Weather_input.py

Purpose:

The purpose of weather_input.py is to take the curated weather data and compile it in way that data_gen_mk2.py can use it. First, the program downloads the latest weather data from GitHub and saves them to the users drive. Then, it uses the pertinent data to create weather objects that are then inserted into a weather list, weather_list[].

Libraries used:

Compared to data_gen_mk2.py, the number of libraries is far fewer here. This is due to the limited scope of what this program aims to accomplish.

```
import requests
import os
import xlrd
import math
```

- Import requests – Allows simple http requests. Here it is used to download the weather excel data for use on the machine.
  https://docs.python-requests.org/en/latest/
- Import os - Allows for the use of operating system dependent functionality. Used for getting current working directory.
  https://docs.python.org/3/library/os.html
- Import xlrd – Allows for reading the weather excel data that is downloaded.
  https://pypi.org/project/xlrd/
- Import math – A handy module that was really only used for its floor and ceiling operations.
  https://docs.python.org/3/library/math.html

Classes:

```
class weather:
    def __init__(self, temp, rain, snow):
        self.temp = temp
        self.rain = rain
        self.snow = snow
        self.temp_avg = 0
        self.weather_reduction = 1
```

Only one class was used for this, weather. In total, weather objects have five attributes associated with them. Temperature, rain, snow, temperature average, and weather reduction.

- temp – This is the average temperature for the day.
- rain - The amount of rain for the day.
- snow – The amount of snow for the day.
- temp_avg – A thirty-day average of the temperature. Used to detect deviant weather days.
- weather_reduction – What will eventually be used to determine item use reduction. This is derived from the execution of the program.

Global variables:

There are not too many global variables used.

```
local_file = 'weather_copy.xls'
weather_list = []
thirty_avg = [] * 30
```

- local_file – This is what the current weather file is called. Used when retrieving the current years data.
- weather_list[] – The list that will store the weather objects.
- thirty_avg[] – List that keeps that tracks the average temperature for the previous thirty days.

Operation:

This program is a supplement to data_gen_mk2.py. When the program is called, what occurs is the current years data is gathered based on the year passed through get_data(). The functionality is

```
def get_data(year):
    if year == 6:
        dls = "https://github.com/jgthornb/files/raw/main/weather_data_capstone.xls"
    if year == 5:
        dls = "https://github.com/jgthornb/files/raw/main/2020_weather.xls"
    if year == 4:
        dls = "https://github.com/jgthornb/files/raw/main/2019_weather.xls"
    if year == 3:
        dls = "https://github.com/jgthornb/files/raw/main/2018_weather.xls"
    if year == 2:
        dls = "https://github.com/jgthornb/files/raw/main/2017_weather.xls"
    if year == 1:
        dls = "https://github.com/jgthornb/files/raw/main/2016_weather.xls"
    page = requests.get(dls)
    #saves a copy to the current folder. can be deleted later.
    #uncomment for mac
    #with open(os.getcwd() + "//output_weather//" + local_file, 'wb') as file:
    #uncommnet for windows
    with open(os.getcwd() + "\\output_weather\\" + local_file, 'wb') as file:
        file.write(page.content)
```

essentially that of a switch statement. The “year” gets passed in and a determination is made for which year to download and store. Once decided, the current directory is found, and the file placed.

Afterwards, data_gen_mk2.py deep copies weather_list[]. This prompts the creation of the list by running the entire program with the current weather data being the last on downloaded. The

function that facilitates this is read_file(). Using the global weather_list, it is first cleansed of any

```
def read_file(year):
    global weather_list
    weather_list = []
    #uncomment for mac
    #loc = os.getcwd() + "//output_weather//" + local_file
    #uncomment for windows
    loc = os.getcwd() + "\\output_weather\\" + local_file
    wb = xlrd.open_workbook(loc)
    if year == 6:
        sheet = wb.sheet_by_name('all')
    if year == 5:
        sheet = wb.sheet_by_name('2020')
    if year == 4:
        sheet = wb.sheet_by_name('2019')
    if year == 3:
        sheet = wb.sheet_by_name('2018')
    if year == 2:
        sheet = wb.sheet_by_name('2017')
    if year == 1:
        sheet = wb.sheet_by_name('2016')
    for i in range(sheet.nrows):
            avg_temp = str(sheet.cell_value(i,2))
            if i > 0:
                avg_temp = math.floor(float(avg_temp))
            else:
                avg_temp = 0
            rain = str(sheet.cell_value(i,3))
            snow = str(sheet.cell_value(i,4))
            weather_list.append(weather(avg_temp, rain, snow))
            thirty_day_avg(weather_list, i)
    weather_list.pop(0)
    populate_weather_mod()
```

