WebScrape Programme

Requirements:

- python 3
- node.js
- Lighthouse
- requests
- tqdm
- bs4

Installation:

Install python and the programme dependencies using the cli command:

`pip install <package name>` e.g `pip install tqdm` to install tqdm

Lighthouse is installed using the npm package manager and the command:

`npm install -g lighthouse` to install it globally on your system.

Usage:

Run the main.py file and follow the steps. You can either run the scraper by pressing 1 or change programme settings by pressing 2.

The programme will prompt the user to enter their desired query and the number of search results you would like to obtain.

Lighthouse Integration

Page performance is taken from using the *node.js* package *Lighthouse* to test page load speeds in a headless chrome browser. This is called from the programme in a call to the command line.

Code Tips

All Functions and methods have been documented. If you look at the code in vscode and hover over the function calls it will give you a quick description of the function and what parameters it takes.

Changelog - 2

- Had to make the user agent a instance variable to line 36
- Wrapped the main function execution in a while True statement and added a user input to check if they want to exit.
- Added a user agent variable to the `query_google()` function that reads the settings.json file. As well as added it to the class constructor call on line 33 of the same function.
- added a return statement to the `check_times()` directory to return a boolean showing if the scraper is ready to be used again. The check is used for in `query_google()` function.

Changelog - 1s

- the package tqdm is used to display a progress bar as the lighthouse method can take some time (NOTE: this is another external package needs to be installed through pip)
- the `get_link_info()` function was changed as two splits are required to get a valid url. The new function is shown below and it also adds the https protocol to the start if the string for use later on.

```
def get_link_info(element):
    """Extracts link data from url element tag

    Parameters:
        element: string html element to be extracted
    Returns:
        extracted link element
    """
    first_split = element.split("//")[-1]
    second_split = first_split.split("&")[0]
    return "https://" + second_split
```

- the `check_times()` function was moved to the utils file. You was right it is more organised this way.
- all input checks have been moved from the main block to a utils function called `check_input()` in the utils file. It has also been expanded to make it more versatile.
- the main block and execution code in general has been separated from the scraper class (which is now located in its own package *classes*) and the block was moved to *main.py* file in the base directory for organisation purposes.
- class variables for the data to be collected was made a the top of the scraper class so these lists can be changed and modified from other methods in the class.
- `json_loads()` added in utils, to quickly load string into python dictionary (necessary function for lighthouse implementation).
- the page title (H3 elements) list was removed as this was causing an exception and was not sure if this was needed for the assignment.
- couple of other functions added to utils for general usage.

Extension

App Specific

More settings can be added to the *settings.json* file in the config directory to customise the experience of the scraper even further.

Lighthouse Specific

The lighthouse configuration can be altered in the *lighthouse.json* directory to what data is contained in report. Currently it is only configured to give data for the performance metrics of a webpage.

For more information on the configuration process and lighthouse in general check out the links below:

- https://github.com/GoogleChrome/lighthouse/blob/master/docs/scoring.md

- https://github.com/GoogleChrome/lighthouse/blob/master/docs/configuration.md

Issues

- Lighthouse requires that a protocol be present for testing a webpage speed as a result some pages will fail and this will be shown through a `N/A` in the data.

Things That Could Be Done

- Could add a timer that shows when it can be used again.
- Could make the addition of additional choices dynamic.
- Use of arrays rather than lists to store data, could be faster.