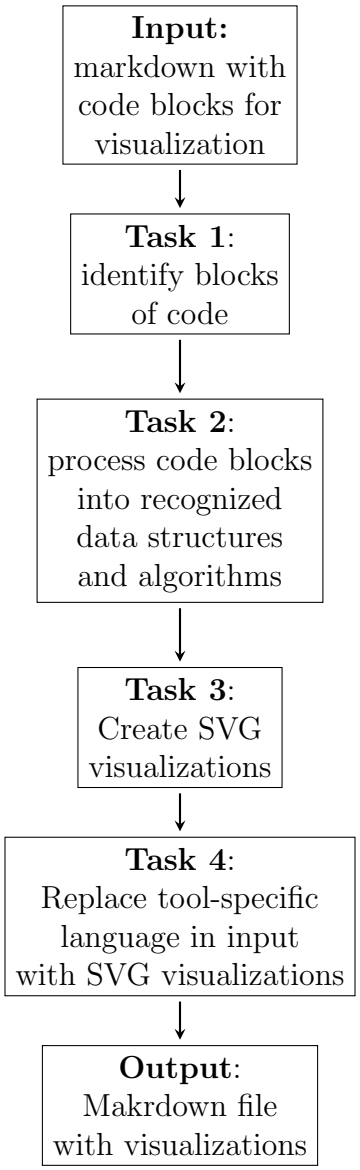
This document consists of a breakdown of the current state and design plan of our project, as I understand it. My goal was to organize a lot of the design notes and decisions spread out across most of our meetings into a single document and then combine that with a description of the current state of our project. It is still pretty rough though and could probably use some revision. Let me know if I forgot or misunderstood anything.

A diagram of the overview for the structure of our project is given below:



I don't plan to describe in detail why we decided to use the modular structure described above, see Oliver's components.md for that. Currently, Anna, John, Doni and I are working on task 3 (visualization) while Oliver and Eyal are working on task 2 (description).

Detailed descriptions of the above tasks are given below. I also include (my understanding of) the current state of our progress on these tasks.

Tool 1: Identifying code blocks

Our project will be given a markdown file with user-defined blocks of code and descriptions of how/what they want visualized.The task of this component will be to identify these subsections of the input markdown and then pass these on to the second tool. Doni has started work on this.

Tool 2: Describing data structures

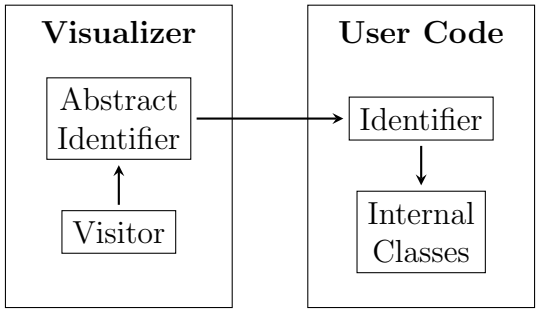
After the first tool identifies the relevant code and description of how it will be visualized, it is the task of this tool to create a language-independent description of the data structure and pass it on to the next tool for visualization.

The first task of this tool is processing the given code block to understand what it is trying to represent. Our current plan for this is to have a user implement an "identifier" class for each data structure that they want visualized. This identifier will implement all of the functions that our tools needs access to in order to understand the structure and contents of the data structure. The motivation for this is that a user may want to visualize the same instance of a class in different ways (e.g. visualizing a tree as a tree proper and later as a generic graph) and this allows us to never need to interact with the user's actual code (ideally).

For an example of the above, see Oliver's tree_visitor_example.py. In this file, we have the following classes:

1. BinaryTreeIdentifier (Visualizer-defined Abstract Identifier): An abstract class that defines what functions we need the user to implement to properly visualize the tree
2. BinaryTreeVisitor (Visualizer-defined Vistitor): The internal function in our code that uses the user-defined implementation of BinaryTreeIdentifier to traverse the given tree and extract the information needed for visualization.
3. UselessBinaryTree (User-defined Internal Class): The tree class that the user defines to use locally. Our code will ideally never touch this class.
4. UselessBinaryTreeIdentifier (User-defined Identifier): Implements the abstract BinaryTreeIdentifier class. The user defines how to gather the relevant information from UselessBinaryTree.

A general diagram of the relationships between these classes is given below:



Although we have a working instance of the above specifically for trees, we do not yet have any other data structure implemented.

After the relevant information is parsed from the user's code, this tool also has the task of writing a language-independent description of the contents so that we visualize it with the next tool. Oliver has taken to calling these description "UConn Language Independent Descriptions of Structures" or UCLIDS. Oliver and Eyal have begun working on a formal specification of the UCLIDS format. The work that has been done on this so far is in UCLIDS_spec.md.

Tool 3: Visualzing Data Structures

All relevant information about the structure and contents of a data structure should be contained in the UCLIDS description and then passed to the visualizer. Thus, the visualizer only needs to concern itself with how to best visualize (i.e. generate an SVG representing) a given data structure (it need not worry about parsing/processing other code). This is not so simple of a task though.

To start, we've identified a few crucial (and relatively simple) data structures and are implementing these independently. These starting data structures are trees, linked lists, sets, and arrays. This is what we are working on now. Once that is finished, we will merge our ideas into a common visualization framework and design. After this, it will be pretty straight forward to add a new data structure for visualization (at least for this tool) assuming that we can figure out how to visualize them.

So far, I've implemented a visualizer for trees and John has implemented a visualizer for linked lists. I believe Anna is working on sets and Doni is working on arrays. The code that we have finished can be viewed at https://github.com/Agizin/Algorithm-Visualization.

Tool 4: Adding SVG to Markdown

This tool needs to remove the code in the markdown file specific to our visualizer (i.e. probably the identifier code described under Tool 2) and add the SVGs file given as output by the previous component. This is the tool that we've worked least on but I also suspect that it will not give us much trouble.