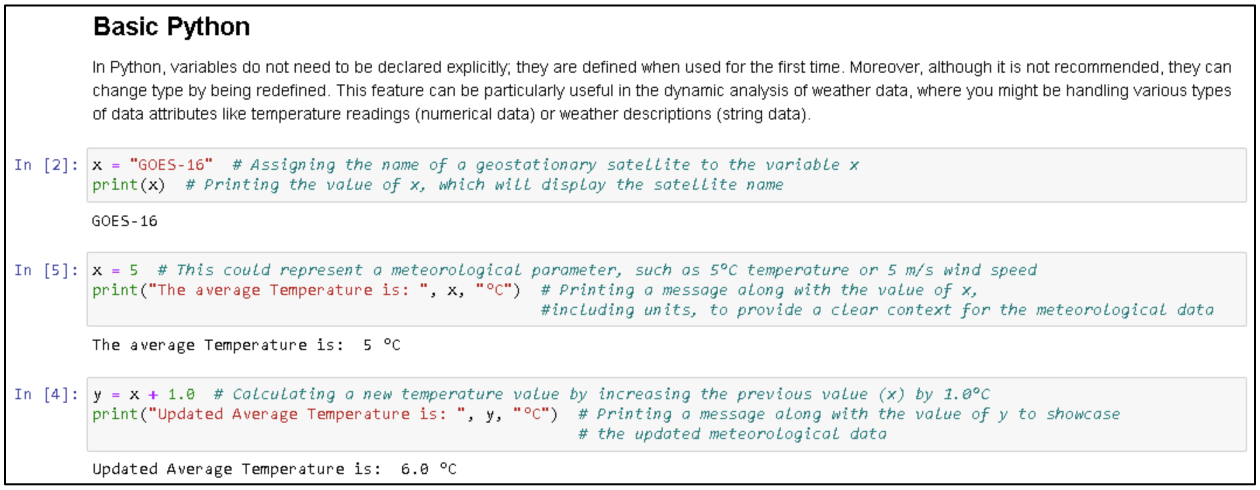
4. *Execute the code cell in #test_cell using Ctrl+Enter to practice running a simple print command.*



```
In [1]: # test_cell
        # This is a comment because it starts with #

        # This is a code cell, an essential tool in analyzing geostationary data.

        # Execute the cell with Ctrl+Enter. Give it a try.

        # The print function can be used to display various outputs, such as weather data insights.
        print("Hello World") # printing a string, later we will use it to display data insights.
```

```
Hello World
```

5. *Run the provided code to see how variables work in Python and observe how a variable can change types from a string to a number. Notice how the 'print' function is used to clarify the output, making the data's context and changes clear and understandable.*

