Shopping on a budget

Terminal app

Wen Lu 09.05.2023

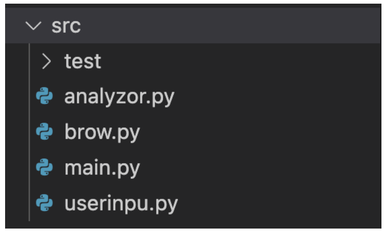
Features

1. Accept user input
user input budget and provide shoplist txt file which would be processed to a dict for the next step

2. According to user input, using browser searching and receive data
use selenium to simulate user browsing activity and beutifulsoup to extract searching data needed

3. Analyzing searching data and comparing with product keywords and budget, save results in file
* calculate total price of each possible combinations
* match keywords and budget in the shoplist provided in first step, get the final matched results and saved in local path as a csv file for user to check

Overview of code



userinpu.py > ...

```python
from os import path
import tkinter.filedialog
import tkinter as tk

class userinput:
    def __init__(self) -> Nor

    def get_budget(self):...

    # open a window to select
    def get_path(self):...

    # open file and save text
    # shoplists dict example:
    def get_shoplist(self):...
```

brow.py > ...

```python
from selenium import webdriver
from selenium.webdriver.support.w
from selenium.webdriver.common.by
from selenium.webdriver.support i
from bs4 import BeautifulSoup

# 'wws example'
class brow:
    # brow object need a item whe
    # shopping website url is ini
    # and search action
    def __init__(self,item) -> No

    def get_soup_pages(self):...

    def get_title_price(self):...
    # close browser manually
    def closewindow(self):
        self.driver.close()
```

analyzor.py > ...

```python
import os
import tkinter.filedialog
import csv

class analyzor:
    # analyzor object need three params when initialized/created
    def __init__(self, budget, user_dict: dict, search_lists: list[dict]

    def search_item_match_records(self):...

    # add each element of 1st list to each ele of 2nd list
    def add_price(self,arr1):...

    # to find if the total cost of each possibility <= budget, then keep
    # each possibility is a lsit of dict like this:
    # {12.9: [['SOGA', '1.4'], ['SOoden Handle', '2.7'], ['La Extra Large
    def price_budget_match(self):...

    # ask user to choose a directory to save results in csv file
    def get_path(self):...

    # each line example:
    # {12.9: [['SOGA', '1.4'], ['SOoden Handle', '2.7'], ['La Extra Large
    def output2csv(self):...
```

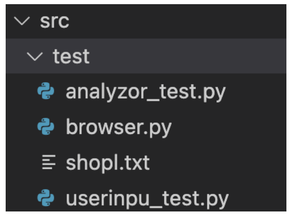
main.py > ...

```python
from userinpu import userinput
from brow import brow
from analyzor import analyzor


try:
    #get userinput instance
    userinputs = userinput()
    # get budget input
    budget = userinputs.get_budget
    # get shoplist input
    user_dict = userinputs.get_sho
    search_lists = []
    # search all items in browser
    for item in user_dict:
        browser = brow(item)
        search_lists += browser.ge
    browser.closewindow()
    analize = analyzor(budget, use
    c = analize.output2csv()
    print('execution end')

finally:
    print('end of listing, leaving
```

Overview of testing

test > analyzor_test.py > ...

```python
from contextlib import contextmanager
import pytest
import sys
sys.path.append('..')
from analyzor import analyzor

testUserDict = {'Egg': ['700g', 'large'], 'Apple': ['1kg'
testSearchLists = [...
searchKeepListReturn = [...

searchKeepListReturn2 = [...

arr = [{'1.3': 'ww1'},...
total = [{'3.7': [['1.3', 'www1'], ['1.3', 'ww12'], ['1.1

total_price_list_return = sorted(total, key=lambda item:

def exitErr():...

@pytest.fixture(scope="module")
def createAnalyzor():
    return analyzor(12, testUserDict, testSearchLists)

def test_search_item_match_records(createAnalyzor):...

def test_add_price(createAnalyzor):...


budgetMatch_return = [{17.3: [['Sunny Queen 12 Extra Larg

def test_price_budget_match(createAnalyzor):...

def test_get_path(createAnalyzor):...

def test_output2csv(createAnalyzor):...
```



