**Summary of CZBiohub Programming Challenge**

**Steps to run the program**

The program has two dependencies which should be installed through pip. Note PyQt5 is a large library.

>> pip install Equation

>> pip install PyQt5

User should then run “CZApp.py” from the command line,

>> python CZApp.py

**How the program should work:**

The purpose of the program is to plot functions of two parameters, therefore we should make plotting functions and comparing them to each other as easy as possible. For this reason the program is structured around several core classes: Functions, Domains, and Datasets. These are explained further below.

A user should be able to easily generate plots, compare the effects of changing parameters on a function, and compare two functions over the same domain. Domains should be generated easily by user input. Descriptions of functions, domains, and datasets should be readily accessible via a single click by displaying in the status bar.

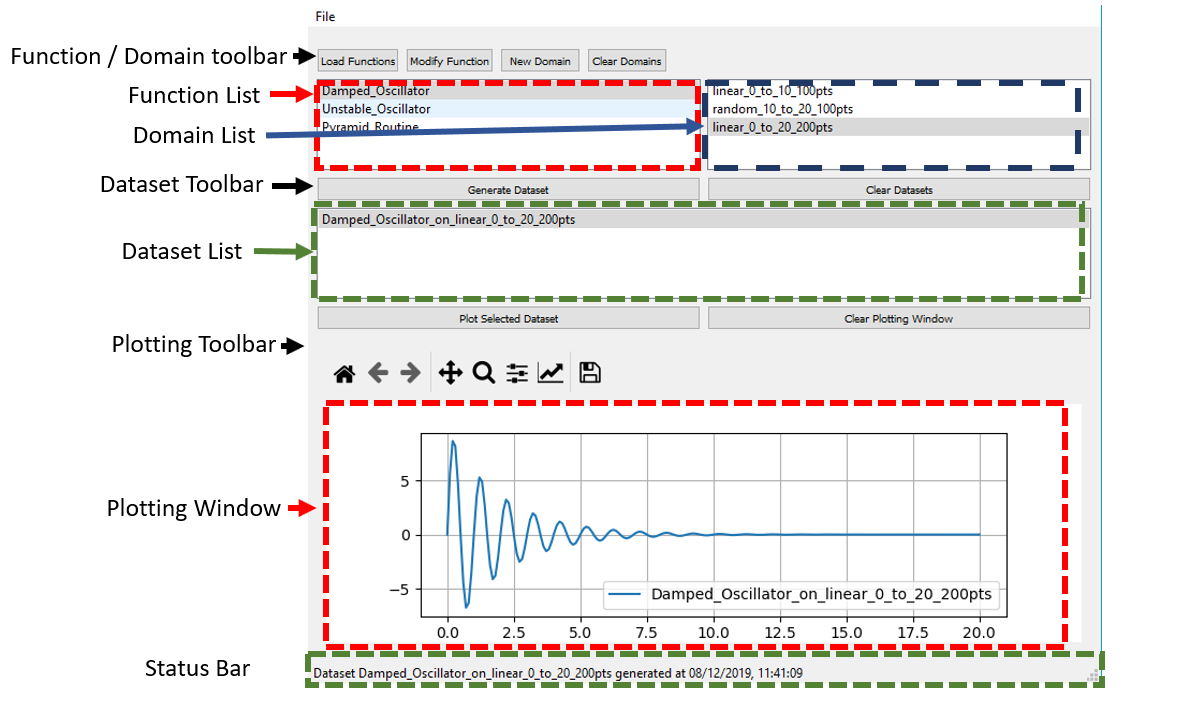
It should be intuitive and obvious how to use the program through layout and tooltips that appear when mouse hovers over a button. Someone should hypothetically be able to run the program and start using it without any instruction.

Program should fail ‘nicely’ as modern software does, with error messages stating what the problem was, avoiding crashing, reverting to safe/previous values. If something goes wrong, a status bar prints out what the likely cause of the error was and steps to remedy it.

