6.2.6

Get the City Weather Data

You and Jack have a quick catch-up over coffee the next morning, and he can hardly believe his eyes. You have already learned how to use APIs to retrieve data from a JSON file. As you look at the project plan, you realize it's now time for an even bigger challenge: retrieving the weather data from 500+ cities.

You and Jack know this weather data will be able to enhance our client's search capabilities and hopefully draw more clients to your website. This is a huge step for you, but you know you can build on what you've learned thus far to pull it off.

Let's use pseudocode to map out, at a high level, how we will get the weather data for each city for the website.

We will need to do the following:

1. Import our dependencies and initialize counters and an empty list that will hold the weather data.
2. Loop through the cities list.
3. Group the cities in sets of 50 to log the process as we find the weather data for each city.
    - Two counters will be needed here: one to log the city count from 1 to 50, and another for the sets.
4. Build the `city_url` or endpoint for each city.
5. Log the URL and the record and set numbers.
6. Make an API request for each city.
7. Parse the JSON weather data for the following:
    - City, country, and date
    - Latitude and longitude
    - Maximum temperature
    - Humidity
    - Cloudiness
    - Wind speed

8. Add the data to a list in a dictionary format and then convert the list to a DataFrame.

Import Dependencies, and Initialize an Empty List and Counters

At the top of our code block, we are going to declare an empty list, `city_data = []`; add a print statement that references the beginning of the logging; and create counters for the record numbers, 1–50; and the set counter.

We will now work in our WeatherPy.ipynb file. Before continuing, make sure the following tasks are completed:

- Import your Requests Library and the weather_api_key.
- Build the basic URL for the OpenWeatherMap with your weather_api_key added to the URL.

Also, import the time library, as well as the datetime module using the following code:

```
# Import the time library and the datetime module from the datetime libra
import time
from datetime import datetime
```

Next, add the following code to a new cell, but don't run the cell. Instead, continue to add on to this code block.

```
# Create an empty list to hold the weather data.
city_data = []
# Print the beginning of the logging.
print("Beginning Data Retrieval     ")
print("-----------------------------")

# Create counters.
record_count = 1
set_count = 1
```

In the code block, we have initialized the counters at 1 because we want the first iteration of the logging for each recorded response and the set to start at 1.

Loop Through the List of Cities and Build the City URL

Next, we need to iterate through our list of cities and begin building the URL for each city, while grouping our records in sets of 50. To do this, use `for i in range(len(cities))` and the index to tell us when we get to 50. Once we get to 50, we tell the program to pause for 60 seconds using the `time.sleep(60)` command. The OpenWeatherMap API only allows 60 calls for per minute on their free tier, so pausing our program for one minute after each set of 50 will prevent time-out errors. We can also retrieve the city from the `cities` list and add it to the `city_url` by using indexing, as shown in the following code:

```
# Loop through all the cities in our list.
for i in range(len(cities)):

    # Group cities in sets of 50 for logging purposes.
    if (i % 50 == 0 and i >= 50):
        set_count += 1
        record_count = 1
        time.sleep(60)

    # Create endpoint URL with each city.
    city_url = url + "&q=" + cities[i]
```

Every time we want to reference the city in our code, we need to use the indexing on the `cities` list. Unfortunately, this will cause programming errors when we are building the `city_url` because it adds the index, not the city name, to the `city_url`. To fix this issue, we need to create another `for` loop to get the city from the `cities` list.

Instead of using two `for` loops, we can use the `enumerate()` method as an alternative way to iterate through the list of cities and retrieve both the index, and the city from the list. The syntax for the `enumerate()` method is the following:

```
for i, item in enumerate(list):
```

Let's use the `enumerate()` method to get the index of the city for logging purposes and the city for creating an endpoint URL. Add the following code below our counters **but don't run it just yet**:

```
# Loop through all the cities in the list.
for i, city in enumerate(cities):

    # Group cities in sets of 50 for logging purposes.
    if (i % 50 == 0 and i >= 50):
        set_count += 1
        record_count = 1
        time.sleep(60)

    # Create endpoint URL with each city.
    city_url = url + "&q=" + city.replace(" ","+")

    # Log the URL, record, and set numbers and the city.
    print(f"Processing Record {record_count} of Set {set_count} | {city}"
    # Add 1 to the record count.
    record_count += 1
```

Let's break down the code so we understand fully before continuing:

