|  |
| --- |
|  |
| Python Web Scraper |
|  |

|  |
| --- |
| Brandon Treston |

Table of Contents

[Abstract 1](#_Toc46775036)

[1.0 Introduction 1](#_Toc46775037)

[1.1 Proposed System 1](#_Toc46775038)

[1.2 Purpose & Motivation 1](#_Toc46775039)

[1.3 Project Objective 1](#_Toc46775040)

[1.4 Technologies & Tools 2](#_Toc46775041)

[1.5 Timeline 2](#_Toc46775042)

[1.6 User Guide 3](#_Toc46775043)

[2.0 Analysis 4](#_Toc46775044)

[2.1 Activity List 5](#_Toc46775045)

[2.2 Context Diagram 5](#_Toc46775046)

[2.3 Data Flow Diagram 5](#_Toc46775047)

[3.0 Functionality and Implementation 4](#_Toc46775048)

[3.1 Code Sample #1 4](#_Toc46775049)

[3.2 Code Sample #2 5](#_Toc46775050)

[3.3 Code Sample #3 5](#_Toc46775051)

[3.4 Code Sample #4 5](#_Toc46775052)

[4.0 Testing 5](#_Toc46775053)

[4.1 Unit Testing 5](#_Toc46775054)

[4.2 White Box Testing 6](#_Toc46775055)

[4.3 Black Box Testing 6](#_Toc46775056)

[4.4 Case Testing 6](#_Toc46775057)

[5.0 Conclusion 6](#_Toc46775058)

[5.1 Limitation 6](#_Toc46775059)

[5.2 Future Enhancement 7](#_Toc46775060)

[6.0 Author 7](#_Toc46775061)

[7.0 Reference 7](#_Toc46775062)

# Abstract

The objective of this software is to leverage HTML parsing web scrapping techniques with Python to obtain product information from various brands’ websites. Targets for this program will include women's clothing across all categories (pants, joggers, jackets, dresses, skirts, jumpers, etc.) The software will deliver two CSV files with the target information, one file containing “best sellers” and the other containing “normal sellers.”

# 1.0 Introduction

The software to which this technical document pertains is intended to be used with the specified list of 12 brand sites provided by Marieke Santema. The result of collaboration with Marieke and the author of this script, Brandon Treston (myself) is a piece of software that delivers two formatted CSV files containing product information the has been obtained via web scraping. While the core functions of this software *can* work on any site, it is important to note that to accurately retrieve the desired information, the target site’s HTML structure must be considered carefully when inputting the parameters upon which the software will execute.

## 1.1 Proposed System

To accurately retrieve the desired information from the list of brand sites, the program will query the site with specific parameters to retrieve a product list page for normal sellers and then a separate page for best sellers. The program will then proceed through the product list pages, visiting each product’s individual store page (if applicable) and scrape that product page for the desired information. This process will vary slightly due to the nature of web scraping and the varied structures of the sites. The script will repeat this process for the desired number of items to be scraped, being sure to not disrupt the site’s traffic with excessive HTTP requests.

## 1.2 Purpose & Motivation

The purpose for the creation of this software.

## 1.3 Project Objective

This software is used to scrape the following brands’ websites:

* American Apparel
* American Eagle
* Bershka
* Forever 21
* GAP
* H&M
* JCPenney
* Old Navy
* Reformation
* Uniqlo
* Urban Outfitters
* Zara

The data output for this program satisfies the following criteria:

* Brand
* Product Name - Description
* Product Id – Item #
* Price In USD
* Properties Fabric Composition - To Be Sorted as:
  + Leather, Cotton, Polyester, Acrylic, Rayon, Modal, Spandex, Nylon, Polyamide, Viscose, Elastane, Linen, Lyocell, Other – Unknown
* Html Page Name
* Time of Extraction

## 1.4 Technologies & Tools

* + 1. Python 3.8.2: The language this software is coded in. While one of the outlined requirements, it is also one of the best tools for this task due to its extensive libraries for web scraping, as well as the excellent documentation for those libraries.
    2. Pandas v1.0.5: Pandas is a fast, powerful, flexible, and easy to use open source data analysis and manipulation tool,  
       built on top of the Python programming language. [source]
    3. Google Chrome (on Windows): version 83.0.4103.116
    4. Selenium 3.141.0: Selenium Python bindings provides a simple API to write functional/acceptance tests using Selenium WebDriver. Through Selenium Python API you can access all functionalities of Selenium WebDriver in an intuitive way. [source]

## 1.5 Timeline

Day 1: 7/10/2020

* Setup environment for Python development
* Create project workspace
* Initialize GitHub Repo
* Establish dependencies
* Create Technical Document

Day 2: 7/14/2020

* Establish data requirements

Day 3-5: 7/15/2020 – 7/17/2020

* Begin code
* First prototype successful
* Refinement of first protype
* Addition of multi-page functionality

Day 6-11: 7/18/2020 – 7/23/2020

* Hiatus (personal circumstances.)

