1. Import all of the solvers that we need
    a. `from pyomo.environ import *`
    b. `from pyomo.opt import SolverFactory`
    c. `import pandas as import pd`
    d. `opt = SolverFactory('glpk')`
2. Import the data the we need and set it up in a data frame
    a. `dfsets = pd.read_excel("sample_data/DataFile.xlsx", sheet_name = "Sets")`
    b. `dfparams = pd.read_excel("sample_data/DataFile.xlsx", sheet_name = "Parameters")`
3. Check that the data frames look the way I want them to
    a. `print(dfsets)`
    b. `print(dfparams)`
4. Prompt the user to check the data
5. Have the user tell the code to continue
6. Solve for Stage 1 using the Pyomo Library
    a. Declare the model
        i. `model = AbstractModel()`
    b. Declare the sets that are needed for stage one
        i. Y: residency years
            1. `model.Y = Set(dfparams[Year Level].unique())# residency year`
        ii. R: residents
            1. `model.R = Set(dfparams[RESIDENT ID])        # set of residents`
        iii. R_i: residents of year i
            1. `model.R_i = Set(model.Y)     # set of residents of year i`
        iv. C: clinic units
            1. `model.C = Set(dfsets[Units])       # set of clinic units`
        v. G: clinic rotational groups
            1. `model.G = Set(dfsets[Clinic_Group])       # clinic rotational groups`
        vi. T: weeks of rotation in a year
            1. `model.T = Set(initialize = "1,2,3,4,5,6,7,8,9,10")       # Is there a better way to do this? # set of weeks`

vii. V: Vacation Unit

   1. `model.V = Set("V")      # vacation unit`

viii. A: Ambulatory care units

   1. `model.A = Set()      # ambulatory care units`

ix. I: inpatient care units

   1. `model.I = Set()      # inpatient care units`

x. E: elective units

   1. `model.E = Set()      # elective units`

xi. H_u: type of resident-year allowed to do a rotation in a unit

   1. `model.H_u = Set()    # resident year needed for unit u`

xii. U: units where rotations can be scheduled, U = A U I U E V C

   1. `model.U = Set(model.A, model.I, model.E, model.V, model.C)`
      `# set of units`

xiii. Q: subset of units that require having at least one group of residents in rotation every single week

   1. `model.Q = Set()      # units requiring a group every week`

xiv. N: Night shift rotations

   1. `model.N = Set()      # night shift rotations`

xv. Theta: subset of units considered in stage 1 aka critical units

   1. `model.theta = Set()  # subset considered for phase 1`

xvi. S: standby unit

   1. `model.S = Set()      # standby unit`

xvii. P: clinic rotational policy (“4+1”, “8+2”)

   1. `model.P = Set()      # clinic rotational policy`

c. Declare the parameters

i. Pi

ii. S

iii. $h_{g,c}$

iv. $l_{r,g}$

v. Zeta_u

vi. alpha_u

vii. lambda_u

viii. $\text{phi}_{i,u}$

ix. tau

x. $omega_{r,u}$

xi. v

xii. m

xiii. $psi_{r,t}$

d. Declare the PRIME decision variables

i. $Xprime_{r,u,t}$

1.
```
Xprime = var(r, u, t, within=binary)  # 1 if resident r
rotates in unit u in week t
```

ii. $Wprime_{r,u,t}$

1.
```
Wprime = var(r, u, t, within=binary)  # 1 if resident r
starts in unit u in week t
```

iii. $Deltaprime_{u,t}$

1.
```
deltaprime = var(u, t, within=binary)  # 1 if unit u begins
a rotation in week t
```

e. Declare the objective function(s)

i. Objective function for vacation

```
Def objective1_rule(model):
        Return sum(model.psi * model.X for r in model.R for u in
model.V for t in model.T)
    model.Schedule = Objective(rule=objective1_rule, sense =
maximize)
```

f. Declare the constraints

i. Hard

1. Constraint 9
2. Constraint 10 might be important since they deal with the 4+1, 8+2 system
3. Constrain 11 deals with making sure the residents stay in their group.
4. Constraint 12

```
def cons_12(model):
    return(sum(model.I * model.h * model.W for r in
model.R for g in model.G) >= model.m)
Model.cons_12 = Constraint(rule = cons_12)
```

5. Constraint 13

```
def cons_13(model):
        return(sum(model.x * model.I for r in
model.R) >= 1)
    Model.cons_13 = Constraint(rule=cons_13)
```

6. Constraint 14

```
def cons_14(model):
    return(min(model.thing)<= sum(model.X for r in
model.R)<=max(model.thing))
Model.cons_14 = Constraint(rule = cons_14)
```

7. Constraint 15

```
def cons_15(model):
    return(sum(model.X for u in model.U) == 1)
Model.cons_15 = Constraint(rule = cons_15)
```

8. Constraint 16 (this one I'm not sure on)

```
def cons_16(model):
    return(sum(Model.X for r in model.Ri) ==0)
Model.cons_16 = Constraint(rule = cons_16)
```

9. Constraint 17

10. Constraint 18

ii. Soft

iii. Non-negativity

1. Non negativity constraints can be handled by defining the types of variables that can be held in each object

g. Run the model

7. Return the PRIME decision variables

a. return(some stuff)

8.

a. Declare everything I will need to use for Stage 2 using the Pyomo Library

   i. Declare the model

   ii. Declare the FINAL decision variables

      1. $X_{r,u,t}$

         a. `X = var(r, u, t, within=binary)  # 1 if resident r rotates in unit u in week t`

      2. $W_{r,u,t}$

         a. `W = var(r, u, t, within=binary)  # 1 if resident r starts in unit u in week t`

      3. $Delta_{u,t}$

         a. `delta = var(u, t, within=binary)  # 1 if unit u begins a rotation in week t`

   iii. Declare the constraints

      1. Hard

         a. Constraint 6

```
def cons_6(model):
    return(model.W >= model.W0)
Model.cons_6 = Constraint(model.Quality, rule =
cons_6)
```

         b. Constraint 7

```
def cons_7(model):
    return(model.X >= model.X0)
Model.cons_7 = Constraint(model.Quality, rule =
cons_7)
```

         c. Constraint 8

```
def cons_8(model):
    return(model.delta >= model.delta0)
Model.cons_8 = Constraint(model.Quality, rule =
cons_8)
```

      2. Soft

      3. Non-negativity

         a. These will be taken care of when defining possible variables within each object

        iv. Declare
        v. Create a for loop that will iterate through all of the objective functions as a permutation
            1. Return the results of every iteration of the for loop as a text file in some folder

b. Use pandas/python to take all of those txt files and turn them into a pretty Excel workbook where each txt file is a different sheet
c. Show user where to get the workbook or something
d.