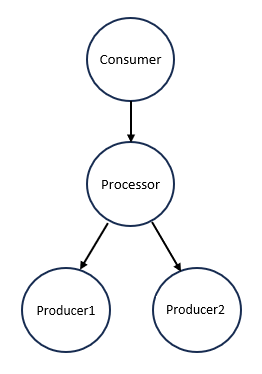
Function Connection Structure for Calling Functions in Reverse Order

Background

I worked on a project that required the usage of a decorator that connected functions in a hierarchical way. Say, Consumer() calls Process() which calls Producer1() and Producer2(). The functions do not have any parameters passed to them. All functions return some data or None. The connections between functions are shown in the picture below.



The implementation of the functions in my project is in top down fasion like this picture. In my project, Consumer() is a field in the UI that displays the calculated value of a security with the latest information. Producer functions are called when there is an update in, say proce of a stock. The issue in the production is that we don’t know when the stock price will be updated, so Consumer() must be called repeatedly without knowing the availability of updates which is a waste of CPU time. And there are multiple consumer functions. Some producer functions are shared among consumer functions. It is like a web.

This decortor solves the efficiency issue. Just wrap the four functions with the decorator without any modification to the scripts. Within this structure only Processor() cannot be called otherwise the intergraty of structure will be broken. After Produder1() or Producer2() has been called to get the update ready, Processor() will be called to process the data followed by Consumer() to consume the data and display the new security price.

Simple Explanation of How It Works

Here is what the decorator does to make it works:

1. The wrapped function, Newbie(), is added to the structure.
2. Find out which of the wrapped functions already in the structure call Newbie() and put their names in the UPPER\_FUNCS list that belongs to Newbie().

In runtime, say, Producer1() is called.

1. It produces something as return which is stored in the structure.
2. The functions in its own UPPER\_FUNCS list are called one by one.
3. This whole process is applied to each UPPER\_FUNC called upon in the last step.

When the UPPER\_FUNC calls Producer1(), Producer1() no longer needs to produce the same return again, instead its stored return is handed to UPPER\_FUNC immediately. That’s why I underscored the storage part in step one.

This process mirrors the reporting process perfectly. When one of the analysts produce the report (and stores it in computer). The manager is notified (calling the UPPER\_FUNCS). When all reports are ready, the manager use them to produce summary report (also stored in computer) and the boss will be notified. If there are still some reports missing, the manager will not create summary report and the boss will not be notified.

Further Enhancement

As of now I only tested it with functions that do not take parameters.

In this structure, there can be actually two types of parameters, one for calling UPPER\_FUNCs (call it UPPER\_PARAMETERS) and one that is already in the implementation of the functions (LOWER\_PARAMTERS). It makes the functions more complex but it facilitates communication from LOWER\_FUNCS to UPPER\_FUNCS. For example, the UPPER\_PARAMETERS can provide information to the UPPER\_FUNC on how to call the LOWER\_FUNCS within It with the right sets of parameters based on the UPPER\_PARAMETERS.

Overtime there may tons of return results in the structure (like too many old reports in the computer) many of which are no longer useful. Purge routine will be needed to keep the structure from growing forever.

Final Words

This structure is, by no means, the one-size-fits-all design.

This structure is not the whole thing in the implementation but the center part of it. There should be functions that call Producer1() and Producer2() from outside of this structure. These functions check the availability of data for Producer1() and Producer2().

This structure works perfectly for functions that takes no parameters. I hope this design could serve as a platform for tuning to fit particular needs like using the parameters as way of communication between functions.

Any comment and idea would be much appreciated.

This is link to this project at GitHub:

https://github.com/howtam/Reserve-Function-Call-Order/tree/main