

DAta scraping/file transfer project

The Fixed Income trading team have assigned three other colleagues and myself to a project, which will automate a process in which they do every morning manually. They have asked us to download a series of files regarding financial trading statistics, manipulate the data to their formatting preferences and save it to their desktop, so that they are able to upload it.



June 29, 2018

Student Name : Jason Fernandes

Tutor : Vy Tran   
Line Manager: Katie Hirchovits

Contents

[**Introduction** 3](#_Toc526114858)

[Project Brief 3](#_Toc526114859)

[Purpose of the Document 5](#_Toc526114860)

[Scope 5](#_Toc526114861)

[Who with? 5](#_Toc526114862)

[**Solution Overview** 6](#_Toc526114863)

[**How the program will work [Task 1 – ICAP]** 6](#_Toc526114864)

[Screenshot when file is downloaded 6](#_Toc526114865)

[Screenshot when file is edited by Kristof 7](#_Toc526114866)

[**How the program will work [Task 6 – MarketAxess]** 8](#_Toc526114867)

[Screenshot of website when run 8](#_Toc526114868)

[Screenshot of excel spreadsheet when edited by Kristof 8](#_Toc526114869)

[Project timescale and dependencies 9](#_Toc526114870)

[**Functional Requirements [Task 1 - ICAP]** 10](#_Toc526114871)

[**Functional Requirements [Task 6 – Market Axess]** 10](#_Toc526114872)

[Communication to Client [Kristof] 11](#_Toc526114873)

[**Process Map (Task 1 – ICAP)** 13](#_Toc526114874)

[**Process Map (Task 6 – MarketAxess)** 14](#_Toc526114875)

[**Development of the Program – Task 1** 15](#_Toc526114876)

[**Version 2** 16](#_Toc526114877)

[Section 1 16](#_Toc526114878)

[Section 2 16](#_Toc526114879)

[Section 3 17](#_Toc526114880)

[Variables used in final code [Task 1 – ICAP] 18](#_Toc526114881)

[**Development of the Program – Task 6** 19](#_Toc526114882)

[**Version 2** 20](#_Toc526114883)

[**Version 3** 21](#_Toc526114884)

[**Version 4** 22](#_Toc526114885)

[Section 1: 22](#_Toc526114886)

[Section 2: 23](#_Toc526114887)

[Section 3: 24](#_Toc526114888)

[Variables used in final code [Task 6 – Market Axess] 25](#_Toc526114889)

[Version Control 26](#_Toc526114890)

[**Testing** 28](#_Toc526114891)

[Testing the Final Program [Task 1 – ICAP] 28](#_Toc526114892)

[Testing the Final Program [Task 6 – Market Axess] 30](#_Toc526114893)

[**Documentation** 32](#_Toc526114894)

[Minimum Hardware and Software requirements 32](#_Toc526114895)

[**Reflection of Code** 33](#_Toc526114896)

[Task 1 – ICAP 33](#_Toc526114897)

[Task 6 – MarketAxess 33](#_Toc526114898)

[**Deployment** 34](#_Toc526114899)

# **Introduction**

Project Brief  
A colleague, Kristof McFerren, in the Fixed Income trading desk within the analytics department at Bloomberg is required on a daily basis to manually download a series of files regarding financial trading statistics, manipulate the data within, combine into a single sheet and upload to Bloomberg databases for reference by clients. This process currently takes approximately an hour of time to complete per day and so is an inefficient use of resources. To assist with this, we have been tasked with automating the process by writing a script that will carry out the necessary tasks, and result in a fully populated excel sheet that will be ready for upload by Kristof. The steps vary for each sheet, as the data comes in different formats, and so the tasks will be distributed between the four people within our team responsible for this. A detailed breakdown of each task is as follows:

|  |  |
| --- | --- |
| **Task 1 ICAP:**  1) Run <http://www3.icap.com/sef/marketdata/igdl/ICAPSEFMarketDataIGDL.csv> 2) Sheet on here updates daily. Download the file from here  3) Highlight Column A >> Data Tab >> Text to Columns 4) Choose “Delimited” then click Next>. 5) For Delimiters choose “Tab” and “Other” and enter the symbol “|” 6) Click on “Finish” 7) Take the sum of all values under Column Z labelled Total Notional \_USD\_NDA 8) Delete the last 3 digits from the sum 9) Label this value as “ICAP IRS” | **Task 2 Tradition:**  1) Run <http://www.traditionsef.com/dailyactivity/SEF16_MKTDATA_TFSU_20180913.csv> 2) Sheet on here updates daily. Download the file from here 3) In Cell “C2” enter the formula: =A2&B2 4) Fill this formula down for the entire length of Column C 5) Copy A1 into C1 6) Copy the entire column C into Column D 7) Delete Columns A-C and shift cells to the left 8) Select Column A >> Data Tab >> Text to Columns 9) Choose “Delimited” then click Next>. 10) For Delimiters choose “Tab” and “Other” and enter the symbol “|” 11) Click on “Finish” 12) Highlight Row 1 and enter a Filter from the Data tab 13) Filter in column G for “IR” 14) Take the sum of all values under Column AA labelled Total\_Notional\_USD\_DA 15) Delete any digits after the decimal point and then delete an additional 3 digits before the decimal point 16) Label this value as “Tradition IRS” 17) Filter in Column G for “CD” 18) Repeat steps 14 and 15 19) Label this value as “Tradition CDS” |
| **Task 3 GFI:**  1) Run <http://www.gfigroup.com/doc/sef/marketdata/2018-09-13_daily_trade_data.xls> 2) Sheet on here updates daily. Download the file from here 3) Select and copy all the data in the table (Columns B-J) excluding the column headers 4) Open the excel sheet I attached called “SEFD GFI Template (nEW) 5) Paste the data in Cell B6 6) Filter in Column J and select values: CLF, CLP, KRW. They will not always all be there but if they are present then we need to select them. 7) Once filtered for those values, remove 2 digits from all cells in Column I 8) Once digits are removed remove the filter from column J so all currencies are showing again 9) Filter for “Interest Rate” in Column B 10) Take the sum of all values in Column K and delete any digits after the decimal point and then delete an additional 3 digits before the decimal point 11) Label this value as “GFI IRS” 12) Filter for “Credit” in Column B 13) Repeat step 10 14) Label this value as “GFI Credit” | **Task 4** **TRUEX:**  1) Run <https://www.trueex.com/download-settlement-data> 2) Sheet on here updates daily. Download the file from here 3) Take the sum of all values in Column AA, Total\_Notional\_USD\_DA 4) Delete any digits after the decimal point and then delete an additional 3 digits before the decimal point 5) Label this value as “TRUEX IRS” |
| **Task 5** **TradeWeb/DealerWeb:**  1) Run https://www.tradeweb.com/institutional/derivatives/sef-center/ 2) Download File called “DW SEF CDS – End of Day Data” 3) Sum all values in Column E 4) Label this value as “DealerWeb CDS” 5) Download File called “DW SEF IRD – End of Day Data” 6) Sum all values in Column E 7) Label this value as “DealerWeb IRS” 8) Download File called “TW SEF CDS – End of Day Data” 9) Sum all values in Column E 10) Label this value as “TradeWeb CDS” 11) Download File called “TW SEF IRD – End of Day Data” 12) Sum all values in Column E 13) Label this value as “TradeWeb IRS” | **Task 6** **MarketAxess:**  1) Run <https://www.marketaxess.com/trading/sef-data.php> 2) Scrape the values from column called “Total Volume USD (‘100) 3) Sum these values 4) Label this value as “MarketAxess CDS” |

## Purpose of the Document

The reason why I am creating this document is that it will show the evolution of the automating task that I plan to carry out for the Fixed Income Trading team. The process will be completed by creating a script in python that will perform the necessary instructions for each task, as documented in the brief. Consequently, the python program should output a fully populated excel sheet that will be ready for upload by Kristof. This document will show each part of the program and cover all the stages in the software development life cycle.

## Scope

Kristof, whose job it is to upload trading data on to Bloomberg will use the programs in which my colleagues and I plan to create, since the process currently takes approximately an hour of his time to complete per day and therefore is an inefficient use of resources. The plan is that he will run these programs daily in order for a spreadsheet to be fully generated in the correct format, so all he would have to do manually is upload the files to a specific place on Bloomberg.