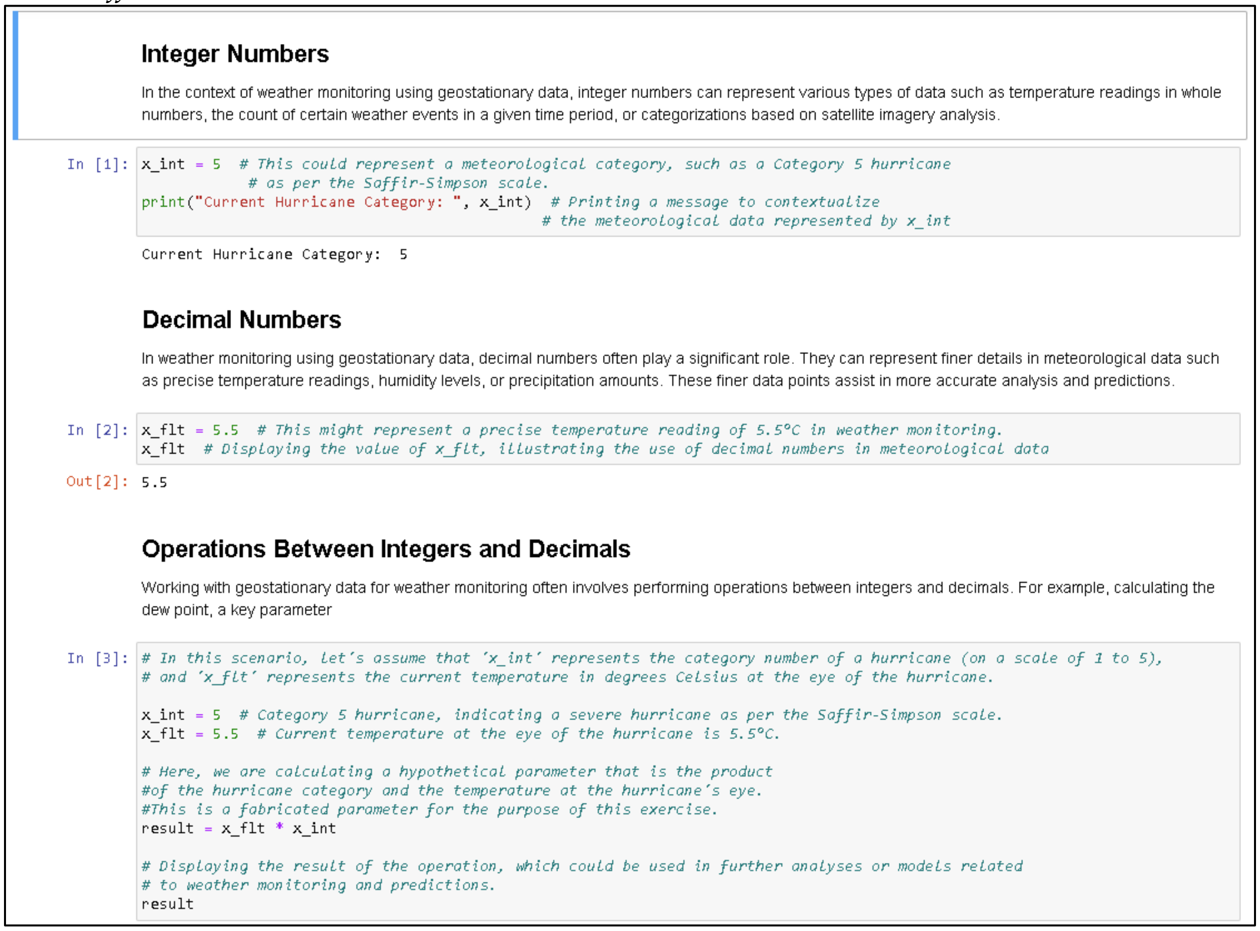


Basic Python

In Python, variables do not need to be declared explicitly; they are defined when used for the first time. Moreover, although it is not recommended, they can change type by being redefined. This feature can be particularly useful in the dynamic analysis of weather data, where you might be handling various types of data attributes like temperature readings (numerical data) or weather descriptions (string data).

```
In [2]: x = "GOES-16"  # Assigning the name of a geostationary satellite to the variable x
        print(x)  # Printing the value of x, which will display the satellite name

        GOES-16

In [5]: x = 5  # This could represent a meteorological parameter, such as 5°C temperature or 5 m/s wind speed
        print("The average Temperature is: ", x, "°C")  # Printing a message along with the value of x,
                                                         #including units, to provide a clear context for the meteorological data

        The average Temperature is:  5 °C

In [4]: y = x + 1.0  # Calculating a new temperature value by increasing the previous value (x) by 1.0°C
        print("Updated Average Temperature is: ", y, "°C")  # Printing a message along with the value of y to showcase
                                                             # the updated meteorological data

        Updated Average Temperature is:  6.0 °C
```

6. *Execute the code to observe how integers and decimals represent different data types in weather monitoring. Pay attention to the operations between them to see their combined effect on the results.*



Integer Numbers

In the context of weather monitoring using geostationary data, integer numbers can represent various types of data such as temperature readings in whole numbers, the count of certain weather events in a given time period, or categorizations based on satellite imagery analysis.

```
In [1]: x_int = 5  # This could represent a meteorological category, such as a Category 5 hurricane
                    # as per the Saffir-Simpson scale.
        print("Current Hurricane Category: ", x_int)  # Printing a message to contextualize
                                                      # the meteorological data represented by x_int

        Current Hurricane Category:  5
```

Decimal Numbers

In weather monitoring using geostationary data, decimal numbers often play a significant role. They can represent finer details in meteorological data such as precise temperature readings, humidity levels, or precipitation amounts. These finer data points assist in more accurate analysis and predictions.

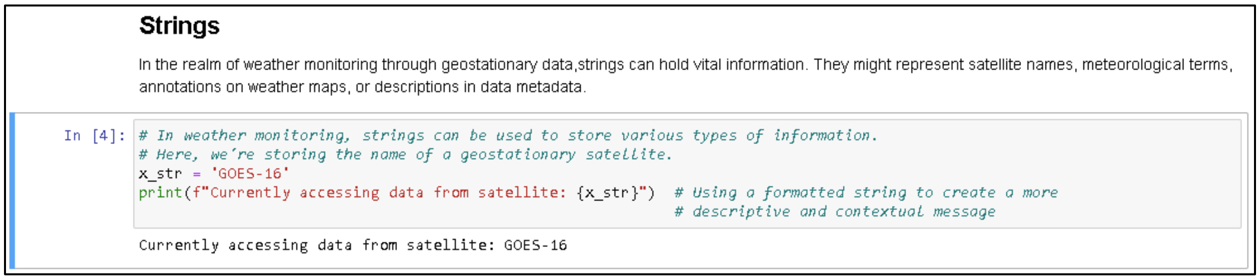
```
In [2]: x_flt = 5.5  # This might represent a precise temperature reading of 5.5°C in weather monitoring.
        x_flt  # Displaying the value of x_flt, illustrating the use of decimal numbers in meteorological data

Out[2]: 5.5
```

Operations Between Integers and Decimals

Working with geostationary data for weather monitoring often involves performing operations between integers and decimals. For example, calculating the dew point, a key parameter

```
In [3]: # In this scenario, let's assume that 'x_int' represents the category number of a hurricane (on a scale of 1 to 5),
        # and 'x_flt' represents the current temperature in degrees Celsius at the eye of the hurricane.

        x_int = 5  # Category 5 hurricane, indicating a severe hurricane as per the Saffir-Simpson scale.
        x_flt = 5.5  # Current temperature at the eye of the hurricane is 5.5°C.

        # Here, we are calculating a hypothetical parameter that is the product
        #of the hurricane category and the temperature at the hurricane's eye.
        #This is a fabricated parameter for the purpose of this exercise.
        result = x_flt * x_int

        # Displaying the result of the operation, which could be used in further analyses or models related
        # to weather monitoring and predictions.
        result
```

