```
import matplotlib.pyplot
import pandas
import math

class timestep:

    # house constants
    SURFACES = [(15., 1400.), (2., 250.),
                (60., 2000.), (30., 2000.)] # R-value, ft^2
    HOUSE_TEMP = 70. # degF, constant house temperature

    # general constants
    BTUHR_PER_WATT = 3.41 # (Btu/hr) / Watt
    CP_WATER = 1.0 # Btu/(lb*degF), specific heat of water
    INTERVAL = 60. # minutes

    # HP performance model
    # https://www.researchgate.net/publication/328096017, figure 2
    # at 75C output; (degC, Watts)
    ELECTRIC_INPUT = [[-7, 8300],
                      [12, 6050]]

    # at 75C output; (degC, COP)
    HP_COP = [[-7, 1.8],
              [12, 2.4]]

    def __init__(self, radius):
        # class variables
        self.temperature_air = None
        self.hp_on = True

        # converts performance data to US units
        self.ELECTRIC_INPUT_US = [] # degF, Btu/hr
        self.HP_COP_US = [] # degF, COP
        for x in self.ELECTRIC_INPUT:
            self.ELECTRIC_INPUT_US.append([x[0] * 1.8 + 32,
                                           x[1] * self.BTUHR_PER_WATT])
        for x in self.HP_COP:
            self.HP_COP_US.append([x[0] * 1.8 + 32, x[1]])

        # tank constants and geometry
        self.RADIUS = radius # ft
        self.HEIGHT = 2 * self.RADIUS # ft (square cylinder tank)
        self.TANK_SA = 2 * math.pi * self.RADIUS * self.HEIGHT # ft^2, sides
        self.TANK_EA = math.pi * (self.RADIUS ** 2) # ft^2, top/bottom
        self.VOLUME = self.TANK_EA * self.HEIGHT # ft^3
        self.MASS_WATER = self.VOLUME * 62.4 # lbs
        self.R_TANK = 60. # R-value
        self.R_TANK_BOTTOM = 10. # R-value

        print('Gallons: ', self.VOLUME * 7.48)
        print('ft^3: ', self.VOLUME)

        self.calculate_insulation()
```

```
    # calculate coefficient from house surfaces array
    # input elements in form [R-value, corresponding area]
    def calculate_insulation(self):
        sum = 0.
        for x in self.SURFACES:
            sum += x[1] / x[0]
        self.HOUSE_LOSS_COEFFICIENT = sum

    # update the outside temperature
    def change_conditions(self, new_temp, new_rad):
        self.temperature_air = new_temp # degF

    # interpolate or extrapolate tabulated linear models
    def interpolate(self, temp, table):
        if temp < table[0][0]:
            index = 1
        elif temp > table[-1][0]:
            index = -1
        else:
            index = 0
            while temp < table[index][0]:
                index += 1

        slope = (table[index][1] - table[index - 1][1]) / \
            (table[index][0] - table[index - 1][0])
        output = slope * (temp - table[index][0]) + table[index][1]
        return output

    def calculate_cop(self, temp):
        return self.interpolate(temp, self.HP_COP_US)

    # calculates the electric load
    def calculate_input(self, temp):
        return self.interpolate(temp, self.ELECTRIC_INPUT_US) # Btu/hr

    # calculates heat produced by the HP
    def calculate_heat_pump(self):
        if self.hp_on:
            return self.calculate_cop(self.temperature_air) * \
                self.calculate_input(self.temperature_air) # Btu/hr
        return 0

    # calculates the heat demand of the house
    # returns 0 if the demand is negative
    def calculate_demand(self):
        hourly = self.HOUSE_LOSS_COEFFICIENT * \
            (self.HOUSE_TEMP - self.temperature_air) # Btu/hr
        if hourly > 0:
            return hourly # Btu/hr
        return 0

    # calculates the loss of the storage tank
    def calculate_loss(self, temp_tank):
        delta = temp_tank - self.temperature_air
        sides = self.TANK_SA * delta / self.R_TANK
        top = self.TANK_EA * delta / self.R_TANK
        bottom = self.TANK_EA * (temp_tank - 50.) / self.R_TANK_BOTTOM
        return sides + top + bottom
```

```
    # calculates change in tank temperature for the interval
    def calculate_temp_change(self, temp_tank):
        heat_change = self.calculate_heat_pump() - \
            self.calculate_demand() - self.calculate_loss(temp_tank)
        heat_change *= self.INTERVAL / 60. # Btu
        # deltaT = deltaQ / (M*Cp)
        temp_change = heat_change / (self.MASS_WATER * self.CP_WATER)
        return temp_change # degF
# end timestep

# loads the file into a dataframe
def load_data(filename):
    data = pandas.read_csv(filename)
    return data

# forecasted weather data
# looks ahead n periods for temperature below threshold,
# starting at index. returns true if found
def look_ahead(data, index, threshold, n):
    index += 1
    result = False
    # returns false if out of bounds
    if data['temp_f'].size < index + n:
        return False
    if data['temp_f'].iloc[index:(index + n)].min() < threshold:
        result = True
    return result

# implements the time loop for the model
# collects results and visualizes
def model(radius):
    data = load_data('abq-data/data.csv')

    # add columns to dataframe for control logic and visualization
    data['temp_f'] = (data['Temperature'] * 1.8 + 32.)
    data['temp_tank'] = 0.
    data['temp_tank_change'] = 0.
    data['heat_demand'] = 0. # house end heat demand
    data['hp_on'] = 0 # the boolean state of the heat pump
    data['heat_loss'] = 0. # heat loss of the tank
    data['hp_output'] = 0.
    data['days'] = 0.
    data['low_temp'] = 130.