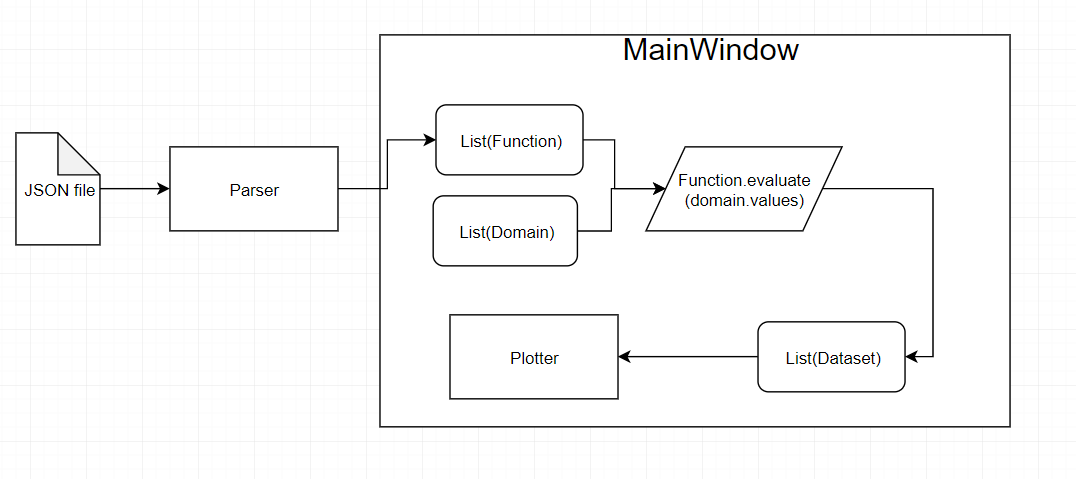
**Design principles:**

* Contractual programming is enforced wherever possible to encapsulate class functionality without breaking external representation
* Defensive programming is enforced against invalid user actions where ever possible through the use of try-except handling for better program stability, getter-setter methods, default initialization of functions
* Escalator principle: if something breaks, the program still works and simply displays a status message about what went wrong
* Sanitizing inputs wherever possible, e.g. when importing functions from a JSON file make few assumptions that the data is ‘good’, but be robust enough to handle missing attributes / auto-generate them if possible

**GUI Layout of major features**



**Conceptual Organization of Code**



The flow of information in the program is as follows:

1. A user may either load functions from a JSON file or use the default functions built-in. Functions populate a list the user can click on to select and view function information

2. A user may either create new domains or use the default domains built-in. Domains populate a list analogous to functions

3. With a function and domain selected, a user may generate a new dataset which is added to a List analogous to Functions and Domains.

4. With a dataset selected, a user can add it to the current plotting window. Plotter can support multiple plots and will append with legend if there is more than one dataset plotted.

5. User can generate as many datasets and domains as they wish to compare / evaluate functions on.

**Features that I anticipated would be useful to implement in the future, which informed design decisions:**

* Saving/Loading datasets, domains, functions to and from JSON files
* Creating multiple plotting windows
* Creating new functions from within the GUI
* Selecting / highlighting multiple functions at once, and running them all at once on a single domain
* Selecting / highlighting multiple datasets at once for plotting
* Replace ListView of objects with TableView, where columns have different attributes (name, description, etc) instead of just name for more visibility of information
* Displaying previous parameter values when a function is modified
* Mathematical operations on datasets (smoothing, FFT, etc)
* Provide methods to ‘update’ datasets to re-evaluate their original function after function parameters have changed, i.e. a “Update all Datasets” button would do this

**Class Descriptions & Important Class Methods**

MainWindow:

* 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
* Annotates warnings and info about functions as it processes them, including:
  + Missing descriptions, unbounded at origin, un-importable functions
* Parser.returnFunction: Returns objects that implement the abstract base class TwoParameterFunction, i.e. dynamically generates classes

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
* Store values for A and B, and optional description text
* Default initialization to prevent user-error in defining new functions from breaking program

Datasets:

* Name generated automatically from function names and current domain settings
* Contain two private fields, \_xvals and \_yvals which are numpy arrays of the same length that are accessed through getters.
* 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
* Returns with different samplings, i.e. linear-spaced, uniform random, exponential, Chebyeshev, etc