Individual Programming Exercise

Academical introduction to Python

Developed by Guido van Rossum, Python is an interpreted programming language known for its readability, straightforwardness, and flexibility. Since the first release in 1991, Python has grown to be a significant programming language for both small and large-scale projects. As Python is straightforward, it supports beginners and experienced programmers and is widely used in various IT fields such as data science and automation (Lutz, 2013).

Python is particularly well-suited for automation work due to several factors, including ease of use, debugging capabilities, extensive library availability, and generally acceptable execution times. Miller et.al (2022) stated that Python is “the most commonly used” language for network automation, which underlines the above-mentioned arguments.

First, the error handling and build in debugger offer interactive debugging and allow to go through the code one by one. The error handling shows clear exception messages and retesting is fast as Python allows quick iteration of the code. One of the strongest advantages is the extensive ecosystem of libraries and frameworks. These libraries cover a wide range of automation needs, e.g., task scheduling. If the standard library is not sufficient, third-party libraries can also be imported to Python. Lastly, also the execution time is important for automation programs. Pythons’ execution time is often sufficient for automation, even if other compiled languages such as C++ perform tasks faster (Miller et al., 2022).

While Python has numerous strengths, it also has some downsides that can be significant, depending on the automation use case. As already stated, Pythons execution speed is slower compared to other programming languages such as C++. Additionally, the high memory usage of Python can be problematic for applications which are memory intensive and lastly Python is not easy to deploy as there can be dependencies on various libraries and the need for specific environment configurations (Power & Rubinsteyn, 2013).

Evaluating Python's suitability for Miele

Understanding these downsides is crucial for developers and organizations when deciding whether Python is the right tool for a particular project, especially when high performance or specific system-level capabilities are required. For my company Miele there are different aspects to consider. First, Miele has a huge IT infrastructure with over 50 subsidiaries and an IT environment which is heavily SAP driven. Second, we have IT departments spread worldwide and need a programming language which is majorly know and can be debugged by most of our employee's.

Integrating Python in Miele's SAP environment can provide several benefits, such as better analytics and automation, but can also lead to challenges e.g., compatibility and integration. Overall, Python is a highly versatile language that fits well in Miele's system infrastructure and can enhance SAP capabilities through data analysis, automation, integration, and machine learning. An example of a current use case are the regression tests implemented with Python, which are running before a new feature deployment. Despite some challenges related to performance and compatibility, the extensive availability of libraries and community support makes

Python a powerful tool for Miele and a worldwide programming solution for selected projects (Miller et al., 2022).

Within the selected projects at Miele, Python's built in functions, but also external resourced can be used. Both have advantages and disadvantages:
Built in functions are directly available, further, they provide a good performance and are safer than third-party alternatives. The difficulties are, that the built-in functions do not cover every edge case and are not always flexible enough (Lutz, 2013).
In contrast, external resources are mainly used to cover specialized tasks that require advanced functionality. They can reduce the development time, as they offer pre-built solutions to solve complex problems, but the security is not always given (Cannon & Wohlstadter, 2010).

As IT security plays an important role, but Miele is also often faced with problems of high complexity, each project manager must evaluate critically, if the use of external resources in python are needed.

Individual Programming Exercise

Time taken to sort the provided file:

Quick sort:	1.5 seconds
Bubble sort:	457.77 minutes
Pythons built in sort:	0.179 seconds

Timings may differ to provided timings, as I had issues with reading the files and encoding the numbers.

Performance analysis

My performance analysis shows that quick sort (1.5 seconds) performs the sorting faster than bubble sort (457.77 minutes) but is still slower as pythons built in sort function (0.18 seconds).

Quick Sort has an average-case time complexity of O(n*log(n)), which makes it very efficient for large datasets. This efficiency arises because it divides the array into smaller sub-arrays, reducing the problem size with each recursive call. Quick sort first selects an element from an array, then rearranges the array in a way that all elements less than the selected one come before it, and all elements greater than the selected one come after it. Finally, it applies this task to the sub-arrays of elements less than the first selected one and elements greater than the first selected one (Durrani & Abdulhayan, 2022).

In comparison, Bubble sort has a time complexity of $O(n^2)$ in the average case. It is therefore significantly slower for large datasets because it repeatedly iterates through the array and makes many comparisons and swaps. Basically, it works repeatedly through a list of elements, comparing each pair of neighboring items, and swapping them if they are in the wrong order (Durrani & Abdulhayan, 2022).

Python's sort function is generally faster than a pure implementation of quick sort for several reasons. The underlying algorithm Timsort (a hybrid sorting algorithm derived from merge sort and insertion sort) is specifically designed to perform well on real-world data, which often contains pattern like runs of already sorted data.
While having the same time complexity as quick sort, it is still faster due to its adaptive nature, stability, and the efficiency of the underlying C-Code. Overall, it is a

highly optimized, hybrid algorithm that takes advantage of low-level optimizations that are not feasible in a pure Python implementation (Yang, 2022).

Why is the number of the second hash different?

A hash function used in python generates a unique hash value based on the contents of the file & the program. Hash functions are extremely sensitive to changes in the input data (in this case, the output file). Even a small change in the file will produce a completely different hash value as “hash functions can only match files exactly for every single bit” (Gurjar et al., 2015). In our case we added three additional rows in the output file, which is a big change and resulted in a new hash (Gurjar et al., 2015).

Academic References

Cannon, B., & Wohlstadter, E. (2010). ‘Controlling access to resources within the python interpreter‘, *Second EECE 512 Mini-Conference on Computer Security. Vancouver, 10-12 November.* 1-8.

Durrani, O. K., & Abdulhayan, S. (2022). Performance Measurement of Popular Sorting Algorithms Implemented using Java and Python. *International Conference on Electrical, Computer, Communications and Mechatronics Engineering:* 1-6. DOI: 10.1109/ICECCME55909.2022.9988424.

Gurjar, S., Baggili, I., Breitinger, F., & Fischer, A. (2015). *An Empirical Comparison of Widely Adopted Hash Functions in Digital Forensics: Does the Programming Language and Operating System Make a Difference?*. Page 57-60 Available from: https://digitalcommons.newhaven.edu/electricalcomputerengineering-facpubs/30/ [Accessed 10 June 2024]

Lutz, M. (2013). *Learning Python* (5th ed.). O'Reilly Media. 13 – 67.

Miller, A., Kobylski, N., Qamar, E., Xiao, J., Veal, N., Kenney, R., Wysocki, N. & Mahmoud, M. (2022). ‘Automating File Operations via Python‘, *International Conference on Computational Science and Computational Intelligence*. Las Vegas, December. 1907-1913. DOI: 10.1109/CSCI58124.2022.00343

Power, R., & Rubinsteyn, A. (2013). *How fast can we make interpreted Python?.* Available from: https://arxiv.org/pdf/1306.6047 [Accessed 12 June 2024]

Yang, A. (2022). *Approaches to the Parallelization of Merge Sort in Python*. Available from: https://arxiv.org/pdf/2211.16479 [Accessed 12 June 2024]

Word count: 1091