#!/usr/bin/python

import argparse

import time

import pandas as pd

from pandas import DataFrame

#all the options for this script

#parser = argparse.ArgumentParser(description='Tool that automatically gathers trophic information on BugGuide into csv file')

#parser.add\_argument('--days', action='store', required=False, help='How often is the script ran')

#args = vars(parser.parse\_args())

#creating variables of inputted options

#days = args.days

#importing libraries

from urllib.request import Request, urlopen

from bs4 import BeautifulSoup

import re

import requests

from random import choice

import nltk

from nltk.corpus import stopwords

stop\_words = nltk.corpus.stopwords.words('english')

from nltk.stem.wordnet import WordNetLemmatizer

lem = WordNetLemmatizer()

from nltk.stem import PorterStemmer

from nltk.tokenize import sent\_tokenize, word\_tokenize

from nltk.tokenize import sent\_tokenize, word\_tokenize

from nltk.corpus import stopwords

#open url at Bugguide with links to all families

html = urlopen("https://bugguide.net/node/view/3/tree/all")

#save html from bugguide page to soup

soup = BeautifulSoup(html, 'html.parser')

body = soup.find('body')

def get\_link3(family):

answer = {}

#searching html for family name and save as object

string = soup.find(string=family)

#save taxon classification as object

f = string.find\_previous("span", class\_="bgpage-taxon-title")

taxon = str(f.text)

#get the span tag parent of that found string

parent = string.find\_parents("span")

#filter the section of the html parent string found for just the url

url = re.compile(r'(?<=href=").\*?(?=tree")')

#return url as a string

url = url.findall(str(parent))[0]

if str(taxon) == 'Species ' or str(taxon) == 'Subspecies ':

last\_tag = soup.find("a", href=str(url)+'tree')

family = last\_tag.next\_element.next\_element.next\_element

#append details to answer

answer = { 'Name': family, 'Taxonomic Rank': taxon, 'Url': url}

return answer

listofheaders = ['Mozilla/5.0 (Windows NT 10.0; WOW64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/54.0.2840.99 Safari/537.36',

'Mozilla/5.0 (Windows NT 6.1; WOW64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/54.0.2840.99 Safari/537.36',

'Mozilla/5.0 (Windows NT 10.0; WOW64; rv:50.0) Gecko/20100101 Firefox/50.0',

'Mozilla/5.0 (Windows NT 6.1; WOW64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/54.0.2840.71 Safari/537.36',

'Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/54.0.2840.99 Safari/537.36',

'Mozilla/5.0 (Macintosh; Intel Mac OS X 10\_12\_1) AppleWebKit/602.2.14 (KHTML, like Gecko) Version/10.0.1 Safari/602.2.14',

'Mozilla/5.0 (Windows NT 10.0; WOW64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/54.0.2840.71 Safari/537.36',

'Mozilla/5.0 (Macintosh; Intel Mac OS X 10\_12\_1) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/54.0.2840.98 Safari/537.36',

'Mozilla/5.0 (Macintosh; Intel Mac OS X 10\_11\_6) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/54.0.2840.98 Safari/537.36',

'Mozilla/5.0 (Windows NT 6.1; WOW64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/54.0.2840.71 Safari/537.36',

'Mozilla/5.0 (Windows NT 6.1; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/54.0.2840.99 Safari/537.36',

'Mozilla/5.0 (Macintosh; Intel Mac OS X 10\_11\_6) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/54.0.2840.98 Safari/537.36',

'Mozilla/5.0 (Windows NT 10.0; WOW64; rv:50.0) Gecko/20100101 Firefox/50.0',

'Mozilla/5.0 (Macintosh; Intel Mac OS X 10\_10\_1) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/39.0.2171.95 Safari/537.36']

def random\_headers():

return {'User-Agent': choice(listofheaders)}

def get\_hierarchy(url):

#open url for html content

header = random\_headers()

response = requests.get(url, headers=header)

#save html from bugguide page to soup

soup3 = BeautifulSoup(response.content, 'html.parser')

#Scraping Taxons

kingdom = soup3.find\_all("span", {'class': 'bgpage-taxon-title'})

kingdoms = soup3.find\_all("span", {'class': 'bgpage-taxon-desc'})

#Predetermined hierarchy

fullhierarchy = {'Kingdom ': 'None', 'Phylum ': 'None', 'Subphylum ': 'None', 'Class ': 'None', 'Subclass ': 'None', 'Superorder ': 'None', 'Order ': 'None', 'Suborder ': 'None', 'Infraorder ': 'None', 'No Taxon ': 'None', 'Superfamily ': 'None', 'Family ': 'None', 'Subfamily ': 'None', 'Genus ': 'None', 'Species ': 'None', 'Subspecies ': 'None'}

mydict = {}

for i in range(len(kingdoms)):

mydict[kingdom[i].text] = kingdoms[i].text

#putting main classification text into an object

fullhierarchy.update(mydict)

return fullhierarchy

def get\_food(url):

#open url for html content

header = random\_headers()

response = requests.get(url, headers=header)

#filter html for content

soup = BeautifulSoup(response.content, 'html.parser')

#save main body text as object

body = soup.find('body')

body\_text = body.get\_text()

#find information on page after food heading

m = re.findall('(?<=Food)(.\*)', body\_text)

return m[1]

import csv

from itertools import zip\_longest

#function to classify trophic level

def classification\_trophic(food\_info):