## Who with?

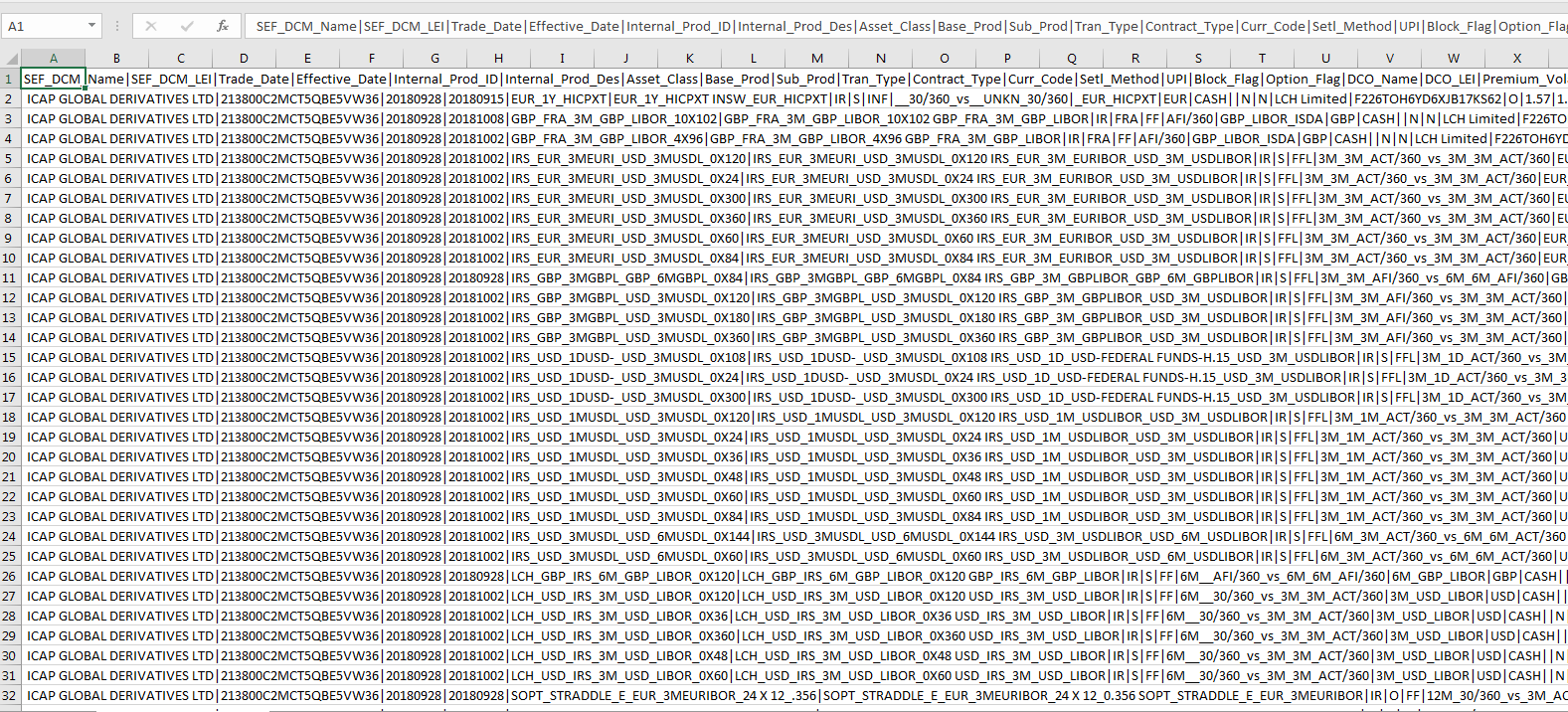
I am working with three other colleagues, as well as myself to complete this project, however I will lead the project since I have taken the responsibility to become the Project Manager. I will delegate each of the six tasks as shown in the brief to myself and the 3 other colleagues, depending on our strengths/weaknesses.   
  
Furthermore, as being the project manager, I will be regularly checking in with Kristof in order to check whether what we are creating is exactly what he wants. Communication between my other colleagues and Kristof will go through myself since we plan to have a structure of questioning, otherwise Kristof will be bombarded with questions from the 4 of us. To add to this, I will also be monitoring version control for all of the six tasks, in which I have set up a shared file between myself and 3 other colleagues and plan to see the evolution of the versions in there

# **Solution Overview**

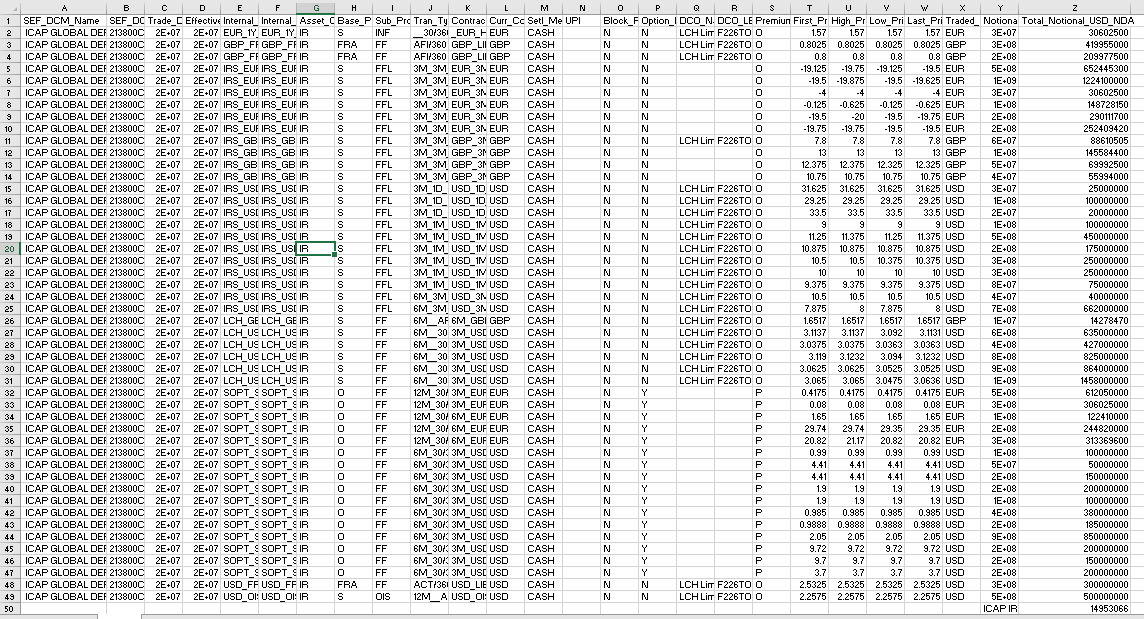
**How the program will work [Task 1 – ICAP]**  
I will be creating a program in which will automate the steps in Task 1 that has been outlined in the brief by Kristof. What the program should do is look for the sheet, <http://www3.icap.com/sef/marketdata/igdl/ICAPSEFMarketDataIGDL.csv> and download all the figures into an excel document.

Once the file is downloaded, various formatting on the file should be completed, such as:  
  
 Highlight Column A >> Data Tab >> Text to Columns  
 Choose “Delimited” then click Next>.  
 For Delimiters choose “Tab” and “Other” and enter the symbol “|”  
 Click on “Finish”  
  
After the formatting has been done on the excel file, a calculation should take place in which a sum up all the values under Column Z labelled Total Notional \_USD\_NDA. Then the 3 last digits of the sum calculation should be deleted and consequently the value should be labelled as “ICAP IRS”. Therefore once the program has run, an excel file should be able to be found in the same folder as the python program. A user will have to manually open the spreadsheet in order to see the correct structured format, obeying to the rules as stated above.

### Screenshot when file is downloaded

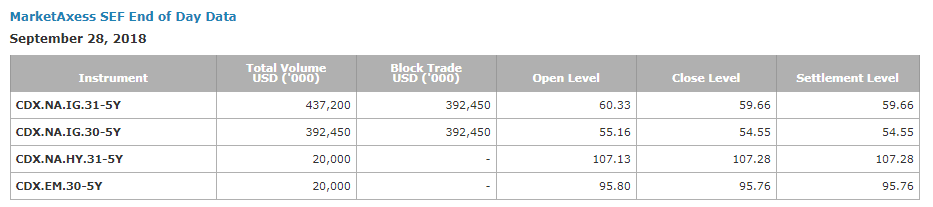
The below screenshot shows the format of the excel sheet when Kristof downloads the file <http://www3.icap.com/sef/marketdata/igdl/ICAPSEFMarketDataIGDL.csv>

### Screenshot when file is edited by Kristof

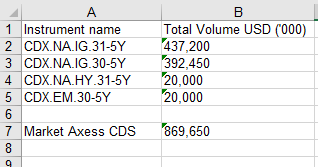
  
Once Kristof has done the specific edits to the spreadsheet as said in the brief, the spreadsheet looks like this:  
  
  
As seen in the screenshot above, every piece of information should be pasted into a single cell. A calculation should also take place in which should sum up all the values under Column Z labelled Total Notional \_USD\_NDA. Then the three last digits of the sum calculation should be deleted and consequently the value should be labelled as “ICAP IR”, as seen below:  
  
\*Please note that the spreadsheet is more than 21 lines and the ICAP IR is in outputted always in the bottom right hand corner of the spreadsheet in column Z.

