Predicting daily local weather information using recorded historical climate data from NOAA’s NCDC

by: Malek Necibi

mnygp – 14340854

Topic/Research:

Using publicly available climate data measured daily between 1981-2010 across the United States from the National Oceanic and Atmospheric Administration (NOAA) to predict the average weather on a given date of the year.

The most advanced modern weather prediction models used by the world’s top meteorologists are only capable of predicting at most 10 days into the future with a reliability above 50%, yet individuals and businesses often need to make plans months or years in advance. To remedy this problem, planners today generally use empirical reasoning to determine the likely weather on a given day, then make plans accordingly. This method is both inefficient and inaccurate due to 2 main issues.

The first problem is the empirical nature of this decision process. To simplify the problem of determining the weather on a given date in the future, people often use the seasons to estimate when it will be a certain temperature. This in turn can lead to an unnecessary delay for businesses, meaning time and money wasted. For example, let’s say a large business was planning to construct a building, and the materials of this construction project require an ambient temperature of at least 60 degrees to set correctly. A human would likely default to waiting for the summer to begin the project, as this would ensure a warm enough temperature, so they set a date of June 20th. The cost of renting equipment to complete the project is higher during the summer, due to an influx of demand from other renters. Had the company checked average daily weather information, they would have seen that the historical minimum temperature in Columbia, MO rises to above 60 degrees on May 31st. This means they could have started the project 4 weeks earlier, saving significant time, as well as rented the equipment at a discount. Now let’s say the project leaders knew they could save time and money by pulling back the starting date of construction. Without a program like this to see real-life data, they run the risk of pulling the date too early, thus ruining the temperature-sensitive materials and setting back the project even further.

The second problem with our current human-centered weather approximation methods is how region-specific our knowledge is. A worker based in Maine has no gauge of the climate patterns in Arizona, so in this growingly interconnected world we now live in, it’s important we understand the climate in other places by using real data.

One potential solution to this problem was derived back in 1725 with the publication of the first Annual Almanac. The Almanac provided farmers with a means of predicting the weather into the future, which allowed them to sow their seeds at the time best for their future harvests. While this idea was well-founded, and annual book sales for the Almanac skyrocketed with just a few short years, the Almanac authors suffered from a lack of measured data to predict future weather patterns, meaning the Almanac was often incorrect. A 5-year study verifying the accuracy of *The Old Farmers Almanac*, conducted by John Walsh at the University of Illinois, found that their monthly temperature and precipitation forecasts were just 50.7% accurate. This hints that the old techniques were almost completely equivalent to the 50% success rate expected by basic chance. Using real data collected over a period of 30 years, I believe the NOAA data can be used to predict far more accurate predictions of likely climate trends.

For these reasons, I created a bash script that uses real historical data measured by the NOAA National Climate Data Center (NCDC) to approximate the weather conditions on a given date. This data was released by the NCDC through ‘Products*’* known as ‘Normals’ which each contain data about one of 7,500 ‘Stations’ posted across the nation. Between 1981 to 2010, the majority of these stations collected data daily which included average/minimum/maximum temperatures, average cloud coverage, snow coverage percentages, and much more. For the sake of this program, I opted to just display temperature data to keep the script more focused.

In the program, I used several of the techniques discussed in class, including variables, if statements, for and while loops, and commands such as sed, awk, grep, read, head, wget, and much more. I also used regular expressions on various occasions when parsing and searching for data, and it allowed me to greatly simplify many of the challenges brought forth by this problem. In addition to the topics from class, I performed my own research and learned how to manipulate and concatenate strings, how to measure the length of a string, and performed mathematical operations both within bash and using the bc command. I dove deep into functions as well, where I learned how to create functions, how to pass data to functions, how to pseudo-return data from a function, and how to use recursion to simplify otherwise repetitive and lengthy code.

Application:

The provided script file ‘predictWeather.sh’ can be run in a Bash system (tested on Red Hat Enterprise Linux 8 64-bit). The program begins by creating a directory named ‘.TempWeather’, to which it downloads 3 files containing station data from the NOAA NCDC.

