Contents

[Analysis 4](#_Toc66116807)

[The Problem: 4](#_Toc66116808)

[Why it suits itself to a computational solution: 4](#_Toc66116809)

[Computational Application: 5](#_Toc66116810)

[Why Use Selenium: 5](#_Toc66116811)

[Abstraction of how selenium works: 6](#_Toc66116812)

[Abstraction of how the web scraper works: 7](#_Toc66116813)

[User Accessibility: 11](#_Toc66116814)

[Limits: 11](#_Toc66116815)

[Requirements: 12](#_Toc66116816)

[Stakeholders**:** 12](#_Toc66116817)

[Interviews: 13](#_Toc66116818)

[Competition: 14](#_Toc66116819)

[Use: 15](#_Toc66116820)

[Decomposition: 16](#_Toc66116821)

[Success Criteria: 16](#_Toc66116822)

[Success Table: 16](#_Toc66116823)

[Design 18](#_Toc66116824)

[Inputting credentials: 19](#_Toc66116825)

[Generating The unique URL 19](#_Toc66116826)

[Scraping The Generated URL 19](#_Toc66116827)

[Algorithm for scraping Items URL: 19](#_Toc66116828)

[Pseudocode for Scraping Generated URL: 20](#_Toc66116829)

[Selenium & Multiprocessing Check out: 20](#_Toc66116830)

[Algorithm for Checkout 20](#_Toc66116831)

[Pseudocode For Selenium & Multiprocessing Check out: 21](#_Toc66116832)

[User Credential input: 21](#_Toc66116833)

[Pseudocode for inputting credentials: 22](#_Toc66116834)

[Generating URLS 23](#_Toc66116835)

[Algorithm for Generating URLs: 23](#_Toc66116836)

[Requests: 27](#_Toc66116837)

[Selenium Checkout: 27](#_Toc66116838)

[Justification: 28](#_Toc66116839)

[-Finding the pattern/using the bot to find said product: 28](#_Toc66116840)

[-Using BS4 for preference availability: 28](#_Toc66116841)

[-Requests: 28](#_Toc66116842)

[-Selenium Checkout: 29](#_Toc66116843)

[Usability Features: 29](#_Toc66116844)

[Usability check list: 30](#_Toc66116845)

[Developmental testing: 30](#_Toc66116846)

[Testing Development Stage 1 30](#_Toc66116847)

[Testing Development Stage 2 30](#_Toc66116848)

[Testing Development Stage 3 31](#_Toc66116849)

[Testing Development Stage 4 31](#_Toc66116850)

[Testing Development Stage 5: 31](#_Toc66116851)

[Development Stage1: 32](#_Toc66116852)

[URL Generation: 32](#_Toc66116853)

[Algorithm: 32](#_Toc66116854)

[Lessons from prototype: 33](#_Toc66116855)

[Review 37](#_Toc66116856)

[Testing table: 37](#_Toc66116857)

[Remedial Action: 37](#_Toc66116858)

[Success Criteria: 38](#_Toc66116859)

[Progress: 38](#_Toc66116860)

[Summary: 38](#_Toc66116861)

[Development Stage 2 39](#_Toc66116862)

[Speeding up URL Generation: 39](#_Toc66116863)

[Lessons for prototype: 40](#_Toc66116864)

[Algorithm: 41](#_Toc66116865)

[Review: 45](#_Toc66116866)

[Testing Table: 45](#_Toc66116867)

[Testing Evidence: 46](#_Toc66116868)

[Success Criteria: 47](#_Toc66116869)

[Progress: 47](#_Toc66116870)

[Summary: 48](#_Toc66116871)

[Development Stage 3 49](#_Toc66116872)

[Lessons from prototype: 50](#_Toc66116873)

[Data structures: 51](#_Toc66116874)

[Test Data: 51](#_Toc66116875)

[Implementation: 52](#_Toc66116876)

[Review: 53](#_Toc66116877)

[Testing: 53](#_Toc66116878)

[Success Criteria: 53](#_Toc66116879)

[Test Evidence: 54](#_Toc66116880)

[Summary: 55](#_Toc66116881)

[Progress: 56](#_Toc66116882)

[Development Stage 4 56](#_Toc66116883)

[Lessons from prototype 57](#_Toc66116884)

[Algorithm: 58](#_Toc66116885)

[Key Variables: 59](#_Toc66116886)

[Data Structures: 59](#_Toc66116887)

[Test Data: 59](#_Toc66116888)

[Implementation: 60](#_Toc66116889)

[Review: 62](#_Toc66116890)

[Testing Table: 62](#_Toc66116891)

[Test proof: 63](#_Toc66116892)

[Remedial Action: 64](#_Toc66116893)

[Success Criteria: 64](#_Toc66116894)

[Progress: 64](#_Toc66116895)

[Summary: 65](#_Toc66116896)

[Development Stage 5: 68](#_Toc66116897)

[Users interface creation: 68](#_Toc66116898)

[Key Variables: 68](#_Toc66116899)

[Implementation: 69](#_Toc66116900)

[Review: 72](#_Toc66116901)

[Testing Table: 72](#_Toc66116902)

[Success Criteria: 74](#_Toc66116903)

[Progress: 75](#_Toc66116904)

[Summary: 75](#_Toc66116905)

[Final testing: 75](#_Toc66116906)

[Post Development Interviews: 76](#_Toc66116907)

[Evaluation: 78](#_Toc66116908)

[Success Criteria: 78](#_Toc66116909)

[Other improvements/features: 80](#_Toc66116910)

[Usability check list: 81](#_Toc66116911)

[Maintainability: 81](#_Toc66116912)

[Limitations: 81](#_Toc66116913)

[Usability check list: 82](#_Toc66116914)

[Usability Features: 82](#_Toc66116915)

# Analysis

## The Problem:

People are trying to obtain sought after luxury items online all of the time but it is quite hard to do so when you have to compete against others trying to obtain the same item as you are. With companies like StockX being the middleman during the trading of sought after clothing and luxury items, by just being the middleman they are making over $1billion, this shows the sheer quantity of items being sold. This means that resellers are everywhere and are attempting to get these items with the intention of making profit. With all this competition it’s impossible to get these luxury items that are extremely hyped up. Our program seeks to be the answer and for those who pay for it they will be able to obtain whatever they want. My program will allow this by using the computer as the buyer of the item because computers are faster than people and speed is the defining factor between obtaining the item and it selling out especially when typical sell to times are on average 6 seconds for the items in question which is simply not enough time for any individual to fill out a whole check out form so why not have a computer do this for them.

## Why it suits itself to a computational solution:

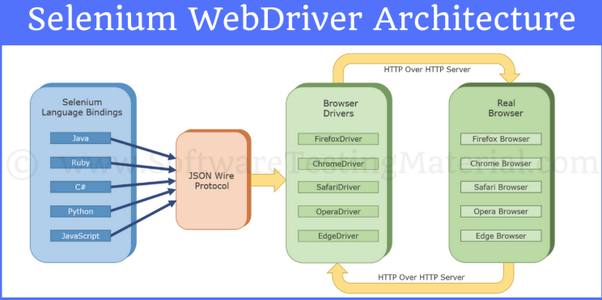
This problem utilises the computers multiple processors to buy multiple items and uses the computer to type at speeds faster than any human because of this the problem has to be solved by a computer and can’t just be solved by typing fast due to the competition and presence of other user utilising bots in the market

Websites are hosted on computers and they act as an interface between a the computer running the website the computer accessing the website and a human telling the computer how to interface the with a computer in real time. If we eliminate the real time decision making and instead give the computer what we want before the item is available. Then the computer isn’t slowed down by a human when talking to the web server as such it is just a computer communicating with a computer and this allows the process to be a lot faster and allows our customers to be the first to get the product they are going for. This is also because the computer does not need access to the user interface when performing the action for us. This is because it does not need to show a human the data so they can make decisions as the decisions are pre-set and without UI having to be loaded when buying an item it allows the computers to communicate with each other a lot faster.

## Computational Application:

when these items drop it’s a race and computers can fill in information many times faster than even a normal human could without a human being present this allows speeds to be limited by a number set by the user as to not trip any bot sensors.

This bot will use web-scrapping techniques to view what fields need to be filled in and then filling them with the given information the user will be also able to have multiple bots running to cop(buy) multiple items this way they can go for as much as possible as fast as possible web scrapping will also allow the user to not have to be directly looking at the screen to use it eliminating more human error. It will also replace the persons need to type which drastically improves speeds.



### Why Use Selenium:

~Diagram above from a Quora article

While most browsers have their own automation features, this allows for very low level actions as we know all code is made from a lower level version of itself so selenium goes the next step and employs features that allow you to further program and automate various tasks. What selenium have done is take this web automation and makes a large complex package of small actions into a simple one-line command that you can use in your code so you can do complex web automation while writing minimal amounts of actual code. This allows us to perform our goal in the time outlined

Here is a simple and technical explanation

Each selenium command interacts with the browser driver by sending a HTTP request

The browser driver is running a browser server which allows it to interpret each of the commands sent.

Once these instructions are interpreted by the browser server it uses the browsers automation software to execute the command

The browser sends it to a HTTP server

The HTTP server then sends whether the instruction has been executed back to the script.

Allowing the script to send the next instruction and then the process is repeated.

In more simple terms we can use the analogy of a medieval general leading ordering his troops:

The leaders have several formations they can use to complete different problems and he needs to perform these fast enough for any situation that arises.so he can’t waste time telling each individual soldier what to do so he says a larger command instead and they all know what to do because it has already been programmed and trained into them.

### Abstraction of how selenium works:

The sergeants view what the outcome of the execution was and report it back to their general and wait on the next command

The General issues command (http request) to go into a certain formation (command) to each sergeant (Browser driver)

Each sergeant interprets what that means for the troops they lead

The troops then go into the formation and execute the order

Our product can utilize this system due to the fact that the script will have to be custom for each website and at times be complex and using the normal web automation would take a while to code all the different variables in.

The pros of using selenium are that I am able to handle all this automation with python which means that I can do more complex tasks within the timeframe given for the project because of the fact python has so much automation of its own. By using selenium it also means I am able to make this bot very usable and easy to install and run with the only thing essential being the web drivers and the program it also is a lot more lightweight and doesn’t have to rely on APIs which a lot of websites tend to not offer due to botting. The website also believes the bot to be a human since it is sending web requests and typing and interacting with the web the same way a very fast human would and no faster.

It will also implement web scraping with beautiful soup. Now, the advantages of BeautifulSoup is that it can parse HTML like a simple XML and return the values required (text) to us.

For example, if we wish to look through a website (for testing only) we may want to locate all the anchor tags (<a></a>) and locate the href property inside. In our case this means we are able to use it to find the tags for the fields or get keywords like “email” and match them up with a list of a keywords in our program with each of those keywords pertaining to some sort of information e.g

### Abstraction of how the web scraper works:

Webscraper looks for next field

Webscraper finds “Email”

Matches with keyword “Email” in program

Enters correct information

This process will only be applied if we can’t find out a pattern of a websites URL so is more a fail-safe then a main method to doing this.

The main way we will be using web scraping is to check if an item or a size/colour of an item is being sold at the current moment we can do this by finding the CSS elements that contain things like the size and making them readable using our bs4(beautiful soup 4) to actually go through the web page and get our information in conjunction with our parser to put it into a readable and saveable format and our request library to actually connect to the URL we have Generated.

The way the request library works is that it takes huge task like connecting to a server through our computer and makes it extremely easy to deal with in python at the cost of a bit of speed in our case its taking care of the process that happens when trying to connect to the sites servers. The way this works is our computer sends a SYN asking if the webserver is ok to connect to the server the server then responds with an ACK which basically means yes I’m online and you can connect here our computer then does that and we scrape the page for the CSS element holding all the shoe sizes and then we use it in our program to display to the user and our bot whether its possible to obtain the item they are trying to get. This also takes into account the protocols we are using so we obviously need headers which we provide.

In coding terms this is how it works:

res = requests.get(GenUrl(model,size), headers=headers)  
page = bs4.BeautifulSoup(res.text, "lxmml")  
ListofSizes = page.select(".size-selector")  
ListofSizes[0].getText()  
Sizes = str(ListofSizes.getText().replace("\t",""))  
Sizes = Sizes.replace("\n\n","")  
print(Sizes)

The function that is explained in “Generating URLs”

Headers we generated

Bs4 being used to scrape the site along with the parser making it readable by us in lxml format

The piece of CSS that contains the shoe sizes that we are attempting to scrape from the site

Takes out the new lines and tab tags to make it more readable

In simple Terms this is a like a chef making a dish at :

The chef then looks for his ingredients (scrapes the webpage) For exactly what he needs an then cooks it all

He then tidy’s the food up so it is presentable

The waiter then collects the food and serves it(Request library closing the connection)

The chef(bs4) receives the order from the waiter(requests) that went and fetched it for him

**Generating URLs**

URL Generation

To find the exact web addresses of the items we want our bot to look for we have to find a pattern within the URLs that gives the exact shoe model and make for each shoe ,colourway and size so we can reconstruct the URL for our bot to know exactly where it needs to go in this case we are using adidas’ shoe URLs.

<https://www.adidas.co.uk/superstar->shoes/EG4958.html?forceSelSize=EG4958\_590

Model number

Model number

Size 6.5

<https://www.adidas.co.uk/superstar-shoes/EG4958.html?forceSelSize=EG4958_600>

Size 7

Model number

Model number

with this information we can stich together our own URLs by making a list of Model numbers for each site we already know the number that tells us the size so stitching together an URL can be done by.

Def GenUrl(name,model,size):

BaseShoeSize = 590

Shoesize = size – 6.5

Shoesize = Shoesize\*20

Rawsize = Shoesize + Base size

And then you would add the size and model number to the correct place in the URL this is what it would look like if GenUrl(EG4958,7)

<https://www.adidas.co.uk/EG4958.html?forceSelSize=EG4958_600>

note that the “superstar-shoes” part is missing that’s because its redundant and the link still works without it

a simpler analogy would be a translator

Message is sent to the translator in French (this is the User putting in the size adidas doesn’t understand)

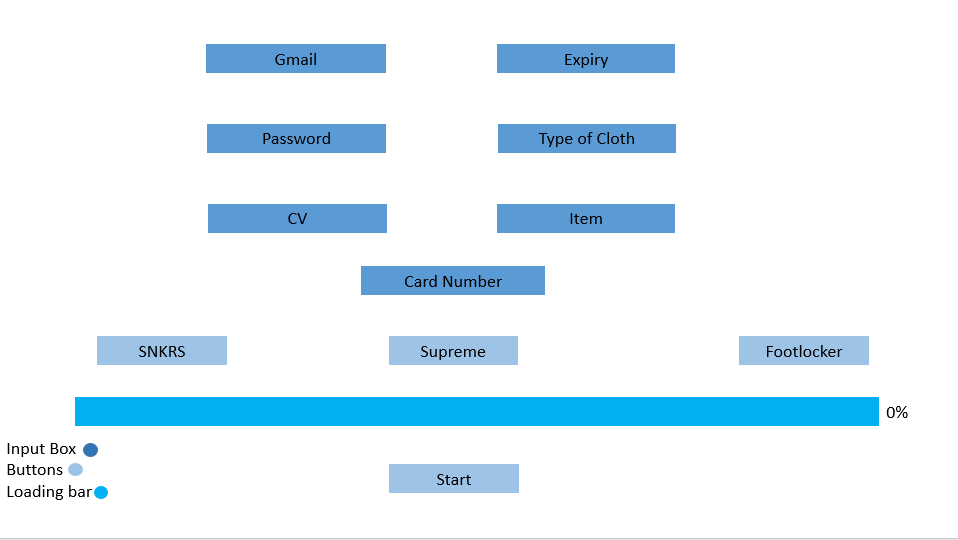
Tells the English speaker the translated information (this is adidas receiving the message in the code it understand)

The English speaker carries out said command or responds back

Changes the language in his head to English (putting it in the code that is understood by the wbesite

This will not work the same for every site since each site is unique finding its patterns will be different but the process will be the same the numbers are the only things that will change this means we can store a list of values with size values and codes for each site what I have just described in the first example is Adidas’ but the second example shows how it will work for most sites.

## User Accessibility:

****

Above is a design of what I would like my UI to look like. This is optimised for any user who   
may be using my program as its simplicity provides ease of use for anybody who isn’t or is computer illiterate.

The loading bar acts as an indicator that something is going on so those who aren’t as computer literate know that the program is responding and active.

The light blue boxes which are buttons help usability as they will be self-explanatory in what they do and saves the user from having to know the exact syntax of what they need to type.

Input Boxes with labels on them mean that users do not have to take a guess at what to put in each box and are instead told immediately

The buttons when clicked will have a visible difference allowing for the user to be able to know that they are working and so the user has visual feedback of it working.

## Limits:

The limitations in the automation would probably go as far as the bot solving captchas our bot won’t do this because in the current time frame it would take a while to train the bot to do so reliably and consistently.

To get past these limits we will allow users to run multiple tasks on the same order which means that multiple bots will be attempting to buy the same thing and some of them won’t get captcha tested allowing at least one to get an order. Or we would simply leave the captcha solving to the human to do or tell the human to do it before entering check out because it will disable the need to do so for the bots.

Our other limitation is anything to do with large servers as this would be expensive and setting these up may not be in possible in the current time constraints.

A limit is also drawn on how much speed the bot will perform at one because I’m working in python and two because some sites detect if you are filling in things too fast and could automatically decline cards.

Another limit has to be drawn partially on unlimited reusability because some of the top bots aren’t always available so thinking mine would be may be aiming to high but it will be attempted.

## Requirements:

The computer should have a higher amount of ram depending on how many tasks that need to be ran it also needs to have a quad core CPU for the fastest performance possible. They will also need to be running windows mac or Linux so they can run the executable. While this program would work on slower computers it wouldn’t be as fast when dealing with multiple tasks.

They also need to be able to download browser drivers so they would need to be on an administrators account. The browser drivers are only available on chrome, IE, Edge and Firefox so you would need one of these. You would also need a computer because the app isn’t compatible with mobile devices as of yet.

It needs a python interpreter as this is the code it will be written in

The program will require a robust user interface as this makes it much easier for users to use. On this interface I will need options to switch the script dependant on website. I will also need something that shows there are processes taking place to show to the user that a crash hasn’t happened. The program also requires inputs so instead of typing it to the console I require an easy way of allowing the user to perform inputs.

## Stakeholders**:**

There a wide range of stakeholders for this product but it is mainly aimed at people at the age of 16-18 attempting to make money through reselling this is because they typically have part time jobs and use that money to buy sneakers and flip them for small profits that go a long way in their day to day. They also know the most about this industry since them and their peers are the main demographic demanding these.it will be either that group or a group of the same age range just trying to not get beat by faster individuals’ could also be a group of older individuals with already lucrative businesses in the reselling market.

The common thing about all of this group is that most will already be internet literate and so making a program that they can use easily shouldn’t be extremely hard. The set up process would be as simple as downloading a few drivers so it shouldn’t be to hard for anybody that knows how to use a computer but there will also be instructions

Most of this group also has large amounts of disposable income this is because they are either not at the age to be taxed yet or don’t have to worry about paying for a house or rent or food giving them more money to use on our products to further make a profit. They also may have a large disposable income because they already have a lucrative spot flipping sought after items. This all means we can charge a lot for our bot since they will make a lot using it and don’t have much else to spend their money on. Furthermore the stakeholders have already spent a lot for other bots in the same area that cost upwards of £300 for something like cybersole meaning that they are willing to spend the money but because there isn’t any competition bots like cybersole completely dominate the market allowing them to make massive profits. This aims to be a cheaper alternative so people don’t have to spend copious amounts of money to get their favourite shoe.

