web_scrape_clean_html_test.py

```python
#
#  Course: COSC 4P02
#  Assignment: Group Project
#  Group: 9
#  Version: 1.0
#  Date: March 2024
#
import pytest # Import the pytest module for automated testing.
import requests # Import the requests module to make HTTP requests.
from langdetect import detect # Import the detect method from the langdetect module.
from bs4 import BeautifulSoup # Import the BeautifulSoup class from the bs4 module.
from web_scrape_clean_html import get_clean_content, clean_text, scrape_and_clean # Import the get_clean_content, clean_text, and scrape_and_clean methods from the custom web_scrape_clean_html.

def test_clean_text():
    #
    # Test Case 1: Testing the clean_text method of the web scraper independently. This will confirm base functionality of the clean_text method.
    # Execution: python -m pytest web_scrape_clean_html_test.py -k "test_clean_text" -s -v # Only use -s to view the contents of the text in the test.
    # The method will replace newline and carriage return characters with spaces, replace quotes with pipe characters, and reduce multiple spaces to a single space.
    # Expected Result: Pass. The cleaned text should match the expected output.
    #
    input_text = "Test line 1.\nTest              line 2.\rTest line                     3. \"quotes\"." # Sample text to be cleaned.
    expected_output = "Test line 1. Test line 2. Test line 3. |quotes|." # Expected output of the cleaned text.
    cleaned_text = clean_text(input_text)  # Call the clean_text method on the URL to be tested.
    assert isinstance(cleaned_text, str)  # Confirm that the cleaned text is a string.
    assert cleaned_text == expected_output  # Confirm that the cleaned text matches the expected output.
    print(input_text) # Print the input text. This line will only show if the -s is included when running the test.
    print(cleaned_text)  # Print the cleaned text. This line will only show if the -s is included when running the test.

def test_scrape_and_clean():
    #
    #  Test Case 2: Testing the scrape_and_clean method with a valid URL independently. For this test, not a Wikipedia page, but a CNN page.
    #  Execution: python -m pytest web_scrape_clean_html_test.py -k "test_scrape_and_clean" -s -v # Only use -s to view the contents of the URL in the test.
    #  This method will test the scrape_and_clean function on a valid URL. The function should scrape the page, clean the content,
    #  and return the page content.
    #  Expected Result: Pass. The content scraped from the URL is a valid string with only permitted characters.
    #
    url = "https://www.cnn.com/2025/03/10/business/usmca-tariff-delay-trump/index.html" # URL to be tested. This is a valid URL.
    content = scrape_and_clean(url) # Call the scrape_and_clean method on the URL to be tested.
    assert isinstance(content, str) # Confirm that the content scraped from the URL is a string.
    assert len(content) > 50  # Confirm that the content scraped from the URL is more than 50 characters long. This ensures that the full content of the URL is not empty.
    assert '\n' not in content # Confirm that the content scraped from the URL does not contain newline characters.
    assert '\r' not in content # Confirm that the content scraped from the URL does not contain carriage return characters.
    assert '"' not in content #Confirm that the content scraped from the URL does not contain quotes.
    print(content) # Print the content scraped from the URL. This line will only show if the -s is included when running the test.

def test_ValidURL1():
    #
    #  Test Case 3: Testing the get_clean_content method with a valid URL. For this test, Wikipedia.
    #  Execution: python -m pytest web_scrape_clean_html_test.py -k "test_ValidURL1" -s -v # Only use -s to view the contents of the URL in the test.
    #  This method will test the get_clean_content method of the web scraper with a valid Wikipedia URL. The URL will be scraped and
    #  the content will be cleaned. The content will be checked to ensure that it is a string and that it is not empty.
    #  Expected Result: Pass. The content scraped from the URL is a valid string and is significantly long.
    #
    #
    url = "https://en.wikipedia.org/wiki/Agile_software_development" # URL to be tested. This is a valid URL.
    content = get_clean_content(url) # Call the get_clean_content method on the URL to be tested.
    assert isinstance(content, str) # Confirm that the content scraped from the URL is a string.
    assert len(content) > 50  # Confirm that the content scraped from the URL is more than 50 characters long. This ensures that the full content of the URL is not empty.
    print(content) # Print the content scraped from the URL. This line will only show if the -s is included when running the test.

def test_ValidURL2():
    #
    #  Test Case 4: Testing the get_clean_content method with a valid URL. For this test, CBS Sports.
    #  Execution: python -m pytest web_scrape_clean_html_test.py -k "test_ValidURL2" -s -v # Only use -s to view the contents of the URL in the test.
    #  This method will test the get_clean_content method of the web scraper with a valid CBS Sports URL. The URL will be scraped and
    #  the content will be cleaned. The content will be checked to ensure that it is a string and that it is not empty.
    #  Expected Result: Pass. The content scraped from the URL is a valid string and is significantly long.
    #
    #
    url = "https://www.cbssports.com/soccer/news/champions-league-bold-predictions-liverpool-will-rely-on-alisson-to-save-them-pedri-shines-for-barcelona/" # URL to be tested. This is a valid URL.
    content = get_clean_content(url) # Call the get_clean_content method on the URL to be tested.
    assert isinstance(content, str) # Confirm that the content scraped from the URL is a string.
    assert len(content) > 50  # Confirm that the content scraped from the URL is more than 50 characters long. This ensures that the full content of the URL is not empty.
    print(content) # Print the content scraped from the URL. This line will only show if the -s is included when running the test.

