Project 1 - Command Line Weather

Isn't it awful when you're in *the zone*, and forget to check the weather? Whenever you finally finish that homework, and forget to put the correct clothing on? Probably not!

Your first project will be building a command line weather application utilizing **bash scripting, API calls, automating with cronjobs, and more!**

Project Goals/Outcomes

This project will cover multiple bases, so hang on tight.

By finishing this project, learners will be able to:

1. Create bash scripts utilizing conditional statements, such as *if statements* and *loops*.
2. Access data online using `curl` calls.
3. Schedule jobs to be run over time utilizing `cron`.
4. Parse through JSON results using `jq`.
5. Utilize the `date` command to grab the current time.
6. "Typecast" bash datatypes (we are going to bully the system.)

What to Submit?

- A ZIP file containing your bash script, named
- A screenshot of your crontab

Part 1: Setting up the API

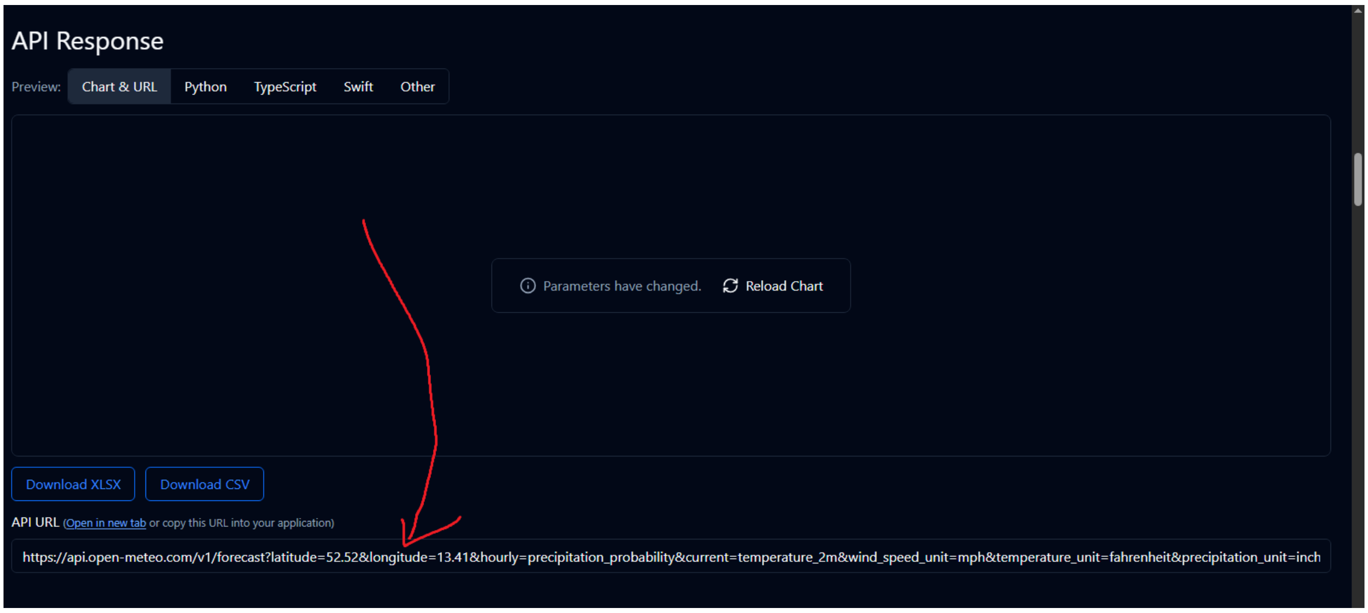
With this assignment, we will be using one of my favorite (read: free) APIs. Open Meteo!

The reason I pick this API also has to do with the readability of the result. Other APIs return ugly results, where this is digestible!

For the **required** part of the assignment, you'll need to select two items: *hourly precipitation probability* and *current temperature_2m*. Be sure to select the 3 day range option.

I would also prefer imperial units, but if your heart so decides I cannot stop metric units.