T’challa lives in Africa and is one of our stakeholders is interested in using our bot because he is trying to make money by reselling but he can’t especially when it comes to hype releases because they either release when he’s work or he is too late and not able to beat everybody else trying to combat this we have adapted our program to be able to buy whatever you need and even give it a time to start, this means that T’challa doesn’t have to worry about being at work when the drop is happening

Octane lives in South America and has used a bot before but the problem is he was ip banned because of it since some companies will do such things if a person has been found to be using automation. To combat this, we have made it so people that have proxies can use them with our program. This means that people can anonymously connect to the websites without the worry of being banned for automation. Furthermore, we have put in place measures to hide the fact certain parts of things that make the fact chrome is automated obvious.

Gibraltar Lives in the US and isn’t very good at using computers so isn’t able to type so fast, his son is a big sneaker head so he likes really rare trainers since Gibraltar can’t compete with the reaction speeds of young individuals it means that he can’t really buy the shoes his son wants online and has to instead rely on camping outside for hype releases of new shoes for his son. Our program is very easy to use this is because our UI is being run by HTML/CSS since this is what websites use our app will be guided and obvious in its use so even people that are not so computer literate like Gibraltar will be able to easily buy his son a pair of shoes.

## Interviews:

**Resellers**

**T’challa:**

**1.Are you satisfied with your current method of acquiring assets to resell??**

“Yes I am I currently use a bot named Cyber Sole and because I am extremely skilled at using it I have made a very lucrative business from it because of its reliability”

**2. Is there anything you don’t or didn’t like about your current bot?**

“At the start I had to practice with it quite a bit as its not the easiest thing to use people think that you just pick up the bot and use it and that’s it but the truth is it takes a lot of learning how to properly use it before you see any of your £300 investment comeback. It would be nice if the interface was more user friendly and obvious and if it had definitions to what everything does or maybe even video guide or a well written documentation.”

**3. Have you ever used auto fill programs if so did you enjoy them??**

“ I used Fillr one time but it’s very unreliable as some sites it will just be caught which can get my IP banned on top of this even when it does work it tends to work a lot slower than if I just decided to use my bot.”

***Casual buyers***

**Gibraltar:**

1. **What do you find most challenging about buying sneakers?**

“I’ve never been fast at typing and I’m not very lucky when it comes to raffles for sneakers and now they have stopped store releases I don’t know how I’m going to get my son the sneakers he wants for Christmas”

1. **Why have you not considered using a bot?**

“I have been told there is a possibility of getting ip banned for botting or even getting your address and card details blacklisted and I wouldn’t want that to happen as I wouldn’t be able to get my son the things he wants on top of that they need a great deal of knowledge to use them effectively and I don’t have the tine time to learn one”

**Bot creator:***We sent an email to the bots creator to find out what problems we might run into and why they chose to make certain decisions with their application.*

1. **You chose to limit the amounts of bots sold why is this, how can an application sell out??**  
    We chose to limit the amount of bots we sell because of the fact that if we do not limit it  
    to many people get the bot which turns down the effectiveness of our bot because the

more people that have the bot means those who already have it get less successful

transactions as there is more competition from people using the same tool as them.

This is very problematic because it directly harms the effectiveness of our product.

1. **Why do you charge an upkeep fee of £100 for your bot does this not deter customers and lose the loyalty of those already using your bot?**

We do this because as said before there is a limit to how much people can use our bot if we do not charge an upkeep fee it means it’s very hard to make any money without hurting the effectiveness for current user by releasing more bots and saturating the products market. So we charge an upkeep and if consumers of our product aren’t happy with that they are free to leave and another space is opened for somebody to buy one of our bot without hurting the effectiveness for our other users.

Both group A and B have had issue pertaining to speed or the reliability of the method that they are using so our product has to be both fast and very reliable as this would allow us to target both groups with our product. We could make sure that the speed is fast by timing the bot during development and editing the code accordingly to make it faster. This could be done with methods such as disabling the graphical interface. The two answers from our bot creator allows us to gauge the market cap of this bot as giving everybody a bot just hurts the customers we already have. We could limit our customers by giving them a unique hash

## Competition:

There is other competition out there such as a bot called Cybersole but these are typically extremely expensive at £300 and a further £100 every 6 months. While this bot is extremely good it costs a lot and is constantly sold out. Which means the only viable way to get them is through a reseller and these bots typically resell at about £3000 which means its near impossible to get these bots so due to availability of the competition it isn’t a sustainable solution so we shall attempt to make this bot as sustainable sale wise as possible.

There are a plethora of other bots but the reason Cybersole is so sought after is because of its ease of use , reliability and speeds which is why these factors will be part of my success criteria.

The market is also quite lacking in available commercialised bots due to them constantly being sold out meaning that It would be quite easy for our bot to fill that demand.

FIllr or other autofill’s are another solution to these problems browsers come equipped with this feature so you wouldn’t even have to actually have to pay for a solution. This method can compete with bots since its just a click for some websites but its always faster than another human manually typing it. The only problem with this is that sites have become accustomed to easily just detecting these auto fillers and will just decline your card if you are caught doing so making this quite unreliable even when even using dedicated applications like Fillr to do so. Websites catch so many that it is not reliable for those trying to make an income off getting these items or even simply the people just trying to get a pair of sneakers for themselves consistently. Which is why it needs to be done by a bot.

Reviews of Fillr:

*Tama luck4 points·*[*2 years ago*](https://www.reddit.com/r/supremeclothing/comments/8ifnya/fillr_discussion_mobile/dyrd7sw/?utm_source=reddit&utm_medium=web2x&context=3)

***Tried coping the helmet with Fillr and the s\*\*\* took like 30 seconds to load all my info, took a fat L”***

This clearly shows that Fillr has varying speeds so it is unreliable as a long term piece of software. This needs to be considered in my own code

***"The Fillr app is “next-gen autofill on steroids”, and it’s specifically created for all your mobile devices (running iOS 8 or later, of course)" writes Techly***

The above review shows that when Fillr works it’s very effective and it seems that it is meant

for use on a mobile application and as there aren’t any bots that works well on the mobile

platform which shows that Fillr can work well on the platform its created on. The problem

with it is that it doesn’t work on PC which is the main platform for shoe resellers. However it

does work well for who its designed for. This review has told me how I can add features to

target specific users and doing this can greatly improve its effectiveness of my program.

## Use:

The information will be given to the bot before the purchase via a user friendly interface and since ease of use is such a big must and is one of the reasons Cybersole succeeds so heavily at the top of the market it is a must for us to have an interface that isn’t just usable but also visually appealing. This is why I will be using either pyqt5 or a different html service to make the application since both of these will give me a lot of freedom while keeping the interface looking as good as I need it to be. Once the information is input the bot will keep it for the time of the transaction. The user will be able to add proxies so they aren’t IP banned and they will be given options pertaining to the speed of the bot and how many tasks should run at the same time. After putting in all of the information the user will be prompted to choose their dimensions and other preferences. Once the tasks are finished running the result will be showed on each of the task tabs so you know if and when you have succeeded.

## Decomposition:

My program will be structured as subroutines with each subroutine being either a needed function for a company’s website or the complete buy process for a company’s website. This means I can have as many options as I want and add as many as I want without breaking my code and it allows me to keep up to date on adding new sites when the need arises. Since each site needs a different method I think this is the perfect process of decomposing this task.

## Success Criteria:

Its success will be measured by the speed, reliability , attractiveness of the user interface, ease of use and reusability.

All of these are important due to the fact that if one lacks it cant compete with top bots like Cybersole and its also what our stakeholders look for most when looking for a bot. While our stakeholders are computer literate they are most likely used to having programs that are easy to use and are low in verbosity this means making these as basic as possible through the interface would be a big measure of success. They also actually want to be able to cop most of the things they go for and if not one then multiple so its speed and reliability to actually obtain items would be a massive measure of success as well as its ability to repeat said results even on extremely hyped up items via using multi-tasking to maximise chance of succeeding. We also want the bot to be extremely reusable because we don’t want our customers to have to go to resellers to get an inflated price since we want to fill that void. However this is less important due to the fact most bots aren’t even able to come close to not being sold out.

We have these as the basis for our success criteria because when looking at reviews and what makes the bots good these are the things that were highlighted the most by users.

The compatibility is another part that plays a massive role in it because of the fact resellers hop from site to site or multiple drops can be happening on multiple sites ,compatibility is a massive measure of success because its very hard to make a living off of a one trick pony and because items can go for different prices everywhere its best to have at least support for the main sites

“Their massive site list, the best UI across all AIO bots and their slick mobile app are also reasons why this bot deserves the top of my list” ~ botsthatwork.com

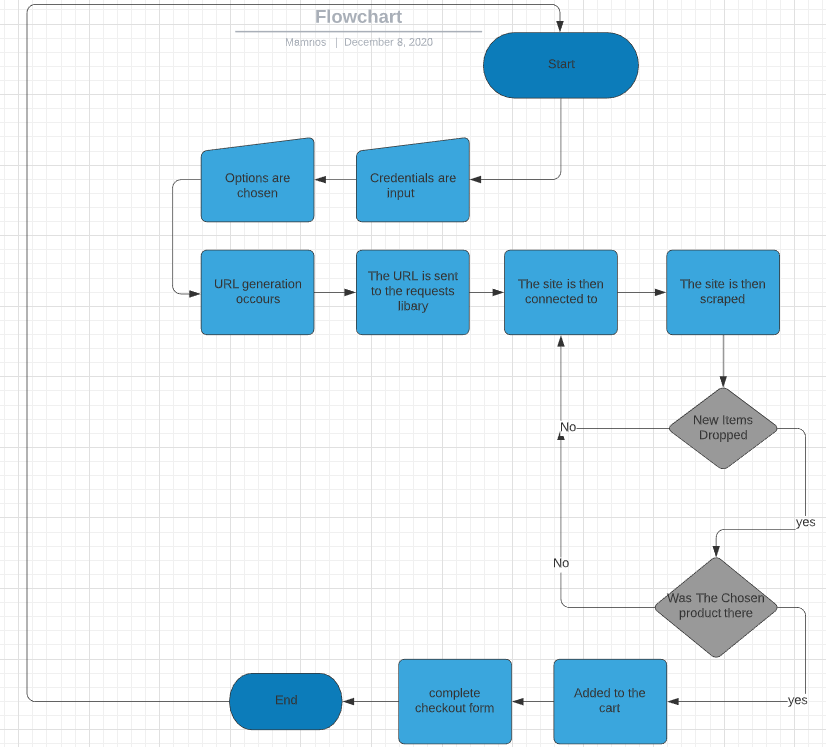
“Cyber AIO is one of, if not the best AIO bots on the market. With a retail of 300€, resale of over 4000$. It has proven itself week after week to be consistent bot cooking on all of its supported sites. Extremely limited user base, super friendly GUI it is a bot people dream of.” ~ appgroove.com

Both these reviews main points are about things from the success criteria so this is why getting these done well is my measure of success.

## Success Table:

|  |  |
| --- | --- |
| Criteria | How to evidence |
| Speed - A speed of less than 20 seconds | Recording or screenshot of time taken for the application to get to get to checkout |
| Reusability - Its able to be used multiple times by the same person | Show that the program can |
| Speed - it runs without needing to have a browsers UI running | Can be shown by no flag code |
| Ease Of Use - Easy to use for computer literate people | Have somebody who isn’t a reseller successfully |
| Versatility – This application can be used for multiple products | Display the code and explain how you can easily make it search for other products |
| UI should be clean and to the point. | Having a very clean output with as little verbosity as possible |
| Make a process that figures out URL generation | Screenshot how we would go about doing it in design and screenshot where it happens in the code |
| The program must be able to buy any item on the websites it supports automatically | Run the program from start to finish and get passed the checkout screen |
| Make an easy way for the user to input the item they are looking for | Screenshot the code that allows you to specify items |
| Make sure the user knows when its loading | Screenshot an output that shows its actively loading something |

# Design

****

This is a basic Overview of the process each process goes through remember this can happen multiple times depending on how many processes of this the user has set.

## Inputting credentials:

## Generating The unique URL

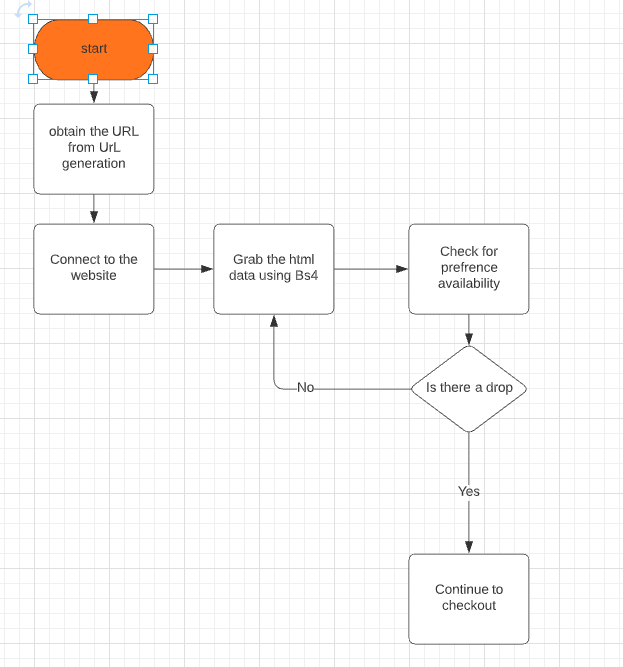
**-**Finding the pattern in each URL myself

-Taking down the code for each product on the website using automation

-Stitching the product id together with the rest of the URL

## Scraping The Generated URL

### Algorithm for scraping Items URL:

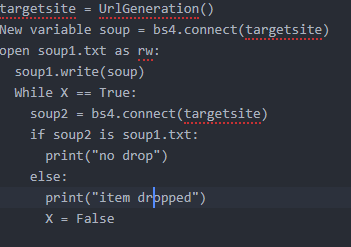


-The drop is detected by cross referencing the data that was grabbed most recently being different from the data previously grabbed

-It is also used to check for preference availability

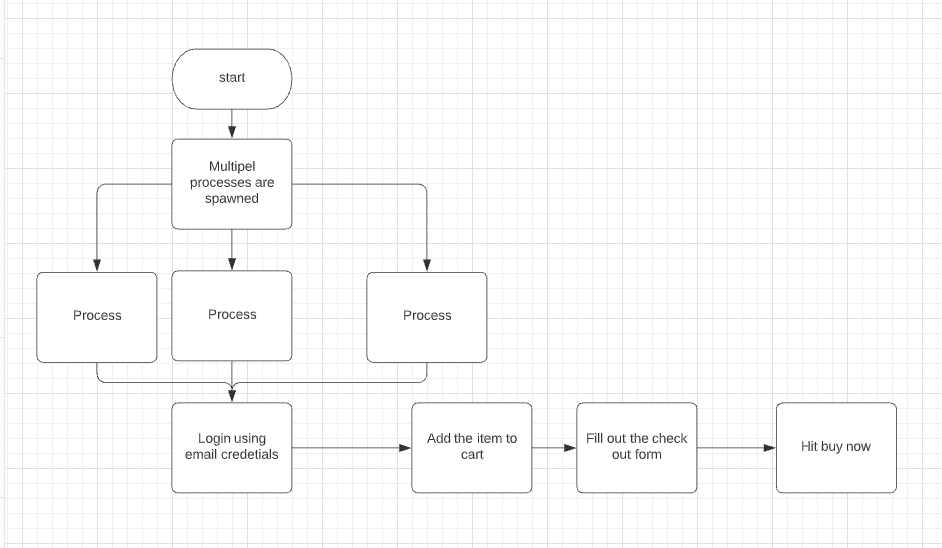
-If a URL cannot be generated that leads directly to the product, we scrape the site to see where exactly our automation needs to click

### Pseudocode for Scraping Generated URL:

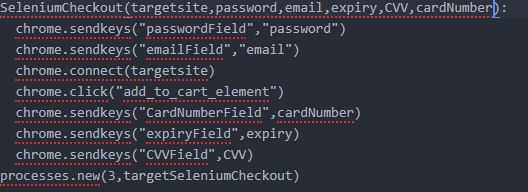


## Selenium & Multiprocessing Check out:

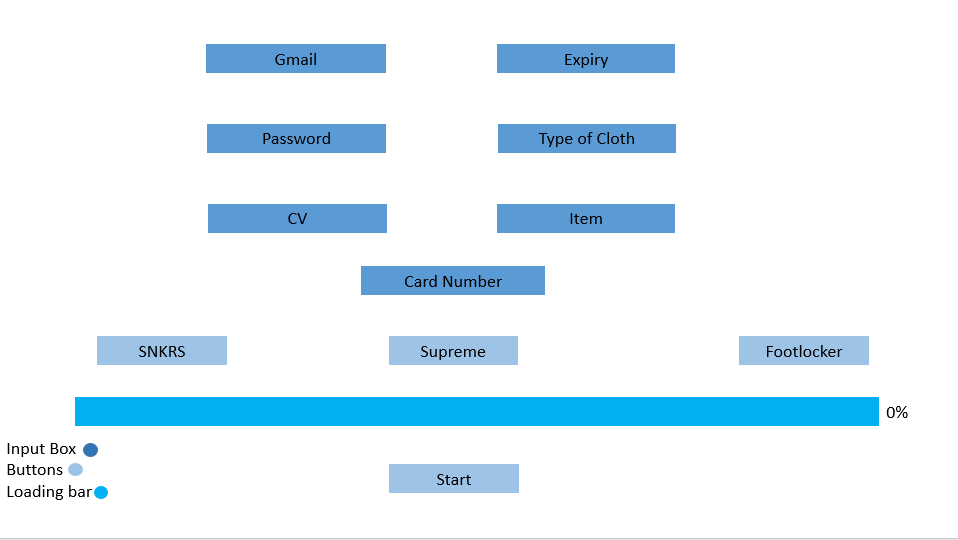
### Algorithm for Checkout

**-**The credentials that were entered at the start are given to selenium to be entered at check out without the person having to be there. This is done with multi-processing.

### Pseudocode For Selenium & Multiprocessing Check out:



## User Credential input:

****

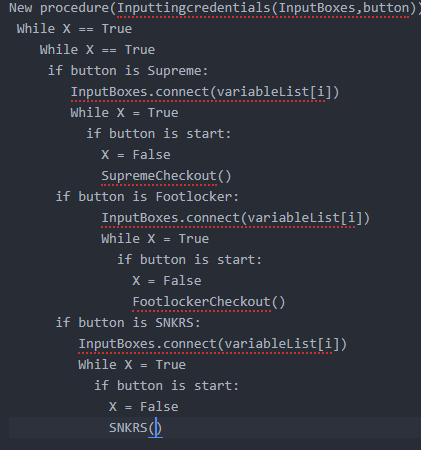
The User inputs their information into each of the fields provided. Fields will be easily accessible, and it will be obvious what information is required and where

For generation of the URL we need to know what the user wants and the best way to do this is via text field/input box. Because there a whole host of different names of items and user inputs that could go in this field and what goes in this field is subjective to what item the user is going for. If we were to instead have predetermined options, we may not have an item that the user wants as new items are constantly creates on websites and there is no way to have them all on a comfortable user interface this also applies to emails. Emails are personal and as such we must have a personalised way for a user to input their unique email and information into the program.

There are four buttons the user can use of the four only 2 need to be used 1 of them is the start button and the other is one of the three website options. I didn’t make these input boxes as the program is limited to as many sites as I have developed for at the time. Allowing for an input box could leave room for human error with capitalisation spelling and syntax as their or many ways you could refer to these websites as such the easy option is buttons.