Once downloaded, it prompts the user for a zip code, where it then checks for stations matching said zip code. If a station is not found (which is not uncommon), it checks for zip codes that begin with the same leading 4 digits, as those zip codes are generally near to the desired target.

Now that nearby stations have been found, the program displays these stations to the user and prompts them to select from which station they wish to acquire their weather data. The program then downloads the specific, relevant data from the NOAA NCDC. While the weather data from all the stations are available in a singular file, this file is over 2 GB in size, so I’ve opted to only download the relevant data from the desired station to optimize for time, network bandwidth, and storage space.

Next, the user is prompted for the date they want to check. If they would like to view info about today’s date, they can input 0 when prompted for the month. Otherwise, they can input any date of the year. Currently, the month and day input doesn’t have any checks for whether a valid input was provided, as the NCDC data has special checks included, so I’ve opted to use their data as the method of checking for a valid date.

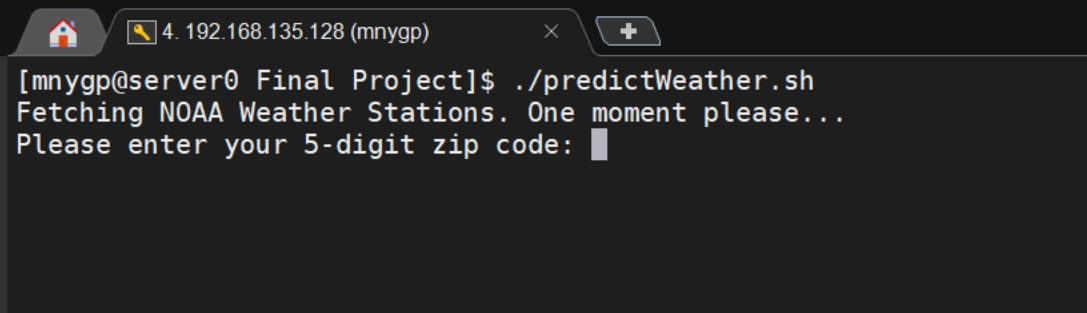
Special Notes:

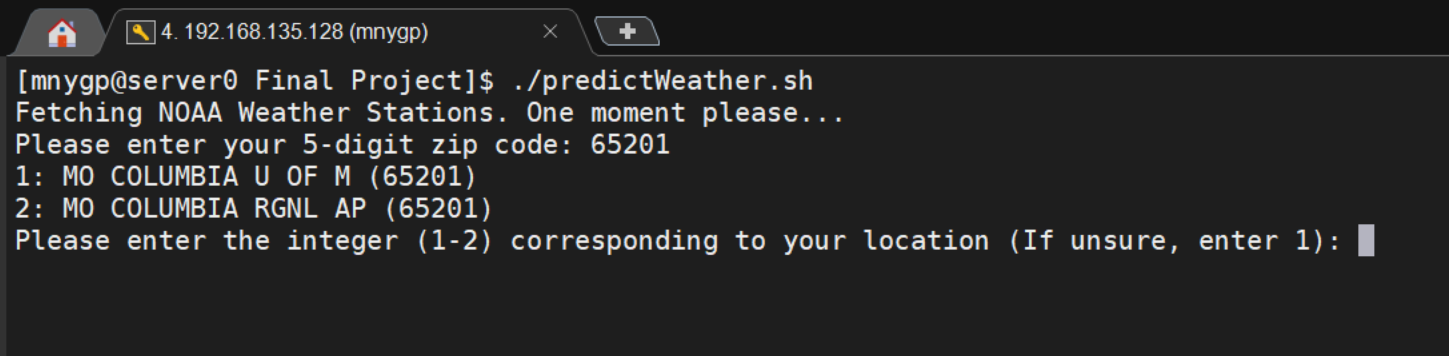
I’ve included the ability to pass in zip code(s) as an argument. When this is done, the program automatically uses data from the first station matching the zip code, as well as uses the current date. I chose to implement this because the majority of the time when a script is called with command-line arguments, it is done by another program. By providing the weather info without requiring keyboard input, the script is now more versatile and useful for others.