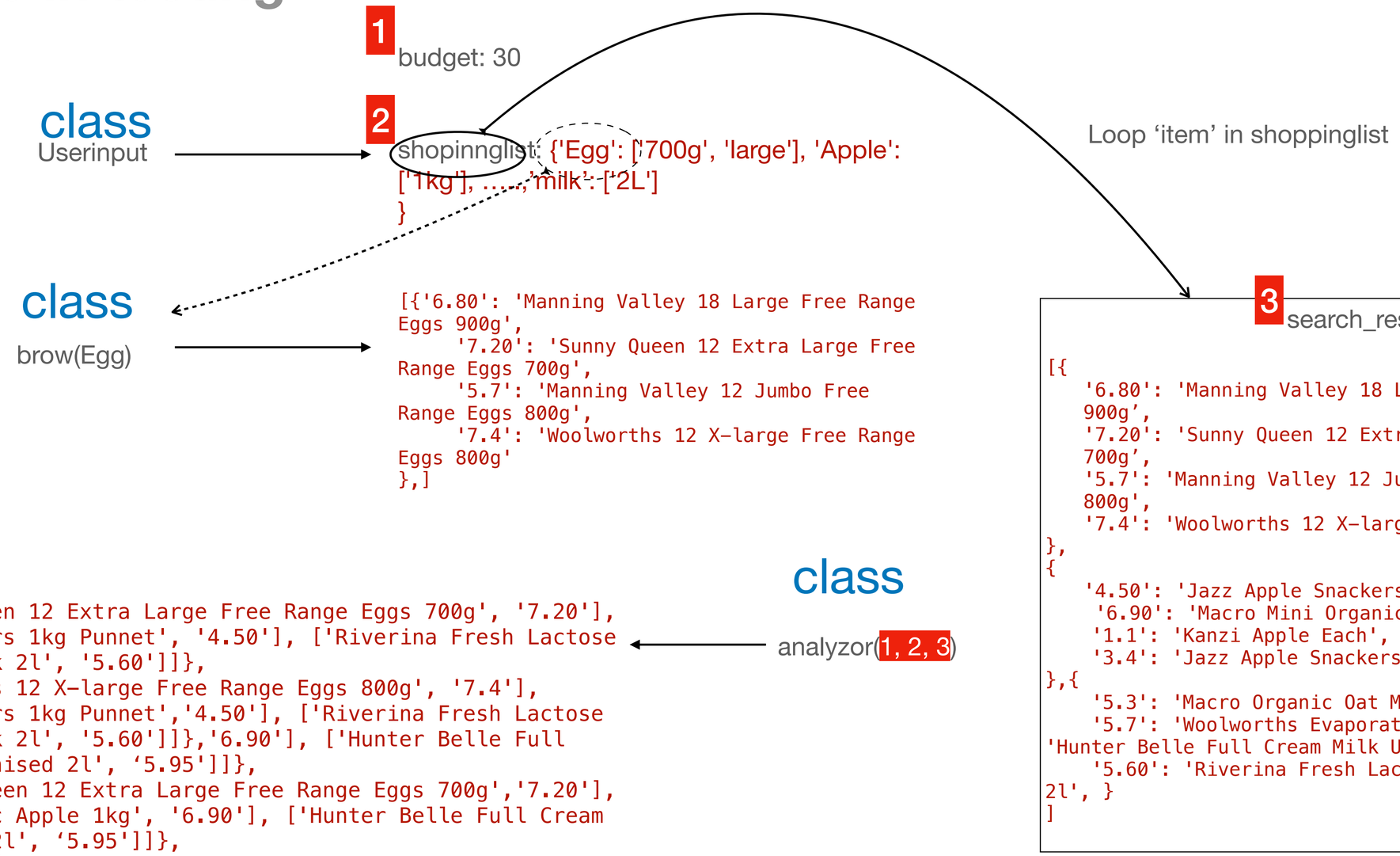
test > browser.py > ...

```python
import pytest
from bs4 import BeautifulSoup
import sys
sys.path.append('..')
from brow import brow

test_item = 'milk'

@pytest.fixture(scope='module')
def createBrow():
    return brow(test_item)

def test_get_soup_pages(createBrow):
    soup, page = createBrow.get_soup_pages()
    assert type(soup) == BeautifulSoup
    assert type(page) == str
    # assert type(page) == None

def test_get_title_price(createBrow):
    titlericeList_return = createBrow.get_ti
    assert type(titlericeList_return) == lis
    assert len(titlericeList_return) == 1
```

test > userinpu_test.py > ...

```python
import pytest
import sys
sys.path.append('..')
from userinpu import userinput

shoplist = {'Egg': ['700g', 'large'],
            'Apple': ['1kg'],
            'milk': ['2L']}

@pytest.fixture(scope='module')
def createUserInpu():
    return userinput()


def test_get_budget( monkeypatch):
    monkeypatch.setattr('builtins.input', lam
    budget = float(input('print your budget:'
    assert budget == float('100')

def test_get_shoplist(createUserInpu):
    with open('shopl.txt', "r") as file1:
        file1.read()
        shoplist_return = createUserInpu.get_
        assert shoplist_return ==shoplist
```