Day 12-15: 7/24/2020 – 7/27/2020

* Conversion to more Object-Oriented approach (added class)
* Testing & debugging
* Final analysis of script, finish technical document

## 1.6 User Guide

To use this script, the user must provide proper XPATH strings for the elements that are to be scraped. In total, the user will must provide proper parameters for the 3 methods of the script. The user will require all of the proper url’s for each site (including multiple urls for pages without dynamic filters.)

The first method sets the url of the catalog page that will be scraped.

The second method’s parameters tell the scraper the XPATH for the individual product links, and XPATH for navigating a sites pagination if appropriate.

The third method tells the scraper the brand and XPATH for the categories to be scraped.

Properly formatted input to the scraper will look like the following:

scraper.setURL('https://www2.hm.com/en\_us/women/products/view

all.html?sort=stock&productTypes=Blazer,Blouse,Body,Cardigan,Coat,Dress,Jacket,Jeans,Jumper,Jumpsuit,Leggings,Pants,Shirt,Shorts,Skirt,T-shirt,Top&image-size=small&image=model&offset=0&page-size=250')

scraper.getProductLinks('//h3[@class="item-heading"]//a[@class="link"]')

scraper.scrape('H&M',

'//h1[@class="primary product-item-headline"]',

'//dl[@class="pdp-description-list"]/div/dt[text()="Art. No."]//following-sibling::dd',

'//dl[@class="pdp-description-list"]/div/dt[text()="Composition"]//following-sibling::dd',

',',

'//span[@class="price-value"]'

# 2.0 Analysis

This section will include the logical diagrams that break down the basic function of the script.

## 2.1 Activity List

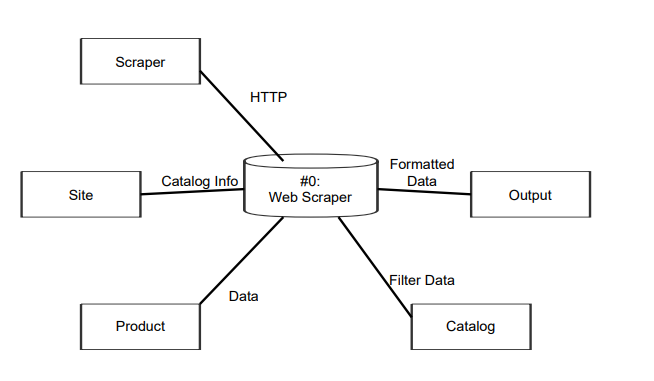
Activities:

* Navigate to catalog
* Check for active filtering
* Visit product
* Scrape data
* Return to catalog

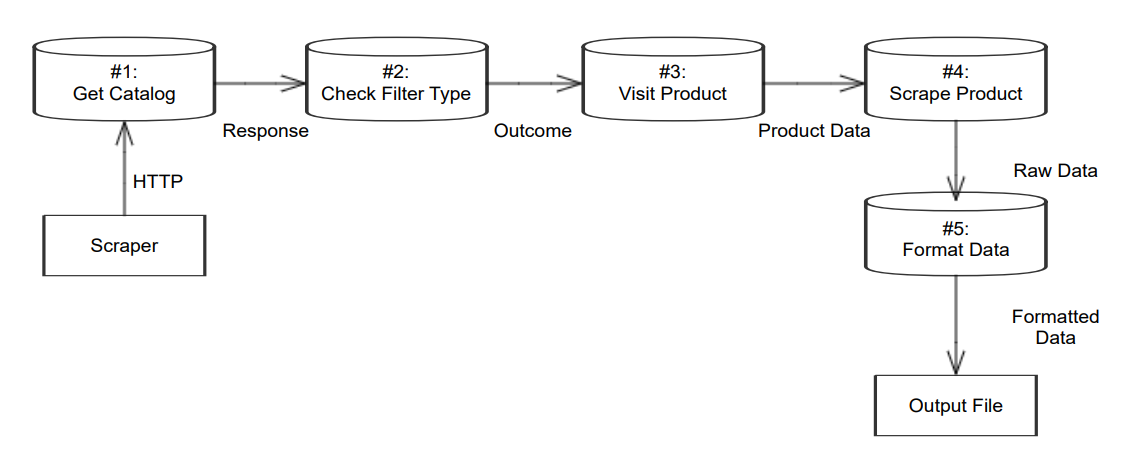
Entities:

* Site
* Product
* Catalog
* Scraper
* Output file

## 2.2 Context Diagram



## 2.3 Data Flow Diagram



# 3.0 Functionality and Implementation

The architecture of the 12 brand sites varies greatly from one another. In terms of product pages and catalogs, there are two approaches that need to be considered. The first is active filtering which lets you select the products you want to see, and the page dynamically renders those products. The second is static category pages which as the name suggests, does not dynamically re-render the page. With these two approaches in mind, the software will include code to navigate these two site architectures. To ensure consistency within the data and reuse code, the algorithm for scraping sites with active filtering will be reused on each static category page.

Another consideration for this software is which sites utilize pagination for their product catalogs. Wherever appropriate, the software will navigate between pages of a site’s catalog to continue scraping without interruption.