def test_InvalidURL():
    #
    #  Test Case 5: Testing the get_clean_content method with an invalid URL. For this test, www.cosc4p02group9fakeurl.com.
    #  Execution: python -m pytest web_scrape_clean_html_test.py -k "test_InvalidURL" -s -v # Only use -s to view the printed error message of the test.
    #  This method will test the get_clean_content method of the web scraper with a custom invalid URL. The web scraper returns "Error:" if the URL is invalid,
    #  so the test will check for this string in the content. If the string is found, the test will pass, the URL was invalid.
    #  Expected Result: Pass. "Error:"" will exist in the content because the URL is invalid.
    #
    url = "http://www.cosc4p02group9fakeurl.com/" # URL to be tested. This is an invalid URL.
    content = get_clean_content(url) # Call the get_clean_content method on the URL to be tested.
    assert "Error:" in content # Confirm that the content scraped from the URL contains the string "Error:". This indicates that the scraper found no URL, hence it is invalid.
    if "Error:" in content: # If the content contains the string "Error:", the URL is invalid.
        print(f"URL '{url}' is not valid.") # Print that the URL is not valid.

def test_NoMainContent():
    #
    # Test Case 6: Testing the get_clean_content method with a valid URL but no main content. For this test, https://www.example.com/.
    # Execution: python -m pytest web_scrape_clean_html_test.py -k "test_NoMainContent" -s -v # Only use -s to view the printed message of the test.
    # This method will test the get_clean_content method on a valid URL that doesn't have any main content. The page may be blank, or there may be no main content. The web scraper returns
    # "Main content not found." if the main content is not found, so the test will check for this string in the content. If the string is found, the test will pass, the URL had no main content.
    # Expected Result: Pass. "Main content not found." will exist in the content because the URL has no main content.
    #
    url = "https://www.example.com/"  # URL to be tested. This is a valid URL without main content.
    content = get_clean_content(url)  # Call the get_clean_content method on the URL to be tested.
    assert isinstance(content, str) # Confirm that the content scraped from the URL is a string.
    assert "Main content not found." in content  # Confirm that the content scraped from the URL contains the string "Main content not found." This indicates that the scraper found no main content.
    if "Main content not found." in content:
        print(f"URL '{url}' has no main content.") # Print that the URL has no main content.

def test_NonEnglishContent():
    #
    # Test Case 7: Testing the get_clean_content method with a valid URL that has content not in English.
    # Execution: python -m pytest web_scrape_clean_html_test.py -k "test_NonEnglishContent" -s -v # Only use -s to view the English contents of the URL in the test.
    # This method will test the get_clean_content method on a page with content in a language other than English.
    # The web scraper should not include non-English lines and return only English text if present.
    # Expected Result: Pass. The content should not contain any lines that contain other languages.
    #
    url = "https://en.wikipedia.org/wiki/French_Wikipedia"  # URL to be tested. This is a valid URL, with a French word in the content.
    content = get_clean_content(url)  # Call the get_clean_content method on the URL to be tested.
    assert isinstance(content, str) # Confirm that the content scraped from the URL is a string.
    assert "Wikipédia en français" not in content  # Confirm that the French phrase "Wikipédia en français" should be filtered out. It should not be in content.
    assert "This edition was started on 23 March 2001, two months after the official creation of Wikipedia." in content  # Confirm the English lines are still present in content. This is the first full English line of the page.
    print(content) # Print the content scraped from the URL. This line will only show if the -s is included when running the test.

def test_TagsInContent():
    #
    # Test Case 8: Testing the get_clean_content method with a valid URL that contains HTML tags embedded in the content. Citations, Footnotes, etc.
    # Execution: python -m pytest web_scrape_clean_html_test.py -k "test_TagsInContent" -s -v # Only use -s to view the contents of the URL in the test.
    # This method will test the get_clean_content method on a page with HTML tags within the content. The web scraper should remove the HTML tags and return only readable text.
    # Most Wikipedia articles contain many HTML tags, so this test could be performed on other Wikipedia articles to showcase the removal of embedded tags. Other sites also have tags, but Wikipedia is a good example.
    # Expected Result: Pass. The content should only contain clean text.
    #
    url = "https://en.wikipedia.org/wiki/Python_(programming_language)"  # URL to be tested. This is a valid URL, with many tags and styles.
    content = get_clean_content(url)  # Call the get_clean_content method on the URL to be tested.
    assert isinstance(content, str) # Confirm that the content scraped from the URL is a string.
    assert "<" not in content  # Confirm there are no tags in the content. The content should not contain any HTML tags. 1/2
    assert ">" not in content  # Confirm there are no tags in the content. The content should not contain any HTML tags. 2/2
    assert "Python" in content  # Confirm there are English words remaining in the content. The content should contain the word "Python".
    print(content) # Print the content scraped from the URL. This line will only show if the -s is included when running the test.


```