7. *Run the code to see how strings function to store and present textual data such as satellite names and note how formatted print statements enhance data readability.*

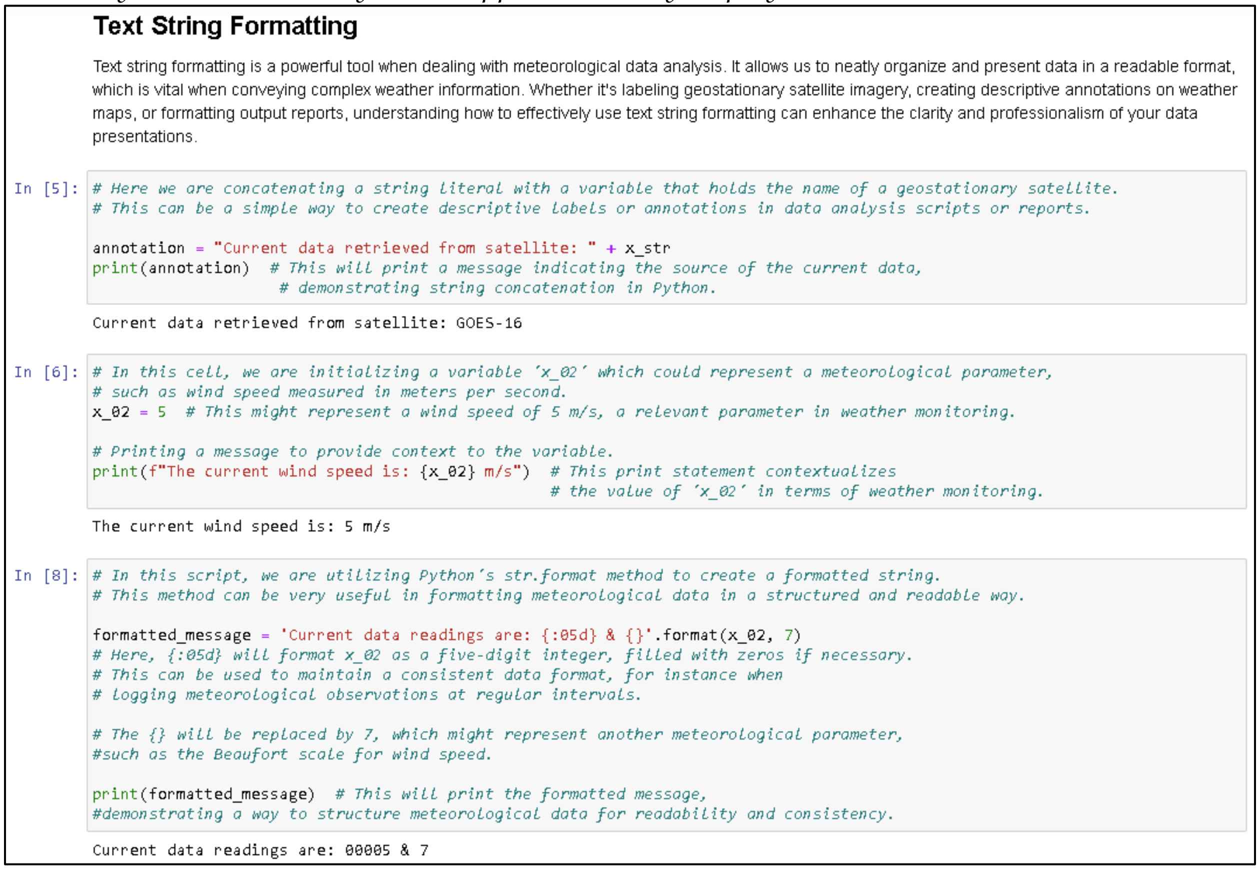


Strings

In the realm of weather monitoring through geostationary data,strings can hold vital information. They might represent satellite names, meteorological terms, annotations on weather maps, or descriptions in data metadata.

```
In [4]: # In weather monitoring, strings can be used to store various types of information.
        # Here, we're storing the name of a geostationary satellite.
        x_str = 'GOES-16'
        print(f"Currently accessing data from satellite: {x_str}")  # Using a formatted string to create a more
                                                                    # descriptive and contextual message

        Currently accessing data from satellite: GOES-16
```

8. *Execute the code examples to observe how different methods of string formatting work in Python and how they can be applied to neatly display weather-related data.*

