Multithreaded Web Scraper/Data Collector

Kieran McGeary

Design Goal:

Get price data from the web on chosen stocks and collect it in a CSV file.

Overall Design

USER THREAD

Facilitate user-Scrape API communication.

SCRAPE API

Simplify Yahoo Finance and facilitate logging to CSV.

YFINANCE API

Facilitate real-time price data collection.

User Thread

Two goals:

1. Take user input and send to the Scrape API
2. Provide functions to get status updates during sampling

User Code

```python
import scraper_api
from os import system


def display_help():
    print('Commands:')
    print('time: displays running export time')
    print('samples: displays current number of samples taken')
    print('tickers: displays ticker list')
    print('prices: displays the price for each symbol')
    print('names: displays the full name of all tickers')
    print('start: begins exporting price values to CSV')
    print('stop: ends CSV export')
    print('clear: clears the terminal')
    print('quit: stops sampling and exits')


if __name__ == '__main__':
    count = 0
    while True:
        tick = input('Enter a ticker (nothing if done): ')
        if tick == '': break  # done getting inputs
        else:
            # Check if our ticker is actually valid
            test = scraper_api.get_price(tick)
            if test == -1: exit(-1)  # Something went wrong
            elif test is None: continue  # Not a valid ticker
            else: scraper_api.tick_list_append(tick)
    if scraper_api.tick_list_len() == 0: exit(0)
```

```python
while True:
    cmd = input('Enter a command (? for help): ')
    match cmd:
        case 'quit':
            scraper_api.done()
            break
        case 'time':
            t = scraper_api.display_time()
            if t == -1: print('Not exporting')
            else: print('Runtime: ' + str(t) + ' s')
        case 'names':
            names = scraper_api.display_names()
            tick_list = scraper_api.display_tickers()
            for i in range(len(names)):
                print(tick_list[i] + ': ' + names[i])
        case 'samples':
            s = scraper_api.display_samples()
            if s == -1: print('Not exporting')
            else: print('Samples: ' + str(s))
        case 'tickers':
            tick_list = scraper_api.display_tickers()
            for t in tick_list:
                print(t)
        case 'prices':
            prices = scraper_api.display_prices()
            tick_list = scraper_api.display_tickers()
            for i in range(len(prices)):
                print(tick_list[i] + ': ' + prices[i])
        case 'start':
            name = input("Enter file name (will be created if missing): ")
```

Scrape API: Public side

- Main purpose is real-time stock data collection and logging
- YFinance API can be simplified into a few functions
- Other convenient functions for status while polling and immediate updates
- Can connect with many programs

Scrape API: Under the hood

- Threading, file management, and data collection (via YFinance) are not visible to the user
- One user thread per ticker
- All happens in one contained process

Scrape API code

```
import cfg
import requests
import threading
import csv
import yfinance as yf
import thread_func


def get_ticker(ticker):
    stock = yf.Ticker(ticker)
    return stock


def get_name(ticker):
    stock = yf.Ticker(ticker)
    try:
        return stock.info['shortName']
    except requests.exceptions.ConnectionError:
        print('Fatal connection error')
        return -1


def get_price(ticker):
    stock = yf.Ticker(ticker)
    try:
        return stock.info['regularMarketPrice']
    except requests.exceptions.ConnectionError:
        print('Fatal connection error')
        return -1
```

```
def done():
    if cfg.recording: stop_recording()
    cfg.done = 1
    for i in cfg.threads: i.join()
    return 0


def display_prices():
    price_list = []
    for tick in cfg.ticker_list:
        price = get_price(tick)
        price_list.append(price)
    return price_list


def display_names():
    tick_list = []
    for tick in cfg.ticker_list:
        name = get_name(tick)
        tick_list.append(name)
    return tick_list


def display_time():
    if cfg.recording: return cfg.t
    else: return -1


def display_samples():
```

```
def display_samples():
    if cfg.recording: return cfg.samples
    else: return -1


def display_tickers():
    return cfg.ticker_list


def start_recording(name):
    if cfg.recording:
        return -1
    else:
        cfg.file = open(name, 'w', newline='')
        writer = csv.writer(cfg.file)
        writer.writerow(['Sample'] + cfg.ticker_list)
        cfg.recording = 1
        cfg.bar = threading.Barrier(len(cfg.ticker_list) + 1)
        for i in range(len(cfg.ticker_list)):
            x = threading.Thread(target=thread_func.scraper, args=(cfg.ticker_list[i], i,))
            cfg.out.append(0)
            cfg.threads.append(x)
            x.start()

        y = threading.Thread(target=thread_func.timer, args=())
        cfg.threads.append(y)
        y.start()
        return 0
```

YFinance API:

- Facilitates real-time stock data collection through Python
- Provided by Yahoo Finance
- Implemented with the requests module

Future expansion

- CSV interpretation and graphing
- Automation
- Remote management

Problems

- YFinance API is very slow and can be unpredictable
  - Requests can take several **seconds** to finish
  - Sampling rate is limited by API response time
- Synchronization
  - Sample timing- threads can’t just run freely
  - CSV updates must happen row-by-row
  - Needs a timer thread to ensure consistency

Demo