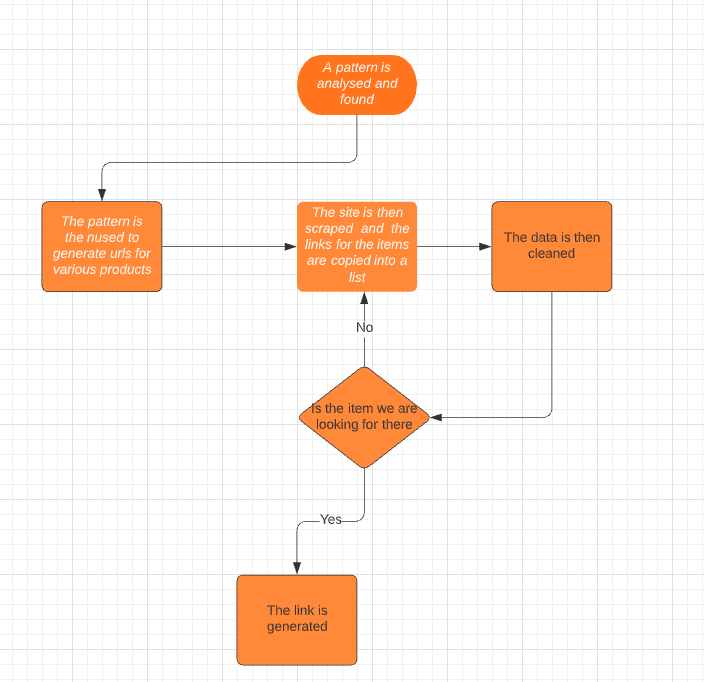
Once all the options are set the start button is press and the process of buying the sneaker begins.

## Pseudocode for inputting credentials:



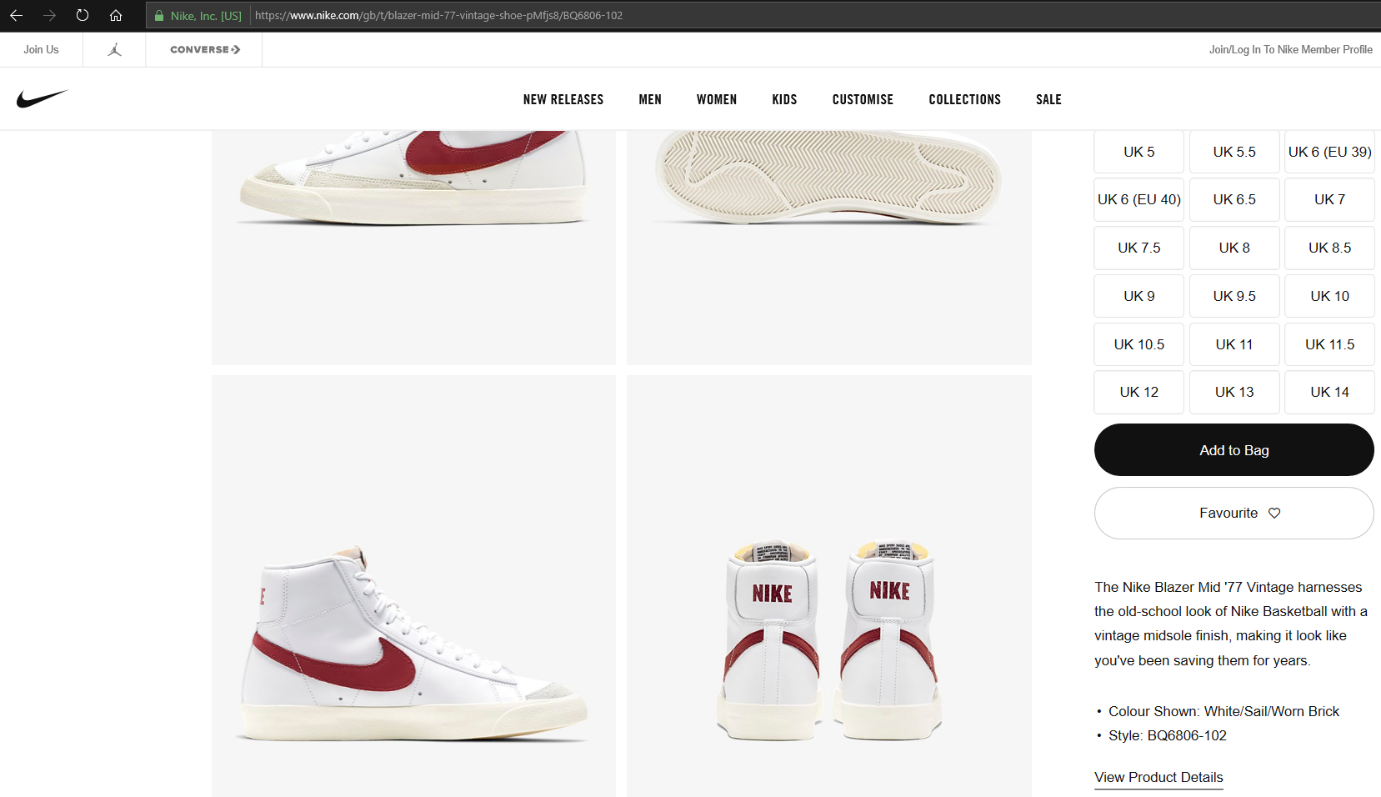
## Generating URLS

### Algorithm for Generating URLs:



**-Finding the Pattern in each URL**

This can be done VIA two ways, and this is different for every website and as such the process changes. I could go and look for the pattern in the product codes for the website and program in the predicted model number of the product that is attempting to be attained e.g



Over here we have an URL that we need to reverse engineer in hopes of finding a pattern or finding a link to allow us to automate this process for the rest of the website.

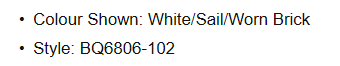
<https://www.nike.com/gb/t/blazer-mid-77-vintage-shoe-pMfjs8/BQ6806-102>

as we can see the URL currently looks like a jumbled mess messing about with it we find that shortening the URL to

<https://www.nike.com/gb/t/BQ6806-102>

still allows it to work

that means that BQ6806-102 is our unique colour and shoe code and I have found out that incrementing the last 3 digits changes the colour. We can also scrape this number off of the site using BS4 since its right here.



This won’t be the same for all sites but for the sites that we can find a pattern for it allows us to be able to have a process down for the whole page.

The second way is the way it’s going to be done when something releases since its harder to stitch together a URL for something we don’t know about. The section of the page where the product will most likely land will be scanned then using BS4 and it will be refreshed and scanned constantly until an item fitting the keywords of the wanted item is found and then that item will be checked for availability and purchased.

Let’s assume that the Nike SB’s have just dropped:



It then proceeds straight to checking out the shoe with the help of selenium



It will then continuously scrape until it finds the word “Shane”



The Bot would go to the section Skateboarding and then to the shoes

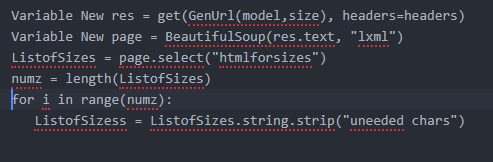
**Other ways we use web scraping**

**-Preference Availability:**

Without going on the site the user can instantly know what items are in stock and fi the criteria you set out of colour/size is. This is done by simply downloading the page in a readable XML format that way we can program what exactly to do with the information whether its to go to the next size up or to alert the user their piece is out of stock but it all stems from making it readable.

**Making it readable:**

Pseudo code for cleaning in URL generation:



This is how I will make the data scraped from BS4 readable to our program and it being in a list stripped allows us to go to each logical point in the list and if something is missing. Let me demonstrate this with the flow chart I used in my analysis

Psudeo code for scraping for the correct urls

res = requests.get(GenUrl(model,size), headers=headers)  
page = bs4.BeautifulSoup(res.text, "lxmml")  
ListofSizes = page.select(".size-selector")  
ListofSizes[0].getText()  
Sizes = str(ListofSizes.getText().replace("\t",""))  
Sizes = Sizes.replace("\n\n","")  
print(Sizes)

The function that is explained in “Generating URLs”

Headers we generated

Bs4 being used to scrape the site along with the parser making it readable by us in lxml format

The piece of CSS that contains the shoe sizes that we are attempting to scrape from the site

Takes out the new lines and tab tags to make it more readable

In simple Terms this is a like a chef making a dish at:

The chef then looks for his ingredients (scrapes the webpage) For exactly what he needs an then cooks it all

He then tidy’s the food up so it is presentable

The waiter then collects the food and serves it (Request library closing the connection)

The chef(bs4) receives the order from the waiter(requests) that went and fetched it for him

**Generating URLs**

Once this happens we can then easily incorporate it to the rest of our code since its in a format that python can easily deal with it further down the line when we are showing it to our user or incorporating it further.

## Requests:

Requests as I explained in my Analysis is a module that allows you to send numerous http queries to a site in a human friendly way so it makes interfacing and connecting to the web a lot easier so we could send a GET request to the site for a manner of reasons and the request library would be doing that for us. This would be done to GET a page on the website or specific information for BS4 to then scrape which it does up there.

## Selenium Checkout:

We use the web automation with selenium I described in my analysis to fill out the forms/text boxes needed to complete the transaction.

We give selenium something to identify all the text boxes which is normally their id

We then take the id and look for certain keywords that easily signal to the program what it is like “last” would most likely be a field for a last name

The code then matches it up and uses the send keys to input the value for last name

In code this would be one simple line on selenium’s side.

driver.find\_element(By.ID,"RESULT\_TextField-1").send\_keys(lastname)

## Justification:

### -Finding the pattern/using the bot to find said product:

While URLs can be input a lot of the time The URL for say an Unreleased item can’t be found for obvious reasons So, to anticipate the drop it needs to be done computationally because if not then we are relying on the reaction speed of the user which doesn’t meet our speed part of the success criteria. However, if we use the method I suggested before and have a bot constantly refresh the page until something meets the criteria given by the user. This method means that we wont have to know what is coming out before hand and we wont have to rely on human reaction speed which fills our speed criteria. Finding the pattern needs to be done for the times that a user wants to buy an item without going through the website to search for it AKA something has already released or predicting what the URL is trivial meaning we can actually predict what the dropped product is going to be under which makes our program even faster

### -Using BS4 for preference availability:

While Pages can be checked for sizes and other factors beforehand this does not meet our ease of use or speed criteria as we are trying to make an AIO(all in one) bot and to contend with what is out there it needs to be both fast and look good as we I have stated in my success criteria and since again this is a human action we have to take it away to meet our success Criteria while simultaneously providing the ease of use so the user doesn’t have to go and check themselves. This is the reason this action also needs to be done computationally. Which is why I am using the method specified above and **making it readable** process allows us to use the knowledge of the availability on multiple parts of our program for example If we would want to use it to go to the next size up and Also, presentation wise it easier to incorporate it into the UI if its readable to humans.

### -Requests:

While the user could input the URL as stated before this can’t happen if they don’t know the URL so the only way to do this without the request library is for the user to get a newly dropped products link which would take time due to the fact they would have to click it as fast as they can and copy and paste the URL into our program. These are a lot of human actions and as we know taking away the part of human error is key to maintaining the speed of the whole program and this is instead all completed by the request’s library which would not only allow us to be faster but allow as to be more AIO putting our ease of use up and therefore further meeting our success criteria.

#### Streamlined:

While we could of use libraries like urlib the problem with these is that it is from a different time and is a lot older than requests as such the code is a lot worse and it is not as streamlined as the shiny new requests library the requests library was born partly form libraries such as urlib so performance wise it is way ahead of urlib. This is also because with new programming practices and innovation the code written now is a lot less “messy” meaning that it is a version that is a lot more streamlined than its old counterparts.

#### Code Writing/ Debugging:

When you run into an error in your code with requests it makes it a lot more clearer when debugging said code for example instead of just stopping it will throw a specific SSL error when it can’t authenticate a certificates as well as a host of other features which are great for debugging code and its documentation clearly covers each bug you may encounter and what to do in the event of doing so.

Requests is also higher level than URLib3 this is because it is basically just an abstracted version of URLib3 this leads to code being easier to write in the given time constraints.

### -Selenium Checkout:

While somebody could enter all of the values themselves with an autofill this can take as much as 20 seconds! Seeming as auto-fills get detected quite easily the only way to do it non computationally would be to manually put it in which would take a terribly long amount of time this means that we can save the most amount of time at this part making it really key to the whole program and on top of that it adds to the stealth because we aren’t doing it with an easy to detect auto-filler, This means it hits both criteria’s of speed and stealth if we perform this computationally using selenium.

While there are other framework selenium is the best for this project due to:

#### Ease of Use and adaptability:

Selenium is compatible with all major browsers such as Chrome, Firefox, Safari, Internet Explorer, Opera, and Edge. This means we don’t have to rewrite our script every time a customer changes browser. It also seamlessly works on multiple platforms this is due to all code being written into an intermediary selenium code which is then written into the final code to be executed. This process means that we don’t have to massively change our code when going from platform to platform. However in QTP you it is only compatible with 4 platforms.

#### Less Hardware intensive:

While there are other automation frameworks such as QTP and UFT selenium is a lot more light weight this means that when it comes to requirements users don’t have to have as much of a powerful computer to be able to successfully run it as intended. This also means that selenium is faster than both QTP and UFT

## Usability Features:

**My User interface should be simple –** The interface should be simple that way it is accessible to individuals with lower computer literacy

**My website should be accessible to those that are visually impaired** – I will make sure there isn’t any clashing colour combinations I will keep it clear and simple for example not using yellows and oranges and use an easy to read font with the words I create being larger than usual

**My user interface will use as little words as possible** – I can do this by not having may paragraphs of words that way those with conditions such as dyslexia won’t have trouble reading my user interface. I will also use input boxes so individuals don’t have to read as many buttons. This also helps individuals with lower English reading levels

**My user interface will have ample spacing between words –** This will also help those with dyslexia as the space between each word will help them not get jumbled together

**My user interface will have simple non-complex wording –** This is so those with a lower English/reading level have an easier time using my application

**The terminal for the application should have simple error messages for easy troubleshooting** –

This should be so any error can be troubleshooted by the user which helps maintainability in the long run as it is less for customer service teams to deal with.

**Loading should be easily communicated** – This is so those less computer literate understand that something is going on.

## Usability check list:

* **My User interface should be simple**
* **My website should be accessible to those that are visually impaired**
* **My user interface will use as little words as possible**
* **My user interface will have ample spacing between words**
* **My user interface will have simple non-complex wording**
* **The terminal for the application should have simple error messages for easy troubleshooting**
* **Loading should be easily communicated**

## Developmental testing:

These are the tests we will employ at each developmental stage:

### Testing Development Stage 1

|  |  |  |  |
| --- | --- | --- | --- |
| Test Number | Test Description | Test input Data | Expected result |
| 1 | Can the program successfully scrape the data from the web page and provide it in a clear concise format | Hooded Facemask Parka - Tigerstripe Camo | Yes, and the data scraped is the only data scarped |
| 2 | Can the program use the scraped data to generate a link | Hooded Facemask Parka - Tigerstripe Camo | Yes and the link works as intended |

### Testing Development Stage 2

|  |  |  |  |
| --- | --- | --- | --- |
| Test Number | Test Description | Test input Data | Expected result |
| 1 | Does the bot hang until it detects a drop | Hooded Facemask Parka - Tigerstripe Camo | Yes, and it communicates this to the user until there is a drop |
| 2 | Does the bot create a list of links of new items only that are able to be checked for the item the user is looking for | Hooded Facemask Parka - Tigerstripe Camo | Yes, and the bot uses this data to differentiate between whether a drop has happened or not. |
| 3 | Does this speed up the process | Hooded Facemask Parka - Tigerstripe Camo, 2 processes | Yes the current speed is at 35 seconds so a decrease of 2 seconds would be great |

### Testing Development Stage 3

|  |  |  |  |
| --- | --- | --- | --- |
| Test Number | Test Description | Expected Result | Test Input Data |
| 1 | Does the program output a readable item title for the item for use later on in the program. | The program produces a list of titles ready to be compared to the item the user input. | Hooded Facemask Parka - Tigerstripe Camo |
| 2 | Does the program cross reference the users input with the list of titles to identify the correct item | Prints out the link/title for the item the user is looking for | Hooded Facemask Parka - Tigerstripe Camo |
| 3 | Does the program attach the users input to a link | The program outputs a usable link that corresponds to the users sought after product | Hooded Facemask Parka - Tigerstripe Camo |

### Testing Development Stage 4

|  |  |  |  |
| --- | --- | --- | --- |
| Test Number | Test Description | Input Data | Expected Result |
| 1 | Is each field of the checkout form entered with the intended data | Hooded Facemask Parka - Tigerstripe Camo, 99089080,11/40 | Yes, the intended data goes in the correct place |
| 2 | Are you able to connect to your email account | [matthewpinnock.mp@gmail.com](mailto:matthewpinnock.mp@gmail.com),  password | Yes you connect to email services and the program continues |
| 3 | Was an item able to be successfully added to the cart | Hooded Facemask Parka - Tigerstripe Camo | Yes |
| 4 | Was the checkout form filled and checkout successful. | Hooded Facemask Parka - Tigerstripe Camo, 99089080, 11/40 | Yes and an item was bought. |
| 5 | Checkout happens in less than 20 seconds with the headless mode disabled GPU | Hooded Facemask Parka - Tigerstripe Camo, 99089080, 11/40 | Checkout takes 20 seconds or less |
| 6 | Can you spawn multiple processes | Hooded Facemask Parka - Tigerstripe Camo | Multiple chrome windows are spawned |

### Testing Development Stage 5:

|  |  |  |  |
| --- | --- | --- | --- |
| Testing Number | Testing Description | Expected Result | Input Data |
| 1 | Does the loading bar clearly communicate the progress through the program | Yes, it does the percentage rises as the bar does | Null |
| 2 | Do all the buttons have functionality | Yes the buttons are clickable and usable | Null |
| 3 | Can the titles for each box be easily changed | Yes they can be changed during the program or before | Product |

# Development Stage1:

## URL Generation:

This stage takes care of the supreme URL generation which is done by scraping every single item on the supreme shop for the links within the a tags this allows us to have URLs directly linking to the product page and after the browser has started this takes less than a second.

## Algorithm:

We then take each item that has been scraped and grab only the children so we get closer to only having the parts we want.

The user chooses the site they want (supreme in this case)

The requests library connects to the main product page

BS4 then scrapes all the inner-article class which holds the links for each product within the a tags along with a lot of other information.

We then manipulate then build a new string with the information we want from each child.

