Design Documentation - Group 1
Blake, Connor, Jeff, Kelly, Kyle, Laura, Nina, Paul

Instructions:
Run from the command line with "python Controller.py". Fill in the text boxes as suggested and choose any of the options for plotting.

Division of Work:
Work was divided roughly according to the Model-View-Controller pattern:

- Model:
  - Parsing
    - Blake
  - Plot
    - Paul
    - Kyle
  - Refine
    - Jeff
  - Kelly
  - Nina
- View:
  - Conner
  - Laura
- Controller:
  - Paul
- Testing
  - Kelly
  - Jeff

Jeff also floated throughout the project to keep the different parts oganized and interfacing correctly.

Design:
The design of our project is centered around the idea of having the graphical frontend and the computational backend interfacing with each other only through one location. Conceptually, the Controller file handles all direct interaction with the GUI itself while the Model file manages all interaction on the backend. The specific ways Model and Controller interact with each other are listed below:

Controller's requests of Model	Model's requests of Controller
Refine - asks Model to initiate a refine with given parameters on the current formulation	Refine - show confirmation of successful refine
Plot - asks Model to return a png to be placed on the GUI	Plot - give Controller a png to display in the GUI

Reset - tells Model to reset all data to a clean state	Reset - update fields of Controller to clean state
Solve - asks Model to initiate a solve with given data	Solve - show confirmation for successful solve (energy error?) OR display error messages/highlight invalid fields
Load - asks Model to load a give file	Load - update fields to match current formulation OR display error message or unsuccessful load
Save - asks Model to save current data to a specific file	Save - show confirmation for successful save OR display error message

For further explanation of the design of our program, we will discuss the GUI in terms of its interactions with the Controller and the backend in terms of its interactions with Model.

Controller:

- Overall construction of GUI: The GUI utilizes kivy's .kv file to construct various elements and print text. Dropdown menus are designed to contain several buttons representing various user options. Text fields allow the user to input and edit data. All required data must be entered before the 'Solve' button is pressed. Faulty input is communicated to the user by changing a text field to red. Once solved the user has can either refine or plot by indicating their action in the respective dropdowns. The reset button may be pressed at any time and clears/resets all fields.
- Information from the fields in the GUI are collected when the user clicks solve via the eponymous method in ViewApp. Information is put in a dictionary as values where the keys are strings representing the names of the fields. If there is missing info, the method does not proceed and highlights missing areas in red for the user to fill out.
- Explanation of how the plots are displayed in the GUI (Laura or Paul)
- Methods for subclasses of Button and TextInput classes are contained here for highlighting invalid/empty fields in red and clearing data/highlights

Model:

- Model uses the memento pattern in order to store all the data pertaining to the current formulation and to handle loading and saving of formulations.
    - Data storage structure
        - InputData is an object for storing the user input data as well as the form. This is the information that is saved and loaded. The convention maintained for storing the user input in the variables dict is as follows:
            - "stokes" = boolean

                - "reynolds" = float/int
                - "transient" = boolean
                - "meshDimensions" = [float, float]
                - "numElements" = [int, int]
                - "polyOrder" = int
                - "inflow" = strings [(inflowRegions, inflowX, inflowY)]
                - "inflowRegions" = SpatialFilter
                - "inflowX" = SpatialFilter
                - "inflowY" = SpatialFilter
                - "outflowRegions" = SpacialFilter
                - "outflow" = strings [outflowRegions]
        - Load & Save
            - Save works for both Stokes and NStokes formulations, but Load does not currently pass all tests
- Initiating a solve
    - Testing data/Passing errors back to Controller
        - Raw data is collected as strings when solve is pressed and then passed to Model and on to ParsingUtils to be tested for validity. If the input is invalid, a dict of errors is passed back to the Controller so the appropriate field can be highlighted in the GUI for the user. If the data is correct, the form is created and solved according to it.
    - FunctionParser
    - ConditionParser
    - Storing data to memento
    - Solving in FormUtils
- Resetting data
- Refine - with FormUtils
- Plotting
    - The Plotter has one main plot method which is called by the model. It then will call one of several other methods depending on what type of plot is requested.
    - When plot is called by the model, the current form is passed in. The Plotter then creates the necessary components i.e. mesh, u1_soln and passes it to the correct plot method. The plot is created and its figure saved to the disk in png format, where it is then picked up by the controller and saved as the current plot.