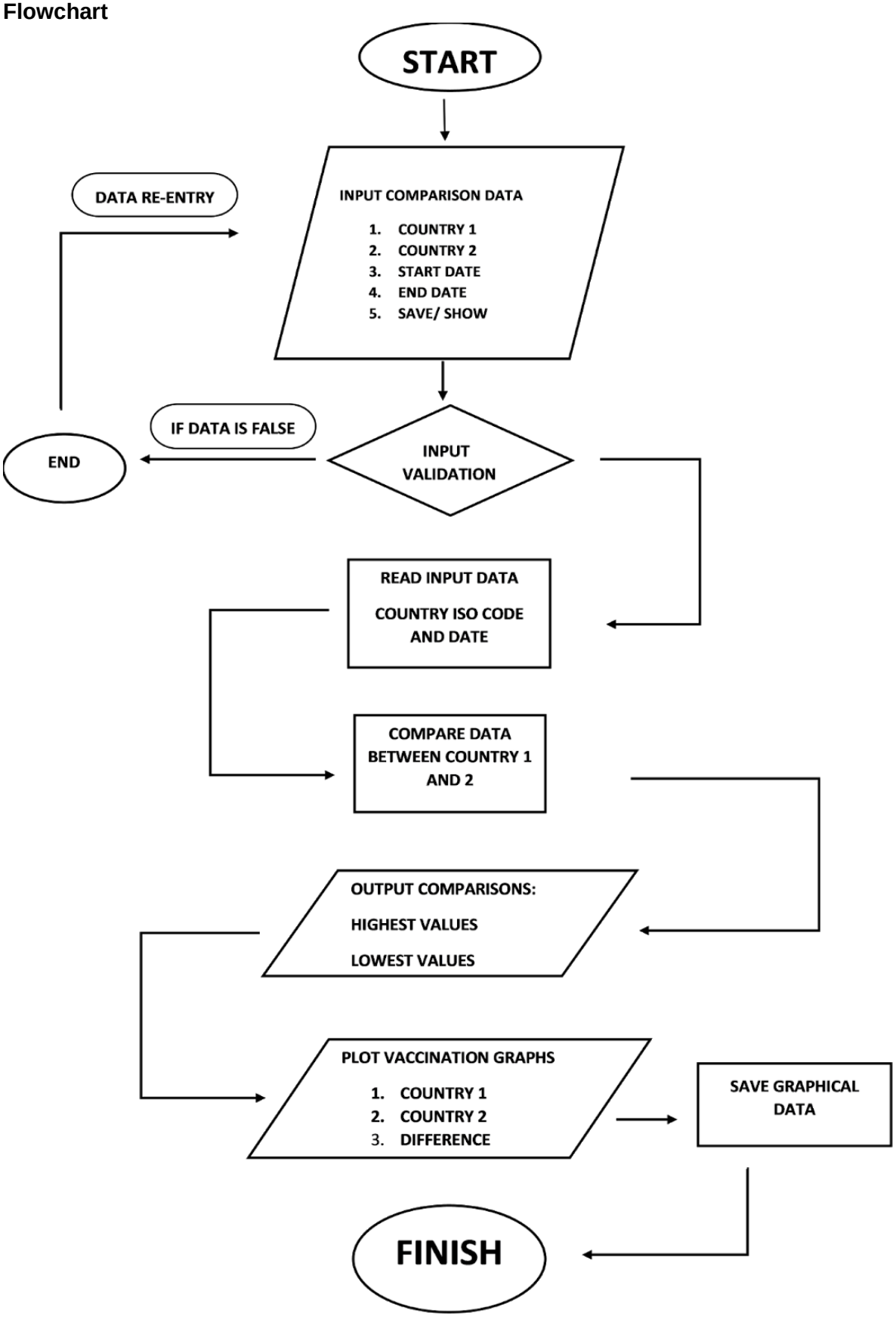
Flowchart


START
INPUT COMPARISON DATA
1. COUNTRY 1
2. COUNTRY 2
3. START DATE
4. END DATE
5. SAVE/ SHOW
DATA RE-ENTRY
IF DATA IS FALSE
INPUT VALIDATION
END
READ INPUT DATA
COUNTRY ISO CODE AND DATE
COMPARE DATA BETWEEN COUNTRY 1 AND 2
OUTPUT COMPARISONS:
HIGHEST VALUES
LOWEST VALUES
PLOT VACCINATION GRAPHS
1. COUNTRY 1
2. COUNTRY 2
3. DIFFERENCE
SAVE GRAPHICAL DATA
FINISH

Command line interface (front-end)

The cli.py script is the main file that imports all other sub-scripts. It includes the Argparse module which handles and stores the arguments given in the terminal by the user such as the iso codes for the countries, the start to end dates and whether to save or show.

```
Paudyal@Paudyal-Hp MINGW64 ~/Programming/Coursework/Group-Coursework-main
$ python cli.py gbr fra 2021-04-01 2021-04-10 show
```

