1.

1. The memory corruption CWE causes undefined behaviors, such as corruption or crashing the program. Part of the CWEVulnerableSum.py code I wrote created a buffer and then wrote past the buffer (using the ctypes module so that the interpreter allows it.) Sometimes Python can catch this, but other times it can crash the program or write to out of bounds memory. In my case, my Python caught it, even using a C module, but this is the idea behind a popular memory corruption CWE.
2. The Input Authentication CWE could lead to SQL attacks if this code was implemented in a database, or DoS attacks. For instance, if this CWEVulnerableSum.py was implemented in some system, due to no input validation, the user could theoretically cause a crash with an extremely long input.
3. The improper Neutralization of Input CWE can also lead to SQL attacks if implemented in a database. For instance, if CWEVulnerableSum.py was included in a SQL database, and the program author didn’t write with good conventions such as the Principle of Least Privilege (PLP) and Minimal Trusted Computed Base (MTCP), then the user may be able to drop the whole database with a single input, causing massive damage to the system.
4. The Improper Authorization CWE can lead to a user accessing admin privileges which is bad when they have malicious intentions. For instance, theoretically in my code, I could have wrote under “if os.name != "nt" and os.geteuid() == 0: “, something to delete the whole file systems contents.

2. I have written multiple programs without proper input validation, such as basic calculators. I don’t have many projects so this is as good an example as I have, but if I incorporated this in a database, it could have left the database susceptible to SQL injection attacks if that database didn’t use the Principle of Least Privilege and Minimal Trusted Computed Base. To follow the Principle of Least Privilege I need to make sure the program doesn’t have any more privileges than what it needs. To follow the Minimal Trusted Computed Base convention, I would isolate security critical code and data from the rest of the system. There may be more methods to implement these conventions, such as memory encryption for MTCB.

3. Say some online service lets users upload and execute their own scripts, they could potentially access unauthorized files, Cause DoS, leak data, and more. Language-based security could solve this problem by rewriting the scripts and removing dangerous sections of the code like forks and infinite loops. This is better than traditional security since it’s more flexible, performant, and secure in general. It allows for a more streamlined and even automated approach to security, since there is a lot less room for the user to be malicious, and a lot less work from the people working on the system to ensure security.

4. The Mutation-Based Fuzzer did find errors in CWEVulerableSum.py in the case when the number in “previous\_amount.txt” became negative or even just had a decimal. It was handling negative numbers as a string and throwing an error because of the leading ‘-‘ symbol, and it was throwing an error with decimal numbers because of the ‘.’ character. The code coverage obviously wasn’t as good as it could have been. It would be better if it handled negative numbers and decimals correctly. I could have covered these in the code but I chose not to for the purpose of including CWE’s and being able to use the fuzzers on this code.

5. The Generation-Based Fuzzer did find bugs in CWEVulerableSum.py in which the input was not properly validated (Improper Input Authentication). When it ran the program with a string as the input, the program had an error because it wasn’t meant to handle strings. It also was throwing an error with extremely large numbers whenever the number in “previous\_amount.txt” was so big that an E was written to it for scientific notation. The code coverage could have been better to handle scientific notation, and it could have had input authorization to prevent string inputs.

6. With the Protocol-Based Fuzzer I included a TCP server in the script. A couple errors arose, such as when the echo command was sent, it did not recognize the command, and when non-utf-8 encoded characters were sent a warning was sent. The code coverage was pretty good, but it didn’t cover commands, and it could have been better with non-utf-8 encoded characters. Overall, it was good though.