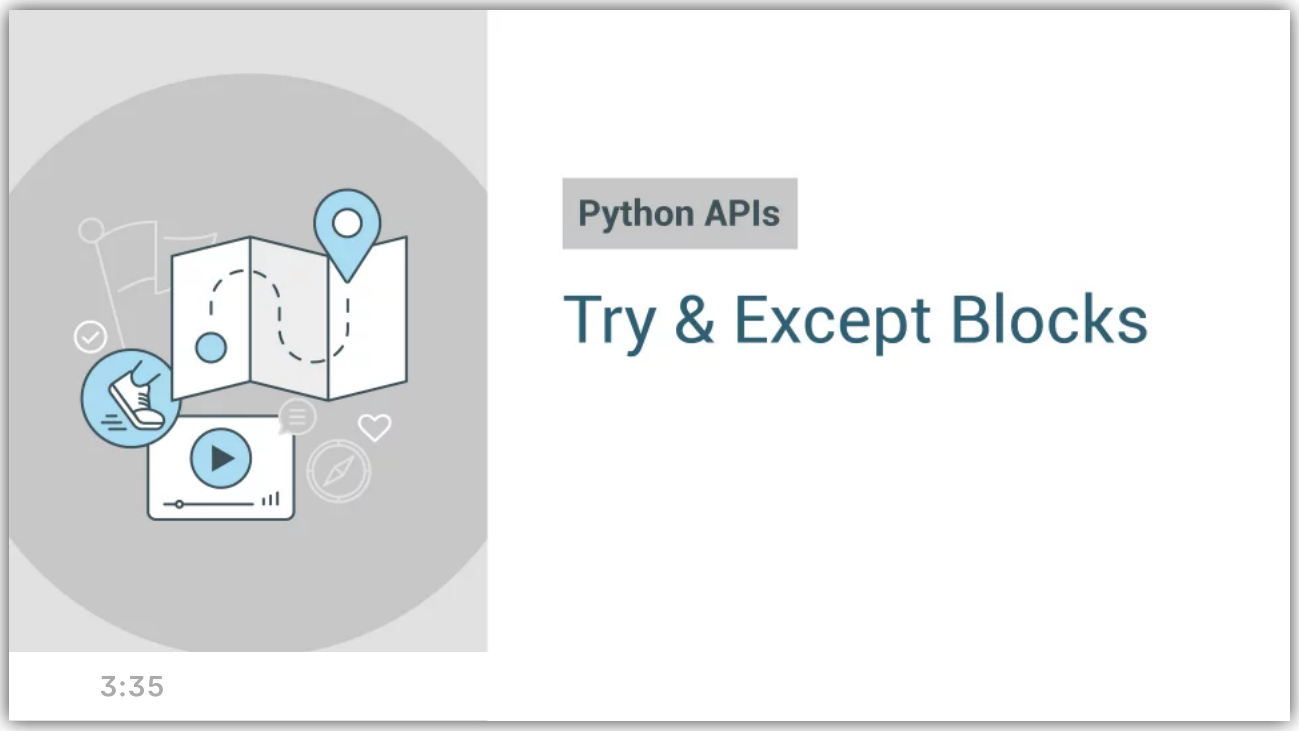
- We create the `for` loop with the `enumerate()` method and reference the index and the city in the list.
- In the conditional statement, we check if the remainder of the index divided by 50 is equal to 0 and if the index is greater than or equal to 50. If the statement is true, then the `set_count` and the `record_count` are incremented by 1.
- Inside the conditional statement, we create the URL endpoint for each city, as before. However, we are removing the blank spaces in the city name and concatenating the city name with, `city.replace(" ","+")`. This will find the corresponding weather data for the city instead of finding the weather data for the first part of the city name.
- Also, we add a print statement that tells us the record count and set count, and the city that is being processed.
- Then we add one to the record count before the next city is processed.

Next, we will retrieve the data from the JSON weather response for each city.

NOTE

When retrieving data from an API, or even when scraping a webpage, make sure there is data to parse. If not, the script might stop at that moment and not finish getting all the data we need.

Handle API Request Errors with try-except Blocks



We have handled request errors for getting the response from a valid city with an API call using conditional statements. Now we'll learn how to handle errors while parsing weather data from a JSON file.