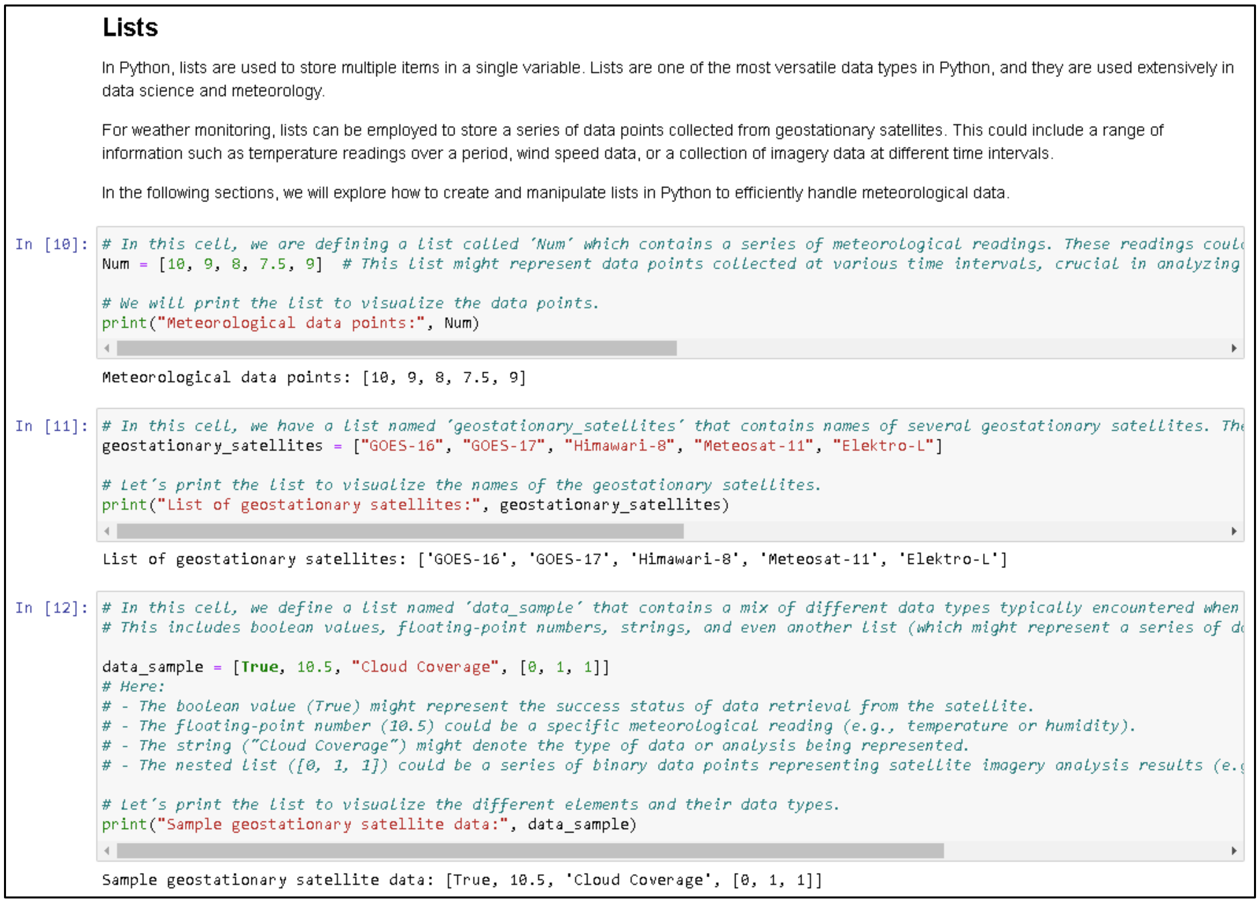


Text String Formatting

Text string formatting is a powerful tool when dealing with meteorological data analysis. It allows us to neatly organize and present data in a readable format, which is vital when conveying complex weather information. Whether it's labeling geostationary satellite imagery, creating descriptive annotations on weather maps, or formatting output reports, understanding how to effectively use text string formatting can enhance the clarity and professionalism of your data presentations.

```
In [5]: # Here we are concatenating a string literal with a variable that holds the name of a geostationary satellite.
        # This can be a simple way to create descriptive labels or annotations in data analysis scripts or reports.

        annotation = "Current data retrieved from satellite: " + x_str
        print(annotation)  # This will print a message indicating the source of the current data,
                           # demonstrating string concatenation in Python.

        Current data retrieved from satellite: GOES-16

In [6]: # In this cell, we are initializing a variable 'x_02' which could represent a meteorological parameter,
        # such as wind speed measured in meters per second.
        x_02 = 5  # This might represent a wind speed of 5 m/s, a relevant parameter in weather monitoring.

        # Printing a message to provide context to the variable.
        print(f"The current wind speed is: {x_02} m/s")  # This print statement contextualizes
                                                        # the value of 'x_02' in terms of weather monitoring.

        The current wind speed is: 5 m/s

In [8]: # In this script, we are utilizing Python's str.format method to create a formatted string.
        # This method can be very useful in formatting meteorological data in a structured and readable way.

        formatted_message = 'Current data readings are: {:05d} & {}'.format(x_02, 7)
        # Here, {:05d} will format x_02 as a five-digit integer, filled with zeros if necessary.
        # This can be used to maintain a consistent data format, for instance when
        # logging meteorological observations at regular intervals.

        # The {} will be replaced by 7, which might represent another meteorological parameter,
        #such as the Beaufort scale for wind speed.

        print(formatted_message)  # This will print the formatted message,
        #demonstrating a way to structure meteorological data for readability and consistency.

        Current data readings are: 00005 & 7
```

9. *Follow the examples provided to learn how lists are defined and used in Python for organizing diverse types of data, such as weather observations and satellite information. Pay attention to how lists can contain different data types and how this flexibility is advantageous in meteorological analysis.*


Lists

In Python, lists are used to store multiple items in a single variable. Lists are one of the most versatile data types in Python, and they are used extensively in data science and meteorology.

