Kaden Scroggins – SRE Automation Exercise – 1-page report

Time spent

In total, I probably spent around 1.5 hours actually working on the code. Since I'm relatively new to Python, I spent a bit more time getting familiar with ways to do different things in the language, maybe an extra few hours on top of that – but I wouldn't count that towards the amount of time I spent working on this.

Methodology

- Define a hardcoded list of allowed packages, which is the list of different architectures contained in the Debian package repository
- Call a function, `get_architecture()`, to either get the first command line argument or prompt the user to input the architecture during runtime
- Pass the result of `get_architecture()` to `get_contents_list()`
- Download the specified contents archive from the package repository, extract it, read each line into a list, and delete temporary files saved on disk
- Map each package name to a dictionary where the key is a count of how many files reference said package
- Get the top 10 largest keys in the packages dictionary and output them

Best Practices

I used Pylint to lint my code, and brought it from a score of ~6 to a score of ~10. Right now Pylint is telling me that my code is rated 9.4/10 because a function call to disable SSL warnings doesn't exist, but it clearly does exist, so I'll say my code is 10/10 by Pylint standards

Organization

I worked on this task on two separate computers, and I used Git with GitHub to sync changes and keep track of history. I kept the repo private since I assumed you did not want this question to be public yet, but I can share the repository with you if you want to see my working history on it. The repo is stored at https://github.com/kadenscroggins/sre-automation-exercise

As for organizing as I wrote my code, I first laid out the necessary functionality with some moderately verbose commenting, then divided up the code into four separate methods that performed the major tasks required for the program. I then used Pylint to help refactor my code, boiled down comments to what I felt was necessary, and then wrote tests once I was done.

Final thoughts

What I have is fully functional, but it is a bit slow for my standards, taking around ten seconds to run on my machine. If I was to continue working on this, the next thing I would implement is multithreading to scan through the millions of lines in these files. I would also add a progress bar that shows how far along the process is, that way a user would not feel like it is hanging after a few seconds.