#categories

with open('keyworddictionaryv2.csv', encoding='ISO-8859-1') as csvfile:

rows = csv.reader(csvfile)

dictkeyword = list(zip\_longest(\*rows))

dictkeywords = [list(filter(None, l)) for l in dictkeyword]

trophiclevel = ['Herbivore, xylophagus', 'Herbivore, folivorous', 'Herbivore, nectarivorous/palnyvorous', 'Herbivore, phytosuccivorous', 'Herbivore, algivorous', 'Herbivore, frugivore', 'Herbivore, radicivorous', 'Herbivore, granivorous', 'Carnivore, parasitoid', 'Carnivore, entomophagous', 'Carnivore, parasitic', 'Decomposer, saprophytic', 'Decomposer, detrivorous', 'Fungivore, fungivorous']

keywordsums = []

#clean food\_info

keep = set('abcdefghijklmnopqrstuvwxyz ABCDEFGHIJKLMNOPQRSTUVWXYZ')

#remove everything not in set

food\_info = ''.join(filter(keep.\_\_contains\_\_, food\_info))

#separate food\_info into individual words

food\_info = word\_tokenize(food\_info)

#remove stop words

for w in food\_info:

if w in stop\_words:

food\_info.remove(w)

#return root of word

food\_info = [lem.lemmatize(w) for w in food\_info]

#loop through each list in dictkeywords

keywordsfound = []

for categories in dictkeywords:

#loop through each keyword

for w in categories:

#adds a count of 1 if keyword shows up in food description

if w in food\_info:

count = 1

keywordsfound.append(w)

else: count = 0

#replace keyword with count

categories[categories.index(w)] = count

#if the list as all integers then sum the list

if all(isinstance(x, int) for x in categories):

g = sum(categories)

#append the sum to categoryindex list

keywordsums.append(g)

#saves the max sum to an object

maxcategory = max(keywordsums)

if maxcategory > 0:

maxb = [i for i in range(len(keywordsums)) if keywordsums[i] == max(keywordsums)]

classification = [trophiclevel[i] for i in maxb]

classification.extend(keywordsfound)

if maxcategory == 0:

classification = "Not Clear"

#returns the max number index object from the trophic level

return classification

dictionarylist2=[]

Scraped\_df = pd.DataFrame(dictionarylist2, columns = ['Name', 'Taxonomic Rank', 'Url', 'Kingdom ', 'Phylum ', 'Subphylum ', 'Class ', 'Subclass ', 'Superorder ', 'Order ', 'Suborder ', 'Infraorder ', 'No Taxon ', 'Superfamily ', 'Family ', 'Subfamily ', 'Genus ', 'Species ', 'Subspecies ', 'Food Info', 'Trophic Classification'])

Scraped\_df.to\_csv('results\_July29\_2.csv', encoding='utf-8', index=False)

taxons = 'Genus Species Family Class Order Superfamily Tribe Superorder Subclass Subphylum Subfamily Suborder Infraorder Subspecies'

for linked in body.find\_all('b'):

dictionarylist = []

i = linked.get\_text()

string = soup.find(string=i)

f = string.find\_previous("span", class\_="bgpage-taxon-title")

taxon = (f.text)

if taxon in taxons:

input\_info = {}

try:

starttime = time.time()

link = get\_link3(i)

input\_info.update(link)

endtime = (time.time() - starttime)

#print(link)

except AttributeError as error:

# Output expected AttributeErrors

NoInfoUpdate = { 'Name': str(i), 'Taxonomic Rank': 'No Taxonomic Rank', 'Url':'No Link Available'}

input\_info.update(NoInfoUpdate)

print(i + " " + "no link available")

#if input\_info['Url'] != 'No Link Available':

#continue

try:

hierarchy\_info = get\_hierarchy(input\_info['Url'])

input\_info.update(hierarchy\_info)

if input\_info['Species '] != 'None':

fullspecies = input\_info['Genus '] + ' ' + input\_info['Species ']

input\_info['Name'] = fullspecies

#print(hierarchy\_info)

except AttributeError as error:

# Output expected AttributeErrors

update = [str(i), 'No Taxonomic Rank', 'No Link Available']

input\_info = update + input\_info

print(i + " " + "no link available")

except ValueError as error:

print('not a url')

#if input\_info['Url'] != 'No Link Available':

#continue

try:

b = get\_food(input\_info['Url'])

input\_info['Food Info'] = b

except IndexError as error:

# Output expected AttributeErrors

NoFoodInfo = 'No Food Information Available'

input\_info['Food Info'] = NoFoodInfo

#print(input\_info['Url'] + " " + "no food information available")

except requests.exceptions.MissingSchema as error:

NoFoodInfo = 'No Food Information Available'

input\_info['Food Info'] = NoFoodInfo

print("no link available")

except HTTPError as error:

if error.code == 502:

time.sleep(600)

b = get\_food(input\_info['Url'])

input\_info['Food Info'] = b

#if input\_info['Food Info'] != 'No Food Information Available':

#continue

try:

c = classification\_trophic(input\_info['Food Info'])

input\_info['Trophic Classification'] = c

except AttributeError as error:

# Output expected AttributeErrors

print(input\_info['Url'] + " " + "is a str")

except requests.exceptions.MissingSchema as error:

print("no link available")

print(input\_info['Name'])

print(endtime)

dictionarylist.append(input\_info)

Scraped\_df = pd.DataFrame(dictionarylist, columns = ['Name', 'Taxonomic Rank', 'Url', 'Kingdom ', 'Phylum ', 'Subphylum ', 'Class ', 'Subclass ', 'Superorder ', 'Order ', 'Suborder ', 'Infraorder ', 'No Taxon ', 'Superfamily ', 'Family ', 'Subfamily ', 'Genus ', 'Species ', 'Subspecies ', 'Food Info', 'Trophic Classification'])

Scraped\_df.to\_csv('results\_July29\_2.csv', encoding='utf-8', index=False, mode='a', header=False)