Read Me for getsoup.py: My web scraper

Instructions

To execute the program,

1. Create a folder `crawled-pages/` inside the present working directory, which contains the program `getsoup.py`
2. Run `python3 .\getsoup.py` in the present working directory.

The code outputs text files containing the name of each webpage it has scraped, its summary and its contents. These files are stored in the crawled-pages folder. The code also outputs a text file containing a list of pages scraped.

About the Code

I use the pre-installed Requests package to get every webpage, and BeautifulSoup4 as my web scraper module of choice. I use lxml as my HTML parser for its speed. I also scrape only the sites in the `animalcrossing.fandom.com` domain by checking for the substring "`animalcrossing`" followed by a dot (.) in the site name.

Global variable

`url` is a global list of pages scraped.

Main function

I define a string `homepage =` https://animalcrossing.fandom.com/wiki/Animal_Crossing_Wiki and call the scraper function on it.

scrape(site : String)

This is the main scraper function. It uses the get method from requests library to get the URL in the string argument site passed to it. It then uses lxml to parse the html file at the URL location and BeautifulSoup to access the parse tree. This parse tree is passed as an argument to the getinfo function to create a document with the webpage title as title, a summary and webpage content as

content.
The code then visits each anchor tag ‘`<a>`’ in the soup and checks if the tag has the ‘`href`’ hyperlink reference argument. If available, we check if the hyperlink contains the string “`animalcrossing.`” or has been already visited by going through the global list `url`. If it has not been visited, scrape is called recursively on the hyperlink.
Error handling of HTTPError, URLError, AttributeError, KeyboardInterrupt, and other exceptions have been built into this function.

getinfo(soup : BeautifulSoup object)

This function creates a title from the webpage `<title>` tag, summary from the `<meta name=”description”>` tag and content from the output of `filter(soup.find_all(string = True), tag_visible)`. Here tag_visible is a Boolean function used to ignore tags whose content we don’t want. A lot of unnecessary text is still saved into the content variable.

tag_visible(element: BeautifulSoup object)

This function has a blacklist containing tags whose text is typically not visible on the webpage, thus making their string attributes not a part of the webpage content. If element is a comment or its parent is a blacklisted tag, the function returns `False`, else it returns `true`.

Conclusion and Scope for Improvement

The quality of content saved for each webpage can definitely improve to be limited to passages of text that are important to an interested reader. For this a better study of the html tags and a better use of the parse tree interface of BeautifulSoup would definitely help.