Testing Plan - Group 1
Blake, Connor, Jeff, Kelly, Kyle, Laura, Nina, Paul

- runTests.py
  - Driver
- TestConditionParser
  - onlyXEquals()
    - test if returned filter for "x = 0" accurately matches points for x = 0
  - onlyXGreater()
    - test if returned filter for "x > 0" accurately matches points for x > 0
  - onlyXLess()
    - test if returned filter for "x < 0" accurately matches points for x < 0
  - onlyYEquals()
    - test if returned filter for "y = 0" accurately matches points for y = 0
  - onlyYGreater()
    - test if returned filter for "y > 0" accurately matches points for y > 0
  - onlyYLess()
    - test if returned filter for "y < 0" accurately matches points for y < 0
  - XY()
    - test if returned filter for "x=0,y<0" accurately matches points for x = 0 and y < 0
  - YX()
    - test if returned filter for "y=0,x>0" accurately matches points for x > 0 and y = 0
  - Doubles()
    - test if returned filter for "x>0.0,y>0.0" accurately matches points for x > 0 and y > 0
  - HalfDoubles()
    - test if returned filter for "x>0.,y>0." accurately matches points for x > 0 and y > 0
  - ManyArgs1()
    - test if returned filter for "y=0,x>0,y=3" accurately matches points for y = 0, x > 0, and y = 3
  - ManyArgs2()
    - test if returned filter for "x>0,x<2,y<2" accurately matches points for x > 0, x < 2, and y < 2
  - OutOfOrder()
    - test if "0=x" throws a ValueError
  - NoComma()
    - test if "x=0 y<2" throws a ValueError
- TestDataUtils
  - getDataList() & getExpectedVars()

        - call each method and confirm that the list and dict contain the expected data
    - populateInputData()
        - Create an instance of InputData and call populateInputData on it, then confirm that inputData contains each expected key and value.
    - generateForm()
        - Generate each type of form to confirm that generateForm does not crash. This method is primarily a switch on the parameter type string. The real testing happens in the following three tests.
    - For the following three tests, create a testForm using DataUtils methods and create an expectedForm using PyCamellia methods. Compare their ElementCount, GlobalDofCount, and EnergyError to confirm that they are equal.
    - generateFormStokesTransient()
        - Follow above testing approach with testForm and expectedForm as transient Stokes forms.
    - generateFormStokesSteady()
        - Follow above testing approach with testForm and expectedForm as steadyStokes forms.
    - generateFormNavierStokesSteady()
        - Follow above testing approach with testForm and expectedForm as steady Navier Stokes forms.
- TestFormUtils
    - For the following tests, create a testForm using DataUtils methods and create an expectedForm using PyCamellia methods. Compare their ElementCount, GlobalDofCount, and EnergyError to confirm that they are equal.
    - energyPerCell()
        - Create form using steadyLinearInit(). Add a wall and an inflow condition. Use energyPerCell() to get the list of energy error per cell of form. Confirm that for each cell, the energy error is the same as foo's.
    - formInit()
        - Using the testing approach described above, test formInit with a transient linear, steady linear, and steady nonlinear form.
    - steadyLinearInit()
        - Create form using steadyLinearInit() and test as previously described.
    - transientLinearInit()
        - Create form using transientLinearInit() and test as previously described.
    - steadyNonlinearInit()
        - Create form using steadyNonlinearInit() and test as previously described.
    - autoRefine()

        - pass
    - linearHAutoRefine()
        - Create form using steadyLinearInit(). Add a wall and an inflow condition. Refine form using steadyLinearHAutoRefine(). Test as previously described.
    - linearPAutoRefine()
        - Create form using steadyLinearInit(). Add a wall and an inflow condition. Refine form using steadyLinearPAutoRefine(). Test as previously described.
    - linearHManualRefine()
        - Create form using steadyLinearInit(). Add a wall and an inflow condition. Retrieve the cell IDs for all active cells. Use those cell IDs to refine form using linearHManualRefine(). Test as previously described.
    - linearPManualRefine()
        - Create form using steadyLinearInit(). Add a wall and an inflow condition. Retrieve the cell IDs for all active cells. Use those cell IDs to refine form using steadyLinearPManualRefine(). Test as previously described.
    - nonlinearHAutoRefine()
        - Create form using steadyNonlinearInit(). Add a wall and an inflow condition. Refine form using steadyNonlinearHAutoRefine(). Test as previously described.
    - nonlinearPAutoRefine()
        - Create form using steadyNonlinearInit(). Add a wall and an inflow condition. Refine form using steadyNonlinearPAutoRefine(). Test as previously described.
    - nonlinearHManualRefine()
        - Create form using steadyNonlinearInit(). Add a wall and an inflow condition. Retrieve the cell IDs for all active cells. Use those cell IDs to refine form using steadyNonlinearHManualRefine(). Test as previously described.
    - nonlinearPManualRefine()
        - Create form using steadyNonlinearInit(). Add a wall and an inflow condition. Retrieve the cell IDs for all active cells. Use those cell IDs to refine form using steadyNonlinearPManualRefine(). Test as previously described.
    - solve()
        - pass
    - steadyLinearSolve()
        - Create form using steadyLinearInit(). Add a wall and an inflow condition. Solve using steadyLinearSolve() and test as previously described.
    - transientLinearSolve()
        - Create form using transientLinearInit(). Add a wall and an inflow condition. Solve using transientLinearSolve() and test as previously described.
    - nonlinearSolve()

