9.5.3 Precipitation Route

The next route will return the precipitation data for the last year. We've kept W. Avy in the loop while we've been coding, of course, but by building this route he'll be able to access this analysis in real time with just a URL.

The next route we'll build is for the precipitation analysis. This route will occur separately from the welcome route.

CAUTION

> Every time you create a new route, your code should be aligned to the left in order to avoid errors.

To create the route, add the following code. Make sure that it's aligned all the way to the left.

```
@app.route("/api/v1.0/precipitation")
```

Next, we will create the `precipitation()` function.

```
def precipitation():
    return
```

Now we can add code to the function. This code may look almost identical to code you've written previously, but now we'll dive deeper into our precipitation analysis and figure out how to best integrate it into our application.

First, we want to add the line of code that calculates the date one year ago from the most recent date in the database. Do this now so that your code looks like the following:

```
def precipitation():
   prev_year = dt.date(2017, 8, 23) - dt.timedelta(days=365)
   return
```

Next, write a query to get the date and precipitation for the previous year. Add this query to your existing code.

```
def precipitation():
   prev_year = dt.date(2017, 8, 23) - dt.timedelta(days=365)
   precipitation = session.query(Measurement.date, Measurement.prcp).\
      filter(Measurement.date >= prev_year).all()
   return
```

SHOW HINT

Finally, we'll create a dictionary with the date as the key and the precipitation as the value. To do this, we will "jsonify" our dictionary. `Jsonify()` is a function that converts the dictionary to a JSON file.

REWIND

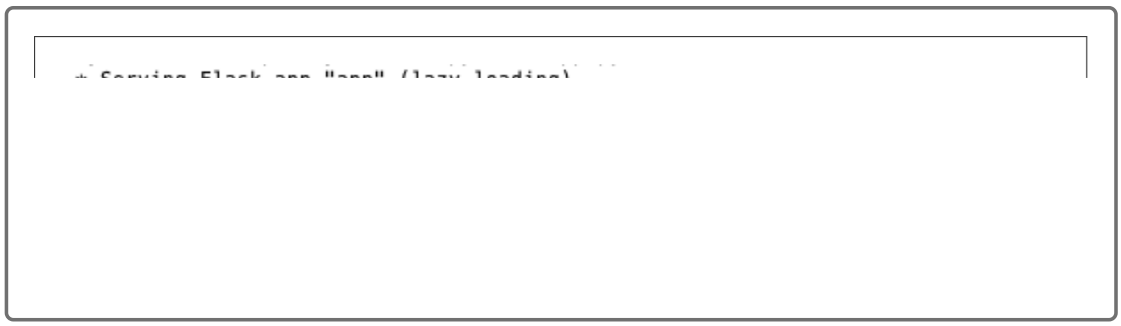
JSON files are structured text files with attribute-value pairs and array data types. They have a variety of purposes, especially when downloading information from the internet through API calls. We can also use JSON files for cleaning, filtering, sorting, and visualizing data, among many other tasks. When we are done modifying that data, we can push the data back to a web interface, like Flask.

We'll use `jsonify()` to format our results into a JSON structured file. When we run this code, we'll see what the JSON file structure looks like. Here's an example of what a JSON file might look like:

```
{
"city" : {
"name" : "des moines",
        "region" : "midwest"
}
```

```
def precipitation():
   prev_year = dt.date(2017, 8, 23) - dt.timedelta(days=365)
   precipitation = session.query(Measurement.date, Measurement.prcp).\
    filter(Measurement.date >= prev_year).all()
   precip = {date: prcp for date, prcp in precipitation}
   return jsonify(precip)
```

Our second route is defined! Now let's run our code. If your application is still running from the previous route, be sure to either close the window or quit the Flask application. Once you run your code, your output will look like the below. As a reminder, you can copy `http://127.0.0.1:5000/` into your web browser to see the results of your code. If you see this window, that means that you've properly set up your Flask application.



At this point, you might be wondering where to view the output of your code. Just like we did previously, copy the web address (`http://127.0.0.1:5000/`) into our web browser. You will need to navigate to the precipitation route in order to see the output of your code. You can do this by adding `api/v1.0/precipitation` to the end of the web address. This is what your output should look like:

Good work! You have your first route with code logic and are now prepared to build the other routes. The next route we'll turn to is the stations route.