CS130 Project 3 - Design Document
==================================

Please answer all questions in this design document. Note that the final feedback section is optional, and you are not required to answer it if you don't want to.

Unanswered or incompletely answered questions, or answers that don't actually match the code/repository, will result in deductions.

Answers don't have to be deeply detailed! We are mainly looking for an overview or summary description of how your project works, and your team's experiences working on this project.

Logistics (7 pts)

L1. [2pts] Enumerate all teammates here.
Zack Dugue, Eli Kugelsky, Kevin Do, Purvi Sehgal

L2. [2pts] What did each teammate focus on during this project?
Zack Dugue - Refactoring, parser, relative/absolute cell references, and writing tests
Purvi Sehgal - Refactoring, copy/move cells, and caching parsed formula
Kevin Do - Refactoring, copy/move cells + tests, and caching parsed formula
Eli Kugelsky - Refactoring, linting, testing, and bug clean up

L3. [3pts] Approximately how many hours did each teammate spend on the project?

9 hours

Spreadsheet Engine Design (9 pts)

D1. [3pts] Moving and copying regions of a sheet are very similar operations, with only a few differences between them. How did your team take advantage of the similarity of these two operations to reduce the amount of code required to provide this functionality?

We had a method called transfer_cells which handled basically all our functionality for move cells and copy cells. Transfer_cells simply had a kwarg called “is copy” and if it was not true, then it would set all of the cells that had been copied to None.

D2. [3pts] Similarly, moving/copying regions of a sheet, and renaming a sheet, both involve formula updates. Was your team able to factor out common aspects of these two operations to reduce the amount of code required to implement these operations? If so, what did you do? If not, why not?

We were not able to factor out the common parts of the formula updates of this. This is because while these operations have similarities in the fact that they update formulas, they are basically completely distinct operations. We found it was best to just reduce coupling by having separate Lark transformers for each functionality. In the future it might be worth it to write a code that wraps in some of the common elements (the idea that you take the parsed code, initialize a transformer, apply the transformer, then reconstruct the parsed code) into a workbook method, but at the time it was not a priority for us.

D3. [3pts] How does your implementation address the challenges of moving or copying a region of cells where the source and target regions overlap?

We address the challenge of moving or copying a region of cells where the source and target regions overlap by first hashing all of the cell contents and then moving them to the target region after all of them have been copied

Static Code Analysis / Code Linting (16pts)

L1. [5pts] The Project 3 spec includes an example of a subtle implementation bug with the Python counts(s, totals) function as written in the spec. Briefly describe the cause of the buggy behavior, and what is the recommended approach for avoiding the buggy behavior.

The buggy behavior occurs because its actually not a bug but a specific design implementation unique to python. There are many programmers who wish it were removed, however, there is no way to tell how many existing code bases rely on this specific implementation to work. It is mostly a problem with mutable default objects. So a way around this is to choose an immutable such as None and check to see if something was passed in the place of None and use that instead. Otherwise, if None then create a new object.

L2. [4pts] What code-linter did your team use on your project? Why did you choose it? Was this the first CS130 project in which you used a linter?

We chose to use RUFF as our linter because it was something we could all understand how to use and it is extremely efficient and quick. It was also highly recommended in the spec and by Professor Pinkston on the first day of class. This was the first project we linted on.

L3. [3pts] How did you automate the execution of your code linter? Did everyone in your team find it easy to run?

We wrote a github action which automatically checks the linting of our code when we push. This allows us to see if we need to make any changes and to where exactly we do. Then we push those changes and see if the linting has been updated. I think we want to move to an offline (not github action) linting next so we don't have to keep checking online every time, but rather run something local and check the results ahead of time.

L4. [4pts] Did the use of the linter improve your overall code quality and correctness? Give some specific details in your answer. Were there any serious issues (e.g. buggy language idioms) you were unaware of?

Our code was already pretty clearly formatted as well as using RUFF while writing helped us not code in any real problems. We were able to catch a lot of bugs early because of this and we avoided the mutable objects bug discussed above for default arguments. We have people with a lot of different programming backgrounds and thus different programming styles and the autoformater gave much better readability by putting us all on the same page.
It also helped us catch a bunch of bare try and excepts in our code which was very useful. We removed some and added more specific errors to catch for others.

Performance Improvement (18 pts)

In this project you must improve the performance of two central areas of your spreadsheet engine - cell updating and cycle detection. In the previous project your team should have written some performance-testing code in preparation for this effort, and should have run it under a profiler to get an initial sense of where improvements can be made. In this project you will follow through on this investigation, and fix performance issues in your code.

P1. [7pts] Give a brief overview of 3-4 of the worst hot-spots you identified in your performance testing and analysis. For each one, describe how your team was able to resolve it.