You'll need to grab the API url that is below the chart. Mine was about a third of the way down, and I've attached an image showing it.



```
https://api.open-meteo.com/v1/forecast?latitude=52.52&longitude=13.41&hourly=precipitation_probability&current=temperature_2m&wind_speed_unit=mph&temperature_unit=fahrenheit&precipitation_unit=inch
```

Go above and beyond if you want. The sky is the limit.

Part 2: Output

Required output:

1. Time and AM/PM
2. Temperature, average chance to rain in the next 3 hours, and City (I hardcoded the city, but if you want it to be location modular, check out `curl ipinfo.io`.
3. Minimum of three different outputs. I did average precipitation > 50, current temperature > 65, and a default statement. **Yours does not have to match mine.**

Output 1:

```
16:11 PM
It is currently 87F in Cookeville, TN, with a 0% chance to rain in the next three hours.
My recommendation: Shorts!
```

```
8:11 AM
It is currently 62F in Cookeville, TN, with a 12% chance to rain in the next three hours.
My recommendation: Jacket!
```

```
8:11 AM
It is currently 71F in Cookeville, TN, with a 60% chance to rain in the next three hours.
My recommendation: Umbrella!
```

Cron

Believe it or not, you can schedule bash scripts to run at specific times!

You'll need to setup your script to run *once a day, everyday,* at 4:30pm.

To access your cron jobs, run `crontab -e`

Part 4: Tips and Tricks

This section will hold the parts that I had to Google profusely.

Cron Output Doesn't Show Up!

Cron jobs run in their own shell, so any output does not print to the screen. You'll need to send the output (>) to a file in your project directory.

Cron is also very very picky with paths. First, run `which bash`. Assuming you're trying to run `./weather.sh`, you'll need to swap it to `{output of which bash} {full path to script}`.

To test your cron job, I will suggest setting it to run every two minutes. In the minutes part of your crontab, you'll specify `*/2`.

Changing a string (holding a number) to an int

Bash does not inherently support floating point numbers. Unfortunately, our API *returns* floating point numbers as strings (what I can assume, in our type agnostic language).

While I have only taught `echo`, bash also comes with `printf`, and works very similar to C's `printf`. Hopefully this sets you down the correct path.

Math in Bash

If you go into a bash script, and try to `echo 5+5`, your script will act just like a teenager, and give you `5+5` as the output. If we want to do math in Bash, you'll have to wrap it in $(( )).

Example:

```
result=$((5+5))
echo $result

10
```

JSON in Bash

You all can thank Mr. Burchfield for showing me `jq`, as I was planning on being evil and make you all use `awk`.

Before we can use `jq`, we *probably* need to install it. (Don't type the dollar sign, that's representative of your command line).

```
$ sudo apt update
$ sudo apt install jq
```

Now that we have it, I'll give a baby crash course on it. Go ahead and save the JSON below to a file, so we can play with it.

```
filename: output.json
{
  "firstName": "Jane",
  "lastName": "Doe",
  "address": {
    "city": "Denver",
    "state": "CO"
  },
  "friends": [
    {"firstName": "Tom", "lastName": "Jackson"},
    {"firstName": "Linda", "lastName": "Garner"}
  ]
}
```

We will be using the pipe command a **lot** with `jq`.

To pretty-print all of the JSON, we can just use a `.`

```
$ cat output.json | jq .

{
  "firstName": "Jane",
  "lastName": "Doe",
  "address": {
    "city": "Denver",
    "state": "CO"
  },
  "friends": [
    {
      "firstName": "Tom",
      "lastName": "Jackson"
    },
    {
      "firstName": "Linda",
      "lastName": "Garner"
    }
  ]
}
```

To access a certain key, we can specify it by using it's label:

```
$ cat output.json | jq.firstName
"Jane"
```

We can also use this to grab nested data:

```
$ cat output.json | jq .address.city
"Denver"
```

Finally, we can specify indexes like normal arrays:

```
$ cat output.json | jq .friends[0].firstName
"Tom"
```