We then take the links once they have finished building and put them into a global Link list

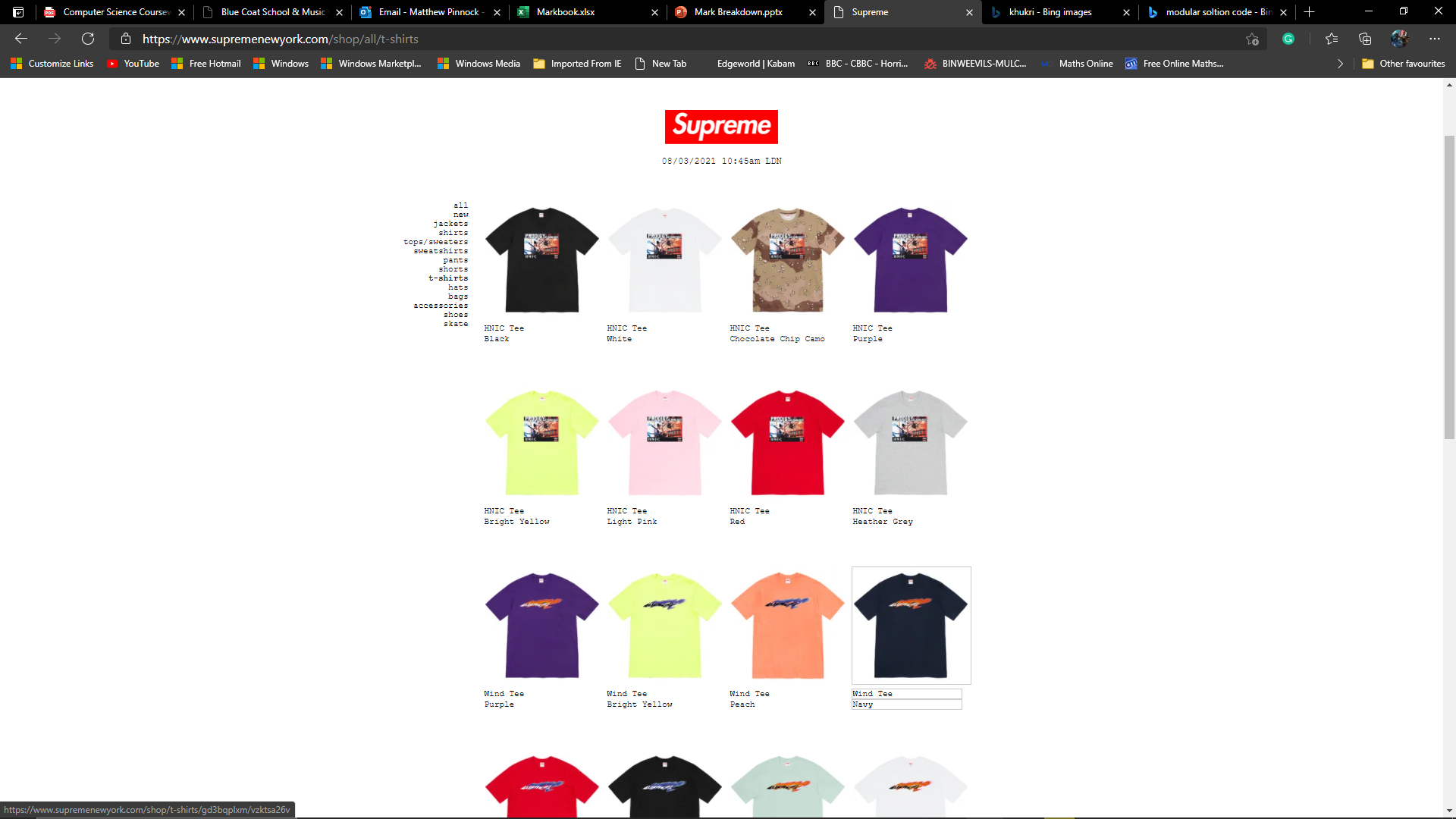
Key Variables/Data Structures:

Headers(string),browser(web driver path),res(string/link),page(parsed lxml page),article(BS4 element),Links(python list),children(list of anchor tags),counter(integer),LinkBuild(string),choice(string)

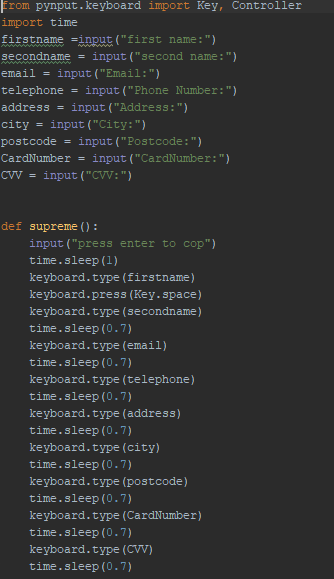
Stage 1 Development:

## Lessons from prototype:

The prototype lacked any URL generation which is part of the reason that it was so slow as the user would have to find the item themselves for this phase which can be very timely as there are a lot of products I each section and finding and selecting the exact product costs a fair amount of seconds. Below is a picture of only some of the products in the t-shirts category and it is so longyou may sctoll down this as it goes on for a while.



This is showing that the prototype lacks functionality to automatically find the product for you



As you can see it just starts with entering your first name which shows that it it is just for filling to the checkout form.

## Implementation:

After this stage we will scrape each link in the list for certain keywords pertaining to the product.

Below this is the chrome-driver and is what my code is using to communicate with my browser it creates a fingerprint that is exactly identical to chrome and allows the input of commands a user would normally do it has all the features of chrome and is what allows my chrome to be controlled by my code it acts as an intermediary between the browser and my code and allows them to communicate by my browser sending information to it and it being executed on chrome.

browser = webdriver.Chrome(executable\_path= "C:/Users/matth/Documents/year 12/Computing Y12/chromedriver\_win32/chromedriver.exe")

Testing:

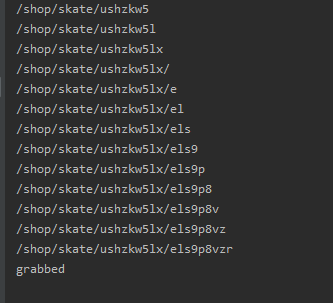
First Error came about when I ran it after trying to make the program use the chrome web driver the problem was that python didn’t understand that the path given was a path. To fix this I added forward / instead of the \ windows uses this is because using \ for paths isn’t understood by pythons Unicode but using / is understandable by all operating systems and it means u don’t need to signify it’s a raw address.

browser = webdriver.Chrome(executable\_path= "C:/Users/matth/Documents/year 12/Computing Y12/chromedriver\_win32/chromedriver.exe")

Another came while scraping the URL for the names of items I first tried to do this by printing everything on the page as I hoped it wouldn’t be that verbose and I could easily distinguish and grab each item. However in practice this was not so simple as the output was extremely verbose and unusable so I had to filter out the redundant parts that way it would become usable

After a couple successful test of getting the webpage to give me a list of all the elements in the products list it ended up being to verbose so I ended up taking each item from the list and making it so I just had the usable parts. I didn’t know how quite to go about this and kept getting errors pertaining to me treating the list of objects as a single item because I thought I could just pull the children of all of the items but it turned out I couldn’t. So I need to do it in more of a sophisticated way I did this by getting each item in the list ,so it wasn’t trying to do this to the whole list of items at one time, this way I could get the children of each of those items and then once I did that and it worked after being tested it still was a bit to verbose so at this point I built a list of strings with those strings and made a loop that would continuously index different parts of the verbose string to make a simpler version of it that was usable for my program once the loop ended and only then would one string of characters be complete so I added an if statement that adds them to the second list when the string is fully formed and then clears the list of the list that is used for building the string.

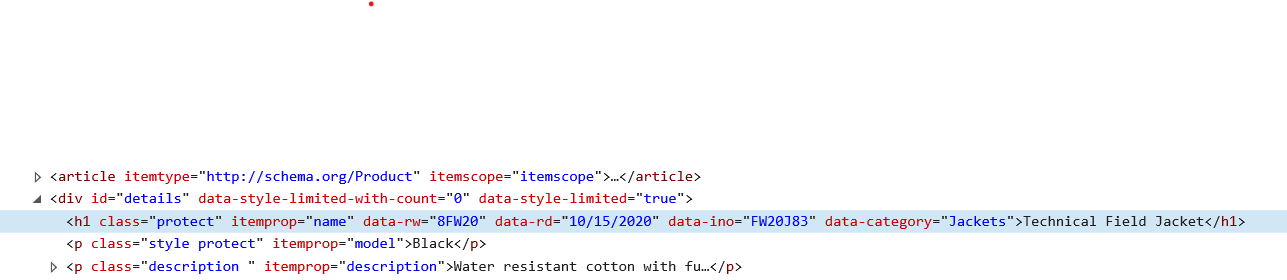
Below is the process of it building



Then once it gets to the bottom which is the URL completed it is then grabbed and put in the “Links” list and then all the incomplete variations are wiped.

As seen here:

for i in article:  
 children = i.findChildren("a" , recursive=True)  
 counter = -1  
 #we find the children of inner article so we get closer to just the link the children of inner article has the link aswell as a href and an image which we dont need  
 children = str(children) # turns children into a string  
 LinkBuild = "" #starts an empty sting  
 #we start a new list to build a string of just the link without the useless information  
 for i in range(28):  
 #28 is how long the link string is  
 counter = counter+1 #appends our own counter  
 LinkBuild = LinkBuild +children[15+counter] # the link starts at the 15th index of the string  
  
 if counter == 27:  
 #we need the counter to know whemn to append it in and no further  
 print("grabbed")  
 Links.append(LinkBuild) #add the built link to the list

The reason we have a counter is because the string is that many characters long so we don’t want it to carry on building redundant information after we have the link as this will cause furt0her issues when trying to use the links later on in the program. The reason the child starts at the index of 10 was because the string would start to build at the wrong position giving me 10 characters that I don’t need and messing up the functionality of the link.

The reason I did it like this instead of just finding the html object by following a line of children was because this was the closest child in the html and so I couldn’t be more targeted then I was here:

I also couldn’t have taken a more generalised approach that would allow my code to be easier to write because by grabbing the whole child and not being targeted in my approach I would have more than just the name of the product which means I would have a lot of redundant information which could cause errors in the long run for example if I was trying to compare a users input to the name of the actual item I wouldn’t be able to do so as easy because I would have to filter out so much information to make what should be matching match.

## Review

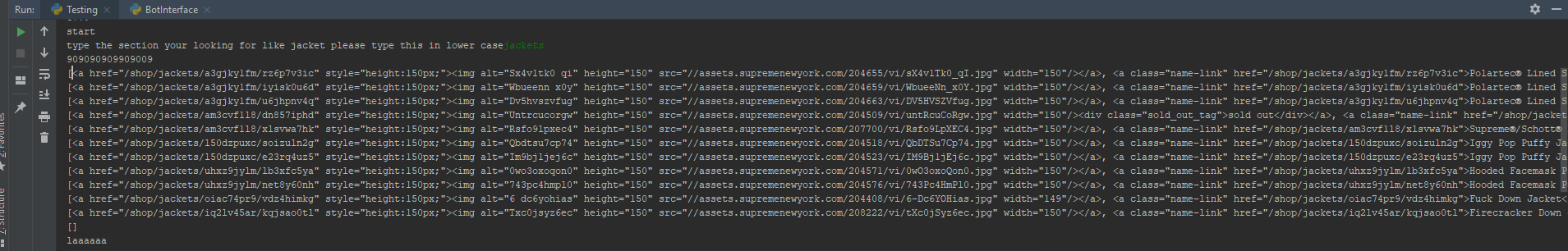
### Testing table:

|  |  |  |  |
| --- | --- | --- | --- |
| Test Number | Test Description | Expected result | Actual result |
| 1 | Can the program successfully scrape the data from the web page and provide it in a clear concise format | Yes, and the data scraped is the only data scarped | The website was successfully scarped however we did not scrape only the data needed to build the link |
| 2 | Can the program use the scraped data to generate a link | Yes and the link works as intended | Yes and the link works as intended |

### Remedial Action:

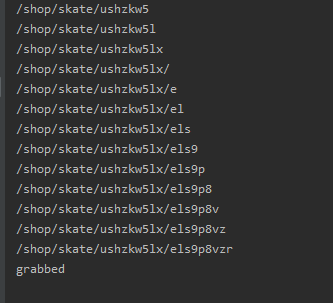
Test 1) we ended up cleaning the data scraped this is evidenced in the process described in what I call as the process of I building the link.

**Example of unbuilt / clean, scraped output:**



[<a href="/shop/jackets/a3gjky1fm/rz6p7v3ic" style="height:150px;"><img alt="Sx4v1tk0 qi" height="150" src="//assets.supremenewyork.com/204655/vi/sX4v1Tk0\_qI.jpg" width="150"/></a>, <a class="name-link" href="/shop/jackets/a3gjky1fm/rz6p7v3ic">Polartec® Lined Sideline Parka</a>, <a class="name-link" href="/shop/jackets/a3gjky1fm/rz6p7v3ic">Tartan</a>]

**Example of cleaned data:**



/jackets/iq2lv45ar/kqjsao0tl

### Success Criteria:

|  |  |
| --- | --- |
| Criteria | Met? |
| Speed - A speed of less than 20 seconds | N |
| Reusability - Its able to be used multiple times by the same person | N |
| Speed - it runs without needing to have a browsers UI running | N |
| Ease Of Use - Easy to use for computer literate people | N |
| Versatility – This application can be used for multiple products | N |
| UI should be clean and to the point. | N |
| Make a process that figures out URL generation | Y |
| The program must be able to buy any item on the websites it supports automatically | N |
| Make an easy way for the user to input the item they are looking for | N |
| Make sure the user knows when its loading | N |

### Progress:

This phase was largely successful in laying the foundations of the program as link generation is essential and while the process I have went about generating the link can be improved I have provided good structure to be able to build upon in the later phases where I speed up URL generation. The way I have structured this part of the program with it being a subroutine means it is modular in nature and so when it comes to speeding this part of the process up changes can be made and tested in isolation to the rest of the program this makes writing and maintaining the code a lot more feasible in the long run. Making sure that my program is modular in design is essential to its maintainability and my development this is because I can continuously improve the code that I have laid down while being able to constantly test it without having to run through the rest oof the code

### Summary:

At this point in time the code is not fully functional and able to be used for what we want to use It for. This is fine as functionality will improve as development progresses. There is not a User interface either but this is also fine as this is better built once the program is finished being created this is because The structure of the program has to be laid out in full before I am able to integrate a User interface into it is also just makes the testing process longer as I have to wait for it to load every time I load up the program.

the reason I have done it in list form is because then it easy to index for use later in my program this is the code in its entirety:

import selenium  
from selenium import webdriver  
import bs4  
import requests  
import webbrowser  
import lxml  
  
headers = {"User-Agent": "Python"}  
browser = webdriver.Chrome(executable\_path= "C:/Users/matth/Documents/year 12/Computing Y12/chromedriver\_win32/chromedriver.exe")  
browser.get('https://www.supremenewyork.com/shop/all')  
  
res = requests.get('https://www.supremenewyork.com/shop/all', headers=headers)  
page = bs4.BeautifulSoup(res.text, "lxml")  
article = page.findAll("div", class\_='inner-article')  
Links = []  
  
def SupremeLinkGen():  
 Links = []  
 for i in article:  
 children = i.findChildren("a" , recursive=True)  
 counter = -1  
 #we find the children of inner article so we get closer to just the link the

children of inner article has the link aswell as a href and an image which we

dont need  
 children = str(children) # turns children into a string  
 LinkBuild = "" #starts an empty sting  
 #we start a new list to build a string of just the link without the useless

information  
 for i in range(27):  
 #28 is how long the link string is  
 counter = counter+1 #appends our own counter  
 LinkBuild = LinkBuild +children[15+counter] # the link starts at the 15th

index of the string  
  
 if counter == 27:  
 #we need the counter to know when to append it in and no further  
 print("grabbed")  
 Links.append(LinkBuild) #add the built link to the list

return(Links)  
Choice = input("what site are you looking to use")  
if Choice == "supreme":  
 Links.append(SupremeLinkGen())

# Development Stage 2

## Speeding up URL Generation:

My aim for this was to make it so getting the correct URL did not take as long as it has needed to and there was an obvious place where I could do it. Scraping all the links in a section was very slow because it relied on the users internet speed to go through this massive list of links that could have been the correct item and each one of these links would have to be scraped to see if this was the correct link for the item that was being looked for. This meant that it went through all of it just to be able to find the correct thing and it means that drops would take an abhorrent amount of time to complete since it is redundantly looking in URLs that aren’t even part of the most recent drop. Instead I wanted to make a way that shortens the list of links by detecting when a drop has arrived and then only using the list formed from the new drop, meaning it would only search for the newest items and compare them to everything else. This is a lot more effective because it allows me to produce a much smaller list meaning that it has a lot less links to go to what would have been 10 or 20 is now 3 or 5. This speeds up the process immensely.

## Lessons for prototype:

There were no lessons to be learned as talked about in the last stage the prototype lacked URL generation so there was nothing to speed up with code. The only way to speed that up in the prototype was for the human using the program practice clicking on a product as fast as possible. However speed wise the prototype had to intentionally delay itself so the user could click each field as the prototype couldn’t go to the next field for the user this meant there was issues because if the users reaction time was not fast enough then they would miss a field slowing the processes down immensely or even missing out on buying the item so speed wise it was very bipolar.

keyboard.type(firstname)  
time.sleep(0.7)  
keyboard.type(secondname)  
time.sleep(0.7)  
keyboard.type(telephone)  
time.sleep(0.7)  
keyboard.type(postcode)  
time.sleep(0.7)  
keyboard.type(address)  
time.sleep(0.7)  
keyboard.type(city)  
time.sleep(0.7)  
keyboard.type(postcode)

the time.sleep was the delay needed as average human reaction time is 0.5 mili seconds 0.7 was a good time to account for those slower.

## Algorithm:

wdqwdqwdqwdqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqwesdqwdqwddddddddddddddddddddddddddddddddddddddddddddddddddddddddddddddddddddddddddddddddddddddddddddddddddddddddddddddddddddddddddddddddddddddddddddddddddddddddddddddddddddddddddddddddddddddddddddddddddddddddddddddddddddddddddddddddddddddddddddddddddddddddddddddddddddddddddddddddddddddddddddddddddddddddddddddddddddddddddddddddddddddddddddddddddddddddddddddddddddddddddddddddddddddddddddddddddddddddddddddddddddddddddddddddddddddddddddddddddddddddddddddddddddddddddddddddddddddddddddddddddddddddddddddddddddddddddddddddddddddddddddddddddddddddddddddddddddddddddddddddddddddddddddddddddddddddddddddddddddddddddddddddddddddddddddddddddddddddddddddddddddddddddddddddddddddddddddddddddddddddddddddddddddddddddddddddddddddddddddddddddddddddddddddddddddddddddddddddddddddddddddddddddddddddddddddddddddddddddddddddddddddddddddddddddddddddddddddddddddddddddddddddddddddddddddddddddddddddddddddddddddddddddddddddddddddddddddddddddddddddddddddddddddddddddddddddddddddddddddddddddddddddddddddddddddddddddddddddddddddddddddddddddddddddddddddddddddddddddddddddddddddddddddddddddddddddd

Then the list is only generated from the new links that are present these links where found by keeping what is new in our new list and deleting what was in the other older list from our new list.