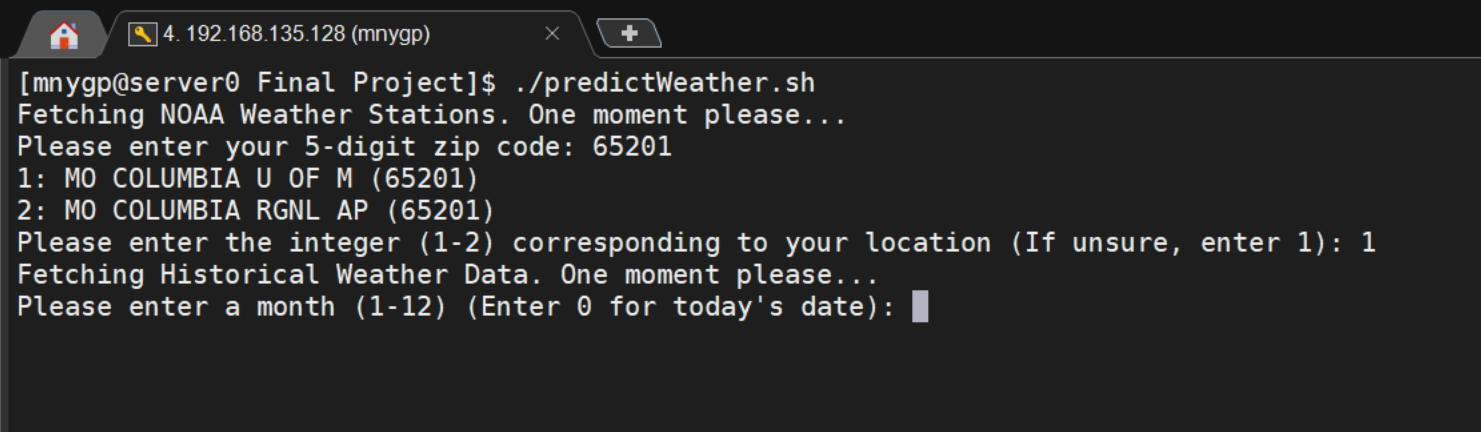
Once a given weather station’s data is downloaded once, it does not need to be re-downloaded again. This means that users based in remote regions for extended periods without internet connections will still be able to have a grasp of how the weather will act throughout the year.