Our performance improvements were mostly motivated on our project 2 shadow grades. We only got less than 10 on two problems, Fibonacci and pascals. We created profiler tests mimicking these (along with other profiler tests) and tried to find hot spots.

By far the largest hot spot we identified was in our parser. Basically every test for our profiler was spending the most total time in earley.py and earley_forest.py which are parts of the Lark Parser. We resolved this by simply caching the parser. IE we would save the parser for each cell in a “parsed_formula” field and then we would access that field any time we needed to recompute the cell value.

Another big hotspot we found was with try and excepts in our code. We had a lot of these everywhere. We removed many try and except blocks (as advised by the TAs). We even managed to be a second faster on our pascals triangle test simply by moving our formula evaluator transformer out of a try except block.

We found that when running our profiler tests for copying very large sheets we spent a lot of time in the “Deep Copy” function. Google said Naive deep copy was not always faster, so I simply created the new sheet and then tried just setting the cell values. It turns out this was about 20% faster. However our profiler tests do not try copying very large sheets with formulas, so this is an area we definitely need to explore further.

P2. [4pts] Did your team try anything to resolve performance issues and find that it didn't improve things at all - perhaps even made things worse? If so, were you able to identify why the intended fix didn't produce the desired benefit?

We had one struggle where we thought our parse caching code wasn’t going faster but it was just because we mistyped an if statement!

We had a major code refactor this project intended to move the Graph operations out of workbook.py and into its own “graph class”. We were pretty worried that this refactor would cause a slowdown. This was because outside of our parser a significant portion of the time spent was in our topological sort method when computing and updating very large formula chains (including pascals triangle). The topological sort function involves traversing the reference graph for formulas. Our prior reference graph implementation had references stored locally, whereas our new one had them stored in a global dictionary. So we’re replacing an instant local lookup to find the parents of a cell, with a lookup in a hash table. Thus we expected the new implementation would be slower. The whole refactor yielded a 12.5% speedup on Pascals triangle, but notably had no effect on the speed of our topological sort (based on our profiler results). We were not able to diagnose the reason for the speedup!

P3. [4pts] How do you feel that your performance updates affected your code's readability and maintainability? Did it make it better? worse? unchanged? Elaborate on your answer.

Our code’s readability and maintainability was improved! Removing a bunch of pointless try excepts was helpful to our code’s performance but it also made it way more readable. The refactor was also made from a place of improving our code’s readability, but did have a non negligible effect in speeding up our code. The “Parser Caching” also lead to making our code more readable, in the sense that we took a look at how we were handling parsing, and found out that our parser shouldn't be stored in the workbook. Now it is just a global object in parser.py .

Overall our code is in a WAY better readability state than at the end of project 2. By the end of project 2 we were very aware that we were deep in tech debt just trying to get the code in a state where it would be passing tests. The refactor really helped our code be much more maintainable.

P4. [3pts] Did your performance updates cause any regressions in functionality?
If so, briefly describe any issues that emerged. How were these issues
identified (e.g. automated test failures, manual testing, etc.)? How
quickly were issues identified?

Honestly, not really! We had super intensive tests for the older projects and none of our speedups ever actually caused anything to break. In many cases improvements to functionality were actually found. For example, the change to sheet copying actually fixed a bug in our code from prior weeks.

Section F: CS130 Project 3 Feedback [OPTIONAL]

These questions are OPTIONAL, and you do not need to answer them. Your grade
will not be affected by answering or not answering them. Also, your grade will
not be affected by negative feedback - we want to know what went poorly so that
we can improve future versions of the course.

F1. What parts of the assignment did you find highly enjoyable? Conversely,
what parts of the assignment did you find unenjoyable?

I'm ngl I think the move and copy stuff was almost too easy? I am worried there are some big errors we massively missed just because it was so easy to implement and all our tests are passing. I kinda wish it had been put in some other project and this project was just performance based.
I think the performance stuff was very fun.

F2. What parts of the assignment helped you learn more about software
engineering best-practices, or other useful development skills?
What parts were not helpful in learning these skills?

I think the performance stuff really helped us take a new eye to our code and to think with a performance mindset.

F3. Were there any parts of the assignment that seemed _unnecessarily_ tedious?
(Some parts of software development are always tedious, of course.)

Diagnosing some of our old bugs from the acceptance tests were really tough. But we have emailed donnie about how some of the test doc strings are unclear.

F4. Do you have any feedback and/or constructive criticism about how this
project can be made better in future iterations of CS130?

I say try and stick move and copy somewhere else and make this project fully focused on performance tests, linting, and refactoring. Maybe include a code review with the TA's that is also graded?
Idk I just felt like the move copy stuff we spent disproportionate amounts of time on solely because it was a new feature.