import urllib.request as urllib2

from bs4 import BeautifulSoup

from urllib.parse import urljoin #for python2

from urllib.parse import urlparse #for python3

import requests

import threading

import time

# You will need to do pip install bs4 to use imported modules in order to save links in plain text

# Your compiler will also have to be python3

# <https://docs.python.org/2/library/urllib2.html>

def String\_checker(website):

try:

result = urlparse(website) # attempts parses uri string to object used to check if port is active

return all([result.scheme, result.netloc, result.path]) # combines return of first level domain and port and/or user and password which is blank

except:

return False # otherwise returns false

def HTML\_checker(website: str) -> bool:

try:

with requests.get(website, stream=True) as response: # attempts to return value of website dict key

try:

response.raise\_for\_status() # uses returned html code as check, returning true is valid

return True

except requests.exceptions.HTTPError: # urllib instead returns false depending on returned num code

return False

except requests.exceptions.ConnectionError: # throws exception for being unable to find website

return False

def crawlin(pages, depth):

url\_void = [] # creates list for storing crawled through html websites

for i in range(depth): # used to set how many layers of urls to append to list

for page in pages: # used to set how many sub layers of urls to append to list

if page not in url\_void:

url\_void.append(page) # adds new url to list if not added already

try:

site = urllib2.urlopen(page) # attempts to open url, storing it as site

except:

print (page % "%s, ain't able to open bruh") # acts as throwing exception

continue

soup = BeautifulSoup(site.read(), "html.parser") # reads contents of webpage, storing it in the main class of soup

Tree = soup('a') # after reading, it finds the sub-links to store

for Branch in Tree:

if 'href' in dict(Branch.attrs): # H(yper)link Ref(erence) in dictionary

url = urljoin(page, Branch['href']) # parses base url with another anchored webpage

if url.find("'") != -1: # fetches last string

continue

url = url.split('#')[0] # gets filename of url keeping its parameters

if url[0:4] == 'http': # seperates "real" urls from garbage

url\_void.append(url) # adds to url list

pages = url\_void

return url\_void # returns url\_void of

class style: # allows text to be bolded out for emphasis

BOLD = '\033[1m' # adds ability to concatenate a start for bold text

END = '\033[0m' # and ability to end concatenation for bold text

def website\_checker(website, stop\_event):

website = input('Your website url here -> ') # asks for user to input website name

while((not website) or (String\_checker(website) is not True) or (HTML\_checker(website) is not True or (not stop\_event.is\_set()))): # first checks if input is valid by string, html and port. Reason is because urllib is picky on string format.

print('\nYour website is not correct:')

front = website[0:8] != 'https://' # set as variable for collectively checking if website is valid

if (front): # slices beginning of input checking if correct format

print('\tYou need to add "https://" to the beginning')

end = (website[-5:]) != ('.com/' or '.org/') # set as variable for collectively checking if website is valid

if(end): # slices end of input checking if correct format

print("\tYou are missing the '.com/, .org/ or .whatever/' at the end")

empty = not website # set as variable for collectively checking if website is valid

if(empty): # checks if string is empty

print('\tWebsite has to have some letters or numbers in it')

if((((not front) and (not end) and (not empty)) and (HTML\_checker(website) is not True))): # if the syntax is correct for all cases, then it checks if it is a valid url. Reasoning is because is you give it junk like 4534fsdf324f, it crashes the program.

print("\tGood try... That is not a real website.")

website = input('\nTry again. Enter your website url here -> ') # prompts user again for website url

def depth\_checker(website, stop\_event):

tries = 0 # adds a little character

print('\nAlright, the second thing is to enter your depth. \n\tFormat needs to be a positive integer less then a 1000. Like this: ' + '\033[1m' + '5' + '\033[0m')

onions\_have\_layers = input('\nYour depth here-> ') # shrek reference for url depth

while((not onions\_have\_layers) or (not int(onions\_have\_layers.isdigit())) or (int(onions\_have\_layers) > 1000) or (not stop\_event.is\_set())): # after string format is checked, next checks if inputed string is a valid number. If not a number, urllib throws a hissy fit.

tries += 1 # increments attempts if string not valid

print('\nYour website is not correct:')

if(not onions\_have\_layers): # checks if depth is empty

print('\tYou have to give a number, ya know 1, 2, 3....')

if ((not int(onions\_have\_layers.isdigit()))): # checks if depth is an int, not other characters or double

print("\tI'm no mathmatician but that ain't a numba g")

else:

if (int(onions\_have\_layers) > 1000): # keeping depth within reasonable crawl depth

print("\tYou're not downloading the entire internet are ya. Needs to be less then 1000")

if(tries > 1 and tries < 3): # little sarcasm wont hurt no body

onions\_have\_layers = input('\nSigh... Try again. Enter your depth -> ')

if (tries > 3): # bringing the sass

print("\tSeriously. I don't have time for this. ")

onions\_have\_layers = input('\nTry again. Enter your depth -> ') # prompts user again for depth url

if \_\_name\_\_ == '\_\_main\_\_': # program starts here

print("Welcome to Ezget. The magic behind the scenes allows you to download web content by crawling through a tree of web pages set by a 'depth' given by you.") # greeting message

print("\nIn this simulation, you are Neo. I am Morpheus. Follow me and 'I'll show you how deep the rabbit hole of truth goes. Time to be set free.'")

print("\nFirst thing to do is to enter your website url. \n\tFormat needs to be like this: " + '\033[1m' + '<https://www.example.com/>' + '\033[0m\n') # instructions to help the user, bolding out the example

website = ''

input\_tick = False

stop\_event= threading.Event()

count = threading.Thread(target=prompt, args=(website, stop\_event))

count.daemon == True

count.start()

time.sleep(5)

if (input\_tick is False):

print("\nBOOOM. The nuke exploded. Bye bye.")

stop\_event.set()

print("Press enter to exit")

if((String\_checker(website) is True) and (HTML\_checker(website) is True)): # last dimensional dolt check

print ("\nThe Matrix has you... Quantifying Metacortex Initialization Phase....") # Morpheus approved

time.sleep(2)

web\_array = [website] # since url is valid, adds to array that will be sent to crawlin to other website

print("\nUnfortunarely, no one can be told what the Matrix is. You have to see it for yourself. - Morpheus\n") # user took the red pill from morpheus "showing you how deep the rabbit hole goes..."

print(crawlin(web\_array, int(onions\_have\_layers)))

print("\nRemember... all I'm offering is the truth. Nothing More. - Morpheus")

else:

print ('Error 502, blue pill ends the story') # user took the blue pill being blissfully ignorant of "believing whatever you want to believe"

'''

import threading

import time

def website\_checker(id, stop):

website = ''

while not website:

website = input('Your website url here -> ')

if stop():

break

stop\_threads = True

def depth\_checker(id, stop):

print("depth check time")

def main(method\_name):

website = ''

stop\_threads = False

workers = []

for id in range(0,1):

tmp = threading.Thread(target=method\_name, args=(id, lambda: stop\_threads))

workers.append(tmp)

tmp.start()

time.sleep(3)

if((not website) and (stop\_threads is True)):

print('\nClick enter to exit')

for worker in workers:

worker.join()

if \_\_name\_\_ == '\_\_main\_\_':

global website

global stop\_threads

main(website\_checker)

main(depth\_checker)

'''