**How the program will work [Task 6 – MarketAxess]**  
  
I will also be creating another program in which will automate the steps in Task 6 that has been outlined in the brief by Kristof.   
  
What the program should do is look for the website, <https://www.marketaxess.com/trading/sef-data.php> and scrape the values in the column called “Total Volume USD (‘100) into an excel spreadsheet. All the values in which appear in the column should be scraped, since the number of instruments changes daily and therefore there is not a specific number of values that appear.   
  
After all of the values are retrieved and outputted in an excel sheet, it should perform a calculation in which will sum up all of these values from this column and label this “Market Axess CDS”.

### Screenshot of website when run

The below screenshot shows the format of the data when Kristof loads the website <https://www.marketaxess.com/trading/sef-data.php>

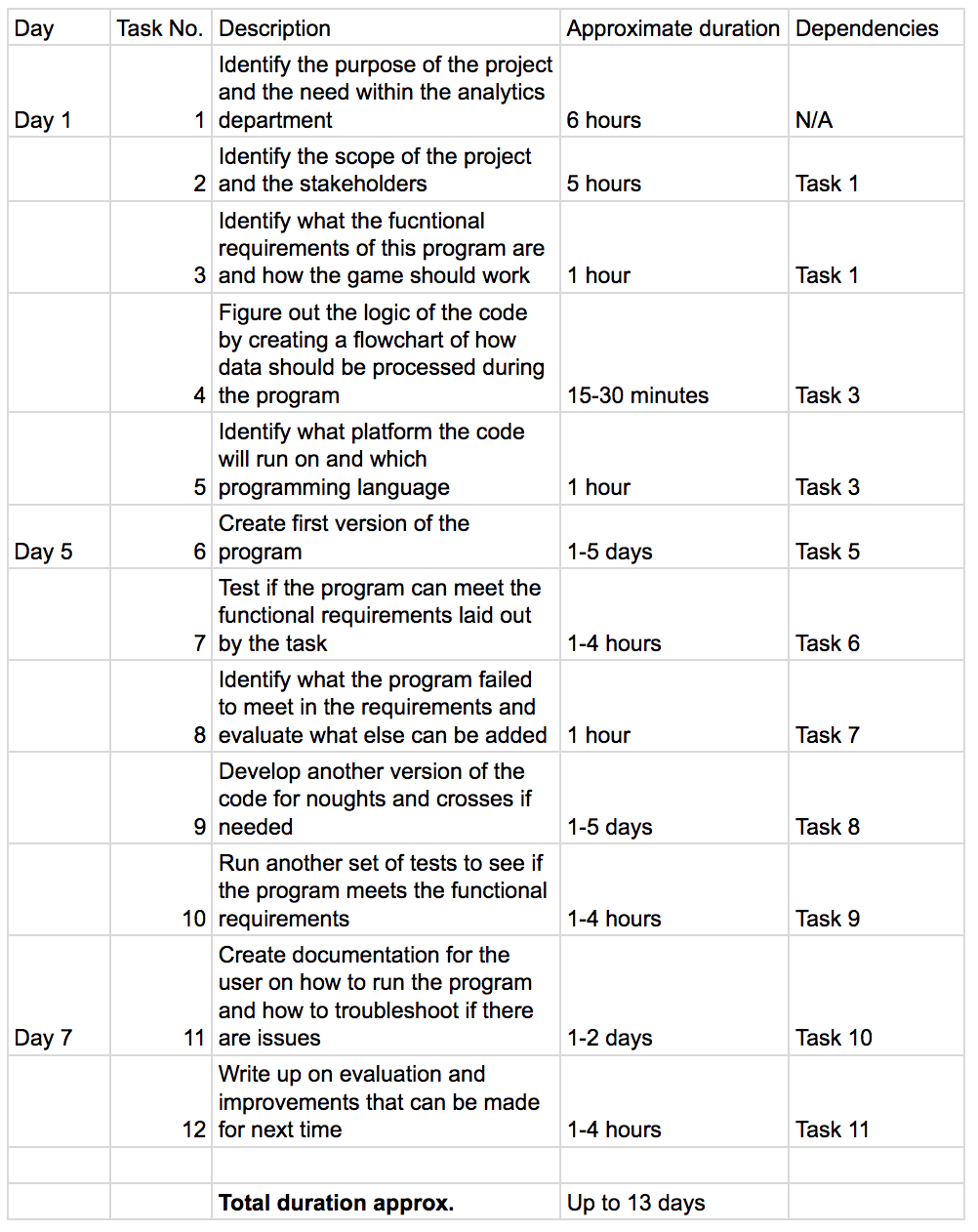
### Screenshot of excel spreadsheet when edited by Kristof



Once Kristof has done the copied and pasted the instruments and total volume values into a spreadsheet, he adds it up all the volumes and labels it “Market Axess CDS

Project timescale and dependencies

I plan to carry out the development of the program for task 1 over 2-3 weeks. I also followed this plan for Task 6 and therefore it is expected to take 5-6 weeks for both tasks to be fully completed. The project timescale takes into account identifying what the code should do, creating multiple versions of the code, and testing if it meets the requirements.



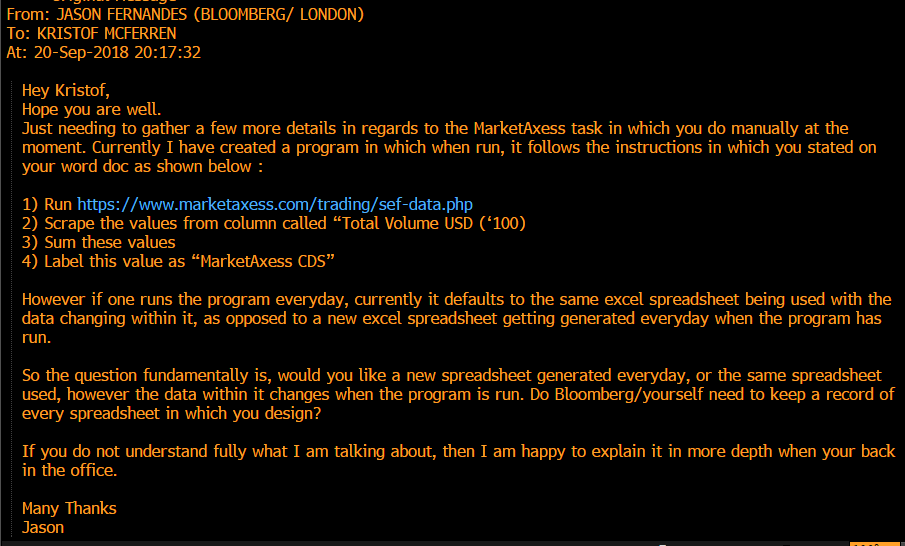
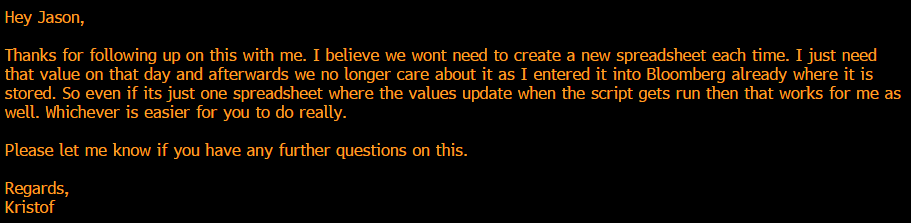
**Identify what the requirements of this program are and how the program should work**

