**Chain of Responsibility:**

The chain of responsibility took a while to implement, but extremely worthwhile. As far as clarity, building out the Handler and concrete handlers is straightforward. It is concise, because it is up to the chain of objects to decide who can handle the clients request. Flexibility is great in this pattern because changing members in the chain or changing the order allows for dynamic adding or deleting of responsibilities. A quick disadvantage is a request must be received but it isn’t guaranteed to be fulfilled by a member of the chain.

**Nested If:**

The nested if-elif was not an elegant way of coding the functionality that was requested. If-elif is very error prone because it is very tempting to copy and paste the statements. I found myself forgetting to change functions a few times when copying and pasting. There is no separation of handlers for each of the cases, so the entire logic is wrapped in one big piece of code. Nested ifs are also very inefficient since it is O(n).

**Dispatch Table:**

The dispatch table was by far the easiest of the problems to implement. The dispatch table is less code and cleaner than the nested if-elif statement to achieve the same result. The handlers for each of the extension types are separated so the actual dispatch does not need to know anything specific about the handlers beforehand. If we needed to add more functions or disable ones that already exist, it is a small change in the extensions dictionary. This takes the O(n) from nested if’s and gives us O (1) a bit faster than a linear scan across nested ifs.