#!/usr/bin/python3

# Python HTML Web Crawler Example

# [Place source link here]

# For Google Docs optimization, use [Tools] > [Preferences...] menu, and disable all autocorrect options.

from html.parser import HTMLParser

from urllib.request import urlopen

from urllib import parse

# We are going to create a class called LinkParser that inherits some

# methods from HTMLParser which is why it is passed into the definition

class LinkParser(HTMLParser):

# This is a function that HTMLParser normally has

# but we are adding some functionality to it

def handle\_starttag(self, tag, attrs):

# We are looking for the begining of a link. Links normally look

# like <a href="www.someurl.com"></a>

if tag == 'a':

for (key, value) in attrs:

if key == 'href':

# We are grabbing the new URL. We are also adding the

# base URL to it. For example:

# www.netinstructions.com is the base and

# somepage.html is the new URL (a relative URL)

#

# We combine a relative URL with the base URL to create

# an absolute URL like:

# www.netinstructions.com/somepage.html

newUrl = parse.urljoin(self.baseUrl, value)

# And add it to our colection of links:

self.links = self.links + [newUrl]

# This is a new function that we are creating to get links

# that our spider() function will call

def getLinks(self, url):

self.links = []

# Remember the base URL which will be important when creating

# absolute URLs

self.baseUrl = url

# Use the urlopen function from the standard Python 3 library

response = urlopen(url)

# Make sure that we are looking at HTML and not other things that

# are floating around on the internet (such as

# JavaScript files, CSS, or .PDFs for example)

if response.getheader('Content-Type')=='text/html':

htmlBytes = response.read()

# Note that feed() handles Strings well, but not bytes

# (A change from Python 2.x to Python 3.x)

htmlString = htmlBytes.decode("utf-8")

self.feed(htmlString)

return htmlString, self.links

else:

return "",[]

# And finally here is our spider. It takes in an URL, a word to find,

# and the number of pages to search through before giving up

def spider(url, word, maxPages):

pagesToVisit = [url]

numberVisited = 0

foundWord = False

# The main loop. Create a LinkParser and get all the links on the page.

# Also search the page for the word or string

# In our getLinks function we return the web page

# (this is useful for searching for the word)

# and we return a set of links from that web page

# (this is useful for where to go next)

while numberVisited < maxPages and pagesToVisit != [] and not foundWord:

numberVisited = numberVisited +1

# Start from the beginning of our collection of pages to visit:

url = pagesToVisit[0]

pagesToVisit = pagesToVisit[1:]

try:

print(numberVisited, "Visiting:", url)

parser = LinkParser()

data, links = parser.getLinks(url)

if data.find(word)>-1:

foundWord = True

# Add the pages that we visited to the end of our collection

# of pages to visit:

pagesToVisit = pagesToVisit + links

print(" \*\*Success!\*\*")

except:

print(" \*\*Failed!\*\*")

if foundWord:

print("The word", word, "was found at", url)

else:

print("Word never found")