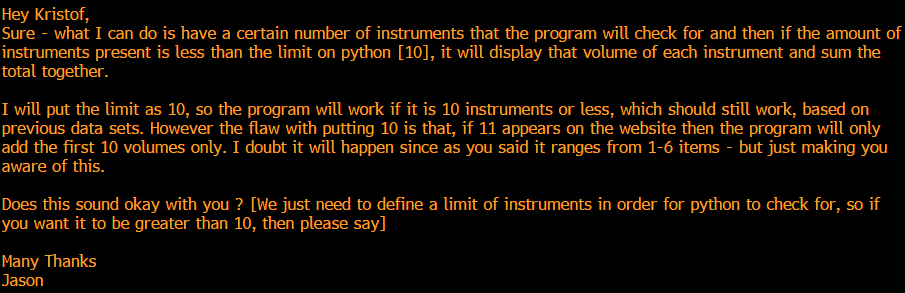
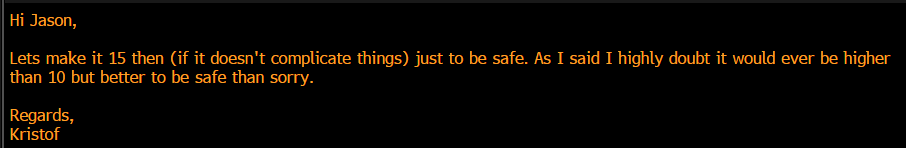
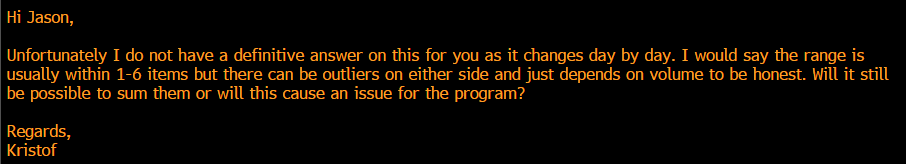
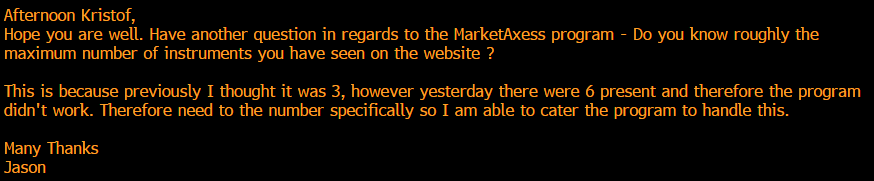
## **Functional Requirements [Task 1 - ICAP]**

I did not have to go into full depth on the analysis stage because Kristof knew exactly what the program had to do/include, as he does this task manually every day. Therefore below are the functional requirements in which he has given to me via the brief, in order to integrate into the program:  
  
1) Run <http://www3.icap.com/sef/marketdata/igdl/ICAPSEFMarketDataIGDL.csv>  
2) Sheet on here updates daily. Download the file from here   
3) Highlight Column A >> Data Tab >> Text to Columns  
4) Choose “Delimited” then click Next>.  
5) For Delimiters choose “Tab” and “Other” and enter the symbol “|”.  
6) Click on “Finish”.  
7) Take the sum of all values under Column Z labelled Total Notional \_USD\_NDA.  
8) Delete the last 3 digits from the sum.  
9) Label this value as “ICAP IRS”.  
  
Through a series of meetings with Kristof, he said that the above steps are the main requirements which the program should follow. However he made it clear that since he himself doesn’t have much programming knowledge/experience, he was happy to do some of the steps above, if this could not be fully coded in python.

## **Functional Requirements [Task 6 – Market Axess]**

Again, for this task I did not have to go into full depth on the analysis stage because Kristof knew exactly what the program had to do/include, as he performs this task manually every day. Below are the functional requirements in which he has given to me via the brief, in order to integrate into the program:  
  
1) Run <https://www.marketaxess.com/trading/sef-data.php>  
2) Scrape the values from column called “Total Volume USD (‘100).  
3) Sum these values.  
4) Label this value as “MarketAxess CDS”.  
  
Through a series of meetings with Kristof, he said that the above steps are the main requirements, which the program should follow. However, he made it clear that since he himself does not have much programming knowledge/experience, he was happy to do some of the steps above, if this could not be fully coded in python.

Communication to Client [Kristof]   
  
Since I am the project manager for this Project, I have been having regular meetings with Kristof on a weekly basis to talk about the requirements of each task – whether that was the ones assigned to myself or those of my colleagues in which I would pass on the feedback.  
  
If I needed to clarify something with Kristof that happened after the meeting for that particular week, I would write him an email so I would be able to get the quickest response possible in order to proceed with the project.   
  
Below is an example of an email which I sent to Kristof, asking him a question about Task 6. The question I asked him was regarding if he wanted the same spreadsheet folder to be overwritten every day or if a new spreadsheet should be created every day.  
  
  
  
  
As seen in the above screenshot, he replied saying that either is good as he only needs the value for that day.  
Another query, which I had to email Kristof about was that as the website updates on a daily basis, there were a different amount of instruments showing on each day. Initially I made my program cater for 3 instruments since I was not aware that more than 3 instruments showed. However, one day 6 instruments were displayed on and therefore since I always had assume the most to be 3 instruments displaying, I proposed the question to Kristof as to how many instruments usually appears on the website, since it was vital that all instrument volumes were recorded on the excel file.

Below shows the emails that exchanged between Kristof and myself discussing about this matter and consequently we agreed for the python code to cater for 15 instruments, since as Kristof said it’s “better to be safe than sorry”.

## **Process Map (Task 1 – ICAP)**

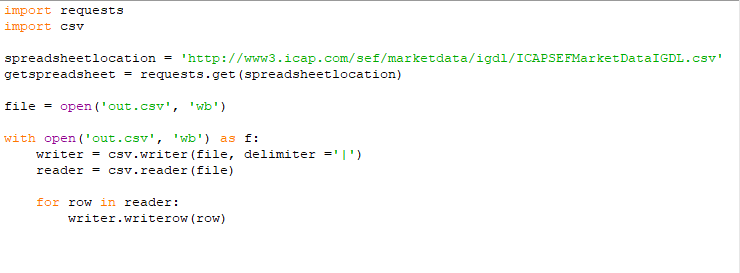


## **Process Map (Task 6 – MarketAxess)**

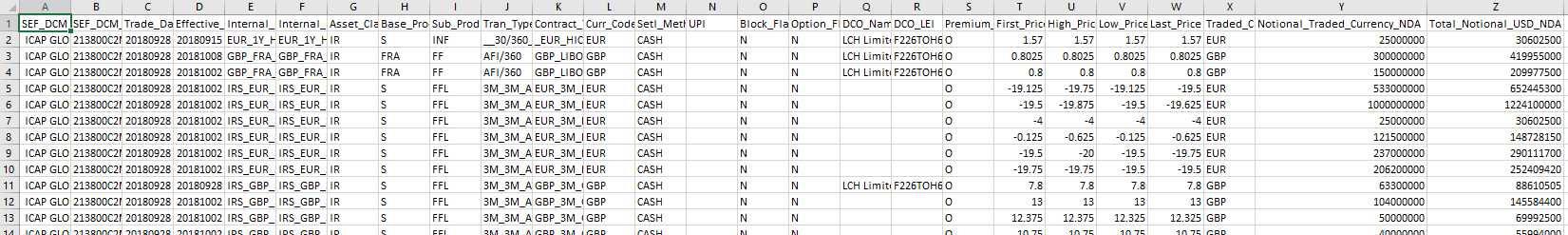


# **Development of the Program – Task 1**

Whilst I was developing the program for task 1, I came up with different ways as to how this game could work. There were two versions in particular which maps out the key stages of the creation upon the final product. Below I will explain each version and why I didn’t settle for that to be the final program – both two versions have been coded in python rather than the Bloomberg platform BQUANT due to security reasons.   
  
**Version 1**   
Version 1 of the code was basic in terms of functionality. It grabbed the .csv file from the internet, created a new csv file called out.csv and wrote each line into it. However, it did not manage to calculate the total of column Z and just outputted a spreadsheet without the total and therefore a version 2 was needed.



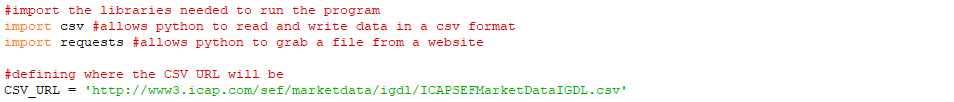
**Output**:



**Version 2**Version 2 of the code had additional lines and was the final version of the code as it was able to fully calculate the total of column Z and add it into the spreadsheet.

