CS130 Project Review
==========

Team performing review: Leopard
Work being reviewed: Ocelot

The first two sections are for reviewing the `sheets` library code itself, excluding the test code and other aspects of the project. The remaining sections are for those other supporting parts of the project.

Feedback comments on design aspects of the `sheets` library

Consider the overall design and structure of the `sheets` library from the perspective of the GRASP principles (Lecture 20) - in particular the principles of high cohesion and low coupling. What areas of the project codebase are structured in a highly effective way? What areas of the codebase could be restructured to have higher cohesion and/or lower coupling? Give specific suggestions for how to achieve this in the code.

I think in general they handle the principles of Grasp very well. Just like us they use a separate graph object in order to handle dependencies, and I think the coupling between this graph class and the workbook is actually pretty low (the graph class is only accessed by the workbook in very direct easy to understand ways). However, I think cohesion is one area where they are really lacking. They said they needed to write lots of tests just to handle the fact that their code is kinda all over the place. As they said, they kind of just made a bunch of helper functions and private methods, so much so that they believe that a lot of it just wasn't necessary and ended up just kind of clogging the codebase. Also, they used value types but it actually seems pretty redundant and unnecessary.

I think one way to improve would be to just cut down on the number of files and different classes broadly. Even if it means just looking at things and going "is this really necessary?" Having to handle redundant stuff that really serves no purpose (like for example their value types) creates unnecessary coupling. Furthermore, I think cohesion could be benefited by reducing the number of python files and distinct classes they have. They have different classes and helper functions for doing basically the same thing, and they have lots of different files which require crazy amounts of imports. I think this could all be consolidated much better.

Feedback comments on implementation aspects of the `sheets` library
---

Consider the actual implementation of the project from the perspectives

of coding style (naming, commenting, code formatting, decomposition into functions, etc.), and idiomatic use of the Python language and language features. What practices are used effectively in the codebase to make for concise, readable and maintainable code? What practices could or should be incorporated to improve the quality, expressiveness, readability and maintainability of the code?

So they had a few issues on this front. First of all they use a lot of try and except statements, even to run very significant and compute-intensive blocks of code. Try and excepts can really really slow down program execution so I have to imagine this was an issue for their performance. Furthermore, their code is absolutely filled to the brim with helper functions to the point where they get redundant. They even said during our discussion that often times they couldn't track down helper functions they had and so they ended up rewriting them. I think the issue here is that they lacked a "utils" file to just store all the random helper functions that they use across multiple files. Their code was also pretty poorly commented and there were no doc strings on basically anything.
Of the things that I liked, I liked that they used the underscore to identify helper methods. They also put a lot of those argument-type things everywhere which is nice.

I think they could improve by adding more comments, and removing the insane amount of try and excepts that they have. Furthermore they should do a big refactor of their helper functions to try and get rid of some of the redundancy. They could also try to add more comments everywhere (even if they just use an AI to document their code better).

Feedback comments on testing aspects of the project

Consider the testing aspects of the project, from the perspective of "testing best practices" (Lectures 4-6): completeness/thoroughness of testing, automation of testing, focus on testing the "most valuable" functionality vs. "trivial code," following the Arrange-Act-Assert pattern in individual tests, etc. What testing practices are employed effectively in the project? What testing practices should be incorporated to improve the quality-assurance aspects of the project?

They write a billion tests! I mean we were astonished by how many tests they had. It was truly ridiculous! Like when I first saw how many tests they had my thought was "what AI are you guys getting to write all this?" I think it definitely did a good job on the coverage side. They explored a lot of edge cases. However, their tests had *insane* redundancy! They wrote all of their "unit-tests" for the various functionalities they implemented in the code. And then a bunch of "integration tests" designed to do multiple things together. From what they told me it seems like they spend a lot of time on tests in part because their code is kind of disorganized (they said themselves they didn't know how to track down what was causing things to break so they needed lots of very detailed tests). I think

it might have been worth it to take some of the time spent on tests and to put it towards a refactor. Even if all they did was go and kill all the redundant helper functions they have, it would cut down on lines of code a lot.

Again tho I was very impressed with their tests overall. These are the kinds of test you would expect on a product actually going out into production. Literally all feature interactions are tested for which is pretty neat. I don't know what sort of advice I would add other than to say that maybe they did go a bit overboard.

Consider the implementation quality of the testing code itself, in the same areas described in the previous section. What practices are used effectively in the testing code to make it concise, readable and maintainable? What practices could or should be incorporated to improve the quality of the testing code?

T**hey use unit test which our team does not like. Pytest is better, so yeah. Other than that I think their test code was pretty good! THey have all of their tests pretty well commented and their tests are pretty concise (ignoring how redundant they might be). We are definitely envious of them regarding their tests! Their tests are also very maintainable, because they only rely on the types of workbook methods accessible to the user.**

Feedback comments on other aspects of the project
Overall I think they did very well! It seems like they have a good grasp on the code. I was really really impressed with how they handle Indirect with Tarjans. Its a very clever way of doing it. Even more impressive given that they are a team of 3! Very Good.

If you have any other comments - compliments or suggestions for improvement - that aren't covered by previous sections, please include them here.

NOTES from our convo:
Parser stuff:
- They don't use any of the auto-visit children things in their Formula Evaluator.
- A lot of the stuff we have as decorators they just call every time.
- They had a specific set of methods for camparing things of different or the same type.
- They use under line notation for the helper methods in their evaluator.
- They also give their workbook to the evaluator in order to get the cell value.

Misc:
- They have a lot of files just for individual classes. Is this a good idea?
- Use a lot of if / else when they could use dictionaries
- They have so many helper functions. Easier to create a new one then to find them sometimes.
- They have headers which are based.

Evaluation:

They create a Deque and then store their topo sort stuff in that.

To me their code for evaluation seems much better organized then ours.

Its super unclear how the Deque actually works though. How is it possible for the evaluator to return a tree? Is the Deque capable of handling changing dependencies?

Bro there idea here is pretty genius

Function Stuff:

Evaluate if Necesassary seems sort of unnecessary?

QUESTION 1:

How did you write so many tests?

They just wrote a bunch of them.

Why did you create a totally separate lark file for cell references?

Implemented for Indirect. Could have been reged

Is it faster to parse that way? NO

You used value types. We discussed doing this early on. Did you find that this was helpful?

They don't think this was that necessary.