For weather monitoring, lists can be employed to store a series of data points collected from geostationary satellites. This could include a range of information such as temperature readings over a period, wind speed data, or a collection of imagery data at different time intervals.

In the following sections, we will explore how to create and manipulate lists in Python to efficiently handle meteorological data.

```
In [10]: # In this cell, we are defining a list called 'Num' which contains a series of meteorological readings. These readings coul
         Num = [10, 9, 8, 7.5, 9]  # This list might represent data points collected at various time intervals, crucial in analyzing

         # We will print the list to visualize the data points.
         print("Meteorological data points:", Num)
```

```
Meteorological data points: [10, 9, 8, 7.5, 9]
```

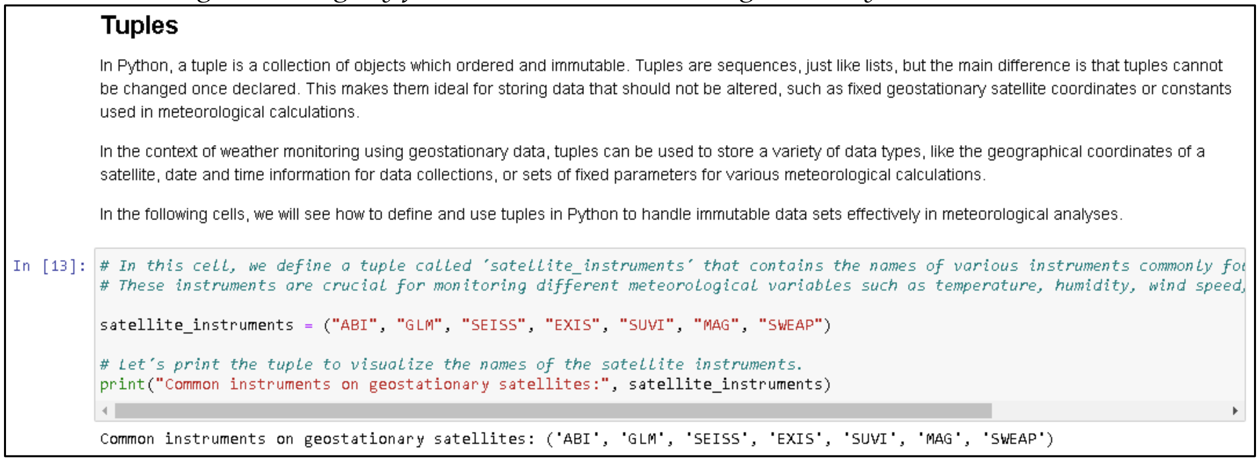
```
In [11]: # In this cell, we have a list named 'geostationary_satellites' that contains names of several geostationary satellites. The
         geostationary_satellites = ["GOES-16", "GOES-17", "Himawari-8", "Meteosat-11", "Elektro-L"]

         # Let's print the list to visualize the names of the geostationary satellites.
         print("List of geostationary satellites:", geostationary_satellites)
```

```
List of geostationary satellites: ['GOES-16', 'GOES-17', 'Himawari-8', 'Meteosat-11', 'Elektro-L']
```

```
In [12]: # In this cell, we define a list named 'data_sample' that contains a mix of different data types typically encountered when
         # This includes boolean values, floating-point numbers, strings, and even another list (which might represent a series of d

         data_sample = [True, 10.5, "Cloud Coverage", [0, 1, 1]]
         # Here:
         # - The boolean value (True) might represent the success status of data retrieval from the satellite.
         # - The floating-point number (10.5) could be a specific meteorological reading (e.g., temperature or humidity).
         # - The string ("Cloud Coverage") might denote the type of data or analysis being represented.
         # - The nested list ([0, 1, 1]) could be a series of binary data points representing satellite imagery analysis results (e.g

         # Let's print the list to visualize the different elements and their data types.
         print("Sample geostationary satellite data:", data_sample)
```

```
Sample geostationary satellite data: [True, 10.5, 'Cloud Coverage', [0, 1, 1]]
```

10. *For this tutorial on tuples in Python, we're focusing on how to create an immutable collection of items, which is useful for storing fixed sets of data such as instrument names on satellites. Observe how once defined, the contents of the tuple cannot be altered, ensuring data integrity for constants in meteorological analysis.*


Tuples

In Python, a tuple is a collection of objects which ordered and immutable. Tuples are sequences, just like lists, but the main difference is that tuples cannot be changed once declared. This makes them ideal for storing data that should not be altered, such as fixed geostationary satellite coordinates or constants used in meteorological calculations.

In the context of weather monitoring using geostationary data, tuples can be used to store a variety of data types, like the geographical coordinates of a satellite, date and time information for data collections, or sets of fixed parameters for various meteorological calculations.