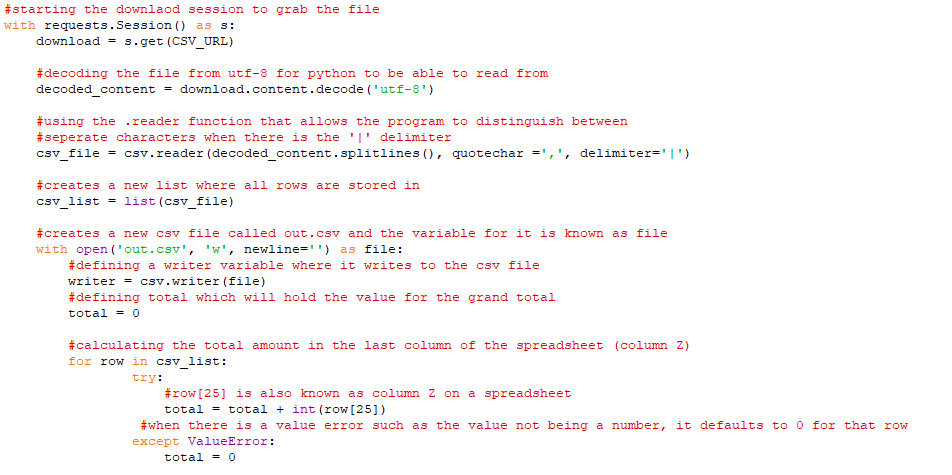
### Section 1

In section 1 this is where the 2 libraries are imported that are used in the program are imported. The csv library allows python to read and write data in a csv format and the requests library allows python to go onto a website and grab data from the internet. I have also defined in this area the URL of where the .csv file will be downloaded



### Section 2

Within this section is where the CSV file downloaded from the internet is analysed and read by python. It is then put into a list called ‘csv\_list’ where each line has its own row. The total is also calculated by adding up each value in row[25] which is also known as column Z on a spreadsheet.



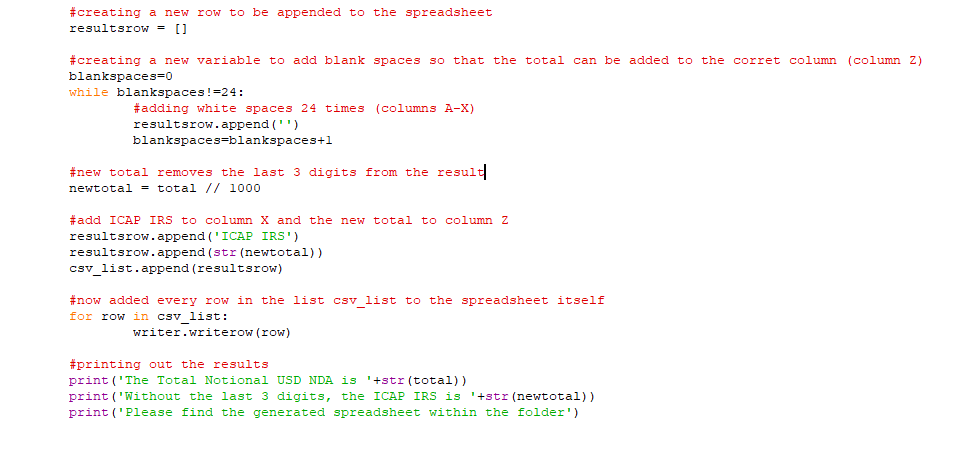
**Output**:

The only output from this section is the creation of the spreadsheet:



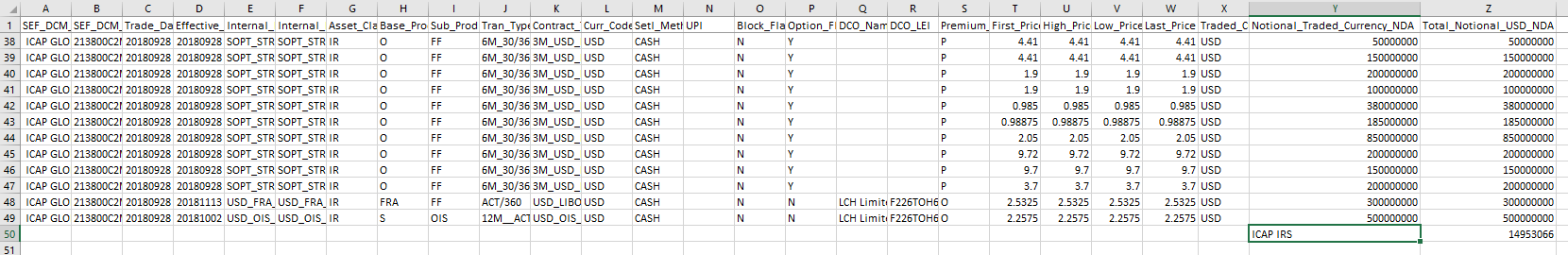
### Section 3

Within this section, the csv file is read and analysed by python and put into rows and columns through the delimiter of ‘|’. The total is also calculated from column Z and added to a new list called ‘resultsrow’ that will be added to the list ‘csv\_list’ which will form a new row to be added. All of the contents in the list csv\_list is written to the spreadsheet. The results are also outputted via python.



**Output:**





### Variables used in final code [Task 1 – ICAP]

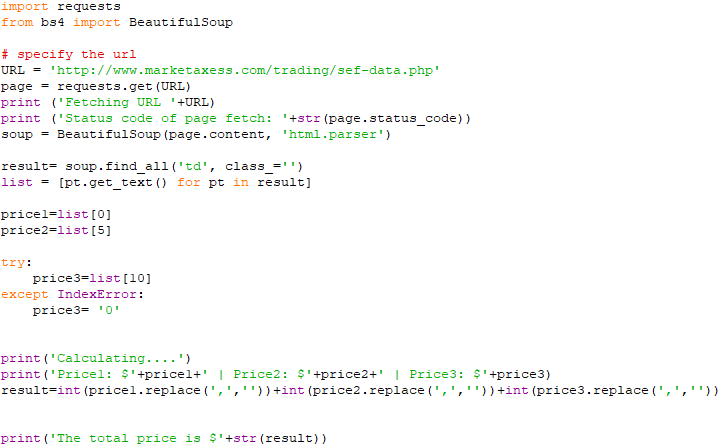
In this code I would use the following variables with specific purposes:

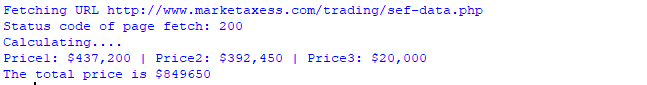
* **CSV\_URL** = This was the variable that defines the website and location of where to download the .csv file from
* **Download** = This was the variable which downloaded the .csv file from the website
* **Decoded\_content** = This variable decoded the .csv file from utf-8 which allows for python to read the .csv file
* **CSV\_file** = This variable read through the csv file and identified what character seperates each value through the delimiter
* **CSV\_LIST** = This list is created from csv\_file variable. Each row is put into its own index within the list ‘csv\_list’
* **Writer** = The variable which defined the function .writer to write each row into the csv file
* **Total** = The variable to contains the grand total of column Z within the .csv file
* **Resultsrow** = The variable that holds the new row to be added to the spreadsheet that contains the total result of column Z
* **Blankspaces** = The variable that will determine how many blank spaces within a row will be added
* **NewTotal** = Contains the total value but without the last 3 digits

