```
'''
Lab 4- Multiprocessing

Trang Van
CIS 41B

Test with Processes - uses Pooling to access APIs and get the data to run
the Weather App

'''

import tkinter as tk
import requests
import json
import tkinter.messagebox  as  tkmb
import tkinter.filedialog
import os

import multiprocessing as mp
import time


zip_codes = [92093, 90013, 95192, 94132, 94720, 95064, 95819, 92697,
93940, 94544]  # global var for cities

'''
GLOBAL: Data Population from URLs - Has to be global b/c you can't pickle
tkinter objects

Returns tuple of information (city, temperature, description)
'''
def getdata(url, city, temps, descrp):
    page = requests.get(url)
    data = page.json()

    city.append(data['name'])
    temps.append(data['main']['temp'])

    descrp_perCity= set()

    for i in range(len(data['weather'])):
        descrp_perCity.add(data['weather'][i]['description'])
        descrp.append(descrp_perCity)

    return city, temps, descrp

class mainWin(tk.Tk):
    def __init__(self):
        super().__init__()
        self._cities = []
        self._temps = []
        self._descriptions = []
        # WIDGETS
        self.title("Welcome to the Weather App")

        tk.Button(self, text = "Choose a city", command =
self._getInformation).grid(row= 0, padx=8)

        # Listbox & Scrollbar
        S = tk.Scrollbar(self)
        self._LB = tk.Listbox(self, width = 75, yscrollcommand=S.set)
        S.config(command = self._LB.yview)
        self._LB.grid(row = 1, column = 0)
        S.grid(row = 1, column = 1,columnspan= 2, sticky = 'ns')

        # Part 3
        self.update()                                       # widgets
appear even though data is still loading

        ### Web Access & Data Pop ###
        appid = '6a2f36aefde82926cc37558161b2d56a'          # unique key
        urls = ['http://api.openweathermap.org/data/2.5/weather?zip='+
str(i)+',us&units=imperial&APPID='+ appid for i in zip_codes]
            # makes a list of URLs for each zip code for pool to access

        # PROCESS/ POOL APP
        pool = mp.Pool(processes = 10)    # creates pool for processes -
allows 10 for 10 zip codes

        start = time.time()        #START Timer
        results = [pool.apply_async(getdata, args=(x,self._cities,
self._temps, self._descriptions)) for x in urls]

        pool.close()         # don't allow any more process to be added
        pool.join()          # wait for all processes in the pool to be
done

        print('Processing - total elapsed time:', time.time()-start,'s')
#END timer

        output = [r.get() for r in results]

        # Takes output and populations instance variables for use in
mainWin and dialogBox
        for info in output:
            self._cities.append(info[0][0])
            self._temps.append(info[1][0])
            self._descriptions.append(info[2][0])

        self._totalInfo=  dict(zip(self._cities,
zip(self._temps,self._descriptions)))        #dictionary with city as key
    '''
    Creates dialog box with city options and displays information in
listbox
    '''
    def _getInformation(self):
        dialog = dialogBox(self,self._cities)
        self.wait_window(dialog)

        choice = dialog.getUserChoice()
        choice_city = sorted(self._cities)[choice]
        choice_temp = self._totalInfo[sorted(self._cities)[choice]][0]
        choice_des = self._totalInfo[sorted(self._cities)[choice]][1]

        lb_display= [choice_city,':',choice_temp,'degrees, ',",".join(i
for i in choice_des)]
        if choice != -1: self._LB.insert(tk.END, ' '.join(str(i) for i in
lb_display))
        if self._LB.size() != 0: self.protocol("WM_DELETE_WINDOW",
self._saveFile)


    '''
    If User X out of app, user will be prompted to save file with user-
defined directory.
    '''
    def _saveFile(self):

        filename = 'weather.txt'
        choice = tkmb.askokcancel('Save', 'Save your search in a directory
of your choice?', parent = self)

        if choice == True:
            choice_p = tk.filedialog.askdirectory(initialdir =".")

            confirm = tkmb.showinfo('Save', 'File '+ filename +' will be
saved in\n' + choice_p, parent=self)
            if confirm == True:
                text = list(self._LB.get(0,tk.END))
                fout = open(os.path.join(choice_p, filename), 'a')
                for i in text:
                    fout.write(i+'\n')
                fout.close()
        else:
            self.destroy()


'''
Displays Cities for User to choose and get information: temp and
description
'''
class dialogBox(tk.Toplevel):
    '''
    Creates dialogBox object and uses self._cities to generate radio
buttons for cities
    '''
    def __init__(self, master, cities):
        super().__init__(master)
        self.transient(master)
        self.grab_set()
        self.focus_set()
        self.protocol("WM_DELETE_WINDOW", self._close)

        self._cities = cities # From list of cities found in main win

        self._controlVar = tk.IntVar()
        self._controlVar.set(0)

        for i, city in enumerate(sorted(self._cities)):
            tk.Radiobutton(self, text=city, variable=self._controlVar,
value=i).grid(row=i, column=0, padx=8, sticky = 'w')

        okBT = tk.Button(self, text = "OK", command =
self.destroy).grid(column = 0)

    '''
    Returns users choice to use in mainWin
    '''
    def getUserChoice(self):
        return self._controlVar.get()

    '''
    If user closes dialog box, choice set to -1 and mainWin won't make a
choice
    '''
    def _close(self) :
        self._controlVar.set(-1)
        self.destroy()

'''
Main - allows other mains to run if they exist
'''
if __name__ == '__main__':
    app = mainWin()
    app.mainloop()

'''
===ANALYSIS===

Method 1 (no threads, process) ==> Elapsed time: 0.5267143249511719 s
Method 2 ==> Processing - total elapsed time: 4.804997444152832 s
Method 3 ==> Threading - total elapsed time: 3.2654874324798584 s

I found that the multiprocessing took the longest. Multithreading took
about 3.27 s to fetch the data while multiprocessing
took about 4.80 s. Running the program without either multithreading or
multiprocessing took about 0.52 s to fetch the data.
Because of these results and according to how I wrote my program, it was
most efficient to run without the threads and processes.
However, if there was a lot more data to fetch from the API, then this
method will not be the quickest. Between threading and
processing, threading with a GUI app will be the better option because it
doesn't require making a global function like processing did.
```

```
Note: Sometimes accessing API took longer than others, but generally
method 1 was quickest and method 2 was longest.
==================================
Method 1:
+ Pros: No extra code required

-  Cons: Not efficient with large sets of data, will take longer to fetch
   and process

==================================
Method 2 w/ Pooling:
+ Pros: Multiprocessing allows true parallel processing
        Good with bigger sets of data to optimize time
        For pool apply_sync, can run when data is not exchanged => saves
time

-  Cons: If processes are running on different processors, it takes longer
   to communicate.
        Cannot integrate with tkinter, have to get getdata a GLOBAL
function => added on time to go out of class

==================================
Method 3:
+ Pros: Makes GUI app more responsive, works better with tkinter
       Runs quicker b/c different tasks are running independently

-  Cons: Speed is only improved if some tasks take a long time
        Not true parallel processing, bc of GIL (one thread at a time)

'''
```