The first item in the list is then constantly checked by a new one that is generated until there is a change and a the “drop is detected”

List of Links is generated by scrapping every single item for its link

wdqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqwesdqwdqwddddddddddddddddddddddddddddddddddddddddddddddddddddddddddddddddddddddddddddddddddddddddddddddddddddddddddddddddddddddddddddddddddddddddddddddddddddddddddddddddddddddddddddddddddddddddddddddddddddddddddddddddddddddddddddddddddddddddddddddddddddddddddddddddddddddddddddddddddddddddddddddddddddddddddddddddddddddddddddddddddddddddddddddddddddddddddddddddddddddddddddddddddddddddddddddddddddddddddddddddddddddddddddddddddddddddddddddddddddddddddddddddddddddddddddddddddddddddddddddddddddddddddddddddddddddddddddddddddddddddddddddddddddddddddddddddddddddddddddddddddddddddddddddddddddddddddddddddddddddddddddddddddddddddddddddddddddddddddddddddddddddddddddddddddddddddddddddddddddddddddddddddddddddddddddddddddddddddddddddddddddddddddddddddddddddddddddddd

Then each one of the links is passed to be connected to reducing the speed since now we don’t have to connect to as much

Key Variables:

|  |  |
| --- | --- |
| Wrl (Work Run List) | This acts as the file in the code and we call it when we need to perform an action on the file. |
| firstrun | This is a Boolean that can either be True or false it is used to differentiate the first run form the others, so things are executed in the correct order. |
| Del\_List | This list contains all the indexes of items that are in both the old and new list this way we can pop exactly the correct one |
| Check | Check is the new list of links that we check the old List against to see if there are any changes this list is generated constantly until the website drops the products and it updates. |
| Links | This is the first list of links we generate, and it is what I keep referring to as the “old list” the first item of this is compared to the first item in the check variable so we are notified when a drop happens. |
| sectionLink | This is used to build complete links its just a like to which section is being scraped |
| counter | This is used to count counteract the fact that the indexes change when something is popped so it is a number that is used to fix the value in Del\_List and make sure it goes to the correct place. |

Data structures:

|  |  |
| --- | --- |
| Wrl | Var(.txt file) |
| firstrun | Boolean |
| Del\_List | Integers |
| Check | Python List |
| Links | Python List |
| sectionLink | String |
| counter | integer |

Test Data:

|  |  |
| --- | --- |
| Jackets | This is the section we want to buy the item from we have chosen Jackets |
| <https://www.supremenewyork.com/shop/all/> | This is key because we add the products section and identifier to the end of this to get onto the correct product page |
| “First run” | This identifies that the run we are in is the first run and it has run as such and built the correct “Link” list. |
| “here1”,” here2”,”here3” | These are printed because of a problem I had where at the end of the subroutine it would run code so the here’s were used to see in what order and where it was being run |
| “no drop detected” | This is self-explanatory and is there, so the user knows why the bot isn’t proceeding it also shows that it is constantly refreshing and checking the site for new |
| “drop detected” | This tells the program that we are ok to go ahead with everything |
| “Reload” | This signals that a new link has been added to the list this was put there, so I knew that the list was being generated correctly |
| Links[0] | This output shows me the first link in the old list so I can compare it to the new one |
| Check[0] | This output told me whether it was changing when new data came since this was the first item of the new list |
| “bang” | This signals the popping of the links that aren’t new and are present in both lists. |

Implementation:

I had to have a way to storing the original verbose list of potential links without having to form it constantly as such I created this process which confirms if it is the first run and if it is it builds the list for you and stores it as a string this means that it doesn’t have to build it constantly and it means that there is a differentiation between the “Links” list (the old one) and the “Check” list (the new one)

with open("LINKS.txt","a") as Wrl: #opens our links file  
 print(firstrun)  
 if firstrun == True:  
 for i in Links:  
  
 Wrl.write(i+"\n") #write the current link into the file  
 print(i)  
 print("First run")  
 print(firstrun)  
 return SupremeLinkGen(sectionLink, False) #starts the subroutine again with

first run as false

qwrqrqwrqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqq

This writes all the links into the file and then adds a new line so it can be easily indexed

This then calls to run the process again but this time with firstrun as false, so It doesn’t rewrite the list again

This makes sure that it’s the first run

This opens the file as append

This next section detects when the drop has happened by seeing when the first items in the old and new lists have changed signalling that new items have been added and there is a new drop after that it then takes all the new Links and adds them to the Links list this is the start of updating our old Link list. After updating the list with the new items, we need to deal with the old ones that are still in the list and so we add all the indexes of the old items these are the ones that appear in the new list into Del\_List.

with open("LINKS.txt","r") as Wrl: #opens our links file

check = Wrl.readlines() #variable holds the lines and data on them  
 print(check[0])  
 print(Links[0]+"\n")  
 if check[0] == Links[0]+"\n": # compares the first old link with the first new

If these are equal it means that there isn’t a drop no drop detected is printed and the code is ran again constantly until a Drop is detected

A change happens the site is updated and drop is detected

Puts the links that aren’t in both lists (the new ones) into the Link list

link in the list and if they are the same  
 print("no drop detected")  
 SupremeLinkGen(sectionLink,False) # starts the process over again as no

drop was found  
 print("here3")  
 Del\_List = []  
if check[0] != Links[0]+"\n":  
 print(Links[0]+"\n")  
 print(check[0])  
 print("drop detected")  
 for i in range(len(Links)):  
 if check[i] != Links[i]+"\n": #compares the current link with the old links  
 Links.append(Links[i]) #if they arent the same this is a new link add

Adds the index of each item that needs to be taken out of Links (all of the old ones)

it to the list of valid new links  
 print(Links[i] + "\n")  
 print(check[i])  
 print("Reload")  
  
 print(1)  
 else:  
 Del\_List.append(i) # if they are the same put this in a list of old links  
 print("bang")

The next section takes the integers for the indexes in Del\_List and uses them to pop the redundant links it does this by grabbing each of the integers from Del\_List and then takes away the value of the counter so It can constantly delete the correct item in the list even if the index of the list fluctuates.

counter = 0  
for i in Del\_List:  
 print(Del\_List)  
 print(Del\_List[counter])  
 print(Links)  
 Links.pop(int(Del\_List[counter])) # pops the link out of Del\_List  
 print("bang")  
 print("val",Del\_List[counter])  
 print("precount",counter)  
 counter = +Del\_List[counter]-counter # accoutnts for a shrink in the list

that the pop created

print("counter",counter)  
print(Links)  
print("laaaaaa")  
return Links

The main problem I came across was the fact it would run and then it would loop back where “here1” is and I tried it multiple times and couldn’t find out why it wouldn’t work so I decomposed the issue took out the loops I tried everything and then realised that I had opened a subroutine in a subroutine causing the subroutine to execute the one I just ran and then go back to the first subroutine and execute that this was effectively doubling the output I got from the list. I fixed this by using.

return SupremeLinkGen(sectionLink, False)

Instead of:

SupremeLinkGen(sectionLink, False)

The return meant that it would return that subroutine instead of keeping it running as another instance within the program

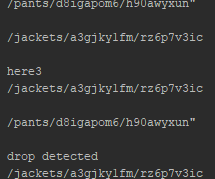
## Review:

### Testing Table:

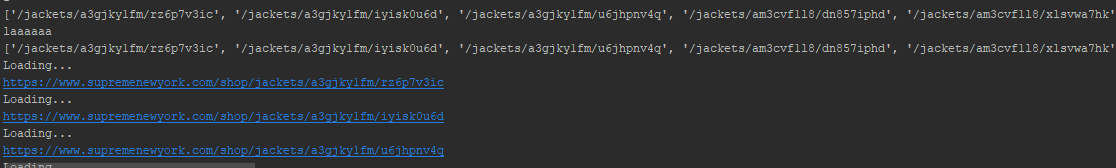
|  |  |  |  |
| --- | --- | --- | --- |
| Test Number | Test Description | Expected result | Actual result |
| 1 | Does the bot hang until it detects a drop | Yes, and it communicates this to the user until there is a drop | Yes, it communicates when a drop takes place however the program does not identify to the user when the page is reloading as a drop hasn’t taken place. |
| 2 | Does the bot create a list of links of new items only that are able to be checked for the item the user is looking for | Yes, and the bot uses this data to differentiate between whether a drop has happened or not. | The list of links is created and then the old items are taken a way giving the bot les to sift through. |
| 3 | Does this speed up the process | Yes the current speed is at 35 seconds so a decrease of 2 seconds would be great | There was a decrease in cart to check out time by 2.5 seconds. As their was less for the bot to sift through |

### Testing Evidence:

1)

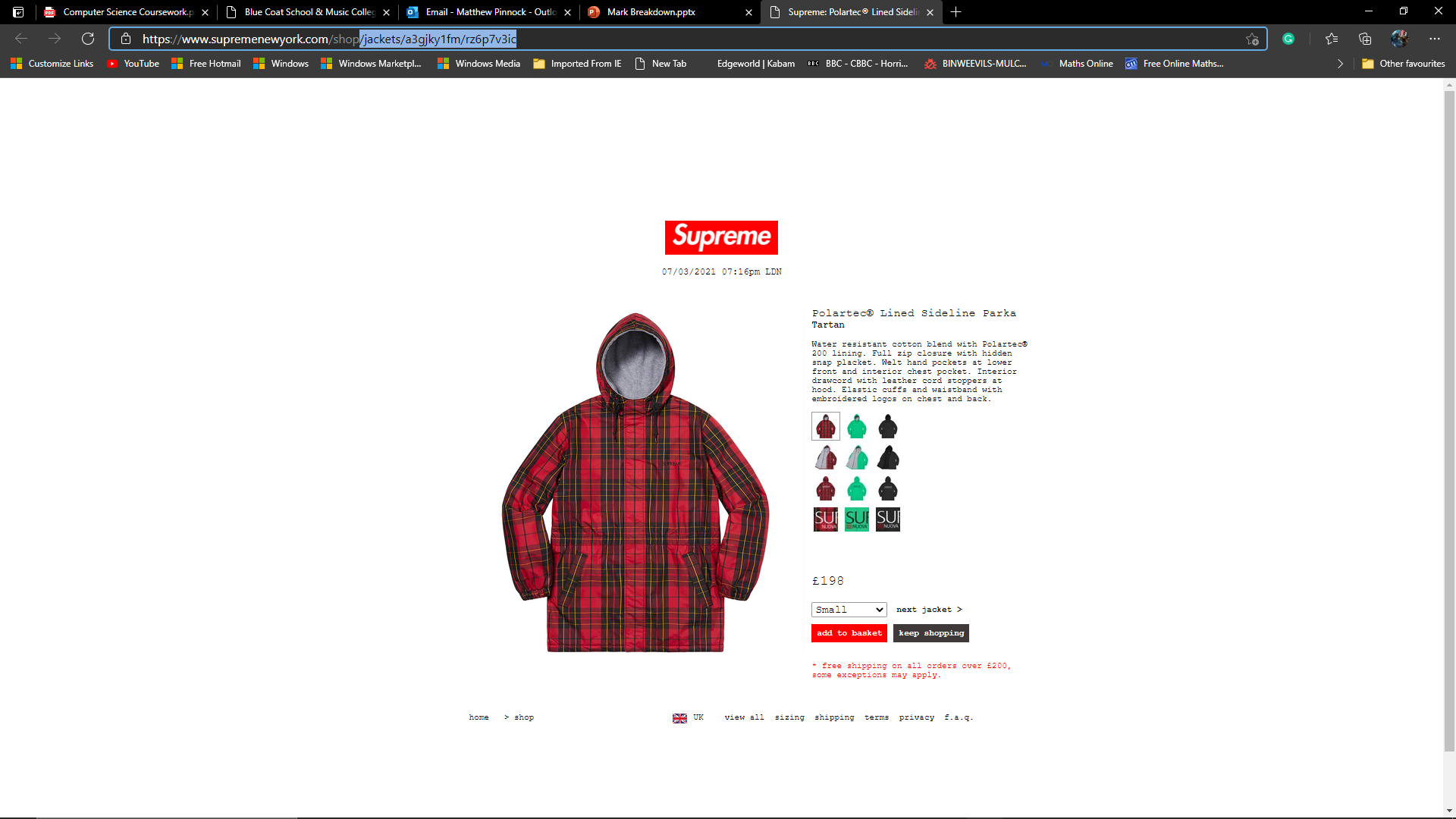


2)

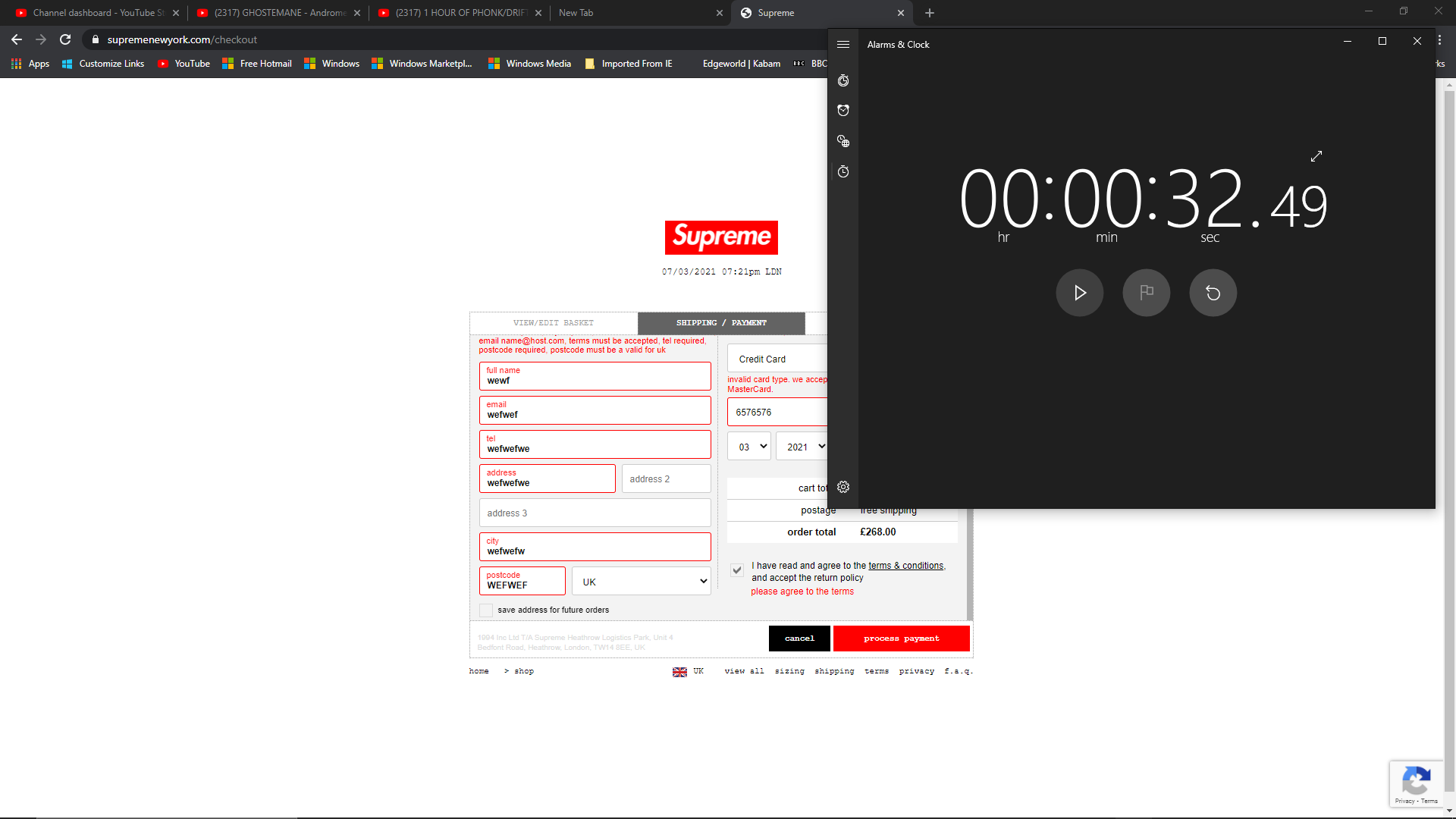


For example if we take the first link which has the ending /jackets/a3gjky1fm/rz6p7v3ic

Which is this new item if it wasn’t a new item:



3)



### Success Criteria:

|  |  |
| --- | --- |
| Criteria | Met? |
| Speed - A speed of less than 20 seconds | N – But we got closer to the targeted number of seconds |
| Reusability - Its able to be used multiple times by the same person | N |
| Speed - it runs without needing to have a browsers UI running | N |
| Ease Of Use - Easy to use for computer literate people | N |
| Versatility – This application can be used for multiple products | N |
| UI should be clean and to the point. | N |
| Make a process that figures out URL generation | Y |
| The program must be able to buy any item on the websites it supports automatically | N |
| Make an easy way for the user to input the item they are looking for | N |
| Make sure the user knows when its loading | N |

### Progress:

The speeding up of the URL generation was an essential part of the program and whilst we haven’t hit our target of a 20s run time at this stage we are closer to it and without this stage it would make it harder to hit the target later on in this program. Since Speed is one of the most important things for this project even a slight speed up of a couple of seconds is a massive win. The only way I could make these changes was by maintaining the modularity of my program by containing the code inside of a subroutine. Making sure that my program is modular in design is essential to its maintainability and my development this is because I can continuously improve the code that I have laid down while being able to constantly test it without having to run through the rest of the code

### Summary:

At this phase we don’t complete much more of the success criteria, but I have highlighted the one pertaining to speed in amber this is because the program has become faster and any increase in speed is movement towards our final goal. The modular nature of the program has proved to be successful because of the fact we have used it to upgrade the program and this has been done independently of the rest of the code in the program in a timely manner. It being modular meant that I Could independently test the time for this part of the application. There is not a User interface either but as stated before this is also fine as this is better built once the program is finished being created this is because The structure of the program has to be laid out in full before I am able to integrate a User interface into it is also just makes the testing process longer as I have to wait for it to load every time I load up the program.

def SupremeLinkGen(sectionLink,firstrun):  
  
 res = requests.get(sectionLink, headers=headers)  
 page = bs4.BeautifulSoup(res.text, "lxml")  
 article = page.findAll("div", class\_='inner-article')  
 print(909090909909009)  
 for i in article:  
 children = i.findChildren("a" , recursive=True)  
 counter = -1  
 #we find the children of inner article so we get closer to just the link the children of inner article has the link aswell as a href and an image which we dont need  
 children = str(children)  
 LinkBuild = ""  
 #we start a new list to build a string of just the link without the useless information  
 for i in range(28):  
 #28 is how long the link string is  
 counter = counter+1  
 LinkBuild = LinkBuild +children[15+counter]  
  
 if counter == 27:  
 #we need the counter to know whemn to append it in and no further  
 print("grabbed")  
 Links.append(LinkBuild)  
 print(909090909909009)  
 with open("LINKS.txt","a") as Wrl:  
 print(firstrun)  
 if firstrun == True:  
 for i in Links:  
  
 Wrl.write(i+"\n")  
 print(i)  
 print("First run")  
 print(firstrun)  
 return SupremeLinkGen(sectionLink, False)  
 #fixed first subroutine not ending  
 print("here1")  
  
 Wrl.close()  
 print('here2')  
 time.sleep(0.5)  
 with open("LINKS.txt","r") as Wrl:  
 check = Wrl.readlines()  
 print(check[0])  
 print(Links[0]+"\n")  
 if check[0] == Links[0]+"\n":  
 print("no drop detected")  
 SupremeLinkGen(sectionLink,False)  
 print("here3")  
 Del\_List = []  
 if check[0] != Links[0]+"\n":  
 print(Links[0]+"\n")  
 print(check[0])  
 print("drop detected")  
 for i in range(len(Links)):  
 if check[i] != Links[i]+"\n":  
 Links.append(Links[i])  
 print(Links[i] + "\n")  
 print(check[i])  
 print("Reload")  
  
 print(1)  
 else:  
 Del\_List.append(i)  
 print("bang")  
 Wrl.close()  
 counter = 0  
 for i in Del\_List:  
 print(Del\_List)  
 print(Del\_List[counter])  
 print(Links)  
 Links.pop(int(Del\_List[counter]))  
 print("bang")  
 print("val",Del\_List[counter])  
 print("precount",counter)  
 counter = +Del\_List[counter]-counter  
 print("counter",counter)  
 #shrunk as it popped  
 print(Links)  
 print("laaaaaa")  
 return Links

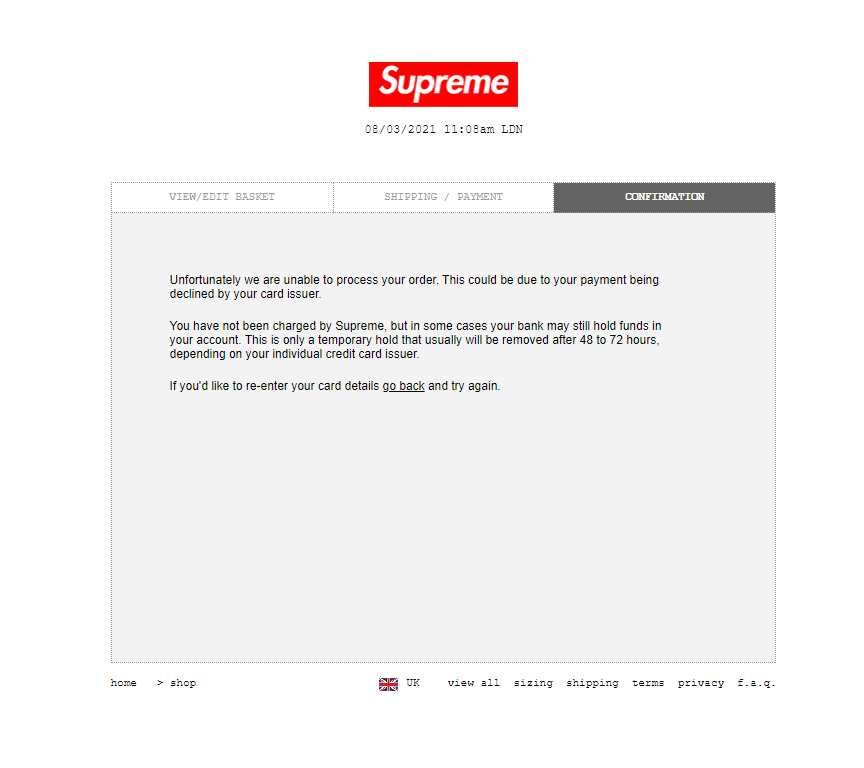
# Development Stage 3

Connection Stage:

This stage focuses on connect to the links that are supplied by the previous link generation subroutine, it scrapes the page for the name of the product and builds a list of each product in the list of links. The reason this needs a whole subroutine is because the initial list of names we get back is filled with a lot of unnecessary html that we don’t need so this code also focuses on eliminating all that redundant code.