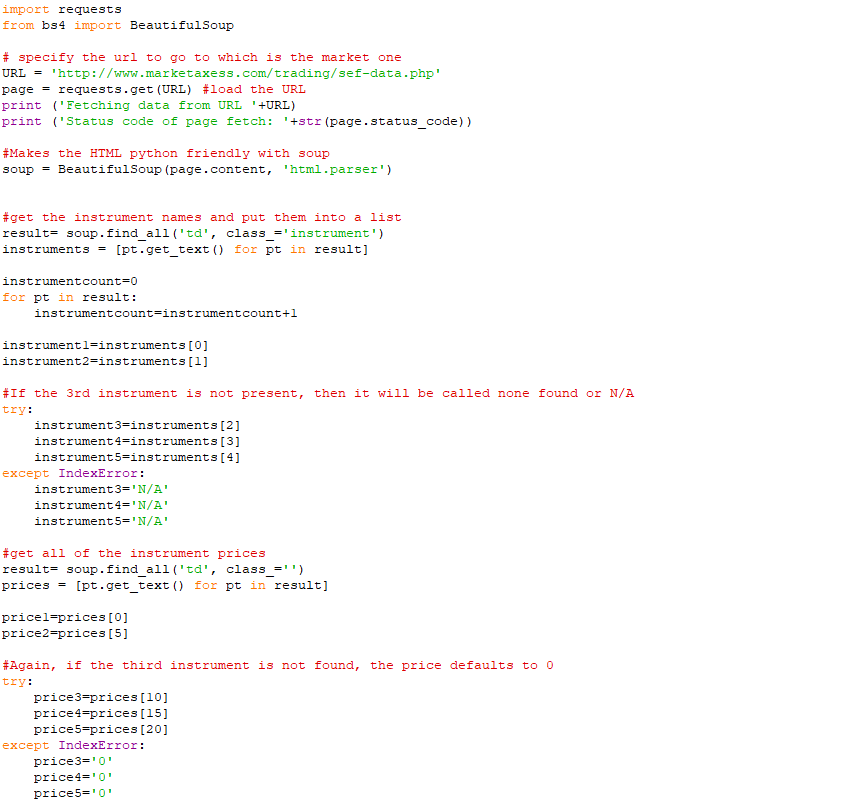
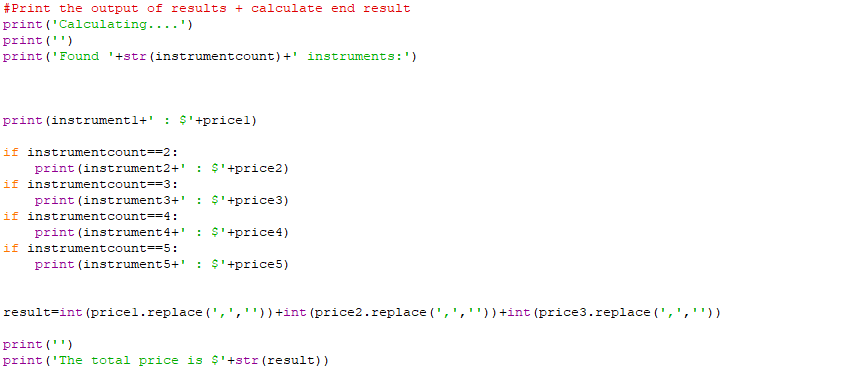
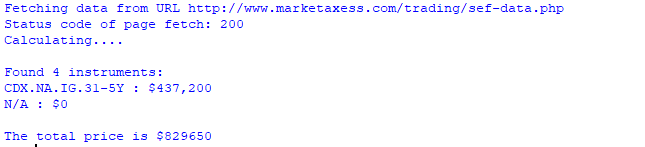
# **Development of the Program – Task 6**

Whilst I was developing the program for task 1, I came up with different ways as to how this game could work. There were three versions which maps out the key stages of the creation upon the final product. Below I will explain each version and why I didn’t settle for that to be the final program – all three versions have been coded in python rather than the Bloomberg platform BQUANT due to security reasons.  
  
  
**Version 1**

Version 1 of the code made use of the requests and BeautifulSoup libraries and was able to load the website and find the prices for each of the volumes. I had also identified that for each of the prices, it was under every 5 ‘td’ html tags within the website code. However it did not save the results to a spreadsheet at this point.

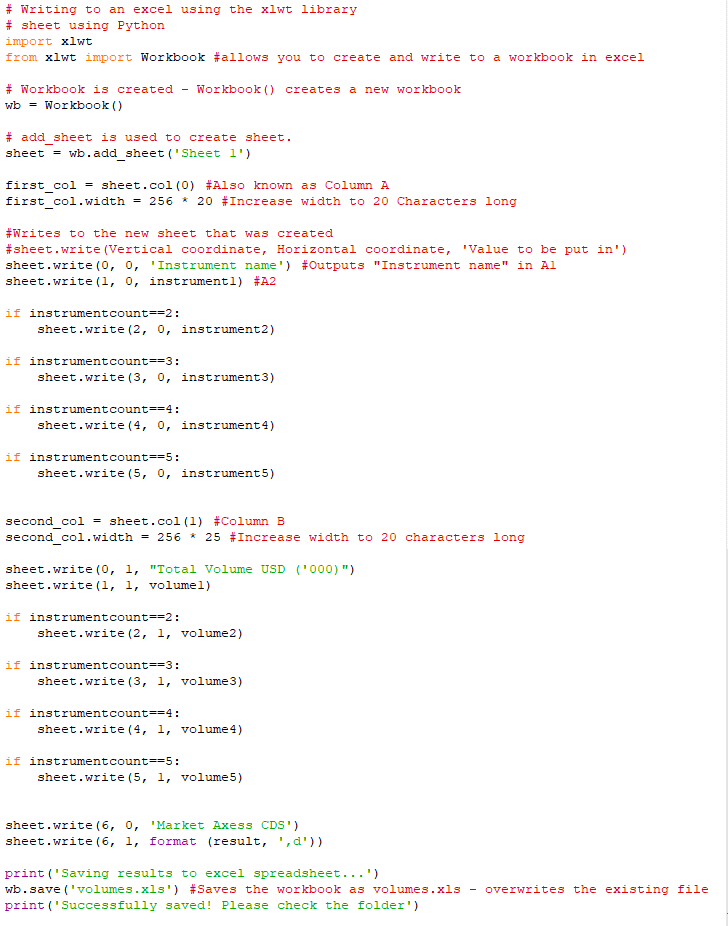


**Outputs:**  


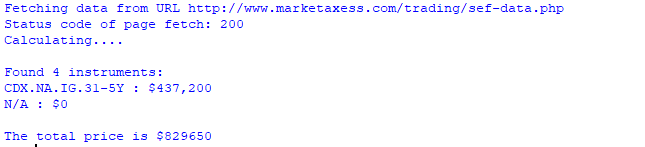
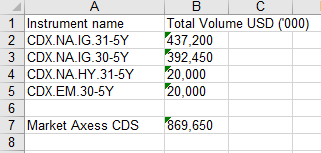
**Version 2**Version 2 did not include the program creating an excel spreadsheet, however it did now analyse and provide the instrument names to match them to the volume prices.   
   
**Outputs**:  


**Version 3**

Version 3 of the code used the same base as version 2, however it added a section below the code to create an excel spreadsheet with all of the results:



**Outputs**:

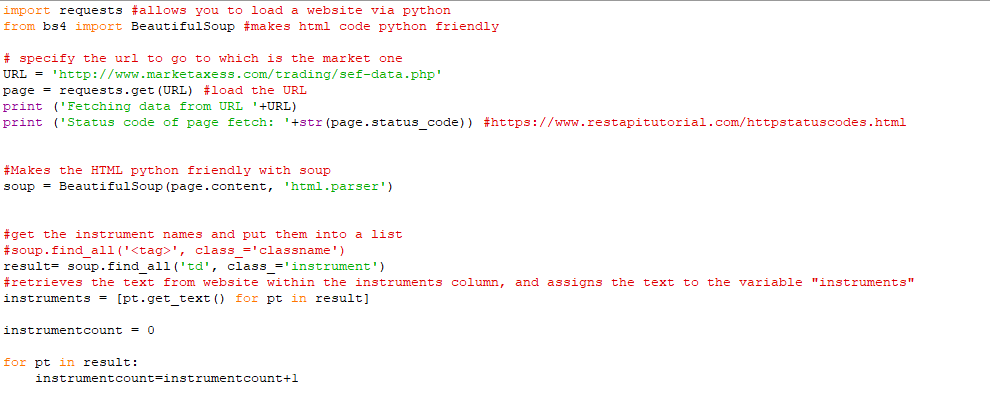


**Version 4**

Version 3 of the code was most efficient piece of code when compared to the other 3 versions. I realized that from versions 1-3 it did not cater for the website have more than 4 instruments and it would then miss out on important volumes and instruments. This final code, dynamically add instruments no matter how many there are. It can also be seen that in the previous versions, it outputted the instrument names as N/A even though there were instruments. The code below has been broken down and will be explained in sections.

### Section 1:

For the first section, the code imports the two libraries needed to make the program work. The requests library allows python to get information from the internet such as the html code or a file. BS4 and Beautifulsoup grabs the html code from a website and makes it python friendly which can be analysed and read through via the html parser. I have also created a new variable called result, where it finds all of the html tags with ‘td’ and the class as ‘instrument’ as this allows for all the instrument names to be identified. There is also an instrument count to count the number of instruments that have been found. All instrument names are put into a list called instruments.