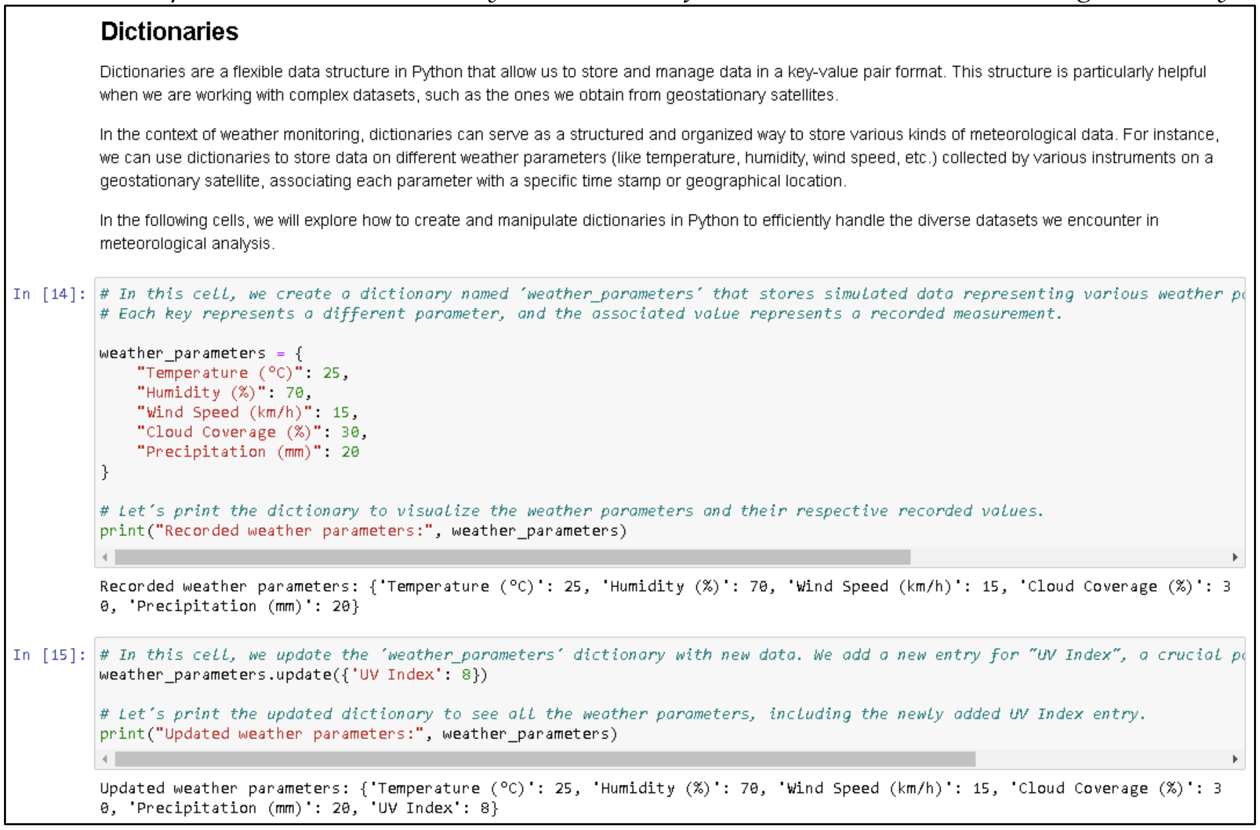
In the following cells, we will see how to define and use tuples in Python to handle immutable data sets effectively in meteorological analyses.

```
In [13]: # In this cell, we define a tuple called 'satellite_instruments' that contains the names of various instruments commonly fo
         # These instruments are crucial for monitoring different meteorological variables such as temperature, humidity, wind speed,

         satellite_instruments = ("ABI", "GLM", "SEISS", "EXIS", "SUVI", "MAG", "SWEAP")

         # Let's print the tuple to visualize the names of the satellite instruments.
         print("Common instruments on geostationary satellites:", satellite_instruments)
```

```
Common instruments on geostationary satellites: ('ABI', 'GLM', 'SEISS', 'EXIS', 'SUVI', 'MAG', 'SWEAP')
```

11. *This portion of the tutorial explains how to utilize dictionaries in Python to store and manipulate weather data parameters. Dictionaries allow us to create a structured dataset with key-value pairs, ideal for meteorological data management. Each key-value pair*

corresponds to a weather parameter and its measured value, making the data easy to access and update, as demonstrated by the addition of the 'UV Index' to the existing dictionary.



Dictionaries

Dictionaries are a flexible data structure in Python that allow us to store and manage data in a key-value pair format. This structure is particularly helpful when we are working with complex datasets, such as the ones we obtain from geostationary satellites.

In the context of weather monitoring, dictionaries can serve as a structured and organized way to store various kinds of meteorological data. For instance, we can use dictionaries to store data on different weather parameters (like temperature, humidity, wind speed, etc.) collected by various instruments on a geostationary satellite, associating each parameter with a specific time stamp or geographical location.

In the following cells, we will explore how to create and manipulate dictionaries in Python to efficiently handle the diverse datasets we encounter in meteorological analysis.