        - This is a helper method, it's main difference from steadyNonlinearSolve() is that it is much less verbose.
        - Create form using steadyNonlinearInit(). Add a wall and an inflow condition. Solve using nonlinearSolve() and test as previously described.
    - steadyNonlinearSolve()
        - Create form using steadyNonlinearInit(). Add a wall and an inflow condition. Solve using nonlinearSolve() and test as previously described.
- TestInputData
    - Memento
        - get() & set()
            - Create an instance of InputData using stokes and create a memento. Call get() and confirm that the resulting dataList contains stokes and does not contain nStokes. Then call set() and give it nStokes. Call get() again and confirm that the resulting dataList now contains nStokes and not stokes.
    - InputData
        - init()
            - Create an instance of InputData using stokes and create a memento. Call get() and confirm that the resulting dataList contains stokes.
        - createMemento()
            - Create an instance of InputData and create a memento. Confirm that the memento is not None and that it contains the initial stokes value in its DataList.
        - setMemento()
            - Create an instance of InputData. Set it's form and add several variables. Create a memento for it and use it to call setMemento() on a new instance of InputData, InputDataNew. Create a memento of InputDataNew and confirm that the resulting dataList contains all of the added variables from the InputData memento.
        - addVariable() & getVariable()
            - Create an instance of InputData using stokes and call addVariable() with transient. call getVariable() for transient and stokes to confirm that they are equal.
- TestModel
    - refine()
        - pass
    - plot()
        - Make a form and call plot with it. Confirm results visually.
    - reset()
        - Make a model and populate its input data, then assert that its InputData variable dictionary is not empty. Next call reset on model and assert that its InputData variable dictionary is empty.

    - solve()
        - pass
    - testData()
        - pass
    - storesData()
        - pass
    - saveStokes()
        - Create a stokes form and a model with the appropriate information in its InputData object, then save it. If it doesn't crash, it was successful. Also, confirm save visually in file.
    - saveNStokes()
        - Create a navier stokes form and a model with the appropriate information in its InputData object, then save it. If it doesn't crash, it was successful. Also, confirm save visually in file.
    - loadValidStokes()
        - First call saveStokes test in order to create something to load, then load it. If it doesn't crash, it was successful.
    - loadValidNStokes()
        - First call saveNStokes test in order to create something to load, then load it. If it doesn't crash, it was successful.
    - loadInvalid()
        - Try to load an invalid file, if it loads raise an error, otherwise return and consider it a success that it failed to load an invalid file.
- TestParseFunction
    - For the following tests, define a simple function with the described operations. Calculate the answer using basic python math operations. Confirm that the parsed and evaluated answer to the PyCamellia function equals the calculated answer.
    - add()
        - Addition
    - subtract()
        - Subtraction
    - divide()
        - Division
    - multiply()
        - Multiplication
    - exponent()
        - Exponentiation
    - negative()
        - Negation
    - parenMultiplty()
        - Parenthetical multiplication, 3(x)
    - xAndY()
        - x and y as parameters
    - noParens()

        - No parenthesis
    - doubles()
        - Doubles and integers
    - parenthesis1()
        - Parenthesis
    - halfDoubles()
        - Poorly inputted doubles, 1. or .4
    - testBasicRoberts()
        - Dr. Robert's test case
    - testBasicRoberts2()
        - Dr. Robert's second test case
    - testBasicRoberts3()
        - Dr. Robert's third test case
    - ePowerOfTen()
        - e used in the power of ten
- TestParsingUtils
    - stringToDims()
        - Confirm float values added for x and y match the string passed as a parameter. Confirm raises exception when input string format is incorrect.
    - stringToElements()
        - Confirm int values added for x and y match the string passed as a parameter. Confirm raises exception when input string format is incorrect or contains float values.
    - stringToInflows()
        - Test if returned filter accurately matches points for x < 8 and function for X and Y evaluate correctly
    - stringToOutflows()
        - Test if returned filter accurately matches points for x < 0
    - formatRawData()
        - Populate rawData and call formatRawData on it, then confirm that the correct data is returned in testData
    - checkValidInputWithValidData()
        - Populate rawData with valid data, then call checkValidInput on it and confirm that there are no errors found
    - checkValidInputWithInvalidData()
        - Populate rawData with invalid data, then call checkValidInput on it and confirm that there are errors found
    - checkValidFile()
        - Check for a the file being tested, ParsingUtils.py, and assert that the result is true.
        - Check for an arbitrary file and assert that the result is false.
- TestPlotterError & TestPlotterP & TestPlotter & TestPlotterStream
    - All tests run different plot on a form and success is confirmed manually