**Outputs:**

This is what the instruments list looks like:

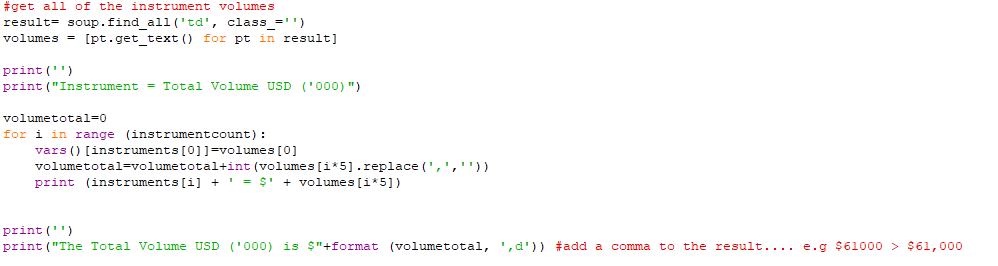


The output of the page status and the URL where data is being grabbed from:



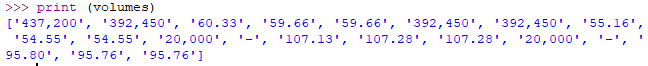
### Section 2:

Within section 2 are where all of the volumes from within the html code are identified. However, the website did not correctly class or identify the volumes properly, so all items with the tag of ‘td’ are put into the volumes list. This will be seen in the volumes list in the output. However, I did manage to correct this by multiplying the number by 5 as every 5 items contained the volume for each instrument.

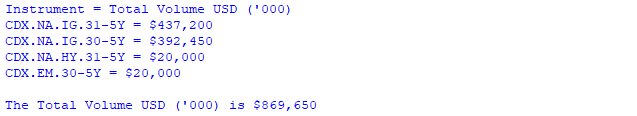


**Outputs:**

This is what the volumes list looks like:

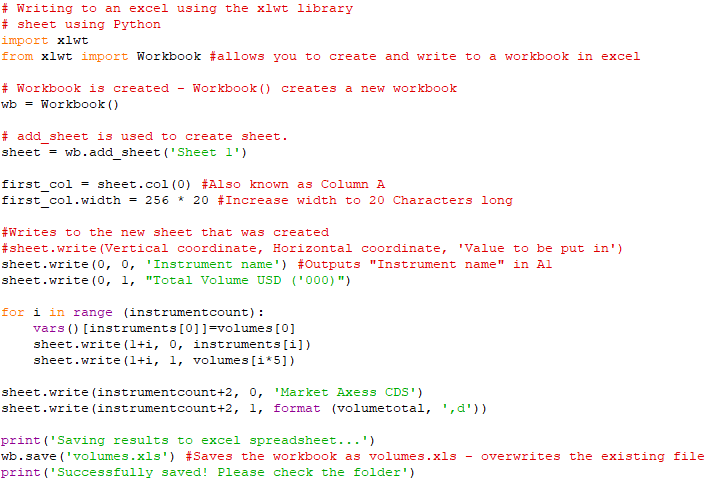


The total is outputted with the correct volume as seen below:



### Section 3:

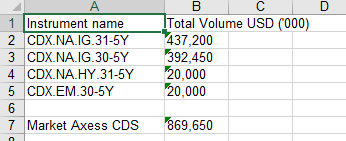
Within this section is where the results are written in an excel spreadsheet. This is achieved using the xlwt library and the workbook function that lets you create and write a workbook that can be opened by excel. The .write function from the xlwt library allows you to write to the sheet that has been created and list all of the instruments within it.



**Outputs:**

The outputs from within python are:





### Variables used in final code [Task 6 – Market Axess]

In this code I have used the following variables with specific purposes:

* **URL** = Defines the URL to grab the html code from
* **Page** = Where python actually loads the page
* **Soup** = Where BeautifulSoup makes the html code python friendly and readable
* **Result** = Where python matches all instrument names within the html class ‘instrument
* **Instruments** = A list where all of the results are placed into
* **Instrumentcount** = A count of how many instruments are found
* **Result** = Where python matches all volumes within the html tag ‘td’
* **Volumes** = A list that contains all potential volumes found on the website
* **Volumetotal** = Contains the grand total of all of the volumes added up
* **Wb** = Creating a new workbook within python
* **sheet** = Creating a new sheet called ‘sheet 1’
* **first\_col** = Defining the first column on the sheet

# Version Control

In order for this project to be a success, I made sure as the project manager that my colleagues, as well as myself uploaded all the different versions of code for the tasks in which were responsible for. We created a shared folder on the function {FILE} within Bloomberg, in which all four of us were able to access throughout the development stage of the project, to ensure that we backed up our work.

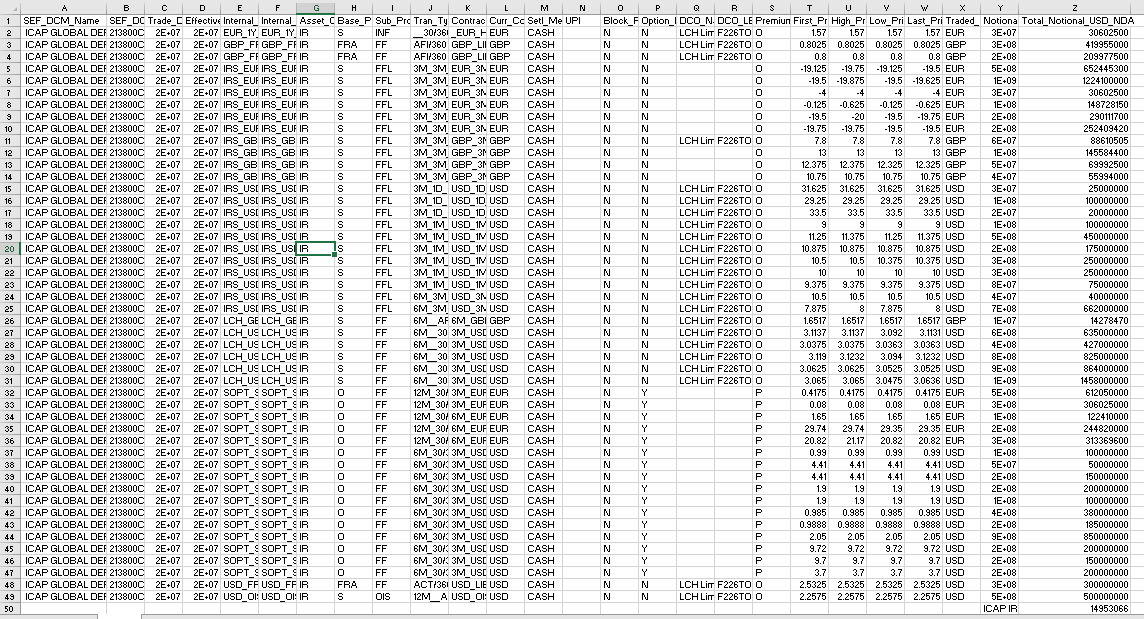
  
  
  
  
  
  
As seen on the screenshot above, my colleagues have also uploaded their progress to their own tasks throughout the development stage. By making this folder, we have been able to monitor and keep track of different versions of each task. We have also been able to go back to previous versions of the code if for any reason we did something wrong, or something in the code seems to not work now when compared to before.  
  
  
  
Below is a screenshot of the privileges that everyone had to the folder. I assigned myself the Administrator permission, since this allowed me to add others to the folder as I was the creator of it. Also, my colleagues was given the access to Read/Write to the folder, but they were not able to add anyone to the folder as only the one who is the admin [myself] is able to do that.

The two versions in which I uploaded to the shared folder for Task 1 –ICAP.

The four versions in which I uploaded to the shared folder for Task 6 –MarketAxess.