## Lessons from prototype:

This section was another feature the prototype lacked. Taking the connection of the products page out of the hands of the user is important as if handled in the wrong way for example the connection taking to long it could cause further stages to fail. Failure here causes further failure due to the fact that you may reach checkout and fill out the form just to be declined at the end as you were to slow in this phase of the program and missed the reserved entry. This an example of the porotype failing for this reason.



Algorithm:

We then connect to each page and take the contents of the title

We then take away the redundant html so we are just left with the name and a <

We index through each link

qwdqwdqwwdddddddddddddddddddddddddddddddddddddddddddddddddddddddddddddddddddVariables:

|  |  |
| --- | --- |
| Connect | This is a variable that acts as a way to connect each one of the links for scraping. |
| Links | This is the links passed from the previous subroutine so they can be connected to. |
|  |  |
| res | This uses the connect variable to actually connect to the page |
| prodpage | Is the scraped pages from each link we have, parsed into lxml. |
| Product | This is the name of the item with all the useless html still attached |
| ProdBuild | This is each letter of the product being built |
| X | It is used to break the loop |

## Data structures:

|  |  |
| --- | --- |
| Connect | String |
| Links | Python List |
| res | Var |
| prodpage | lxml |
| Product | String |
| ProdBuild | String |
| X | Boolean |

## Test Data:

|  |  |
| --- | --- |
| Links | This is printed at the start so I know that the amount of links has been abstracted down and its not more then it needs to be. |
| “Loading” | This just tells the user that its in the process of loading its there for me to know its looping correctly. |
| “End” | This tells me whenever a product has done being built. |
| Prodlist | This is shows me all products built that way I can make sure its adding them to the list properly. |
| ProdBuild | Shows me the item title once its finished building. |

## Implementation:

This first section enumarting through all the links and connects to them individually it then parses each of these links into a readable lxml so it can be manipulated by python. It then goes and scrapes just the Title from the page as this is all we need to build our Products name. the commented out code was used to half the amount of links that was given back from the previous subroutine this is because we had the error where it ran both instances that I talked about in the implementation side for speeding up the link generation.

def SupremeConnect(section):  
 print(Links)  
 Numb = len(Links)  
 #for i in range(int((Numb/2))):  
 # Links.pop(i)  
 #print(Links)  
 for i in range(len(Links)):  
 print("Loading...")  
 #we use the built links to find the product we are looking for  
 connect = 'https://www.supremenewyork.com/shop'+str(Links[i]) holds the

This connects to the page provided by the connect variable above

This parses it in lxml for use in python

This grabs just the title as well as all of its useless html

link for the product we need to connect to  
 print(connect)  
 res = requests.get(connect, headers=headers) #connects to the link

generated  
 prodpage = bs4.BeautifulSoup(res.text, "lxml") #converts the page into

readable lxml  
 Product = prodpage.find("title") #gets the products title and assigns to a

variable

nThis next section is where we make the product name readable for this I have made a loop that doesn’t stop adding characters to a string named ProdBuild until it finds the < character the reason we start at 15 is because the 15 string is where the actual name starts and since the end of the name varies we couldn’t just put in a index for it to stop there so we used the < that came with the element. Once it finds that < it adds the new char to a python list named Prodlist.

This is responsible for enumerating through all of the links every time the loop ends

X = True  
 try:  
  
 while X == True:  
 for i in range(100000):  
  
 ProdBuild = ProdBuild + Product[15+i]  
 #printed the same as the way i build links  
 if Product[15+i] == "<":  
 Prodlist.append(ProdBuild)  
 break  
 #tried to break it with X = false but it woudlnt do so since that doesnt break the for i in range loop  
 X = False  
 #put it outside of column so the loop would properly break  
 except:  
 print("End")  
 print(Prodlist)  
 print(ProdBuild)  
 break  
  
return Prodlist

## Review:

### Testing:

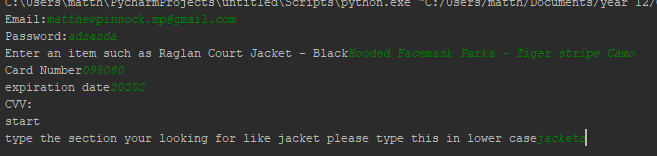
|  |  |  |  |
| --- | --- | --- | --- |
| Test Number | Test Description | Expected Result | Actual Result |
| 1 | Does the program output a readable item title for the item for use later on in the program. | The program produces a list of titles ready to be compared to the item the user input. | The program produces a list of titles for each new item for use later in the program. |
| 2 | Does the program cross reference the users input with the list of titles to identify the correct item | Prints out the link/title for the item the user is looking for | Prints out the name of the user’s product 3 times after comparing with other items. |
| 3 | Does the program attach the users input to a link | The program outputs a usable link that corresponds to the users sought after product | The program output a link that works and is the item intended for the user. |

### Success Criteria:

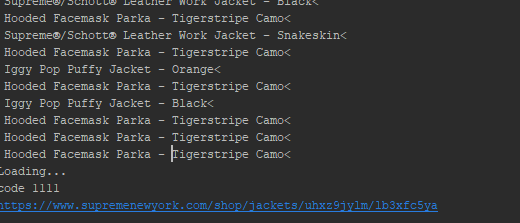
|  |  |
| --- | --- |
| Criteria | Met? |
| Speed - A speed of less than 20 seconds | N |
| Reusability - Its able to be used multiple times by the same person | N |
| Speed - it runs without needing to have a browsers UI running | N |
| Ease Of Use - Easy to use for computer literate people | N |
| Versatility – This application can be used for multiple products | N |
| UI should be clean and to the point. | N |
| Make a process that figures out URL generation | Y |
| The program must be able to buy any item on the websites it supports automatically | N |
| Make an easy way for the user to input the item they are looking for | Y |
| Make sure the user knows when its loading | Y |

### Test Evidence:

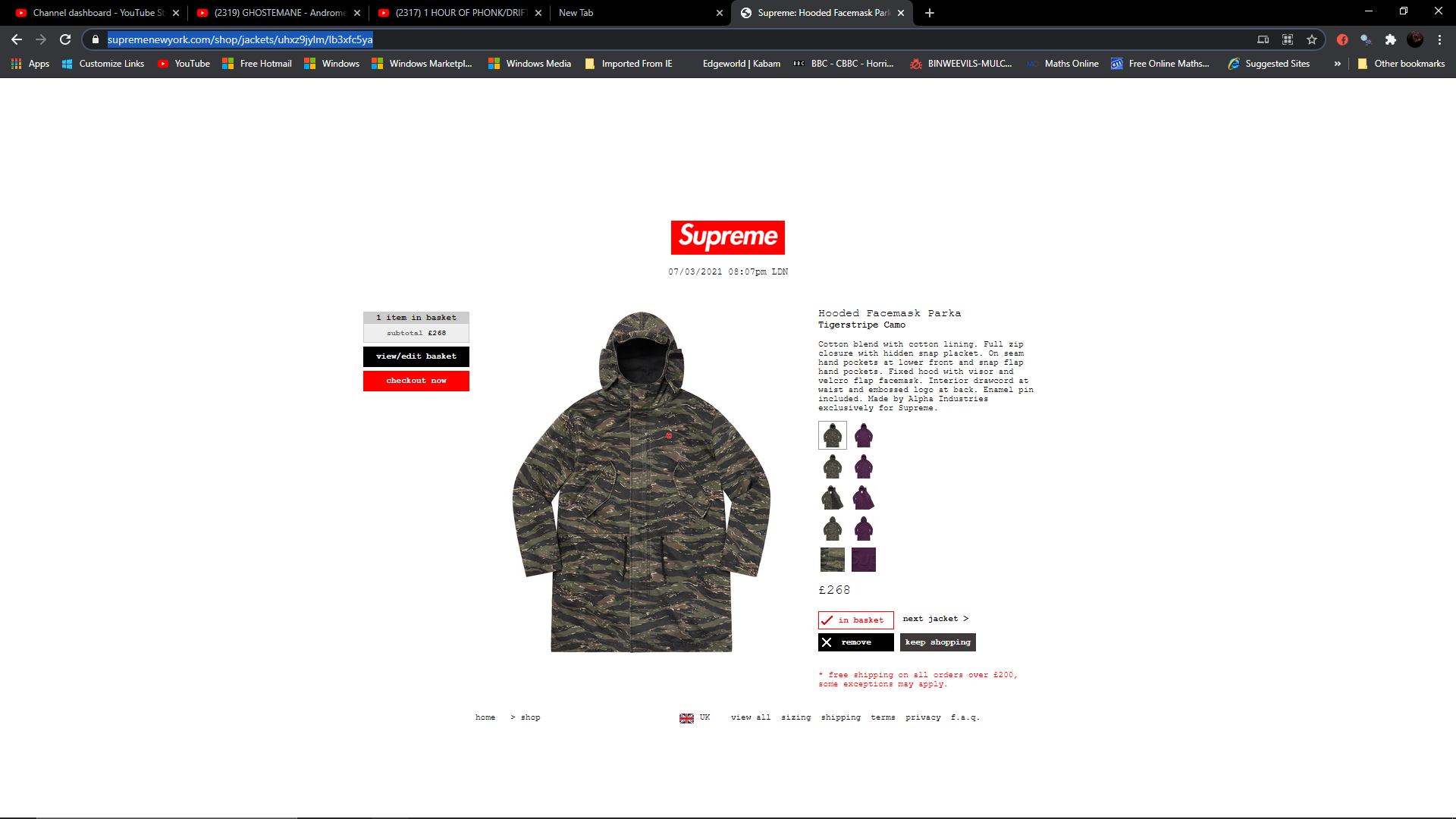
Users input:



Program output:



The bottom output is evidence for test 3 as seen here



### Summary:

We have fully developed a way for the user to input the product they are looking for and for the bot to automatically navigate to the page with that item. This successfully takes the job of doing this outside of the human’s hands unlike how it was in the prototype. This means that this process is not privy to human error as it was before. This also shows that the program works as it is able to spawn processes of chrome on its own and navigate through the webpages Infront of the users own eyes this shows that there is success in the code we are writing and allows us to see it visually. This is also the first instance of connection to a page this is good because it gives us the ability to time each connections average speed and change the code accordingly. The fact this code is modular in design and the fact this part of the application is held independently in its own subroutine means that updates to the way the connection is made only needs to be changed in the SupremeConnect() subroutine. This means it can also be tested independently and updated independently going forward which helps with the maintainability of the overall program.

### Progress:

We have completed two objectives of our success criteria **Make an easy way for the user to input the item they are looking for** & **Make sure the user knows when its loading.** The first of the two was completed as now the user can type the name of a product in and be taken to the page of the product by the bot, The second was completed as now any time the program hangs for loading reasons text saying loading is output. There is not a User interface either but as stated before this is also fine as this is better built once the program is finished being created this is because The structure of the program has to be laid out in full before I am able to integrate a User interface into it is also just makes the testing process longer as I have to wait for it to load every time I load up the program. The bot is now proven to be working in terms of browser automation.

Code:

print(Links)  
Numb = len(Links)  
#for i in range(int((Numb/2))):  
 # Links.pop(i)  
#print(Links)  
for i in range(len(Links)):  
 print("Loading...")  
 #we use the built links to find the product we are looking for  
 connect = 'https://www.supremenewyork.com/shop'+str(Links[i])   
 print(connect)  
 res = requests.get(connect, headers=headers)  
 prodpage = bs4.BeautifulSoup(res.text, "lxml")  
 Product = prodpage.find("title")  
  
 # do a while loop that breaks if < is found  
 Product = str(Product)  
 ProdBuild = ""  
  
 X = True  
 try:  
  
 while X == True:  
 for i in range(100000):  
  
 ProdBuild = ProdBuild + Product[15+i]  
 #printed the same as the way i build links  
 if Product[15+i] == "<":  
 Prodlist.append(ProdBuild)  
 break  
 #tried to break it with X = false but it woudlnt do so since that doesnt break the for i in range loop  
 X = False  
 #put it outside of column so the loop would properly break  
 except:  
 print("End")  
 print(Prodlist)  
 print(ProdBuild)  
 break  
  
return Prodlist

# Development Stage 4

ProdInspect (Adding product to cart):

This part focuses on first logging into your Gmail account to get some free captcha bypasses and then adding the need item to the cart by obtaining the exact link for the item connecting to it and adding the item to the cart and then checking out our item. The log in of the email is done so the browser comes across as if a “normal” user is using it. This stage is also where the multi-processing is introduced because it is part of the checkout phase and it30 is key since we have found the exact product were going for we need the same actions to be played out on multiple browsers for a higher chance of success and more items. This currently spawns 4 processes, but the User can put the number of processes up or down depending on their preference.

## Lessons from prototype

The prototype did not have to connect to Gmail this is because it acted as a keyboard and not as an automated browser while this meant commands were slower it also meant it was successful in not constantly being hit by Recaptchas. However, since it acted as only a keyboard it did not have the functionality to click which was needed for the drop downs and accepting the terms and conditions part of the checkout. Leaving this for a human to do could cause error enough for you to be forced to start the checkout process again and lose the item you were trying to buy in the processes so this is something I needed to fix for my program by automating it.

keyboard.type(firstname)  
time.sleep(0.7)  
keyboard.type(secondname)  
time.sleep(0.7)  
keyboard.type(telephone)  
time.sleep(0.7)  
keyboard.type(postcode)  
time.sleep(0.7)  
keyboard.type(address)  
time.sleep(0.7)  
keyboard.type(city)  
time.sleep(0.7)  
keyboard.type(postcode)

as you can see the code uses Keyboard type to simulate keyboard presses however an option such as mouse click is not available. Also as talked about before the switching between fields was not automatic since there was no click function as such this was left to the user and if the user clicked away to fast or slow it could lose them the item as they would have to fix the error they made this happened multiple times and caused errors like the post code being written in the telephone number box as the users reaction speed wasn’t fast enough. As seen below:



To remedy this, we automated the process of switching between fields as well as a human error could no longer be responsible for not getting the item.

## Algorithm:

All processes are spawned Connect to Gmail and login in a separate tab to the page of the product

Enumerate through the product list to find the same item we have chosen

Once we have found the index for our item in the list

We then use the same index for where the item name is in the list for where the link is within the link list since they are ordered the same

We then connect to that correct page and add the item to the cart

We then go to checkout and the purchase is finished

## Key Variables:

Prodlist1, item1, Links1, CardNumber, cvv, exp, email

|  |  |
| --- | --- |
| Prodlist1 | The list of product names that were obtained from the previous subroutine |
| Item1 | This is the item we have chosen to buy |
| Links1 | This is the list of all new links |
| CardNumber | This is the users card dumber that is needed to purchase the item |
| Cvv | This is the 3 integer number on the back of the card need to complete the purchase |
| Exp | This is the expiry date of the users card |
| Email | This is the email the user needs to use when checking out. |

## Data Structures:

|  |  |
| --- | --- |
| Prodlist1 | Python List |
| Item1 | String |
| Links1 | Python List |
| CardNumber | Var String |
| Cvv | Var String |
| Exp | Var String |
| Email | Var String |

## Test Data:

|  |  |
| --- | --- |
| browser.current\_url | Whenever this is printed its done so I know where the program is even in headless mode |
| Prodlist1[i] | This is printed so I can compare this to the item we chose so I can see if the problem is with the formatting of the users input that causes an error. |
| Code1111 | This is a success code so I know the “finish line has been crossed” |
| Check | This is to show whether we got hit by captcha or not |

## Implementation:

This section after making a new tab and checking the URL is correct it inputs the email.

print(browser.current\_url)  
print(2)  
#seemed like it was fine so we added the 2 for error checking  
print(Prodlist1)  
  
body = browser.find\_element\_by\_tag\_name("body") # assigns the body of the site to a variable  
body.send\_keys(Keys.CONTROL + 't') #uses the variable holding the body and makes a new tab  
connect = "https://chrome.google.com/sync" # holds the link to the login page of chrome  
browser.get(connect) #connects to Gmail login  
browser.find\_element\_by\_id("identifierId").send\_keys(email) #types the users email

This is where we connect to the Gmail account

The body was identified because we needed an element to press the shortcut to open a new tab without affecting the other page

Sends the users email to the field with the id given.