Data usage in testing



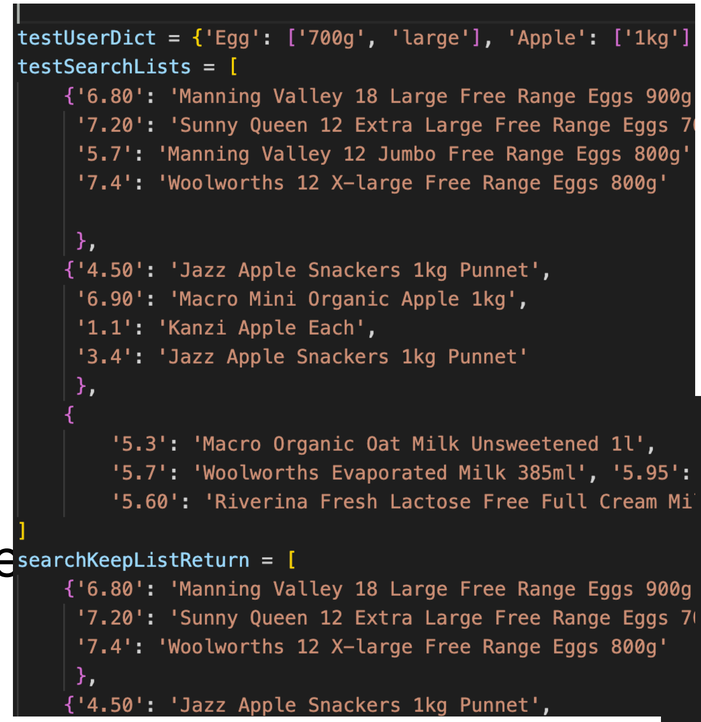
Testing

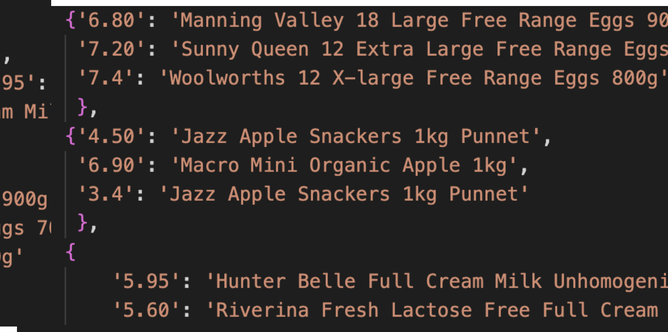
Testing file is according to relevant module file, each file tests functions in that module file.

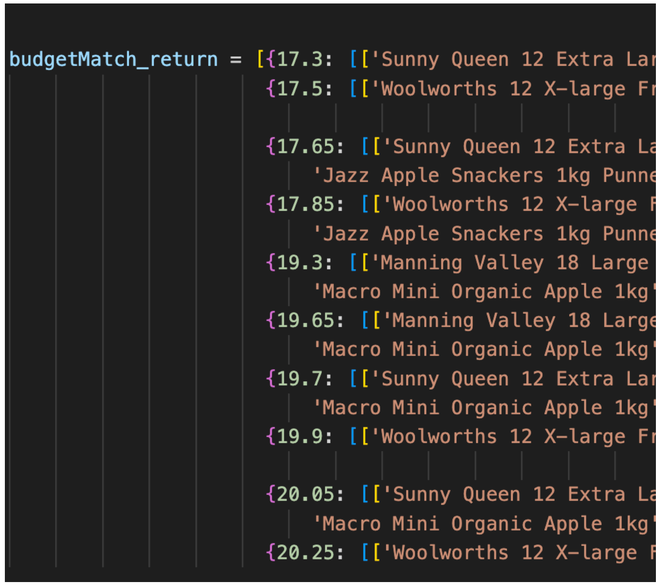
In the analyzor testing file, I provide some simple data for testing the functions, one purpose for the testing is to make sure I have a right data type returned.

I know the testing is not professional in this project and not very automatic, but it makes me realize if knowing testing correctly, it does help a lot for fasting the development process and manually testing can mostly be avoided .