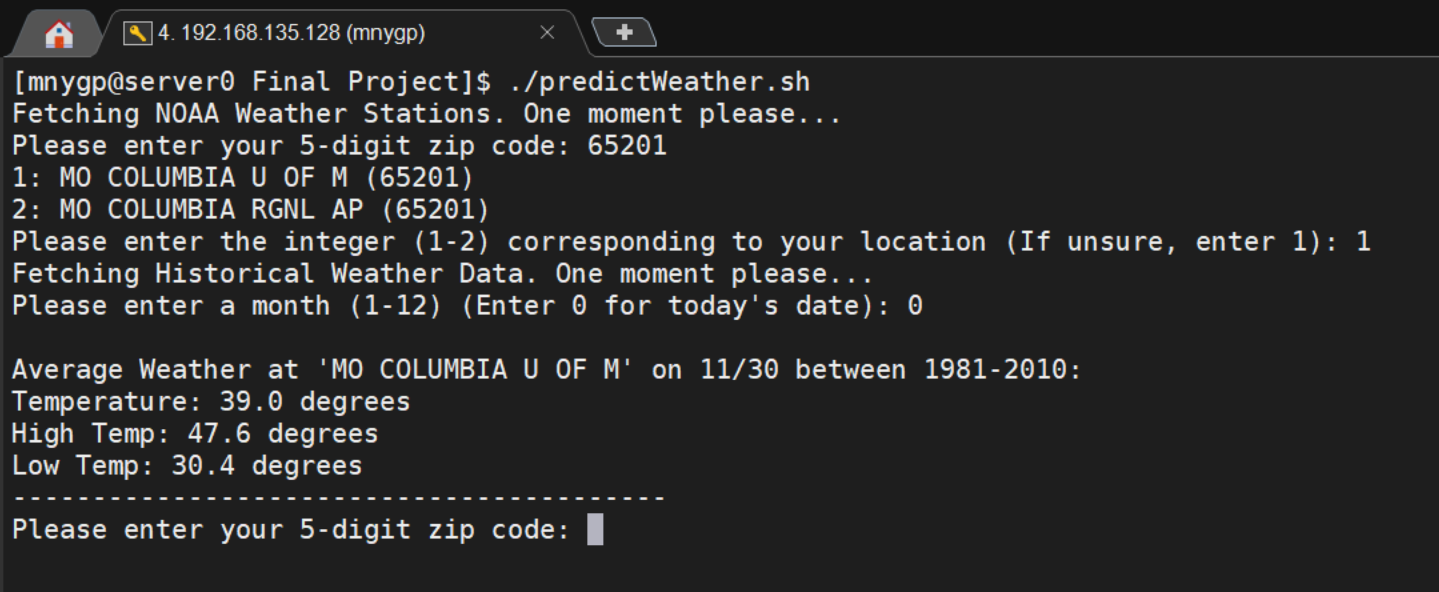
Although the station data files are relatively small, they have considerable amounts of data that can be displayed to the user. I’ve chosen to display 3 factors: average temps, high temps, and low temps, but I wrote the functions in such a way that it would take only 3 more lines of code to display another row of simple data. I’ve included two examples of this, commented out, at the bottom of the script file. The first example is a simple one that's directly pulling 75th percentile precipitation data, while the second is a slightly more complex example that calculates the average precipitation level on a given day in inches.