Furthermore, another issue that this program will navigate is the matter of site’s use of complex JavaScript to render html containing product information dynamically. To resolve this issue, Selenium will be used to automate the browser and run the JavaScript to render the appropriate HTML.

## 3.1 Code Sample #1

The following code sample will explain the getProductLinks() method, which as its name suggests, retrieves the urls for each product on the site’s catalog page(s).

def getProductLinks(self, productSelector, pages=0, productCount=250,

paginated=False, buttonSelector=''):

        elements = [] #temporary storage

        for i in range(0,5):

            self.browser.execute\_script("window.scrollTo(0, document.body.scrollHeight);")

            sleep(3)

        if paginated == True:

            for page in range(0,pages):

                #find all product links. returns list, stores at elements.

                \_elements = (self.browser.find\_elements(By.XPATH, productSelector))

                for element in \_elements:

                    elements.append(element)

                self.browser.find\_elements(By.XPATH, buttonSelector).click()

        else:

            \_elements = (self.browser.find\_elements(By.XPATH, productSelector))

            for element in \_elements:

                    elements.append(element)

        #slice the list if it exceeds product count

        if len(elements) > productCount:

            elements = elements[:productCount]

        for we in elements:

            #get the href attribute of each web element in elements

            self.links.append(we.get\_property('href'))

Parameters

----------

brandName : str

Name of brand from which the article of clothing is from.

nameSelector : str

XPATH string for XML/HTML web element containing productg name.

idSelector: str

String containing XPATH for the product id web element

compositionSelector : str

String containing XPATH for the composition web element

compositionSplitBy : str

String with the character that seperates the values of the composition, if exists.

priceSelector : str

String containing XPATH for the price web element

delay : int (optional)

Sets the delay to scrape each product, as specified in a site's robots.txt file. Defaults to 3 seconds.

## 3.2 Code Sample #2

Scrape() method

## 3.3 Code Sample #3

getType() method

## 3.4 Code Sample #4

processData() method

# 4.0 Testing

## 4.1 Unit Testing

The ‘pilot’ site that this software was run on to create the first prototype was American Apparel. Through many unit tests of the scraper on this site, the overall scraper’s functionality was completed and established a reusable framework that would allow for various types of input that would specify how the scraper would navigate the complexity of other site’s structure, while still maintain a consistent and meaningful output.

## 4.2 White Box Testing

White box testing showed that with the first 10 items from H&M and American Apparel’s catalog pages, as long as all the product’s composition descriptions were consistently formatted, the script would successfully scrape the data, package it into a dataframe and export to a csv file.

## 4.3 Black Box Testing

Black box testing with 250 items showed that within the items on a site, inconsistencies with the product composition make it impossible to export the dataframe with the data to a readable format (csv).

## 4.4 Case Testing

Case testing once again illustrated the same limitation observed above. The greater the number of products, the higher the chance of inconsistency in the data set. The result is a critical failure in properly parsing the product compositions into separate csv columns. However, the script is successful in producing a csv with product composition as a whole string in a single csv column.

# 5.0 Conclusion

## 5.1 Limitation

As to be expected with web scraping, the diversity in structure between sites is to be expected. This program has been designed with that in mind, to operate in as general a manner as possible and allow the script to be used on any site, given the information is there and consistent throughout the site.

This has been the biggest challenge in this project. Within a single site, especially for a product’s composition, the way the data is formatted can be inconsistent. Examples of such inconsistencies are present in American Apparel’s catalog site. One example of a product’s composition is formatted as ‘50% Cotton / 50% Polyester.’ This layout is perfect for parsing into a dataframe for csv export. However, another example from American Apparel is, ‘a timeless classic, our 100% cotton sweater fabric is essential for layering.’ In addition to this, some products have not any composition information. These inconsistencies have made it so that separating the contents of the composition string into separate columns in a csv file is not plausible for very large sets of data where the user cannot monitor the composition string for each individual product.

Additionally, another important limitation to consider is how some sites react to chrome operating in headless mode. For some sites (like H&M), using the scraped while operating chrome in headless mode will not work when scraping products.

Unfortunately, during the development of this project a personal matter had come up that kept me from working directly on the code for nearly a week. Had this not occurred, I am sure that I would have figured out how to navigate these inconsistencies, as the current implementation does so to a certain extent.

## 5.2 Future Enhancement

Being able to parse composition strings into columns is the biggest feature I would complete in this project. In addition to that, a GUI that would allow users to interact with a website to select elements to be scraped would eliminate the need for users to use complicated XPATH strings. Finally, implementations of AI and machine learning could lead to more efficient data scraping and help to better navigate inconsistencies within sites’ structures.

# 6.0 Author

Brandon Treston: I am a current undergraduate university student attending the New York Institute of Technology, pursuing a bachelor’s degree in Information Technology. I am slated to graduate in December 2020. I possess a background in information technology and computer science, with emphasis in full-stack web development using React and Node.js. I am versed in many different programming languages and can easily transition to new languages due largely to my deep understanding of the underlying logic that programming languages adhere to. When I am not coding or learning new technology, I host game servers, research current and upcoming pc hardware, create mods for video games, read, write, and watch science fiction stories and cook.

# 7.0 Reference