This part first clicks the next button to get to the password then inputs the password. The reason the While and try loop is needed is because it takes a while to load up the page that requests your password so we basically make it only break the try loop when the password is successfully inputted since otherwise we would constantly get an error saying there is no password element since the

browser.find\_element\_by\_xpath("/html/body/div[1]/div[1]/div[2]/div/div[2]/div/div/div[2]/div/div[2]/div/div[1]/div/div").click() #clicks the xpath for the login element  
X = True  
while X == True:  
 try:  
 browser.find\_element\_by\_name("password").send\_keys(password)  
 X = False  
 except:  
 print("loading")  
browser.find\_element\_by\_xpath("/html/body/div[1]/div[1]/div[2]/div/div[2]/div/div/div[2]/div/div[2]/div/div[1]/div").click()

Notifies the user that loading is taking place and they need to be patient this will print until that password enters

We have to use the actual X path because the field that has an id isn’t clickable and just deals with the actual text/html and not the is that allows you to click so we have to use the xpath since it’s the most precise and easiest thing to use in this situation

This part focuses on adding the item to the cart. To do this it enumerates through the whole list by indexing it until it finds the product name that’s = to our item. It then finds the place in the list that the name is and searches for a link in the same place in the link list. It then connects to that link and clicks the add to cart button. The site needs a second for the add the checkout button to be clickable I didn’t realise this until I constantly tested it I figured out the browser wasn’t waiting when I watched outside of headless mode and as such fixed it like this and since its after we add the item to the cart so we use the browser wait until its clickable I didn’t use the same while loop method as before since selenium has a built in way to deal with waiting for things to be clickable I used that.

for i in range(len(Prodlist1)):  
 print(Prodlist1[i])  
 print(item1+"<")  
 #the problem was that i copied and pasted the item lacked the <

Allows me to compare for input errors

Once the correct item is found the same index is used to find the correct link in the link list

Adds the item to cart

character all the ones in the list had  
 #Then it required a fingerspace because thats what everysingle one in

the list had had  
  
 if Prodlist1[i] == item1+"<":  
 print("Loading...")  
 print("code 1111")  
 connect = 'https://www.supremenewyork.com/shop' + str(Links1[i])  
 print(connect)  
 res = requests.get(connect, headers=headers)  
 prodpage = bs4.BeautifulSoup(res.text, "lxml") #turns page to lxml   
 browser.get(connect) #connects the products page

browser.find\_element\_by\_name("commit").click()

button = WebDriverWait(browser,10).until(EC.element\_to\_be\_clickable

((By.XPATH,'/html/body/div[2]/div/div[1]/div/a[2]')))button.click()  
 #this was one because it takes a second to become clickable  
 print(1)  
 print(browser.current\_url)  
 print(1)

Supposed to be on the same line

This next part marks the end of the program it consists of simply entering the users details and buying the item and then checking if we were successful at finishing checkout. sThe only problem I ran into was how to input a to a dropdown menu but I looked around and found out that you can use a method that selects visible text.

browser.find\_element\_by\_id("cnb").send\_keys(CardNumber)  
dropdown = Select(browser.find\_element\_by\_name("credit\_card[month]"))  
dropdown.select\_by\_value(exp[0] + exp[1])  
dropdown = Select(browser.find\_element\_by\_name("credit\_card[year]"))  
dropdown.select\_by\_value("20" + exp[3] + exp[4])  
browser.find\_element\_by\_name("credit\_card[ovv]").send\_keys(cvv)  
  
browser.find\_element\_by\_xpath("/html/body/div[2]/div[1]/form/div[2]/div[2]/fieldset/p/label").click()  
browser.find\_element\_by\_name("commit").click()  
# wasn’t sure if it worked so I used this  
Success = browser.find\_element\_by\_tag\_name("p")  
check = Success.text  
print(check)

Clicks checkout

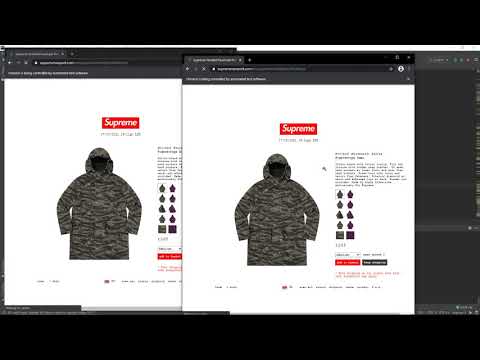
In this phase I also decided to turn on the options headless , GPU disabled , and disabled extensions to allow the code to run faster.

## Review:

### Testing Table:

|  |  |  |  |
| --- | --- | --- | --- |
| Test Number | Test Description | Expected Result | Actual Result |
| 1 | Is each field of the checkout form entered with the intended data | Yes, the intended data goes in the correct place | Yes the intended data goes in the correct place except for expiry date |
| 2 | Are you able to connect to your email account | Yes you connect to email services and the program continues | Yes I was able to login with a valid email address however the bot was clicking the login button before it could load |
| 3 | Was an item able to be successfully added to the cart | Yes | Yes |
| 4 | Was the checkout form filled and checkout successful. | Yes and an item was bought. | Checkout was unsuccessful as we were hit by a Recaptcha |
| 5 | Checkout happens in less than 20 seconds with the headless mode disabled GPU | Checkout takes 20 seconds or less | Checkout happens in less than 20 seconds with the headless mode disabled GPU |
| 6 | Can you spawn multiple processes | Multiple chrome windows are spawned | Multiple chrome windows are spawned with process mirrored on them |

### Test proof:

**Test 1,2,3,4,6)** [https://youtu.be/JDCWX\_y1f5Q**[](https://www.youtube.com/embed/JDCWX_y1f5Q?feature=oembed)**](https://youtu.be/JDCWX_y1f5Q#)

The reason we did not go through with checkout is because I would not like to be charged on my card while testing. The reason this test was successful even though all the fields weren’t filled in is because the normal reseller already has their shipping address and information saved however the Card number which was the intended data cannot be saved there and that’s where our program comes in.

**Test 5)** <https://youtu.be/3otqMPHWYTY>

[](https://www.youtube.com/embed/3otqMPHWYTY?feature=oembed)

When “code 1111” is printed that signals that the checkout process has finished and to confirm that the program also outputs the paragraph listing the terms and conditions. The timer started at 10 seconds into the video and finished just before the 30 second mark at that meets our goal.

### Remedial Action:

Initially entering the expiry date didn’t work properly as this is a drop-down menu rather than just text and as I didn’t know how to use selenium to pick options from a drop down menu I did some research to find out how



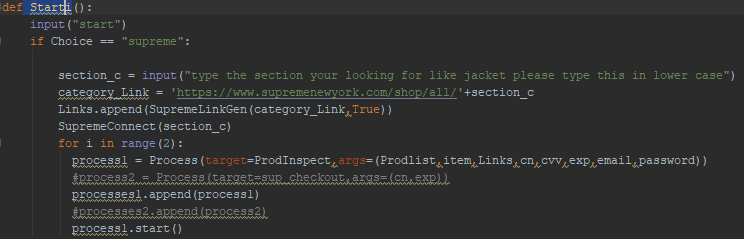
I ended up reading <https://intellipaat.com/community/4266/how-to-select-a-drop-down-menu-option-value-with-selenium-python> and found out that the drop down menus have a value that can be selected so I took the CVV numbers for example 11/23 and then selected the values that correspond to those.

### Success Criteria:

|  |  |
| --- | --- |
| Criteria | Met? |
| Speed - A speed of less than 20 seconds | Y |
| Reusability - Its able to be used multiple times by the same person | Y |
| Speed - it runs without needing to have a browsers UI running | Y |
| Ease Of Use - Easy to use for computer literate people | Y |
| Versatility – This application can be used for multiple products | Y |
| UI should be clean and to the point. | N |
| Make a process that figures out URL generation | Y |
| The program must be able to buy any item on the websites it supports automatically | Y |
| Make an easy way for the user to input the item they are looking for | Y |
| Make sure the user knows when its loading | Y |

### Progress:

There was a lot of progress made during this phase, this is because this was the last phase of actions that the bot must do, and it holds the main functionality of the program. We were able to direct the bot to checkout and have it checkout this is the first thing we did. Since our program is modular and allows for improvements to be made easily, We then improved upon this by adding the functionality to login with your Gmail so it would hide the fact that the bot was a bot. This feature was added without having to touch anything outside of the subroutine responsible for checking out. The programs modularity is also show as adding multi-processing was as simple as adding it to our Starti() subroutine which is the subroutine that all the others are run from:



This is where the multiprocessing was added

As well as this phase successfully developing the main functionality of our bot it also is where the most drastic difference in speed comes as we turn on the options seen below:

chrome\_options = Options()  
chrome\_options.add\_argument("--disable-extensions")  
chrome\_options.add\_argument("--disable-gpu")  
chrome\_options.add\_argument("--headless")  
#each flag makes my program faster becausew there is less load for the cpu to deal with

### Summary:

In this phase we completed most of the objectives in the success criteria. We achieved**; A speed of less than 20 seconds –** We achieved this by changing the options our chrome driver ran with by disabling extensions and disabling the gpu, **Its able to be used multiple times by the same person-** A user is actually able to do this in parallel now thanks to the multi-processing that was added, **it runs without needing to have a browsers UI running –** This is thanks to the headless option that we enabled, **Easy to use for computer literate people –** This is true because we added a bunch of error code so the user knows what’s going on at each code, **This application can be used for multiple products-** A user is actually able to do this in parallel now thanks to the multi-processing that was added so multiple products can be bought at the same time. The only part of the success criteria that has not been met is the one pertaining to a clean Ui as the User interface has still not been created. However, as we know the structure of the back end of our program in the next stage we can go onto making the front end Ui for our program. The modularity has been maintained throughout the program especially here as the checkout is its own whole subroutine any change the website makes in regard to the route to checkout or the checkout form can be changed here accordingly this is good for the maintainability of the program**.**

Code:

This part is part of the subroutine that calls all of the subroutines in the program the 2 in the loop would normally be replaced a variable holding the number of tasks the user wants to run with this. This is where the multiple processes are spawned, and I have decided to put 2 just so it isn’t that demanding but I know it works, Make sure the user knows when its loading

for i in range(2):  
 process1 = Process(target=ProdInspect,args=(Prodlist,item,Links,cn,cvv,exp,email,password))  
 #process2 = Process(target=sup\_checkout,args=(cn,exp))  
 processes1.append(process1)  
 #processes2.append(process2)  
 process1.start()

def ProdInspect(Prodlist1,item1,Links1,CardNumber,cvv,exp,email,password):  
 print(browser.current\_url)  
 print(2)  
 #seemed like it was fine so we added the 2 for error checking  
 print(Prodlist1)  
  
 body = browser.find\_element\_by\_tag\_name("body")  
 body.send\_keys(Keys.CONTROL + 't')  
 connect = "https://chrome.google.com/sync"  
 browser.get(connect)  
 browser.find\_element\_by\_id("identifierId").send\_keys(email)  
 browser.find\_element\_by\_xpath("/html/body/div[1]/div[1]/div[2]/div/div[2]/div/div/div[2]/div/div[2]/div/div[1]/div/div").click()  
 X = True  
 while X == True:  
 try:  
 browser.find\_element\_by\_name("password").send\_keys(password)  
 X = False  
 except:  
 print("loading")  
 browser.find\_element\_by\_xpath("/html/body/div[1]/div[1]/div[2]/div/div[2]/div/div/div[2]/div/div[2]/div/div[1]/div").click()  
 #browser.find\_element\_by\_xpath("/html/body/div[1]/div[1]/div[2]/div/div[2]/div/div/div[2]/div/div[2]/div/div[1]/div").click()  
 #use enter if it doesnt work  
  
 #  
 for i in range(len(Prodlist1)):  
 print(Prodlist1[i])  
 print(item1+"<")  
 #the problem was that i copied and pasted the item lacked the < character all the ones in the list had  
 #Then it required a fingerspace because thats what everysingle one in the list had had  
  
 if Prodlist1[i] == item1+"<":  
 print("Loading...")  
 print("code 1111")  
 connect = 'https://www.supremenewyork.com/shop' + str(Links1[i])  
 print(connect)  
 res = requests.get(connect, headers=headers)  
 prodpage = bs4.BeautifulSoup(res.text, "lxml")  
 browser.get(connect)  
 browser.find\_element\_by\_name("commit").click()  
 button = WebDriverWait(browser, 10).until(EC.element\_to\_be\_clickable((By.XPATH, '/html/body/div[2]/div/div[1]/div/a[2]')))  
 button.click()  
 #this was one because it takes a second to become clickable  
 print(1)  
 print(browser.current\_url)  
 print(1)  
 browser.find\_element\_by\_id("cnb").send\_keys(CardNumber)  
 dropdown = Select(browser.find\_element\_by\_name("credit\_card[month]"))  
 dropdown.select\_by\_value(exp[0] + exp[1])  
 dropdown = Select(browser.find\_element\_by\_name("credit\_card[year]"))  
 dropdown.select\_by\_value("20" + exp[3] + exp[4])  
 browser.find\_element\_by\_name("credit\_card[ovv]").send\_keys(cvv)  
  
 browser.find\_element\_by\_xpath("/html/body/div[2]/div[1]/form/div[2]/div[2]/fieldset/p/label").click()  
 browser.find\_element\_by\_name("commit").click()  
 # wasnt sure if it worked so i used this  
 Success = browser.find\_element\_by\_tag\_name("p")  
 check = Success.text  
 print(check)

# Development Stage 5:

## Users interface creation:

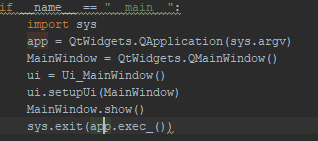
In this stage we will focus on creating a workable clear and concise user interface so the user can easily use our program. This phase is focussed on completing the part of the success criteria that outlines that the **UI should be clean and to the point**. We will also keep in mind the usability features talked about in the design stage for this part of the process. The UI will be created using reference to the model of the mock UI made up in Design.

## Key Variables:

|  |  |  |
| --- | --- | --- |
| Variable Name | Data Type | Variable Description |
| app | Object | Holds the widgets for use wen the application is spawned |
| ui | Object | Holds the Main window that the rest of the Ui is spawned onto |
| Self.label | Object | give the object label a name so we can interact with it easily |
| Self.label\_2 | Object | give the object label 2 l a name so we can interact with it easily |
| Self.label\_3 | Object | give the object label 3 a name so we can interact with it easily |
| Self.label\_4 | Object | give the object label a name so we can interact with it easily |
| Self.label\_5 | Object | give the object label 5 a name so we can interact with it easily |
| Self.label\_6 | Object | give the object label 6 a name so we can interact with it easily |
| Self.label\_7 | Object | give the object label 7 a name so we can interact with it easily |
| Self.pushButton | Object | spawns a button for the user to interact with |
| Self.pushButton\_2 | Object | spawns button 2 for the user to interact with |
| Self.pushButton\_3 | Object | spawns button 3 for the user to interact with |
| Self.pushButton\_4 | Object | spawns button 4 for the user to interact with |
| Self.pushButton\_5 | Object | spawns button 5 for the user to interact with |
| self.textEdit | Object | allows us to add text field so the user can easily interact with our program |
| self.textEdit\_2 | Object | allows us to add text field 2 so the user can easily interact with our program |
| self.textEdit\_3 | Object | allows us to add text field 3 so the user can easily interact with our program |
| self.textEdit\_4 | Object | allows us to add text field 4 so the user can easily interact with our program |
| self.textEdit\_5 | Object | allows us to add text field 5 so the user can easily interact with our program |
| self.plainTextEdit\_2 | Object | allows us to add text field 2.2 so the user can easily interact with our program and make sure its plain |
| self.plainTextEdit | Object | allows us to add text field 2.1 so the user can easily interact with our program and make sure it’s plain |
| Self.menubar | Object | spawns the menu for the user to interact with. |
| Self.statusbar | Object | spawns the status for the user so they understand processes are taking place. |

## Implementation:

The program is run from for a name== “main” loop this is to main usability and intergratibility into the code so that way it is easy to switch processes to the back end. This is also so the program maintains reusability as if it was not in a loop it would close very quickly after it is opened so to keep it reusable which is in line with our success criteria, we must have this loop



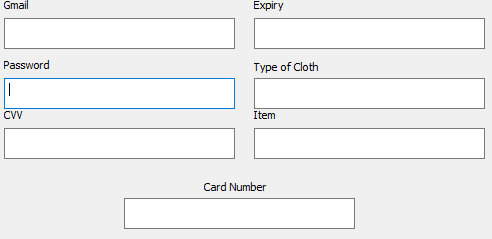
The program is structured as a class that stores multiple subroutines essential to the use of the program. The reason it is in a class is because it is easily maintainable as it is modular. Each method for the class can be tested individually and updating the UI in the long run with new languages and usability features can be easily done and encapsulated since it is modular





As you can see we have 2 different subroutines the setupUI subroutine spawns each object onto our UI.

The top half of the Ui seen here:



This is the code responsible for generating it.

self.textEdit = QtWidgets.QTextEdit(self.centralwidget) #allows us to add text feild  
self.textEdit.setGeometry(QtCore.QRect(130, 200, 231, 31)) #we set the sizing and postition  
self.textEdit.setObjectName("textEdit")#give the object a name so we can interact with it easily

self.textEdit\_2 = QtWidgets.QTextEdit(self.centralwidget) #allows us to add text feild  
self.textEdit\_2.setGeometry(QtCore.QRect(10, 130, 231, 31))#we set the sizing and postition  
self.textEdit\_2.setObjectName("textEdit\_2")#give the object a name so we can interact with it easily

self.textEdit\_3 = QtWidgets.QTextEdit(self.centralwidget) #allows us to add text feild  
self.textEdit\_3.setGeometry(QtCore.QRect(260, 80, 231, 31))#we set the sizing and postition  
self.textEdit\_3.setObjectName("textEdit\_3")#give the object a name so we can interact with it easily

self.plainTextEdit\_2 = QtWidgets.QPlainTextEdit(self.centralwidget) #allows us to add text feild  
self.plainTextEdit\_2.setGeometry(QtCore.QRect(260, 130, 231, 31))#we set the sizing and postition  
self.plainTextEdit\_2.setObjectName("plainTextEdit\_2")#give the object a name so we can interact with it easily  
self.textEdit\_4 = QtWidgets.QTextEdit(self.centralwidget) #allows us to add text feild  
self.textEdit\_4.setGeometry(QtCore.QRect(10, 20, 231, 31))#we set the sizing and postition  
self.textEdit\_4.setObjectName("textEdit\_4")#give the object a name so we can interact with it easily  
self.textEdit\_5 = QtWidgets.QTextEdit(self.centralwidget) #allows us to add text feild  
self.textEdit\_5.setGeometry(QtCore.QRect(260, 20, 231, 31))#we set the sizing and postition  
self.textEdit\_5.setObjectName("textEdit\_5")#give the object a name so we can interact with it easily

As shown in the comments each part input box is spawned by

QtWidgets.QPlainTextEdit(self.centralwidget)

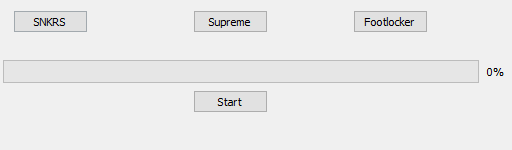
And then the sizes are set

self.textEdit.setGeometry(QtCore.QRect(10, 80, 231, 31))

and then finally the generated box is attached to the program and given a name so it is differentiated to the other boxes

self.textEdit.setObjectName("textEdit")

This is the bottom part of my program



This entails the buttons and loading bar the code for this is below

self.progressBar.setGeometry(QtCore.QRect(10, 300, 511, 23))#we set the sizing and postition  
self.progressBar.setProperty("value", 0) #gives the progress bar a starting value that will go up as progress is made  
self.progressBar.setObjectName("progressBar")#give the object a name so we can interact with it easily

self.pushButton = QtWidgets.QPushButton(self.centralwidget) # spawns a button  
self.pushButton.setGeometry(QtCore.QRect(200, 330, 75, 23)) #we set the sizing and postition  
self.pushButton.setObjectName("pushButton") #give the button a name so we may interact with it

As shown in the comments each part button is spawned

self.pushButton = QtWidgets.QPushButton(self.centralwidget)

The size of the button is set

self.pushButton.setGeometry(QtCore.QRect(200, 330, 75, 23))

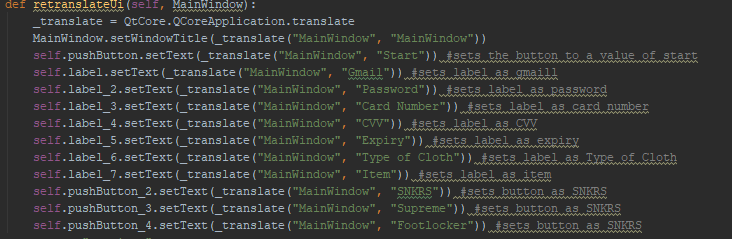
and then finally the generated button is attached to the program and given a name so it is differentiated to the objects and usable by the program

self.pushButton.setObjectName("pushButton")

The process is the same for the progress bar however a value is also given to it to represent how much it has loaded

self.progressBar.setProperty("value", 0) #gives the progress bar a starting value that will go up as progress is made

The second subroutine Retranslates the UI so that if an individual speaks a different language we can easily change it there and give them that version of the application or we could choose to make the retranslation trigger with a button press to set it to another language. This is responsible for giving each, text label and piece of text you see, actual words/titles and can be changed mid program if we would like to. This modularity helps maintain the program with updates and changes iin the long term



As you can see by the code

self.label.setText(\_translate("MainWindow", "Gmail")) #sets label as gmaill

it takes the labels created in the first subroutine and changes their text. The label is spawned here

self.label = QtWidgets.QLabel(self.centralwidget ) #creates a label for the fields

and changed here

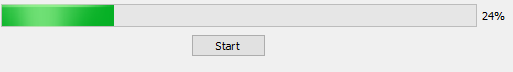
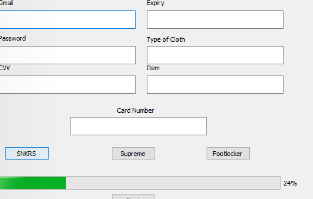


## Review:

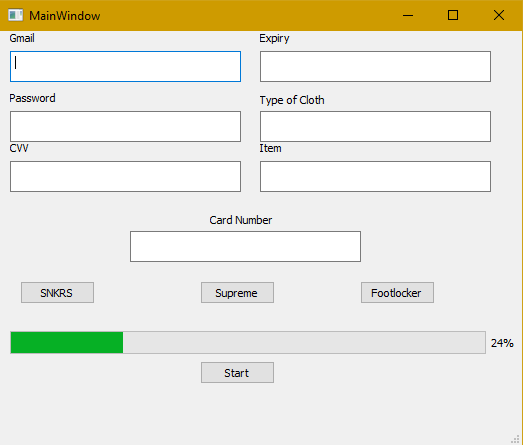
### Testing Table:

|  |  |  |  |
| --- | --- | --- | --- |
| Testing Number | Testing Description | Expected Result | Actual Result |
| 1 | Does the loading bar clearly communicate the progress through the program | Yes, it does the percentage rises as the bar does | The percentage and bar go up as the program gets closer to being complete |
| 2 | Do all the buttons have functionality | Yes the buttons are clickable and usable | Yes the buttons light up when being clickable |
| 3 | Can the titles for each box be easily changed | Yes they can be changed during the program or before | Yes changing the retranslate code changes the title selected |

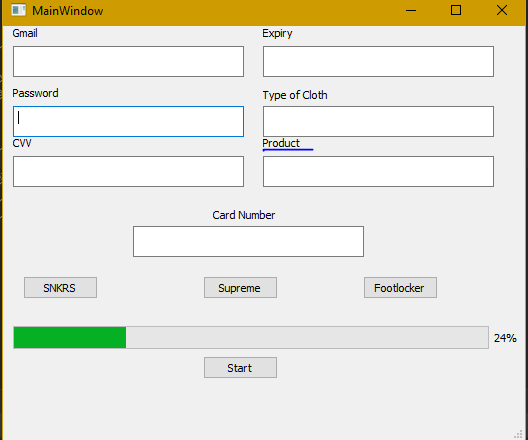
Testing Proof:

1. 
2. 

