Module 10: Debugging K-means Clustering

JHU EP 606.206 - Introduction to Programming Using Python

Introduction

In the workplace you will most often inherit the code of others to modify rather than writing an application from scratch. One of the initial challenges here is that you have to spend time figuring out how the code works and determining if it works correctly. While good style and comments can reduce this effort, inevitably you will encounter code that contains bugs that you'll have to fix. There are a number of different methods for debugging and fixing code, like the ones we've discussed in class this week.

Skills: debugging, exception handling, error types

Programming Assignment: Debugging K-Means Clustering

For this assignment you will be given a solution to the K-means clustering assignment from Module 5, but the solution contains 22 errors of varying types. Before we break out the individual types, there are a few notes you should be aware of: Multiple errors can occur on the same line. Part of the challenge will include the fact that python will not always point you directly to the error. Make sure to take time to think about what python is telling you if you see an error occur. **There may be cases where the same error is repeated multiple times in nearby code. In these cases each error should counted individually.**

The 22 errors are broken out as follows:

1. Lexical errors (1)
2. Syntax errors (1)
3. Semantic errors (5)
4. Logic errors (13)
5. Input errors (2)

Your job is to debug the provided code so that it once again outputs the correct solution provided in Module 5.

The goal of the assignment is to debug an arbitrary piece of code. Please do not refer to your solution to get the program running correctly though you're free to refer to the Module 5 Assignment as a refresher. Your task is to find and document the 22 bugs organically. Additionally, please debug any issues with the input file by running the code, not by visual inspection. That is not to say you can't look at the input file, but one of the requirements is that you must specify the exception caused by each bug (including input errors) and patch them, so simply modifying the input file is not sufficient; **you'll want to implement exceptions to handle the two input errors**. Though looking at the input will likely lead you to finding the input errors for this assignment, real-life inputs are prohibitively long so manually reviewing an input that contains, say, 250 million entries, would be very difficult. In addition, some of the errors may take extra thought in uncovering why there is an error, and may not be obvious at first glance.

Note: there can sometimes be ambiguity between logical errors and semantic errors. Try your best to match the error counts we've provided above, but if your numbers differ slightly don't agonize over it. If you find yourself stuck between two different error types, please just be sure to justify your selection carefully and we'll be gentle on the grading side!

For each bug you find, please provide the following information:

1. A description of the error/bug:
    1. Why it's a bug
    2. When is it triggered
    3. How did you find it
    4. What type of exception did it cause (if any)
2. The type of error: lexical, syntax, semantic, logic, or input
3. A bug fix
    1. Write the actual code that corrects (and in some cases, prevents) the bug
    2. For any input errors, please add code that actively prevents/handles the bug
4. A description of your bug fix
    1. Ex: **"*To prevent the use of x before it is initialized (a semantic error), I declared and initialized x at the beginning of my code.*"**
5. A screenshot of the exception caused by the bug/error (if applicable)

Deliverables

readme.txt

So-called "read me" files are a common way for developers to leave high-level notes about their applications. Here's an example of a README file for the Apache Spark project. They usually contain details about required software versions, installation instructions, contact information, etc. For our purposes, your readme.txt file will be a way for you to describe the approach you took to complete the assignment so that, in the event you may not quite get your solution working correctly, we can still award credit based on what you were trying to do. Think of it as the verbalization of what your code does (or is supposed to do). Your readme.txt file should contain the following:

1. **Name**: Your name and JHED ID
2. **Module Info**: The Module name/number along with the title of the assignment and its due date
3. **Approach**: a detailed description of the approach you implemented to solving the assignment. Be as specific as possible. If you are sorting a list of 2D points in a plane, describe the class you used to represent a point, the data structures you used to store them, and the algorithm you used to sort them, for example. The more descriptive you are, the more credit we can award in the event your solution doesn't fully work.
4. **Known Bugs**: describe the areas, if any, where your code has any known bugs. If you're asked to write a function to do a computation but you know your function returns an incorrect result, this should be noted here. Please also state how you would go about fixing the bug. If your code produces results correctly you do not have to include this section.

Please submit your ***kmeans.py*** source code file, screenshots of your exceptions from Step 2 in the "Programming Assignment" section, your writeups for the bugs that you found, and a screenshot of the fully working output generated by **kmeans.py** in a PDF called JHEDID_mod10.pdf (ex: jkovba1_mod10.pdf). Please do not ZIP your files together.

Recap:

1. readme.txt
2. Writeup detailing information on bugs found
3. The patched version of **kmeans.py**
4. A PDF containing screenshots of your exceptions from Step 2 in the "Programming Assignment" section and a screenshot of the fully working output generated by **kmeans.py**.

Please let us know if you have any questions via Teams or email!