**Screenshots of normal output:**

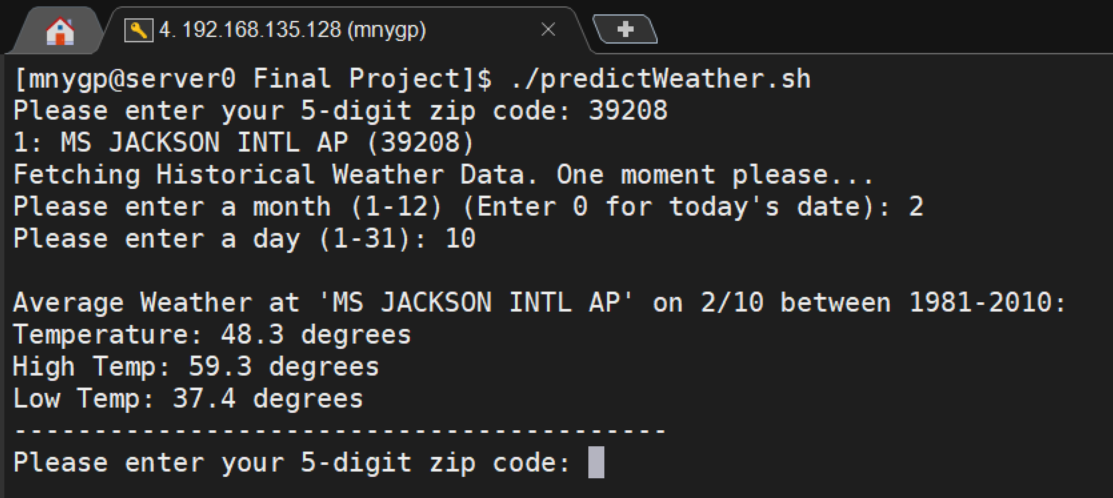




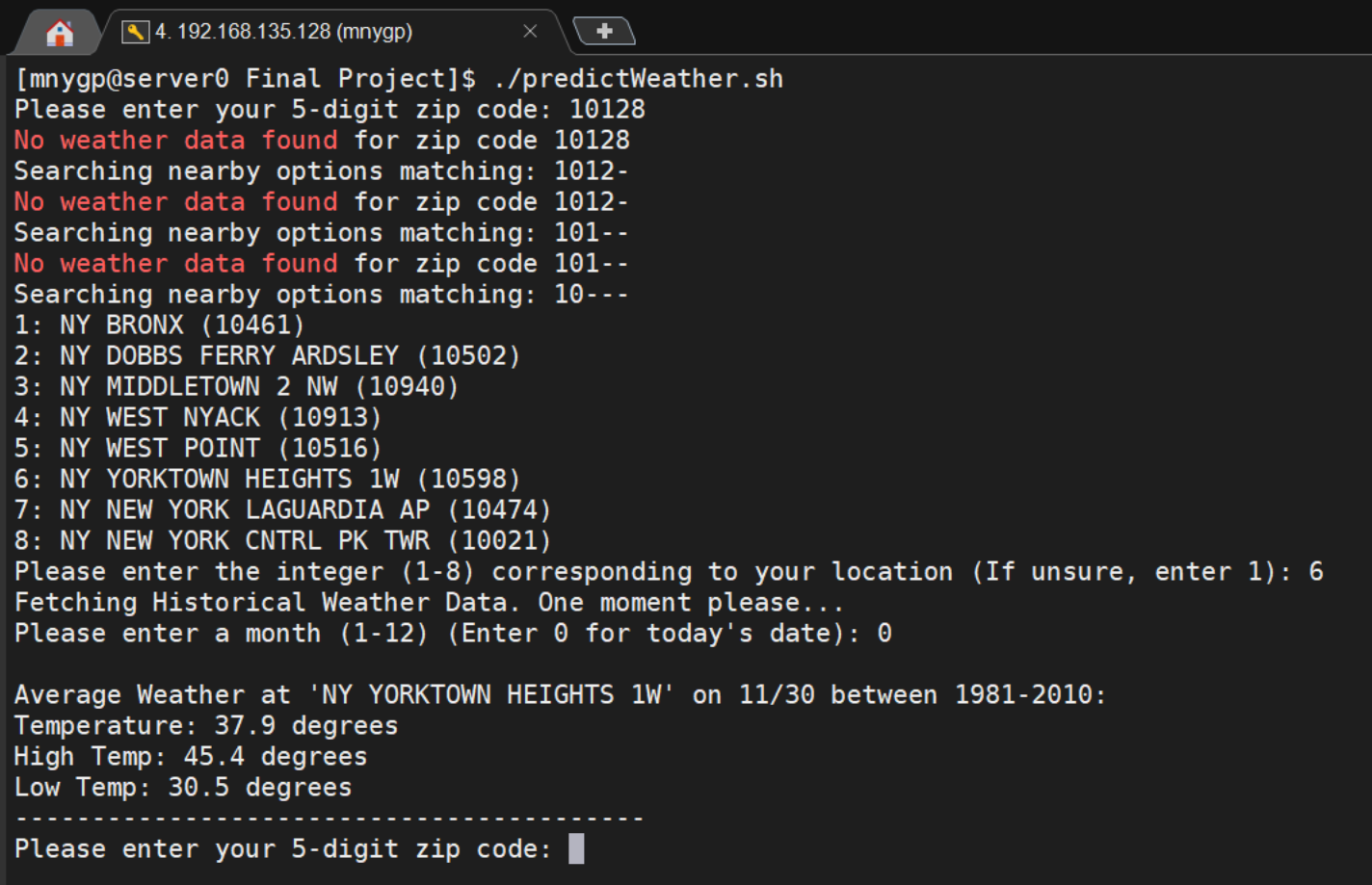




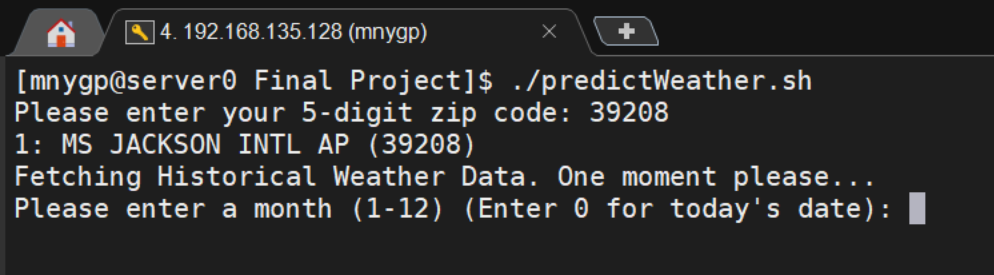