We'll add a `try-except` block to our code to prevent the API request from stopping prematurely if the `city_weather` request isn't a valid response. If the request isn't valid, the code will not find the first item requested, which is the dictionary `"coord"` with the code `city_lat = city_weather["coord"]["lat"],` and skip the city and continue to run.

The `try-except` block has similar syntax and structure as the if-else statement. The basic format is as follows:

```
try:
  Do something
```

```
except:
  print("An exception occurred")
```

We can add a `try-except` block to our code and, below the `try` block, we will parse the data from the JSON file and add the data to the cities list.

Let's add a `try` block. Then, below the `try` block, do the following:

1. Parse the JSON file.
2. Assign variables for each piece of data we need.
3. Add the data to the cities list in a dictionary format.

Add the following code after `record_count += 1`.

```
# Run an API request for each of the cities.
    try:
        # Parse the JSON and retrieve data.
        city_weather = requests.get(city_url).json()
        # Parse out the needed data.
        city_lat = city_weather["coord"]["lat"]
        city_lng = city_weather["coord"]["lon"]
        city_max_temp = city_weather["main"]["temp_max"]
        city_humidity = city_weather["main"]["humidity"]
        city_clouds = city_weather["clouds"]["all"]
        city_wind = city_weather["wind"]["speed"]
        city_country = city_weather["sys"]["country"]
        # Convert the date to ISO standard.
        city_date = datetime.utcfromtimestamp(city_weather["dt"]).strftim
        # Append the city information into city_data list.
        city_data.append({"City": city.title(),
                          "Lat": city_lat,
                          "Lng": city_lng,
                          "Max Temp": city_max_temp,
                          "Humidity": city_humidity,
                          "Cloudiness": city_clouds,
```

```
                                "Wind Speed": city_wind,
                                "Country": city_country,
                                "Date": city_date})

# If an error is experienced, skip the city.
    except:
        print("City not found. Skipping...")
        pass

# Indicate that Data Loading is complete.
print("-----------------------------")
print("Data Retrieval Complete      ")
print("-----------------------------")
```

Let's review the code:

- We parse the JSON file for the current city.
  - If there is no weather data for the city, i.e., a `<Response [404]>` then there is no weather to retrieve and `City not found. Skipping...` is printed.
- If there is weather data for the city, we will retrieve the latitude, longitude, maximum temperature, humidity, cloudiness, wind speed, and date and assign those values to variables.
  - We could write a `try-except` block for each one of these parameters to handle the KeyError if the data wasn't found, but since these parameters are always present in the response this won't be necessary.
- We append the cities list with a dictionary for that city, where the key-value pairs are the values from our weather parameters.
- Finally, below the `try` block and after the `except` block, we add the closing print statement, which will let us know the data retrieval has been completed. Make sure that your `except` block is indented and in line with the `try` block, and that the print statements are flush with the margin.
- Under the print statement in the `except` block, we add the `pass` statement, which is a general purpose statement to handle all errors encountered and to allow the program to continue.

IMPORTANT

Generally, it isn't good coding practice to add the `pass` statement to the `except` block. Ideally, we want to handle or catch each error as it happens and do something specific (e.g., add another `try` block or print out

the error).

Now you have all your code to perform the API calls for each city and parse the JSON data. Let's run the cell!

As your code is running, your output should be similar to the following image:

```
Beginning Data Retrieval
-----------------------------
Processing Record 1 of Set 1 | tuktoyaktuk
Processing Record 2 of Set 1 | hermanus
Processing Record 3 of Set 1 | bluff
Processing Record 4 of Set 1 | port alfred
Processing Record 5 of Set 1 | outjo
Processing Record 6 of Set 1 | agadez
Processing Record 7 of Set 1 | paso de los toros
Processing Record 8 of Set 1 | avarua
Processing Record 9 of Set 1 | mancio lima
City not found. Skipping...
Processing Record 10 of Set 1 | taltal
Processing Record 11 of Set 1 | airai
Processing Record 12 of Set 1 | samusu
City not found. Skipping...
```

After collecting all our data, we can tally the number of cities in the `city_data` array of dictionaries using the `len()` function.

IMPORTANT

If you didn't get more than 500 cities, run the code to generate random latitude and longitude combinations and all the code below it. Or increase the size of the latitude and longitude combinations.

NOTE

For more information about the `try-except` blocks, see the **documentation on errors and exceptions (https://docs.python.org/3.7/tutorial/errors.html)** .