Unit 3 Lab: Object-Oriented Programming

Overview

Welcome to the Unit 3 lab! Now that we've learned some new data structures, let's put them to use in our weather forecasting application.

Goals

In this lab, you will:

- Write and use complex data types to organize weather data efficiently.
- Write classes and methods to calculate and display the weather forecast.

Restructuring Data

Suppose we want to offer the user the opportunity to select the temperature scale in which they view their weather forecast. Create a variable, `temperature_scale`, that is a `set` with the following strings:

- "Fahrenheit"
- "Celsius"
- "Kelvin"

Print the variable `temperature_scale`.

In [ ]:
```
# Enter your code below:
```

Weather data is typically taken and recorded every hour. Let's combine the following lists into a single, table-like structure of data we can review.

Create a list called `weather_data`. This list will be composed of dictionary objects for each hour over a 24-hour span. Each of the dictionaries should have the following four keys: `"hour"`, `"temperature"`, `"humidity"`, and `"rainfall"`. Begin counting at hour `0` and end at hour `23`.

Then, iterate through `weather_data` and print out each dictionary — one dictionary per line.

In [ ]:
```
one_day_of_hourly_temperatures = [67,67,68,69,71,73,75,
one_day_of_hourly_humidity = [60,65,65,70,70,70,70,75,7
one_day_of_hourly_rainfall = [0,0,0,0.1,0.1,0.05,0.1,0.

# Enter your code below:
```

Using your `weather_data` list, print the following with one line of code for each `print` statement:

- Temperature at 2 p.m.
- Humidity at 11 p.m.
- Rainfall at 9 a.m.

In [ ]:
```
# Enter your code below:
```

Creating a `Forecast` Class

Start by creating a `Forecast` class that accepts one argument, `location`, and assigns it as an instance attribute to `self.location`.

In [ ]:
```
# Enter your code below:
```

Calculation Methods

Create two methods within the `Forecast` class:

- `get_daily_high()`
- `get_daily_low()`

Use the built-in `max()` and `min()` functions on `one_day_of_hourly_temperatures` to return the daily high and low temperatures.

Create a third method within the `Forecast` class called `get_daily_chance_of_rain()`. This method should:

- Create a variable named `number_of_years_of_data` and set it to `10`.
- Create a variable named `times_it_has_rained` and set it to `0`.
- Calculate the sum of rainfall for all 24 hours of `one_day_of_hourly_temperatures`.

- If the sum of all 24 hours is greater than 0, increase `times_it_has_rained` by 1.
- Convert `times_it_has_rained` to a percentage by dividing it by `number_of_years_of_data` and multiplying it by 100.
- Return the final value.

Note: We'll modify these three methods to process live data in future labs. For now, we're setting up the basic logic on test data.

In [ ]:
```
# Copy and paste the Forecast class you built in the pr
```

Run the following cell after you're finished to test your output:

In [ ]:
```
test = Forecast("Austin,TX")
print("High:", test.get_daily_high())
print("Low:", test.get_daily_low())
print("Chance of Rain:", test.get_daily_chance_of_rain(
```

Display Methods

Back in Lab 1, we printed `"The weather forecast for today is: High of 85, low of 69, with a 15.0% chance of precipitation."` using the four following variables and the values we assigned to them:

- `message_to_user`
- `todays_high`
- `todays_low`
- `chance_of_precipitation`

Now, we're going to display this same message but with the calculated values from the methods we just created.

Create a `display_daily_forecast()` method within the `Forecast` class that, when called, will print out a string matching the one above.

This method will insert the high temperature, low temperature, and chance of rain by calling the following internal class methods, respectively:

- `get_daily_high()`
- `get_daily_low()`
- `get_daily_chance_of_rain()`

Next, create a `display_weekly_forecast()` method within the `Forecast` class:

- When called, this method will print out a message in the format

shown in the code block below.
- *Note: We only have one day of data we're currently using, so `High`, `Low`, and `Rain` will all be the same for now.*

```
This week's weather forecast:
    Monday:    High 82, Low 65, Rain 10.0%
    Tuesday:   High 82, Low 65, Rain 10.0%
    Wednesday: High 82, Low 65, Rain 10.0%
    Thursday:  High 82, Low 65, Rain 10.0%
    Friday:    High 82, Low 65, Rain 10.0%
    Saturday:  High 82, Low 65, Rain 10.0%
    Sunday:    High 82, Low 65, Rain 10.0%
```

Hint: Use `\n` to create new lines in your `print` statement (i.e., `print("First line\nSecond line")`).

Hint: Use `\t` to simulate a tab character in your `print` statement (i.e., `print("First Column\tSecond Column")`.

Bonus: Convert class methods that are internally called to private methods via the use of the `'underscore'`, `__method_name`.

In [ ]:
```
# Copy and paste the Forecast class you built in the pr
```

Run the following cell after you're finished to test your output:

In [ ]:
```
test = Forecast("Austin,TX")
test.display_daily_forecast()
test.display_weekly_forecast()
```

Nice work!