

# Lab 10: File Input, Exceptions and Data Analysis

## Learning Objectives

*After completing this activity, students should be able to:*

- Read from files
- Parse file data and save it to dictionaries
- Handle file exceptions
- Perform data analysis of file data

## Part 1 File Input and Exceptions

Three weather stations situated in a variety of locations near Colgate University reported these temperatures (in Fahrenheit) during a recent week. The disagreements among the stations are in part a reflection of the fact that the stations sent their data at varying times of day. This data is in the file `rawdata.txt`, which we provided you with:

Station	Monday	Tuesday	Wednesday	Thursday	Friday
Station 1	42.79	48.3	52.03	53.02	55.11
Station 2	46.63	45.3	53.66	52.98	52.19
Station 3	54.04	47.42	49.77	40.5	42.04

We also provided you with starter code in `lab10.py`. **Before you proceed, make sure that all the files we provided you with are in the same folder.**

1. In `lab10.py`, finish implementing the function called `get_data` that takes in the name of a file, opens it and then prints the content of the file line by line. Note that the `open` function for files takes a parameter called `encoding` (for example, `ascii` or `UTF-8`) which comes in handy especially when working with files across different operating systems. Therefore, instead of using `open(filename, "r")`, get practice using `open(filename, "r", encoding = "UTF-8")`.

Note that the data would be organized well by a dictionary, using the station identifier as key and its daily temperature data (converted from strings to floats) as value, like so:

```
{'Station1': [42.79, 48.3, 52.03, 53.02, 55.11],  
 'Station2': [46.63, 45.3, 53.66, 52.98, 52.19],  
 'Station3': [54.04, 47.42, 49.77, 40.5, 42.04]}
```

2. Edit the `get_data` function so that the data is collected into such a dictionary. Return the dictionary to main and print it in main, to see if yours matches what is above. Test your code by changing the input file to `rawdata2.txt`.

3. Test `get_data` with the `rawdata2.txt` file to check that it works correctly.

4. Run `get_data` with the `rawdata_corrupted.txt` file by calling `get_data` in `main`. In a comment in `main` explain what happens and the reason for it. Edit `get_data` so that as the function reads the data, it should discard the data for any station that contains corrupted data (that is, not save to the dictionary), and report the corruption to the user via a message printed to the screen. All other uncorrupted stations should be included in the dictionary as usual.

## Part 2 Data Analysis

5. Write a function called `get_station_maximums` that returns a dictionary of the maximum temperatures for each station. You may also consider writing a utility/helper function `print_station_maximums` (to be called in `main` with the return list from `get_station_maximums`) that prints the results in a helpful way, such as:

```
Station1 maximum temperature: 55.11
Station2 maximum temperature: 53.66
Station3 maximum temperature: 54.04
```

6. Write a function called `get_global_maximum` that takes in the dictionary and returns the maximum temperature reading among all stations, and have `main` print the result.

## Part 3 Challenge problem

This is a two-part problem. We would like to write a function that returns the day on which the average temperature is maximum, and print the day along with its average temperature in `main`. To accomplish this, we will need a list of the days that appear in the top row of the text file; because various text files may have different first rows, we cannot simply hard-code this list.

7. Write a function called `get_header_items` that returns a list with the items in the header row.

8. Now that we have a list of the days, we can send it, along with our dictionary, to a function called `get_max_average_temperature` that will return a list containing both the maximum average temperature and the day on which that maximum occurred. Write that function in `lab10.py` and call it in `main` to test it.