```
# Lab 4 - web api, multithreading, multiprocessing, os, review of json, gui
# Names: Trien Bang Huynh and Marcel Gunadi
# lab4thread.py - Multithreading

import urllib.request
import json
import os
import tkinter as tk
import tkinter.messagebox as tkmb
from collections import defaultdict
import time
import tkinter.filedialog
import threading
import queue
from dotenv import load_dotenv


# You can ignore and comment out two lines of below code if you don't store your
API in .env file.
# ---------------------------------
load_dotenv() #                      |
myAPIkey = os.getenv("API_KEY")#     |
# ---------------------------------


# API key is a secret but you can hardcode your API key here to fetch data
# HEADERS = {"X-Api-Key": "INSERT-YOUR-API-KEY-HERE"}
HEADERS = {"X-Api-Key":f"{myAPIkey}"}


class MainWin(tk.Tk):
    """
    A main tk window for the application, allowing users to select states and fetch
national park data.
    """
    def __init__(self):

        super().__init__()
        self.title("US NPS")
        self._stateSelection = []
        self._dataAPI = []

        with open("states_hash.json", 'r') as fh:
            self._states_dic = json.load(fh)

        tk.Label(self, text="National Park Finder", fg="green", font=(
            'Times', 17)).grid(row=0, column=0, columnspan=3, pady=10, padx=10)

        self.L1 = tk.Label(self, text="Select up to 5 states",
                           fg="black", font=('Times', 15))
        self.L1.grid(row=1, column=0, columnspan=3, pady=10)

        # Listbox and Scrollbar
        self.listbox = tk.Listbox(
            self, width=50, height=10, selectmode="multiple")
        self.scrollbar = tk.Scrollbar(
            self, orient=tk.VERTICAL, command=self.listbox.yview)
        self.listbox.configure(yscrollcommand=self.scrollbar.set)
```

```
        # populate the listbox with US's states
        for state in self._states_dic.values():
            self.listbox.insert(tk.END, state)

        self.listbox.grid(row=2, column=0, ipadx=5,
                          padx=20, pady=20, sticky="nsew")
        self.scrollbar.grid(row=2, column=1, sticky="ns")


        # Select button
        self.B = tk.Button(self, text="Submit choice", font=(
            'Times', 15), command=self.onClicked)
        self.B.grid(row=3, column=0, columnspan=2, padx=20, pady=20)

        # status label
        self.L2 = tk.Label(self, text="", font=('Times', 15))
        self.L2 .grid(row=4, column=0, columnspan=3, pady=10, padx=10)

    def onClicked(self):
        """
        A callback function for submit button to check if the selection is valid,
        and proceed to fetch park data for the selected states.
        """
        # Clear the previous state selection
        self._stateSelection = []

        # Check the number of selected states
        if len(self.listbox.curselection()) == 0 or
len(self.listbox.curselection()) > 5:
            tkmb.showerror("Error", "Please select between 1 and 5 states.")
            self.listbox.selection_clear(0, tk.END)
            return
        else:
            for index in self.listbox.curselection():
                codeState, nameState = list(self._states_dic.items())[index]
                self._stateSelection.append((codeState, nameState))
            self.parksFinder()

    def requestAPI(self, endpoint, stateName):
        """
        A method which fetches data from the given API endpoint and update the
status.
        """

        # Make API request and get response
        req = urllib.request.Request(endpoint, headers=HEADERS)
        response = urllib.request.urlopen(req)

        # Parse JSON data from response
        data = json.loads(response.read().decode())

        # append to dataAPI list
        self._dataAPI.append({stateName: data})

        # update status
        tempStr = stateName + ": " + data['total']

        self.q.put(tempStr)
```

```
    def checkQueue(self):
        '''
        A method which checks the queue to get data every 50ms
        '''
        while not self.q.empty():
            tempInfo = self.q.get()
            if tempInfo == "end":
                return

            self.L2["text"] += tempInfo + "  "

        self.after(50, self.checkQueue)


    def parksFinder(self):
        """
        A method which fetches park data for selected states using multiple
threads,
        show the fetched data in the listbox and the fetching status in a label.
        """
        self.L1['text'] = "Select parks to save parks info to file"
        self.B['text'], self.B['command'] = "Save", self.saveFile

        self.listbox.delete(0, tk.END)
        self.status = []

        threads = []
        self.q = queue.Queue()

        for choice in self._stateSelection:
            # an ex of choice = "AL": "Alabama"
            stateCode, stateName = choice

            # Configure API request
            endpoint = f'https://developer.nps.gov/api/v1/parks?
stateCode={stateCode}'

            t = threading.Thread(target=self.requestAPI,
                                 args=(endpoint, stateName))


            threads.append(t)


        self.checkQueue()

        # start recording fetching time
        start = time.time()

        for t in threads:
            t.start()

        for t in threads:
            t.join()

        print(f"Time of fetching data from API for all threads: {time.time() -
start:.2f}s")
```

```
        self.q.put("end")

        # populate the listbox
        for dataObj in self._dataAPI:
            for stateName, dicOfParksInfo in dataObj.items():
                for park in dicOfParksInfo['data']:
                    tempStr = stateName + ": " + park['name']
                    self.listbox.insert(tk.END, tempStr)


    def saveFile(self):
        """
        A method which saves selected park information to a JSON file in the chosen
directory.
        """
        if len(self.listbox.curselection()) == 0:
            tkmb.showerror("Error", "Please select at least 1 park")
            return

        # each key (statename) will have list of selected parks
        selectedParksWithStates = defaultdict(list)

        for i in self.listbox.curselection():
            stateAndPark = self.listbox.get(i)
            stateName = stateAndPark.split(":")[0].strip()
            parkName = stateAndPark.split(":")[1].strip()
            selectedParksWithStates[stateName].append(parkName)

        # open current directory
        directory = tk.filedialog.askdirectory(initialdir='.')

        filesSave = []

        # if user choose a directory
        if directory:
            for stateName, listOfParks in selectedParksWithStates.items():
                listToSave = []
                for park in listOfParks:
                    parkDictForFile = {}
                    tempDict = {}
                    # loop through each parks in selected states to find the parks
chosen from the listbox
                    for dataObj in self._dataAPI:
                        for state, dicOfParksInfo in dataObj.items():
                            if stateName != state:
                                continue
                            for parkDict in dicOfParksInfo['data']:
                                if parkDict['name'] == park:
                                    tempDict['full name'] = parkDict['fullName']
                                    tempDict['description'] =
parkDict['description']
                                    tempAcList = []
                                    for act in parkDict['activities']:
                                        tempAcList.append(act['name'])
                                    tempDict['activities'] = ", ".join(
                                        tempAcList)
                                    tempDict['url'] = parkDict['url']
                                    break
```

```
                    parkDictForFile[park] = tempDict
                    listToSave.append(parkDictForFile)

                # saving file in selected directory
                filename = stateName + ".json"
                pathToSaveFile = directory + "/" + filename
                filesSave.append(filename)

                with open(pathToSaveFile, 'w') as fh:
                    json.dump(listToSave, fh, indent=3)

            # create a confirm messagebox
            filesSaveStr = ", ".join(filesSave)

            confirmed = tkmb.askyesno(
                "Confirmation", f"Save files: {filesSaveStr}")
            if confirmed:
                self.closeWin()

        else:
            self.listbox.selection_clear(0, tk.END)

    def closeWin(self):
        """
        A method which will close the application window.
        """
        self.destroy()
        self.quit()


def main():
    app = MainWin()
    app.mainloop()

if __name__ == '__main__' :
    main()
```