previous weather data as to not continuously add data from the previous run. Next, the excel file is opened and read based on which year we are working from. The if statements are identifiers for which sheet needs to be read within the weather data. Once the sheet is opened, the for loop collects a row at a time and creates weather objects that are then placed in the weather list. Attributes that are collected from the excel document are temperature, rain, and snow for that day. With an entry in weather_list[], the program can then begin finding the thirty day average for the weather using thirty_day_avg().

```
def thirty_day_avg(self, day):
    avg_time = 30
    if day < avg_time:
        for i in range(day):
            self[day].temp_avg += self[i].temp
        self[day].temp_avg = self[day].temp_avg / (day+1)
    else:
        thirty_back = day - avg_time
        while day > thirty_back:
            self[day].temp_avg += self[thirty_back].temp
            thirty_back += 1
        self[day].temp_avg = self[day].temp_avg / (avg_time + 1)
```

Thirty_day_avg() takes in two variables, the object and the day. Setting avg_time to 30, there are two segments to this funcition. What happens the day is less than thirty and what happens once we are at thirty and beyond. If the day is less than thirty, then an average of the days up till that day is calculated. Once day is equal to or greater than thirty, then the rolling average begins.

After thirty_day_avg() completes, the for loop continues until all days are accounted for. The next line is weather_list.pop(0). The need to pop the list is important as this includes the headers from the excel file. Removing this cleans up the list and only leaves the important information.

The last function is populate_weather_mod(). This is a lightweight loop that runs over everyday and

```
def populate_weather_mod():
    for i in range(len(weather_list)):
        weather_mod(weather_list, i)
```

calculates what the weather reduction should be. To do this, weather_mod() is called, sending in the weather list and the current day. Once in the function, we modify the rain value for the day by

```
def weather_mod(self, day):
    self[day].rain = float(self[day].rain) * 10
    if self[day].rain > 0:
        rain_log = math.log(self[day].rain, 10)
        rain_nlogn = self[day].rain * rain_log
    #After calculating the nlogn value,
    #rain_nlogn will hold the % deficit value.
        rain_percent = rain_nlogn / 100
    else:
        rain_percent = 0
    if rain_percent > 0:
        rain_deficit = (1-rain_percent)
    else:
        rain_deficit = 1
    self[day].weather_reduction = rain_deficit
    temp = self[day].temp
    snow  = float(self[day].snow)

    if temp < 32 and snow > 0.25:
        snow_deficit = 0.25
        self[day].weather_reduction = snow_deficit
    elif snow > 0 and temp > 32:
        snow_deficit = 1
        self[day].weather_reduction = snow_deficit
```

multiplying it by 10. We do this so we can use a nlogn approach to how rain can affect foot traffic. Our aim was to have a little rain effect item use minorly, and large amounts of rain to basically halt customers from coming into the store. So, slow growing but raises quickly once it gets to a certain point. If there is no rain, then we set the rain modifier to 0. We chose to do this as we can subtract from 1 and get a percentage value that is then multiplied by average item use. Since it is below one, this was a good way to take about a certain percentage of item use.

After the rain modifiers, the program checks for snow within weather_mod(). This is a straight check for if it snowed or not and by how much. The first if checks for the temp of the day (if less than 32) and if snow is greater than 0.25 inches. If so, then weather_reduction is attributed the value of 25%. We chose this value instead of zero mainly because there will always be someone who leaves during a

snowstorm. The else checks if there is snow and the temperature is also above freezing and modifies the weather_reduction to 1. Once the program is complete, these values are then copied to data_gen_mk2.py and weather_reduciton is used to directly modify the use of items based on the current day.

For information on file_check(), please refer to the documentation covering data_gen_mk2.py.