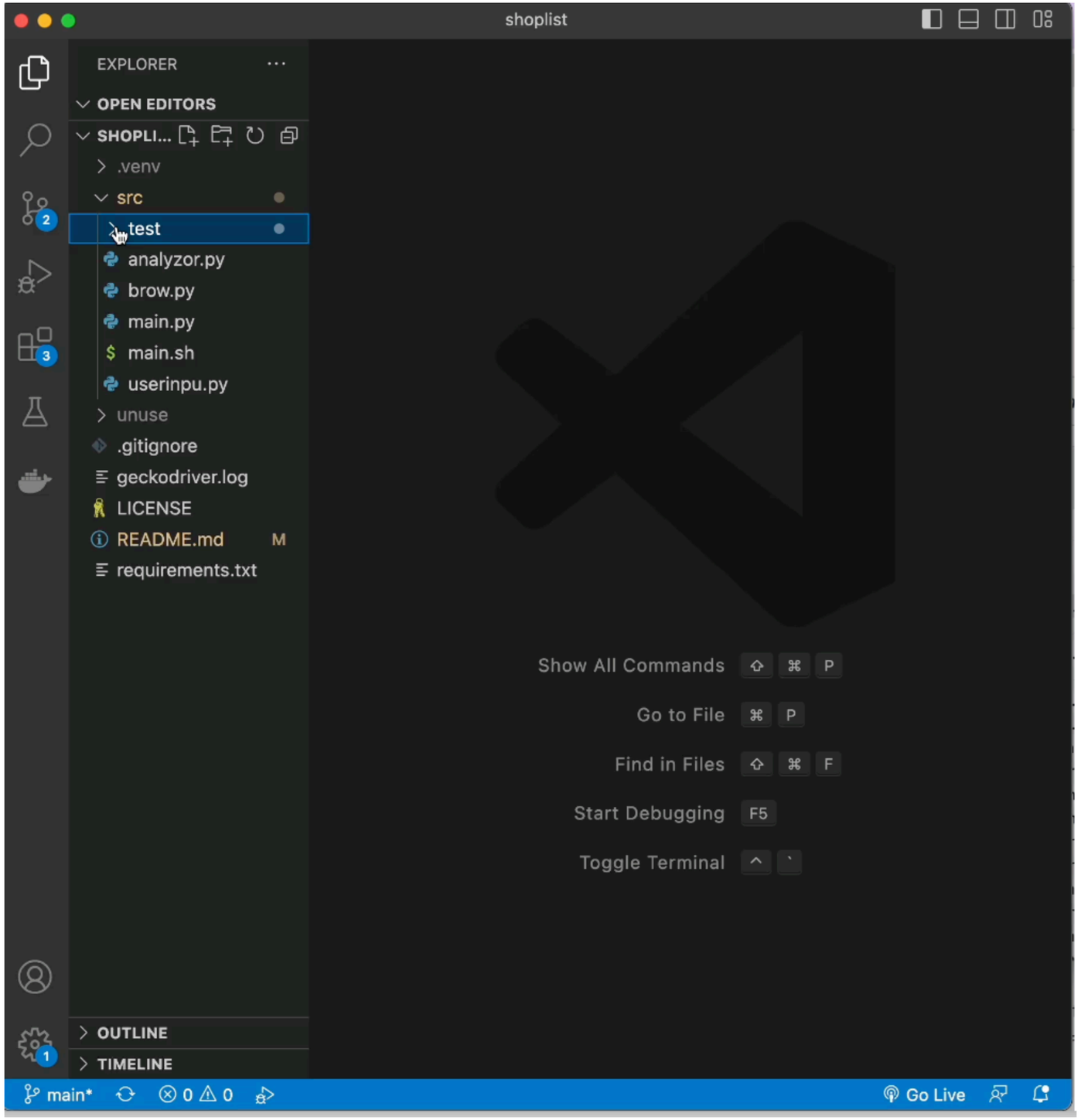
If data is not accessed locally, maybe using fake data to test is wiser.







Demo


shoplist
EXPLORER
OPEN EDITORS
SHOPLI...
.venv
src
test
analyzor.py
brow.py
main.py
main.sh
userinpu.py
unuse
.gitignore
geckodriver.log
LICENSE
README.md
M
requirements.txt
Show All Commands
Go to File
Find in Files
Start Debugging
F5
Toggle Terminal
OUTLINE
TIMELINE
main*
Go Live