Fall 2019

CS 5001: Intensive Foundations of Computer Science

Instructors

Tony Mullen

Email: a.mullen@northeastern.edu

Submitting the assignment

Submit your assignment folder to Bottlenose in the form of a *zip* archive. This folder should contain your program files.

1. **Commit and push changes** Before submitting your assignment, first commit and push all changes to GitHub.
2. **Zip the assignment folder** Right-click on the folder and select "Send to > Zip file" (on Windows) or "Compress" (on Macs). This will generate a zipped (.zip) file.
3. **Upload the zipped file to Bottlenose** Log into Bottlenose and navigate to the Bottlenose page for this class (https://handins.ccs.neu.edu/courses/68). Find the submission area for the appropriate lab or assignment and upload the .zip file to submit the assignment. Double check you're submitting to the correct assignment's submission point.

Homework Assignment 5

Water Consumption

This week you'll write a program that performs some analysis on actual, real-life data! My friend Brad moved into a new house — one that has a device that monitors how much electricity is used. Once a minute, it records the power usage on each of the electrical circuits in the house. The water comes from a well, and it dawned on him that he could calculate how much water the house is using by looking at the power data.

He knows that the pump produces two gallons of water per minute when it's running. So in order to calculate how much water is used, we can look through the power usage data for the pump, and count the number of data points (minutes) where the pump is drawing power. The value of each line describes the number of watts of power drawn by the pump during the corresponding minute, and the pump should be expected to draw about 1000 watts a minute when it's running.

Once we know how many minutes it ran, we can multiply by the flow rate to get the number of gallons produced. (For example, if I see that the pump ran for five minutes, we know that it produced 10 gallons, since it produces two gallons per minute.)

You'll write a program called `pump_analysis.py` to read data from a file and do some analysis on it, including determining water use.

Analyzing the data

First, download the data (secure/pump_data.zip). Unzip that file to find two text files. `pump_data.txt` contains minute-by-minute power consumption data for the well pump from January and February of this year. The `short_data.txt` file contains just one hour's worth of data from the pump. (The short file is small enough that you could calculate the expected results by hand and use that to verify that your program is working correctly.)

To keep things simple, place both of these text files directly into the directory where you save your Python script for this assignment.

Look at the content of the data files in a text editor so you understand how the data is structured.

Reading data from a file

Opening a file and reading it line by line is easy in Python. Read the docs about `open()` here: https://docs.python.org/3/library/functions.html#open (https://docs.python.org/3/library/functions.html#open)

The name of the file will be passed in by the user after a prompt. The `open()` function takes a string argument representing the name of the file, and returns a file object. The file object will behave as a generator for lines of the file. This means that you can use a plain `for` loop to loop through the lines of the file. Each line is a string ending with a new line character, so you will need to strip off the new line character and convert the result to an int in order to use it as a number. The `.rstrip()` string method may be helpful here.

Creating the report

Your program should use the data it reads to compile a report of pump operation and power use. It should do the following:

- Report the duration of the data file in both hours and days
- Report both the total number of gallons produced and the average daily consumption.
- Report the total power used by the pump. As shown below, I'm reporting this is both Watt minutes and the more traditional Kilowatt Hours (kWh). You can divide the Watt minutes value by 1000 to turn the Watts into Kilowatts, and then divide by 60 to turn minutes into hours. (Power costs about

11 cents per kWh here, so once you've got kWh you could also estimate the cost of the electricity used.)
- Print how long it took to consume a certain quantity of water. In my output below I'm reporting results for 5 gallons and for 100 gallons.

Here is some sample output from a run of my program. The name of the file is input by the user.

```
$ python pump_analysis.py
Please enter the file name: short_data.txt
Data covers a total of 1.0 hours
(That's 0.041666666666666664 days)

Pump was running for 7 minutes, producing 14 gallons
(That's 336.0 gallons per day)

Pump required a total of 6795 watt minutes of power
That's 0.11325 kWh

It took 11 minutes of data to reach 5 gallons.
It took -1 minutes of data to reach 100 gallons.
```

The last line, saying it took '-1' minutes, means that in this sample, 100 gallons is never reached.

Handling exceptions

What happens if the user enters the name of a file that does not correspond to a file in the directory of the program? At present, the program will crash, reporting `FileNotFoundError: [Errno 2] No such file or directory: 'fakefile.txt'`.

A user inputting an invalid file name is a good example of what's called an *exception* in programming. An exception is a possible state of the system that is not a part of the designed behavior of the system. *Exception handling* requires anticipating possible things that could go wrong and making sure that they are dealt with by the program in a useful and clear way.

Python's exception handling is done using `try:` and `except:` blocks. Python tries to execute the code in the `try:` block. If it executes successfully, then the code proceeds to execute as expected. If the code in the `try:` block raises an exception, then the code in the `except:` block will be executed instead. Read the documentation for exception handling here:
https://docs.python.org/3/tutorial/errors.html#handling-exceptions
(https://docs.python.org/3/tutorial/errors.html#handling-exceptions)

Use `try:` and `except:` blocks to modify your code so that the user receives a more user-friendly message if they enter an invalid file name, like this:

```
$ python pump_analysis.py
Please enter the file name: no_such_file.txt
Unable to open no_such_file.txt
```

After printing the message to the screen, you can call `return` which will end the execution of the function (this assumes that your code is in a `main` function, as it should be!)

Extra credit

If you finish the code above and are anxious for more, consider taking on this extra challenge.

The pump's water softener (which removes minerals from the water) needs to operate for a sustained period in order to "recharge". So in order to analyze the behavior of the pump's water softener, it is useful to know when there have been particularly long stretches of continuous operation of the pump.

The goal for the extra credit challenge is to look through the data file for times when the pump runs for at least 120 minutes in a row, and report when the long run started and how long it lasted. This information is of interest for answering several questions about the water softener, but all you need to do for the extra credit is find those long runs and report them. Add the necessary code to your program to print output something like this when run on `pump_data.txt` (it shouldn't get anything on `short_data.txt`):

```
Information on water softener recharges:
179 minute run started at 9799
173 minute run started at 29944
228 minute run started at 48978
244 minute run started at 69133
```

This assignment is worth 100 points without the extra credit, and you can receive 15 points, maximum, for your efforts on the extra credit problem.

Style Guide

Please familiarize yourself with the PEP 8 Python Style guide (https://www.python.org/dev/peps/pep-0008/#a-foolish-consistency-is-the-hobgoblin-of-little-minds). These are excellent tips for writing clear Python code and you should follow this style.

Before you submit your assignment, go through the checklist below and make sure your code conforms to the style guide.

- No unused variables or commented-out code is left in the class
- Your code is appropriately commented
- All numbers have been replaced with constants (i.e. no "magic numbers").
- Proper capitalization of any names used: snake_case for ordinary variables and functions, CapWords for class names, and ALL_CAPS for constants
- Use white space to separate different sections of your code (follow the PEP8 linter's guidance)

Using the PEP8 linter

In addition to the checklist above, use the PEP8 linter in your editor to make sure you're catching small style issues of spacing and consistency. The graders will use the PEP8 linter as a guide for enforcing PEP8 style, which should simplify the process for them and you. It's easy to track down issues with the

linter and you should make sure that the linter report is completely error and warning free before submitting.

This assignment is adapted from an exercise by Brad Richards.

Tony Mullen a.mullen@northeastern.edu

This website was last updated at 2019-09-23T18:45:44.857Z