The item becomes clickable and lights up as blue to communicate that

1. 

Before the title change of the “item” label





“item” label changed to “product”

### Success Criteria:

|  |  |
| --- | --- |
| Criteria | Met? |
| Speed - A speed of less than 20 seconds | Y |
| Reusability - Its able to be used multiple times by the same person | Y |
| Speed - it runs without needing to have a browsers UI running | Y |
| Ease Of Use - Easy to use for computer literate people | Y |
| Versatility – This application can be used for multiple products | Y |
| UI should be clean and to the point. | Y |
| Make a process that figures out URL generation | Y |
| The program must be able to buy any item on the websites it supports automatically | Y |
| Make an easy way for the user to input the item they are looking for | Y |
| Make sure the user knows when its loading | Y |

### Progress:

This is the last stage of development as it adds the User Interface to be able to be integrated with our back end. The code for the User interface and the code for the application back end are two separate files this is to maintain compatibility across devices as well as for maintainability that way I can maintain and add to both the Ui and code completely independently that way if I want to make changes to the extensions and modules for the user interface that file is the only on affected. It also means that those wanting a non-user interface more lightweight version of the program are able to have it. As well as this it means testers have an easier time changing and adding to the back end of it without having to worry about changing the font end.

### Summary:

We have met all success criteria and in doing so have completed the development process. Each part of the development is independent of the others and modularised which allows for easy upgrade and maintainability.

# Final testing:

|  |  |  |  |
| --- | --- | --- | --- |
| **Test Description** | **Test Data** | **Expected result** | **Actual result** |
| **Destructive Testing**: I will try and input an invalid email and password to the program. | Enter the string “big boy” into the Gmail field with the password “ smol” | The program should hang as there is a wrong email/password and the reason for this stoppage should be communicated to the user | The program did hang and printed loading constantly to show to the user that something was happening the browser the program was running on then popped up and communicated to the user that the email is incorrect. However, when running in the no flag there was no indication to the user that the program was hanging because of an incorrect email |
| **Destructive Testing**: I will attempt to make enter a wrong subcategory of clothing | Enter the string “ Johnny boy”  When the program asks for a clothing subcategory | The program hangs and communicates to the user that the category doesn’t exist | The program stopped completely and threw an error. The user was not informed of the reason for the error |
| **Destructive Testing**: I will attempt to enter incorrect card information | Enter the integer 990090 in the card number field | The program hangs and communicates that the card number input is incorrect | The program hanged on Supremes website and the window open communicate it the user that this was because the card they entered was not correct |
| **Destructive Testing**: Entered the product name in lower case without the necessary Dashes | Enter the product “Hooded Facemask Parka - Tiger stripe Camo” | The program hangs and communicates to the user that the item they entered was incorrect | The program stopped completely and threw an error. The user was not informed of the reason for the error |

## Post Development Interviews:

1. **Was it easy to use the bot?**

**Gibraltar:**

Yes it was quite easy to use it once it had been set up as the chromedriver as well as everything else needed was there already it just felt weird how some options were typed into the Interface while others I had to type into the programs window directly.

**T’challa:**

Yes it was easy to use as every step had some sort of description telling me how to go about executing it. It was not very complicated at all for me to use as other bots I have used were a lot more complicated.

**Bot Creator:**

It wasn’t the most complex of bots so yes, some features you normally see in bots was missing such as proxies, but the core features worked well enough so yes it was easy to use.

1. **Did you run into any errors when trying to run the bot if so, was it easy to figure out how to remedy them?**

**Gibraltar:**

I ran into a couple errors as it was my first time and I wasn’t doing everything correctly, but they were easily solvable as most told me what I was doing wrong. I struggled with one error as I didn’t realise you needed to add a dash between the colour and item name and nothing really indicated me to except for the syntax for the example item that it shows you in the program.

**T’challa:**

I didn’t run into many errors as I knew what I was doing it was easy once I figured out the syntax.

**Bot Creator:**

There wasn’t much to mess up so no I didn’t only thing I had to figure out was the ample amount of this process to have running parallel.

1. **What feature would you like the bot to have?**

**Gibraltar:**

I would like there to be videos or instructions showing you how to utilise the bot the best and things like setting mow many processes you want as I did not quite understand that part of the program. Also an easy way to tell if your purchase was successful or not.

**T’challa**:

I would like there to be a way to enable us to use proxies. As well as having it compatible with more sites than it currently is.

**Bot Creator:**

I would like there to be both in app proxies as well as a solution to automatically getting past

ReCaptchas as this takes a lot of speed from the bot and the addition of an in-app proxy.

1. **Would you use the bot again?**

**Gibraltar:**

Yes I would it was easy and I was able to use it to buy an item very easily.

**T’challa:**

Yes because for the price that you are suggesting I think it is quite effective 20 seconds is enough time to get any item as long as it doesn’t have a tonne of hype surrounding it.

**Bot Creator:**

No because it doesn’t have all the advanced features that other more highly priced bots have and so far it cant get the extremely sought after items as this has a drop to checkout time of 20 seconds and the most highly demanded items sell out in around 15 seconds.

**Review:**

All the stakeholders found it quite easy to use the bot even Gibraltar who hasn’t used it before this is good because ease of use was one of our main goals. The Bot creator and T’challa have used previous solutions extensively and they both requested in app proxies even though proxies are easily configurable. This may be because the bot community normally uses bots on the network sent from the owners IP however our bot acts as the owner so simply turning on a VPN would do the trick when it comes to hiding one’s identity. For that reason, we will not add proxies in app as this would also slow the bots functioning and we don’t believe that this is a good trade off.

The comments about the bots speed are valid and are something that we ware going to be looking at improving in the future doing this may mean changing the code in its entirety to see where we can cut corners and because of this it is out of the scope of this project. It would require us to time each connection made and it’s a importance and then condense and rewrite the code to make each connection shorter. However for the price the bot is at 20 seconds is an ample amount of time for the part of the bot market we are going for which is the mid-tier of bots this is because it allows us to put our tool in as many peoples hands as possible and it gives them a good way to get product that the average reseller and consumer would get. Due to that we will leave the scarcest products to the bots at the higher end of the market and as the only person who wouldn’t use the bot again was the representative for the Bot Creators who are not the target audience I think we meet the needs of our intended audience.

# Evaluation:

## Success Criteria:

|  |  |
| --- | --- |
| Criteria | Met? |
| Speed - A speed of less than 20 seconds | The program only meets this speed when the browser is headless, the GPU is disabled, and extensions are disabled which is fine as the user doesn’t need to visibly see the process happening |
| Reusability - Its able to be used multiple times by the same person | The program is extremely reusable with the multiprocessing that was added it means that a user can input a number of how many of this program they want to run in parallel |
| Speed - it runs without needing to have a browsers UI running | This has been done as well and doing it allowed the program to run even faster. |
| Ease Of Use - Easy to use for computer literate people | I got my mother who isn’t that skilled with computers to try and use it. She was successful and said that it wasn’t hard at all and was as simple as filling in a form and listening to the instructions the program gave her |
| Versatility – This application can be used for multiple products | You can switch between sizes colour and the types of items every product on the site is accessible to buy. However, there isn’t as many sites it supports as I would have liked to have. |
| UI should be clean and to the point. | The Ui is very clean and simple to follow there is no unnecessary fields or words. All fields are labelled so the person knows exactly what to type and where |
| Make a process that figures out URL generation | The URL generation works simply and is as little steps as possible it is relatively short to the rest of the program. |
| The program must be able to buy any item on the websites it supports automatically | The program is able to buy the item the only part it cannot get passed is the I am not a robot check which will still have to bed done by a human and sadly slows down the program quite a bit. |
| Make an easy way for the user to input the item they are looking for | The item has a simple field where the user is told to enter the item however they must use a dash to input the colour of the item and it is case sensitive. |
| Make sure the user knows when its loading | We have a loading bar as well as this the program prints loading when loading is taking place. |

**Further Development:**

When interviewed the representative for the bot creators suggested adding something to solve captchas. I think this would be a great are to improve because technically the purchase int made till the captcha is solved and that is the largest bottle neck to this bot right now. Some Solutions I have found are:

**2 Captcha / Recaptcha & Captcha farms:**

These are farms of actual workers that sign on and get paid for solving captchas it is easily intergrateable through API key. What happens is the captcha is sent by the program to the farm and a worker picks it up and attempts to solve it this process takes 8 seconds on average meaning it would be great to allow for 12 seconds of delay in the program for the captcha to be solved in case of any anomalies. The problem with these farms is that ethically these may not be seen as best for our customers as the pay per captcha is quite low and I would have to pay a measly 2.99 for 1000 captchas and workers get paid $0.3 per 1000 captchas solved. The problem is even worse when looking at Recaptchas where it takes about 39 seconds for Recaptcha solving and they get payed $1 per 1000 captchas. This means they get $1 for 10 hours of work. It also would add significant time to my program. The only benefit is that it automates the captcha / Recaptcha process.

**Modifying the browser and Chromedriver:**

This requires a lot of research as the only way I have found to do this is to change the binary of the chromedriver using a hex editor to one that takes out the flags that gives away that the browser is automated. While this is a valid way to do this, this hurts the maintainability of my solution as bot detectors are always becoming more sophisticated. This means that we would have to constantly find what part of our browser binary doesn’t seem human and adjusting it which could mean constant updates and downtime while we look for the next tell that our program is a bot. A way to fix this is by making our bot generate human like behaviour and having it have its own fingerprint as a human would as for a lot of sites if they think your browsers behaviour seems human they won’t Recaptcha / captcha check you.

**Artificial Intelligence:**

Another way may be to train an artificial intelligence on the source that captchas use which right now is google maps. We could gather a list of data that shows the most asked captchas and recaptchas and make train a bot to identify each item in them and train it to answer correctly this has already been done before to recaptchas using tensor flow. The technology for solving recaptchas is also there with powerful AI such as GPT-3 could mimic a human’s response to these recapthcas. This solution could take a while to research and develop, and the time invested may not be worth it when considering the place in the market our bot is meant to fill which is the middle f the market fast and automated enough to purchase normal items that aren’t extremely scarce items.

## Other improvements/features:

**Speed from Item drop to checkout:**

As the representative for the Bot creators told us the time from drop to check out need to be 15 seconds or less for the most sought-after items. Even though we extensively went into the reasons the improvement isn’t essential it is still good to have as it keeps us competitive against other bots in our market space. Speed is the best way for our solution to come out on top against the others.

**Adding in app proxies:**

As mentioned before this isn’t essential however it is nice to have everything in one app rather than having to go to many different apps. Furthermore, a VPN could slow down the connection time to the site while the difference in speed between the two is negligible it adds up with the amount of connections being made.

**Anti-pirating measures:**

It may be a good idea to have these measures as currently pirating the software is as easy as asking your friend for it. This is a problem as people could take our software and sell it for a lower price. As well as this why would you buy the software from us if you could just pirate it and download it by your friend giving it to you. This doesn’t just hurt our profits but it also means the efficacy of our bot goes down as so many people will use our bot and it will no longer be an advantage Over the average check out time since everybody is checking out in 20 seconds with our bot that means fewer products for our customers.

To fix this we could employ measures where users have to receive an email before they use the bot that entails a specific hash, they need to enter to gain access to it. Or we could make the program connect to a central server that holds data / code needed to continue with the program and give them each a unique hash ID and if 2 of that unique hash ID attempts to run the code the bot is forced to stop.

Putting these measures in place may allow people to still share their bot but it can only ever be used by one person at any time. This means that the problem of to many people having it and that reducing the efficacy is fixed as well as the problem of people selling it to many others in order to undercut us.

**Making it compliable:**

If the program is compliable it means that people wont have to manually download the long list of libraries and frameworks that make this program work and instead they would just have to download and run it this helps with ease of setup. It also stops people from being able to steal the source code so they cannot make their own.

## Maintainability:

The code is quite maintainable as it is simple commented and modular in design. This means that anyone helping maintain the program can easily understand what each part of the program does this means that if anyone wanted to add any features they can already see how each part o the program functions which makes it more obvious as to where and how the feature can be implemented while maintaining the integrity of the program. It is split into subroutines which makes it modular in nature the benefit to this is that each subroutine is encapsulated and can be tested individually which is good for debugging. The code is also indented to help with readability

Different parts of the project is split into different files this helps with maintainability as it means that changing one thing such as the UI wont mess with the integrity of the back end application an d vice versa.

However, some subroutines are dependant on others to run this somewhat breaks the complete encapsulation they have from one another.

## Limitations:

One limitation is that our program uses the requests library and has to connect to pages of websites to load them this is a limitation as someone’s internet connection could massively change the speed of our bot and those with bad internet connection may have a bad experience with the bot. it would be great if the program could run in the background while waiting for pages to load but since the program is a web scraper the page needs to be loaded for the program to work.

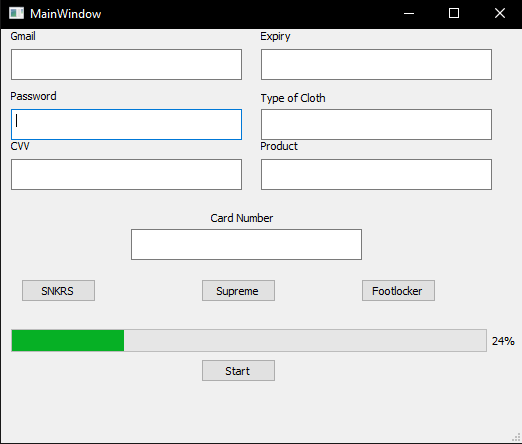
Another limitation Is the speed the individual can solve a captcha as this will have to be the only process done by a human the user will have to be fast about this and there is no way we can make this process faster for the user.

Another limitation is the lack of inbuilt anonymity as highlighted by a stakeholder in the form of asking for proxies. Bots can be found and if found some people find IP or email ban while we have discussed the steps to avoid that users that don’t want to use a vpn or cant as they are not allowed in countries like china don’t have an in app way to have anonymity. Also, the app provides no way yet to hide/spoof someone’s email.

## Usability check list:

* **My User interface should be simple - Y**
* **My User interface should be accessible to those that are visually impaired - Y**
* **My user interface will use as little words as possible - Y**
* **My user interface will have ample spacing between words - Y**
* **My user interface will have simple non-complex wording - Y**
* **The terminal for the application should have simple error messages for easy troubleshooting - Y**
* **Loading should be easily communicated – Y**

## Usability Features:



These labels are there so those using the program understand what goes where. The labels language can also easily be changed

The loading bar communicates to the user what stage the program is in and also serves as something to tell the user something is happening.

Buttons are evenly spaced out as to not cause confusion that way for users with visual impairment or dyslexia the words of each button and label merge.

The borders turn blue when something is selected that way the user knows where they are typing

Input boxes are present to keep down the amount of text on screen this also allows users to voice type if they have trouble using a keyboard.

## Usability Interview:

Questions

Question 1) Are you happy with the current usability features

Question 2) Is there anything you are not happy about

Question 3) What would you want to add

Casual buyers

Answers

1. Gibraltar – Yes they helped me a lot without the loading bar I wouldn’t have known that processes were taking place and might have ran into problems as I would have thought it was frozen.
2. Gibraltar – I’m not happy with how small the text is on the page I would like it to be a bit bigger as I need to put my glasses on to read it and even then I still struggle a bit
3. Gibraltar- I would like it to say when an item has been successfully or unnsuccesfuly bought

Regular Bot Users

Answers:

1. Tchalla – Its alright its straight forward but I would like there to be more information on the screen
2. The fact that you must sometimes refer back to the terminal
3. Definitely a clear indicator that says when a product was bought or an indication that the bot failed

Bot Creator

Answers:

1)Bot Creator- Yes as I have dyslexia sometimes I find it really hard to read extremely cluttered User Interface and this only had the essential information

2) Bot Creator - No I would say it’s a good start apart from the fact that in the night time its extremely bright

3) Bot Creator - you may want to add a dark mode to combat eyestrain for people as white light can be quite harmful and consider and adding Tabs or multiple windows means that you can keep from having a cluttered UI while also having more information.

## Improvements for the future:

I think adding more of the information pertaining to the completion of the purchase and how many processes are spawned would help the user be able to more easily use the bot. As 2 of the interviewees said they would like information on their UI of if the bot was successful or not. I also think that having the terminal as a place only to refer to if an error happens is a better approach to have then require the user to use the terminal and Ui as the representative for the casual buyers highlighted. I also think that a dark mode would be great however I don’t think we need more information on the current screen but a solution like that suggested by the Bot Creators of using tabs maintains minimality as well as information. As well as an inbuilt proxy

A way to also get the bot past captchas consistently that way more purchases can take place and users are more satisfied with their purchases.

Turning the UI into a html interface with CSS will mean that there is a lot more I can do with it visually so I think it will make it look a lot better.

Finally, adding compatibility with more sites in the future as it only works with Supreme so far.