**Design Doc**

**How the program should work:**

A user can load a list of functions or create functions in the application window, choose domains over which to evaluate them and plot the results. Multiple functions can be plotted at once, and the plot window allows for zooming, panning, saving screenshots and other common functionality. Evaluating a function over a domain creates a dataset which can be stored or accessed later. Both lists of functions and lists of datasets can be saved and imported.

A status bar informs the user of different application processes and outputs and errors and warnings are saved to an external log.

**Design principles:**

* Contractual programming is strictly enforced between all classes
* Defensive programming is enforced against invalid user actions
* Most functionality is implemented in a way that reduces memory overhead, however some decisions were made which affect memory to allow more flexibility in user interaction

GUI Window contains basic elements:

* List of functions
* List of datasets
* Plotting window
* Domain settings

**The general flow of modules and user actions is:**

Functions(domain) -> Dataset

Plotter(dataset) -> Plot

TODO: Block Diagram describing the flow of information / objects

For now, only a single plotting window is viewable at once. Datasets can be added to the plotting window by double-clicking them. The plotting window can also be cleared of all datasets.

To generate a new dataset, the user can double-click a function and the current domain settings will be used to generate the dataset. Or a user can select / highlight a function and click Generate.

* Future considerations: deleting single data-sets,

Functions can be loaded and saved to .json files. Users can make new functions via a pop-up dialog that requests certain fields to be filled out, then adds this function object to the list of functions.

* Future considerations: deleting single functions, handling multiple instances of functions.

Main Window Class:

* Sets up UI and button functionality and acts as container for the plotting window, list of functions, and list of datasets
* Animates status bar on user actions to give system information

Parser:

* Parses json files and returns list of **valid** functions
* Valid functions defined to have:
  + mathematically executable text that may or may not take two parameters, but contains x
  + name
* Parser will generate name if necessary, and also notes about the function if description is missing
* Parser automatically adds two zero parameters A and B if none are present in the function definition to functions
* Returns objects that implement the abstract base class TwoParameterFunction
* Annotates warnings and info about functions as it processes them, including:
  + Missing descriptions, unbounded at origin, un-importable functions

Functions:

* Implement TwoParameterFunction base class
* Protected setter and getter methods for parameters A and B
* Can take in a single value or a numpy array and return a numpy array along
* Store values for A and B, and optional description text
* Future considerations: current implementation relies on Equation library’s Expression object, which has well-defined scope and properties but our program should container-ize this more for safety (i.e. Expression object not fully explored in-depth)

Datasets:

* Name generated automatically from function names and current domain settings
* Contain two private fields, \_domain and \_range which are numpy arrays of the same length
* Two public fields, name and timestamp created
* Future considerations: handling of NaN values within the datasets, having functions that could zero-pad, smooth datasets, return discrete derivatives, generate interpolating functions over larger domains

Domain:

* Takes a start and stop and step-size
* Should be able to return different samplings, i.e. linear-spaced, exponential, Chebyeshev, etc, with number of sample points always equal to (stop-start)/step