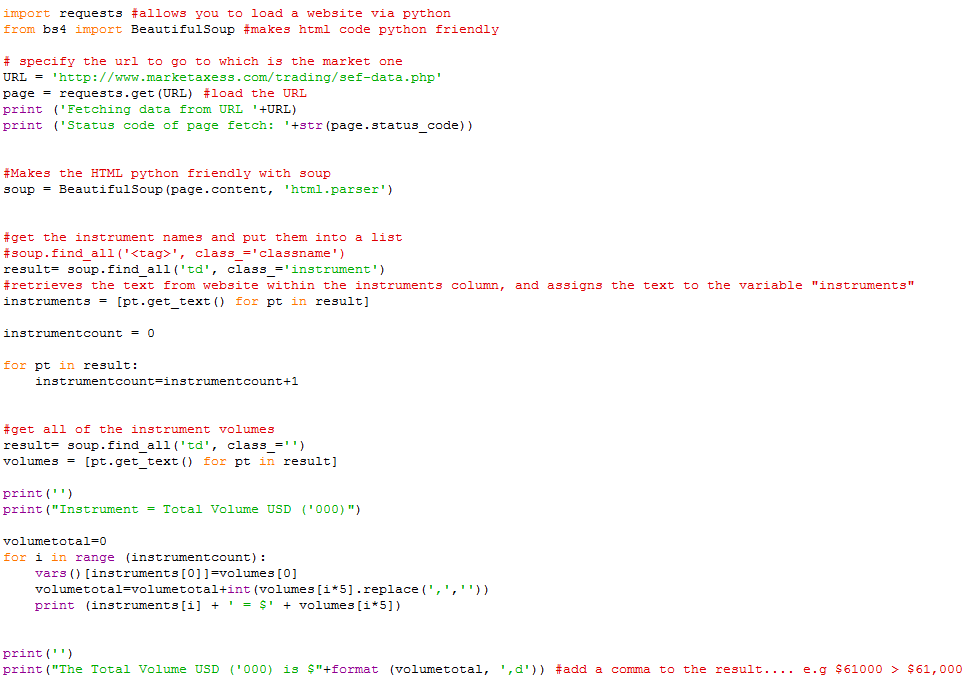
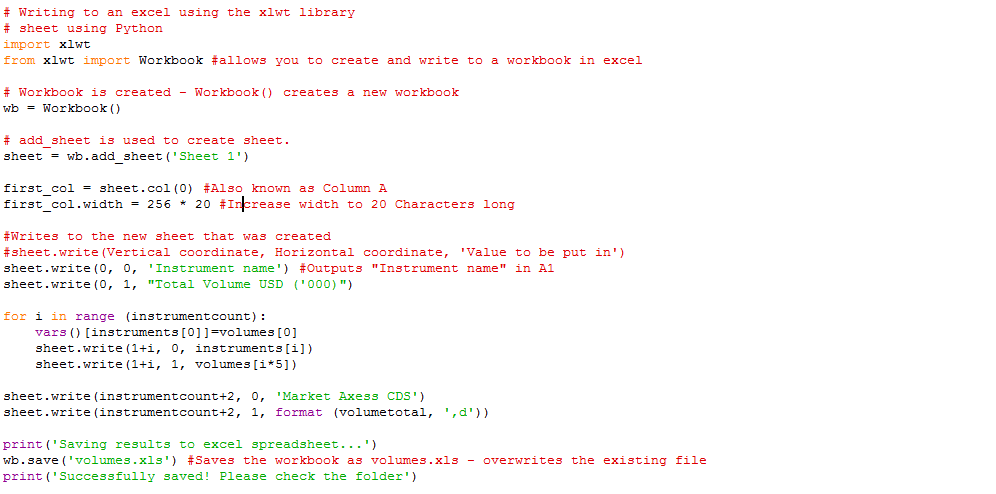
# **Testing**

## Testing the Final Program [Task 1 – ICAP]

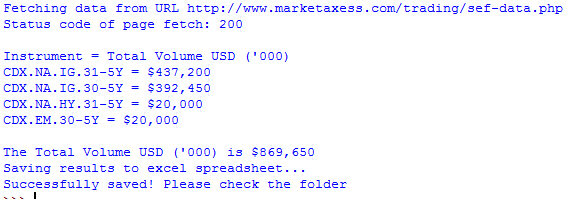
**Outputs:**When the program is run, the following is outputted by the python code.   
A spreadsheet is downloaded and saved in the same location where the python program is run, as shown in the screenshot below:   
  
As seen in the above screenshot, the spreadsheet is formatted correctly. Each piece of information is in its own cell and the total IR is calculated by adding up all values in column Z and deleting the three digits at the end of it.

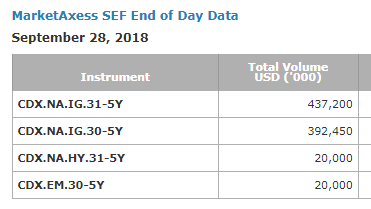
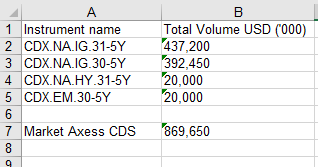
**Test Passed**

## Testing the Final Program [Task 6 – Market Axess]



**Outputs:**When the program is run, the following is outputted by the python code.



  
A spreadsheet is downloaded and saved in the same location where the python program is run, as shown in the screenshot below:As seen in the above screenshots, both the website values and excel values are the same. The spreadsheet in particular outputs each of the instruments volumes in a new cell and then adds the all the volumes, and labels the value “Market Axess CDS”.

**Test Passed**

# **Documentation**

## Minimum Hardware and Software requirements

**Python – Minimum Requirements for Python**

As well as BQUANT. employees are also encouraged to use python when away from the office since both programming platforms use the same methodology. As stated on the python website, below are the minimum requirements needed to run python.

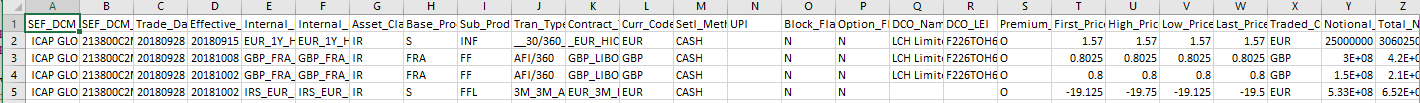
* Processors: Intel Atom® processor or Intel® Core™ i3 processor
* Disk space: 1 GB
* Operating systems: Windows\* 7 or later, macOS, and Linux
* Python\* versions: 2.7.X, 3.6.X
* Included development tools: conda\*, conda-env, Jupyter Notebook\* (IPython)
* Compatible tools: Microsoft Visual Studio\*, PyCharm\*
* Included Python packages: NumPy, SciPy, scikit-learn\*, pandas, Matplotlib, Numba\*, Intel® Threading Building Blocks, pyDAAL, Jupyter, mpi4py, PIP\*, and other

# **Reflection of Code**

## Task 1 – ICAP

Although the code works as expected and is able to sum up the last values in an efficient way there could be some improvements to be made to it:

1. **The width of the columns within the CSV file could be made to adapt to the length of the numbers / text**

When the CSV file is created, some text and numbers are truncated and this could be improved on in the next version of the code to widen them so that the user does not have to manually do it:

1. **The way to add a row to the spreadsheet is very inefficient**

To add the row that contains the result of column Z, it requires a loop using the variable ‘blankspaces’ to keep adding a blank space to the ‘resultsrow’ list until it reaches 24. This could be improved on by finding a better way to add a row within the code for next time.

## Task 6 – MarketAxess

Although the code works as expected and is able to grab the instrument names and volumes from the website there are some improvements that can still be made:

1. **Counting an instrument is not really necessary**

When the instruments are saved to the instruments list, the code does a loop for every result found in the list and results the result under the variable ‘instrumentcount’. However, this is not needed and the code could be made more efficient next time but just doing the length of the instruments list to find out how many instruments there are.

1. **Status codes could be explained to the user**

When the code is run, the user a provided a status code depending on how the website loads. There are so many potential codes that it could be useful to say to the user what the code means for next time.

**Deployment**

When I had completed both Task 1 and 6 in which were assigned to me, I made sure to check with my colleagues to see what stage they were at with their tasks. I helped them all to finish their programs, since it had approved to be more difficult them first thought. Eventually all of the tasks were fully completed and therefore we decided to make the 6 different programs into one single program, since this required less work on Kristof’s side. It took a couple of hours to do this as a group and consequently the single program showed this to Kristof. We did a live demonstration with Kristof as to how to use the program for future reference. He was very impressed and delighted that he only had to upload the spreadsheets that were generated. I made sure that every day for the first week that Kristof was running the program, either myself or one of my colleagues sat with him in order to make sure there were no errors. I am glad to say that this project was a success and the program will be used daily within the Fixed Income trading team.