def splitRoute(): #definition for subroutine

route = [] #creates an empty list

#f=open('http://www.mdl.nws.noaa.gov/~naefs\_ekdmos/text/naefs\_tempcdf\_00.txt/','r')

#f=open('http://www.ems.psu.edu/~young/meteo473/Data/EKDMOS00.txt/','r')

f=open("/home/meteo/sdg152/Meteo\_473/MS3/AllStations.txt","r") #grab the txt file and opens it

for line in f.readlines(): #loops through all of the lines in f

line = line.strip('>') #gets rid of something on the end

line = line.replace("\t","") #gets rid of the escape character t

line = line.replace("\n","") #gets rid of the escape character n

apple = line.split() #splits each line off

route.append([apple]) #appends each line to the end of the list apple

f.close() #closes the file f

return route #returns the whole list appended inside apple

Subroutine or function definition: sets an empty list (route), opens the text file used, reads each line, strips off the extras (tabs and new lines), appends each line to the end of route, and returns route when main program calls.

paul = [] #creates an empty list

paul = splitRoute() #calls the function and puts it into a variable

pricklypear = paul[1][0] #puts a single line in a variable

aprofit = 0 #sets the accumulator

Empty list Paul is used to hold the information from the function call. The line used here corresponds to the first line of station K11R, at the first hour. This line is put in a temporary holder, pricklypear. Any line could be used. The first index is for the station model (every 76 lines is a different station model), the second corresponds to the hour. Program could be changed to loop through all of the hours for a particular station model. Just exchange the zero in the second index with a variable that loops through 75 values (2-76 or any multiple of 76).

f = open('/home/meteo/sdg152/Meteo\_473/MS4/MS4python.txt', 'w') #overwrites the textfile

f.write(' ') #with a space

f.close #then closes

f = open('/home/meteo/sdg152/Meteo\_473/MS4/python.txt', 'a') #opens the textfile and allows append

Opens and closes the output file so that it overwrites each time the program is run. Then opens the text file again and allows appending.

acost=input('What is the cost of you electricity:') #user input for cost, string

bcost = float(acost) #change string to float

Outputs request for cost of electricity. The input is assigned to acost (a string) then changed to a float and stored in bcost.

p=0 #sets p, a counter

Temp = [] #creates an empty list

for r in range(15): #sets for loop to run 13 times

if p >= 2 and p <= 12: #these are the numbers we need to use

apple = float(pricklypear[p]) #changes string to floating point number

Temp.append(apple) #adds the new floats to the end of the list

p = p + 1 #increments p

else: #takes care of the numbers I don't need

p = p + 1 #increments p

Assigns a counter for pulling the values out of the temporary holder pricklypear, changes the values from string to float and appends each value onto the empty list Temp, and increments p.

apple = Temp[0] #used because I can't use Temp[] in equation

pear = Temp[1]

x = pear+(2\*(apple-pear)) #equation for interpolate 00 value

Temp.insert(0,x) #inserts the 00 value at the beginning

apple = Temp[10] #used because I can't use Temp[] in equation

pear = Temp[11]

x = pear+(2\*(pear-apple)) #equation for interpolate 1h value

Temp.insert(len(Temp),x) #inserts the 1h value at the end, appends

Interpolating the 00 and 100 value

cdf = [.01, .05, .05, .1, .1, .1, .1, .1, .1, .1, .1, .05, .05] #creates a list for the cdf values

Creates a list of the cdf

apple = Temp[12] #sets tthe top value for math

pear = Temp [0] #sets the bottom value for math

a = (apple-pear)/0.10 #number of bins, float

n = int(a) #changing a float to an integer

T = Temp[0] #Sets T for incrementation

Ntemp = [] #creates an empty list for New Temp

Ntemp.insert(len(Ntemp),0) #Sets first list item in new temp to zero

CDF = [] #creates an empty list for CDF

CDF.insert(len(CDF),0.0) #sets the CDF, used to make pizza

Tbot = Temp[0] #sets the lower bound on temp

Cbot = cdf[0] #sets the lower bound on cdf

j = 0 #counter for changing Ttop and Ctop

Ttop = Temp[j] #sets the upper bound on temp

Ctop = cdf[j] #sets the upper bound on cdf

The set up: Creates the number of bins, the empty lists Ntemp and CDF, Sets initial values

for i in range (n): #we are going to loop through all n bins

T = T + 0.1 #increment temperature

if T > Ttop: #if what we increment equals the top of the bin

j = j + 1 #increment my changer

Tbot = Ttop #change top to bottom for temp

Ttop = Temp[j] #and create a new top for temp

Cbot = Ctop #Change top to bottom for cdf

Ctop = cdf[j] #and create a new top for cdf

Changes the top of the bin, if the value, T, incrementing becomes equal to the current top of the bin

pizza=Cbot+(((T-Tbot)/(Ttop-Tbot))\*(Ctop-Cbot)) #this is my interpolation

CDF.insert(len(CDF),pizza) #adds the interpolation to the end of the list

k = i - 1 #counter for difference

pdf = CDF[i] - CDF[k] #difference of CDF, doesn't need list, used once

Math to change Cbot and create a list of CDF values

Ntemp.insert(len(Ntemp), T) #creates list for the midpoint calculation

Mid = (Ntemp[i]+Ntemp[k])/2 #middle of bin, doesn’t need to be list, used once

price = 1.36 \*(abs(Mid-55)) +20 #price of the middle

Creates a list for Ntemp, calculate midpoint and price

Pindicator = (price - bcost) #calculates the profit

popcorn = Pindicator \* pdf #calculates the profit for each day

dailyp = format(popcorn, '.2f') #sets 2 decimal places

aprofit += Pindicator \* pdf #calculates profit for the whole week

theprofit = format(aprofit, '.2f')

Calculates the difference between price and cost, and formats the output.

if dailyp < 0: #when profit is less than 0

f.write("You net a loss of ") #displays loss

else: #when profit is greater than 0

f.write("You net a profit of ") #displays output

f.write(dailyp) #displays amount in dailyp

f.write(" \tThe accumulated profit is ") #output for accumulated profit

f.write(theprofit) #accumulated profit

f.write("\n") #makes a space

f.close #closes the textfile

writes the profit to the file and closes the file after all of the bins have been calculated.