Documentation for the Scraper.py program

By. Darrell Owens. Project Developer and Primary Author of Scraper.py

Introduction:

The Scraper program is part of the Transit App project in CSE 115A to produce real-time bus arrival information for Santa Cruz Metro and the Loop Bus. web_scrapper_stops is written in Python 3, where an input file specifies a bus line and direction of a bus and the web scrapper parses through the HTML copy of the Santa Cruz Metro website for bus stops corresponding with the input. The program writes to a JSON or Python List()-based output file of a list of bus stops that correspond to a textfile that the user can select from on the front-end.

User Story

User story: “I want to be able to select a Santa Cruz Metro bus stop and receive real-time information when the bus arrives.”

The sendRequest() function fulfills the real time information aspect of the user story; while the sendStopRequest() function fulfills the select bus stop aspect.

Intended Audience

This is an internal program the customer will never directly see. This document is for developers and project owners only.

System Requirements

The program sources functions / methods from several Python libraries that must be installed and reachable in your executable path in order to work appropriately. Only guaranteed to run on Python 3.9.0 or newer.

- Import requests: Produces a scrapper object which does an HTML call to generate the webpage.
- from bs4 import BeautifulSoup: Converts the HTML response and body message into an array or list format that can be parsed with HTML icons via a method
- import re: regex library to parse out desired input from strings
- import csv: open and close file operators

- import json: to write python-structured (JSON) dictionary to output file
- import socketio: opens a socket for back-end operations
- import sys: back-end operations with the socket
- import time: back-end time out operations

Program Breakdown for sendRequest(data)

Summary: Produce real time bus arrival data, bounds and map code for line generation.
--

Lines 14 - 21 are back end operations creating a socket to be connected to the back end.

Lines 22 - 80 are initializations of dictionaries of bus lines and bus direction URL snippets. When the user's specified input matches up with the bus line key, the bus stop key, and the direction key, the outputs will construct part of the new URL.

Note that the input file is passed in through the function protoype, structured as a dictionary. Lines 81 - 83 copy the user's selected line and selected direction into local variables.

Lines 85 - 101 check for unique bus lines where the direction of the bus makes a material change in the URL generated. For most bus lines, the stop names are identical outbound and inbound, so the direction won't matter when accessing the bus arrivals. For the ones in the if-else if structure, different bus stops will generate, thus the proper URL code will generate depending on input. Denoted by "_I" and _O" codes in the dictionary.

The next line creates the URL to send to the HTML request.
"https://cruzmetro.com/simple/routes/" is the start and /direction/ parameter is added. Because the bus stops number in the hundreds, the stop URL data is stored in a separate file.

Lines 105 - 117 loop through a local file in the repo with the string name for every bus stop in Santa Cruz Metro delimited with a vertical hyphen. Example: "Bus Stop | Bus Stop URL Code." The Bus Stop URL code, cleaned up with newline artifacts removed, will be returned in a local variable with the "/stop/", bus stop URL code, and "/pattern/" URL configured. The URL code for the bus ETA is now ready.

Lines 119 - 122 does the actual web scrapping. the request library sends the HTML request for the configured URL. BeautifulSoup library takes the message body of the response, which is an inspect element of the webpage, and parses the html code into iterable code that can be accessed by string version of HTML codes.

Line 124 - 131 find the appropriate element by searching the HTML tags for "li" and then append the bus arrival times to a Python list.

Finally the list is given to the back end via an emit call from the socket. Inside is the bus arrival time, directional bound and map data code.
--

Program Breakdown for sendStopRequest(data)

Summary: Produce unique list of stops for selected bus and direction.

Lines 163 - 168 initialize variables that'll be used to flag errors for the output data and a list to hold all the bus stops. Local variables absorb the input data consisting of the bus line and the direction. The purpose of this program is to generate all the bus stops for this specific line and direction.

The try-catch block to deal with KeyError problems where during the construction of the URL with lines and directions, if the value in lines is not a valid Santa Cruz Metro bus line. Line 174 skips this block if it's a UCSC bound bus as they loop around and don't have unique bus stop lists based on directions as a result (they default to outbound.)

Otherwise the block should take you to a unique Outbound / Inbound page where the for loop "for direction in job_elements" compares the user request bound up against the website's bounds, and then returns the URL that is hyperlinked to Inbound or Outbound in lines 182 to 187.

The Regex pulls out the hyperlink's specific URL code while discarding the rest, and then it puts that code behind the "/directions/" component of the URL. This completed URL provides all the bus stops for that line and direction. The web parser is activated, the URL is called via HTML get request built into the parser's request library, and returns an object storing the HTML code of the webpage. The webpage is translated into traversable HTML code known as "soup."

In the event of a connection error or glitch, the program will provide an error flag indicating the HTML could not be read or the URL could not be connected to.

For lines 194 - 204, we use the HTML object to parse out the bus stops which are under the HTML tag of "li" and have HTML arrow classes. That will produce many more HTML lines of code beyond just the bus stops so we use REGEX to isolate only the qualifying code with a "pattern" HTML artifact which corresponds with the bus stops.

The lines of code are cleaned up of artifacts and then stored into the list. Lines 205 - 208 deal with producing error results for the stack to manage in the event of issues. Otherwise, Lines 209 - 212 write the list of bus stops to a JSON file to be printed on the front end for the user and is also sent via a backend request. (the latter is more important).

Lines 214 - 221 deal with socket connection debugging irrelevant to the function of the scrapper.