**Example of choosing a date:**



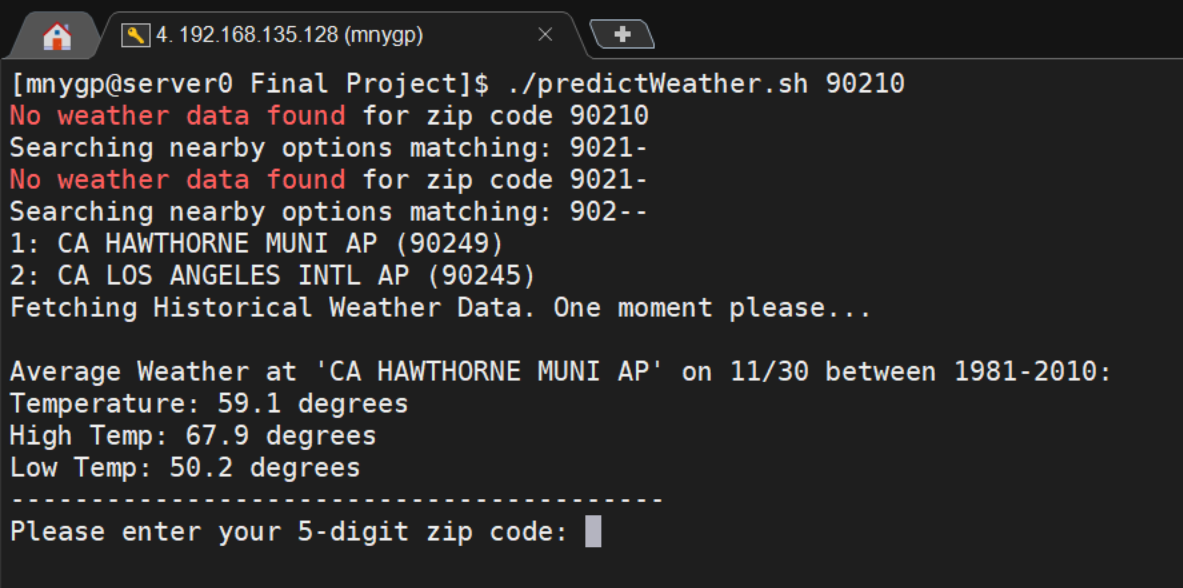
**Example of correcting for when no stations are found for a given zip code (10128 -> 10021):**