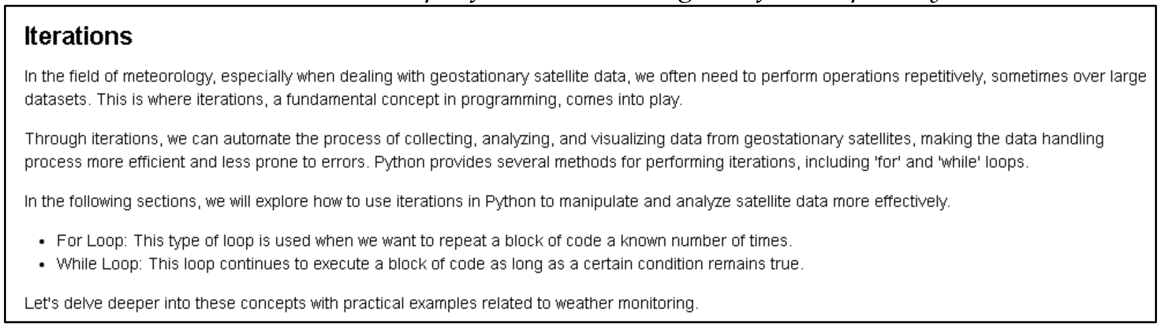
In [14]:
```
# In this cell, we create a dictionary named 'weather_parameters' that stores simulated data representing various weather pa
# Each key represents a different parameter, and the associated value represents a recorded measurement.

weather_parameters = {
    "Temperature (°C)": 25,
    "Humidity (%)": 70,
    "Wind Speed (km/h)": 15,
    "Cloud Coverage (%)": 30,
    "Precipitation (mm)": 20
}

# Let's print the dictionary to visualize the weather parameters and their respective recorded values.
print("Recorded weather parameters:", weather_parameters)
```

```
Recorded weather parameters: {'Temperature (°C)': 25, 'Humidity (%)': 70, 'Wind Speed (km/h)': 15, 'Cloud Coverage (%)': 3
0, 'Precipitation (mm)': 20}
```

In [15]:
```
# In this cell, we update the 'weather_parameters' dictionary with new data. We add a new entry for "UV Index", a crucial pa
weather_parameters.update({'UV Index': 8})

# Let's print the updated dictionary to see all the weather parameters, including the newly added UV Index entry.
print("Updated weather parameters:", weather_parameters)
```

```
Updated weather parameters: {'Temperature (°C)': 25, 'Humidity (%)': 70, 'Wind Speed (km/h)': 15, 'Cloud Coverage (%)': 3
0, 'Precipitation (mm)': 20, 'UV Index': 8}
```

12. *This section introduces the concept of iterations using the `for` loop in Python.*



Iterations

In the field of meteorology, especially when dealing with geostationary satellite data, we often need to perform operations repetitively, sometimes over large datasets. This is where iterations, a fundamental concept in programming, comes into play.

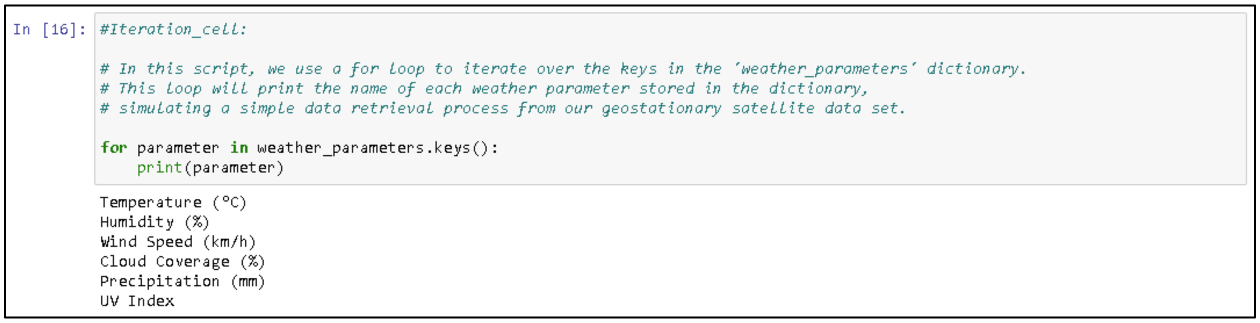
Through iterations, we can automate the process of collecting, analyzing, and visualizing data from geostationary satellites, making the data handling process more efficient and less prone to errors. Python provides several methods for performing iterations, including 'for' and 'while' loops.

In the following sections, we will explore how to use iterations in Python to manipulate and analyze satellite data more effectively.

- For Loop: This type of loop is used when we want to repeat a block of code a known number of times.
- While Loop: This loop continues to execute a block of code as long as a certain condition remains true.

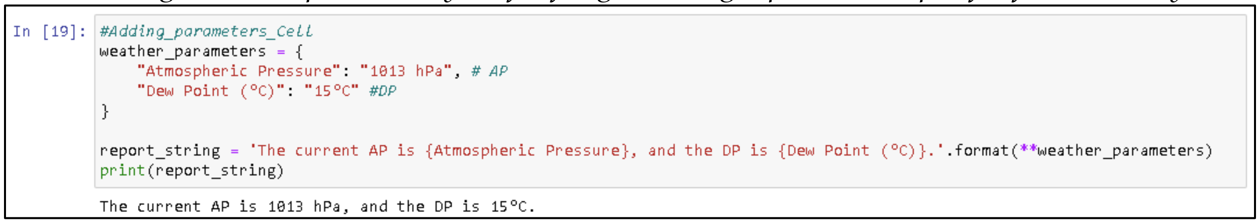
Let's delve deeper into these concepts with practical examples related to weather monitoring.