    step = timestep(radius)

    temperature_in_tank = 150. # initial tank temp 150F
```

```
    # loops over every row in the dataframe
    for index, row in data.iterrows():
        # update the weather conditions
        step.change_conditions(row['temp_f'], row['DNI'])

        high_temp = 160.
        low_temp = 130.

        # heat pump cycles to keep the minimum tank temperature higher
        # if the forecasted temperatures drop below 20F
        if look_ahead(data, index, 20., math.ceil(step.VOLUME / 6.8)):
            low_temp = 150.
            data.at[index, 'low_temp'] = low_temp

        # heat pump is controlled to cycle between the high and low temp
        if temperature_in_tank > high_temp:
            step.hp_on = False
        elif temperature_in_tank < low_temp:
            step.hp_on = True

        # defines the low operating temperature of the heat pump
        # it decreases if the forecasted temperature is too low
        low_air_temp = 30.
        if look_ahead(data, index, 20., math.ceil(step.VOLUME / 6.8)):
            low_air_temp = 20.

        # heat pump will not run below the low operating temperature
        if step.temperature_air < low_air_temp:
            step.hp_on = False

        # if the tank temperature falls below 120F and
        # the air temperature allows for operation, the HP turns on
        # HP cannot operate below -13F
        if step.temperature_air > -13. and temperature_in_tank < 120.:
            step.hp_on = True

        # if it is never below 62F in the forecasted temp,
        # it is assumed that heating is not required and HP is turned off
        if not look_ahead(data, index, 62., math.ceil(step.VOLUME / 6.8)):
            step.hp_on = False

        # calculate the change in tank temperature
        # and add it to the total temperature in the tank
        data.at[index, 'temp_tank_change'] = \
            step.calculate_temp_change(temperature_in_tank)
        temperature_in_tank += data.at[index, 'temp_tank_change']

        # record tank and HP data for visualization
        if step.hp_on:
            data.at[index, 'hp_on'] = 1
        data.at[index, 'temp_tank'] = temperature_in_tank
        data.at[index, 'heat_demand'] = step.calculate_demand()
        data.at[index, 'heat_loss'] = step.calculate_loss(temperature_in_tank)
        data.at[index, 'hp_output'] = step.calculate_heat_pump()
        data.at[index, 'cop'] = step.calculate_cop(step.temperature_air)
        data.at[index, 'input'] = step.calculate_input(step.temperature_air)
        data.at[index, 'days'] = index / 24.
```

```
    # scaled HP boolean state
    # matplotlib.pyplot.plot(data['days'], data['hp_on'] * 50)

    matplotlib.pyplot.plot(data['days'], data['temp_tank'])
    matplotlib.pyplot.plot(data['days'], data['temp_f']) # outside temperature
    matplotlib.pyplot.plot(data['days'], data['low_temp'])
    matplotlib.pyplot.show()

    """
    # determines total power consumption by HP or resistance heating
    print('kWh-hp: ', (data['hp_on'] * data['input']).sum() / \
        (step.BTUHR_PER_WATT * 1000))
    print('kWh-res: ', (data['heat_demand'].sum() / (step.BTUHR_PER_WATT * 1000)))

    # determines monthly HP and resistance equivalent power demands
    demands = {}
    inputs = {}
    for i in range(2011, 2023):
        demands[i] = {}
        inputs[i] = {}
        for j in range(1, 13):
            demands[i][j] = 0.
            inputs[i][j] = 0.
    for index, row in data.iterrows():
        n = data.at[index, 'Year']
        m = data.at[index, 'Month']
        demands[n][m] += data.at[index, 'heat_demand']
        inputs[n][m] += data.at[index, 'input'] * data.at[index, 'hp_on']

    for i in range(2011, 2023):
        for j in range(1, 13):
            demands[i][j] /= (step.BTUHR_PER_WATT * 1000)
            inputs[i][j] /= (step.BTUHR_PER_WATT * 1000)

    # prints peak monthly HP power consumption
    df = pandas.DataFrame(inputs)
    for index, row in df.iterrows():
        print(index, row.max())

    # prints peak monthly resistance heating pwoer consumption
    df = pandas.DataFrame(demands)
    for index, row in df.iterrows():
        print(index, row.max())
    """

    # masks summertime phantom heat demand data
    for index, row in data.iterrows():
        if not look_ahead(data, index, 62., math.ceil(step.VOLUME / 6.8)):
            data.at[index, 'temp_tank'] = 130.
            data.at[index, 'heat_demand'] = 0.

    # calculates the heat demand that exists when HP can't operate (below -13F)
    sum = 0.
    for index, row in data.iterrows():
        if data.at[index, 'temp_f'] < -13:
            sum += data.at[index, 'heat_demand']
    print(sum / (step.BTUHR_PER_WATT * 1000))

model(3.)
```