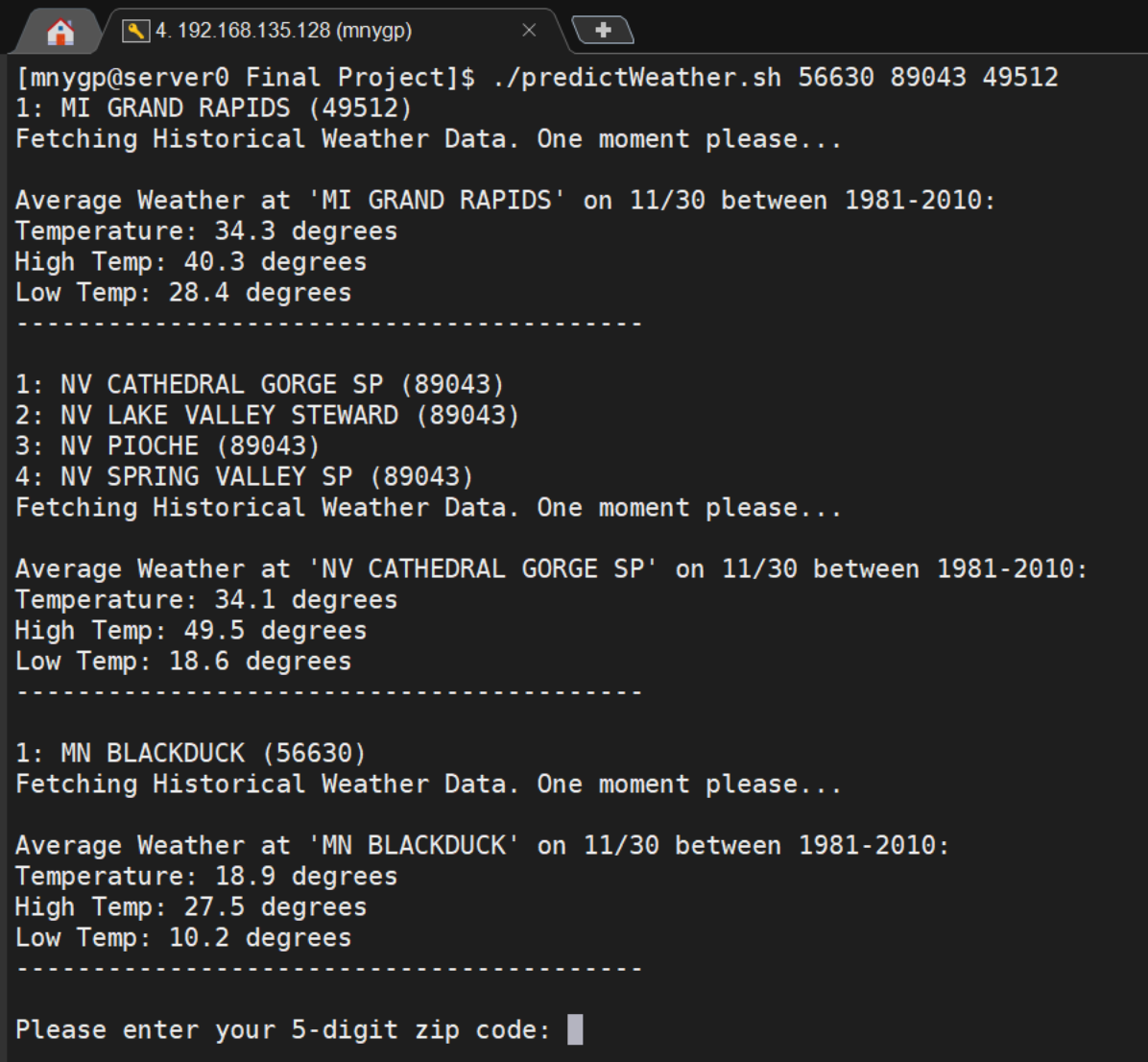
**Example of only 1 station (auto-selected):**



**Example of command-line argument auto-completion (no user input):**



**Example of multiple zip codes provided as arguments:**



Sources

* Overview of NOAA’s U.S. Climate Normals
  + <ftp://ftp.ncdc.noaa.gov/pub/data/normals/1981-2010/documentation/1981-2010-normals-overview.pdf>
* 1981-2010 Climate Normals ReadMe
  + <https://www1.ncdc.noaa.gov/pub/data/normals/1981-2010/readme.txt>
* Normals Daily Documentation
  + <ftp://ftp.ncdc.noaa.gov/pub/data/cdo/documentation/NORMAL_DLY_documentation.pdf>
* Main source of data
  + <ftp://ftp.ncdc.noaa.gov/pub/data/normals/1981-2010/>
* Bash Scripting Cheat Sheet
  + <https://devhints.io/bash>
* Meteorology Reliability
  + <https://scijinks.gov/forecast-reliability/>
* Almanac Accuracy Study
  + <https://extension.illinois.edu/blogs/garden-scoop/2017-08-16-our-moon-and-old-farmers-almanac>