#Iteration_cell: *By iterating over the keys of the `weather_parameters` dictionary, we print out each weather parameter, demonstrating a simple yet effective way to enumerate the keys in a dictionary. This operation mimics a common task in data handling where listing all the parameters in a dataset is necessary.*



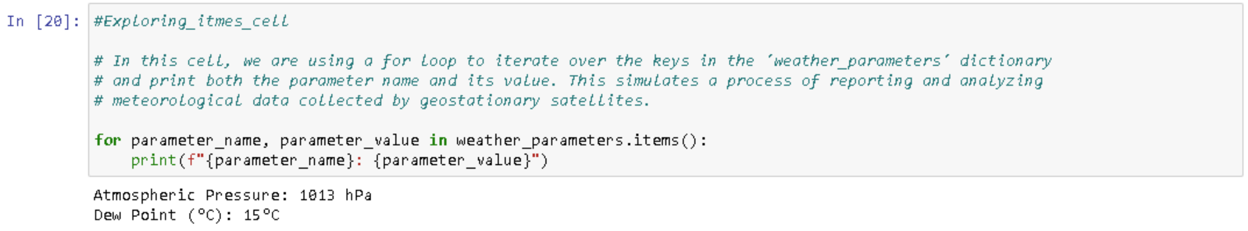
```
In [16]: #Iteration_cell:

         # In this script, we use a for loop to iterate over the keys in the 'weather_parameters' dictionary.
         # This loop will print the name of each weather parameter stored in the dictionary,
         # simulating a simple data retrieval process from our geostationary satellite data set.

         for parameter in weather_parameters.keys():
             print(parameter)

         Temperature (°C)
         Humidity (%)
         Wind Speed (km/h)
         Cloud Coverage (%)
         Precipitation (mm)
         UV Index
```

***#Adding_parameters_Cell**: Here we see how to enhance the `weather_parameters` dictionary by adding new entries. Specifically, we introduce 'Atmospheric Pressure' and 'Dew Point' with corresponding values. Additionally, the snippet showcases string formatting in Python by using the `format()` method. With this technique, we can construct a readable and informative sentence that incorporates our data directly into the string, which is particularly useful for generating reports or output for further analysis.*



```
In [19]: #Adding_parameters_Cell
         weather_parameters = {
             "Atmospheric Pressure": "1013 hPa", # AP
             "Dew Point (°C)": "15°C" #DP
         }

         report_string = 'The current AP is {Atmospheric Pressure}, and the DP is {Dew Point (°C)}.'.format(**weather_parameters)
         print(report_string)

         The current AP is 1013 hPa, and the DP is 15°C.
```

***#Exploring_itmes_cell:** In this code block, we continue to explore iterations with a `for` loop, this time iterating over both keys and values of the `weather_parameters` dictionary by utilizing the `.items()` method. This approach is crucial when one needs to process both the name (key) and the recorded measurement (value) of each weather parameter. The loop prints out each parameter and its corresponding value in a format that is suitable for reviewing collected meteorological data or preparing it for presentation or further computation.*



```
In [20]: #Exploring_itmes_cell

         # In this cell, we are using a for loop to iterate over the keys in the 'weather_parameters' dictionary
         # and print both the parameter name and its value. This simulates a process of reporting and analyzing
         # meteorological data collected by geostationary satellites.

         for parameter_name, parameter_value in weather_parameters.items():
             print(f"{parameter_name}: {parameter_value}")

         Atmospheric Pressure: 1013 hPa
         Dew Point (°C): 15°C
```

Ok, you have completed the tutorial!

Turn-in

For this assignment, you will be demonstrating your skills in manipulating and presenting meteorological data using Python dictionaries and iterations. Please follow the steps outlined below to complete your task and submit your work:

1. Add a new weather parameter to the `weather_parameters` dictionary:

- Choose a weather parameter that is commonly monitored by geostationary satellites (for example, Solar Radiation, Visibility, etc.).
- Insert the new weather parameter into the dictionary with a hypothetical value.
- Print the updated dictionary to verify the addition of the new parameter.

2. Create a formatted output using an f-string:

- Craft a message using a formatted string literal (f-string) to include dynamic data from the `weather_parameters` dictionary.
- Your message should be clear, informing the reader about one of the parameters and its value, mimicking a real-world data report.

3. Prepare your submission documents:

- Generate a PDF document that captures the final state of the `weather_parameters` dictionary, including the new parameter you added.
- The document should also display the formatted message you created with the f-string.
- Additionally, create a "readme" or instruction file that outlines the steps you followed to complete the assignment. This document should serve as a guide for anyone reviewing your code or results.

4. Upload your documents to the designated submission platform:

- Your submission should include the following:
- The PDF document with your updated `weather_parameters` dictionary and formatted message.
- The original Python script used to produce these results.
- The "readme" or instruction file detailing the steps of your process.
- Name your submission as "Weather Data Analysis for [Your Chosen Parameter]" to reflect the content of your work.
- In the description or abstract, provide a brief overview of the contents of your submission, emphasizing the addition of the new weather parameter and the use of f-strings in your analysis.

Ensure that all files are correctly formatted, legible, and free of errors before submission. If you encounter any issues or have questions about the assignment, please reach out to your instructor or teaching assistant for guidance.