The iso codes are crosschecked which prevent the program from crashing and instead shows an invalid input message.

```
isoCodes= ('TLS', 'GHA', 'OWID_AFR', 'BES', 'DJI', 'PAN', 'ABW', 'MDV', 'PNG', 'SEN', 'BMU', 'MYS', 'AZE', 'IMA', 'FRO', 'HTI',
'CHN', 'GEO', 'NGA', 'TUN', 'NRU', 'SUR', 'EGY', 'MDG', 'MNG', 'SHN', 'NOR', 'OWID_SAM', 'PSE', 'ZMB', 'COD', 'NAM','CYP', 'CHE',
'EST', 'PYF', 'GUY', 'KHM', 'IND', 'ARM', 'SLB', 'DNK', 'ATG', 'BHS', 'CMR', 'AUT', 'MKD', 'OMN', 'CUW','VUT', 'THA', 'BRA', 'MDA',
'SSD', 'ISL', 'POL', 'SOM', 'GBR', 'CAF', 'FRA', 'GRD', 'MSR', 'ITA', 'BFA', 'UGA', 'JPN', 'KGZ', 'LIU', 'OWID_WRL', 'BHR', 'YEM',
'KWT', 'OWID_OCE', 'AUS', 'NIC', 'CRI', 'OWID_NAM', 'PRY', 'ESP', 'TON', 'SYR', 'COL', 'JAM', 'NZL', 'AFG', 'FIN', 'BIH', 'GAB',
'AND', 'MOZ', 'LAO', 'LBY', 'SDN', 'GGY', 'COG', 'GIN', 'TUR', 'VEN', 'GIB', 'BGR', 'TZA', 'BGD', 'ETH', 'BDI', 'BLR', 'BOL',
'SLV', 'MAC', 'PRI', 'WSM', 'DEU', 'JEY', 'TCD', 'MMR', 'SWZ', 'PER', 'ISR', 'HUN', 'LKA', 'ROU', 'TWN', 'MAR', 'NLD', 'COM', 'MLI',
'SGP', 'UKR', 'LBR', 'KOR', 'AGO', 'MCO', 'BTN', 'LIE', 'LUX', 'LSO', 'SXM', 'PAK', 'VNM', 'FLK', 'ERI', 'OWID_EUR', 'HND', 'TTO',
'GNB', 'BEL', 'CHL', 'QAT', 'USA', 'WLF', 'CZE', 'IRQ', 'ZWE', 'TCA', 'CPV', 'TUV', 'RWA', 'ARG', 'HKG', 'DZA', 'OWID_INT', 'AIA', 'IRN',
'SRB', 'DOM', 'ECU', 'KAZ', 'NPL', 'OWID_KOS', 'MRT', 'LBN', 'LCA', 'BLZ', 'SLE', 'FSM', 'MLT', 'CUB', 'MNE', 'RUS', 'MHL', 'SAU', 'SYC',
'SVN', 'ALB', 'BWA', 'JOR', 'BRN', 'HWI', 'SWE', 'CYM', 'TKM', 'PHL', 'GIM', 'OWID_EUN', 'NCL', 'VAT', 'GRL', 'IMN', 'IDN', 'URY',
'OWID_CYN', 'FJI', 'OWID_ASI', 'BEN', 'TGO', 'GNQ', 'GMB', 'GRC', 'STP', 'UZB', 'CAN', 'KNA', 'IRL', 'ARE', 'CIV', 'LVA', 'MUS', 'ZAF',
'CI','NER', 'BRB', 'HRV', 'KEN', 'SMR', 'IJK', 'MEX', 'SVK')
```

```
Paudyal@Paudyal-Hp MINGW64 ~/Programming/Coursework/Group-Coursework-main
$ python cli.py asda asd 2021-04-01 2021-04-10 show
Invalid iso code
```

The main() function houses most of the code, there is also a doc string that highlights the format for the input in the terminal. The program would crash whenever the user failed to provide any arguments thus this was our solution.

```
if len(sys.argv)==1: #if user fails to provide any arguments the program stops
    print("""\nYou must provide 5 positional arguments (2 countries , 2 dates: start+end and save/show)""")
    sys.exit(1)
```

Using Argparse you can set the positional and optional arguments; our program only included positional arguments thus the user is required to provide all the information such as iso codes and dates. There is also an optional help menu which can be accessed using (-h) which instructs the user with the required arguments along with the iso codes and the date format. The parser adds each argument and stores the value temporarily which is then assigned to a variable allowing it to be used elsewhere in the program e.g., Graphing.

```
parser.add_argument("country1",type=str, help="initial of country 1")   #
parser.add_argument("country2",type=str, help="initial of country 2")   #
parser.add_argument("date1",type=str, help="start date")                #positional/mandatory arguments
parser.add_argument("date2",type=str, help="end date")                  #
parser.add_argument("saveOrShow", type=str, help="saves or shows plot") #
args=parser.parse_args()

country1=args.country1.upper() #
country2=args.country2.upper() #
date1=args.date1               #user's inputs stored as variable
date2=args.date2               #
saveOrShow=args.saveOrShow     #
```

Source for Argparse: https://docs.python.org/3/library/argparse.html

All the user's inputs are non-case sensitive thus the program will still run if an input is given, the save variable stores a Boolean value which is used in the graphing script to save or show the plots.

```
if saveOrShow.upper()=="SAVE": #saves or shows plot according to user
    save=True
elif saveOrShow.upper()=="SHOW":
    save=False
else:
    print("Please specify whether to show or save")
    sys.exit(1)
```

Lastly the imported functions from the sub-scripts are executed, fileHandling. getNewData updates the csv file whenever it is run thus the user can input any date in the terminal; this makes the entire program self-sustaining. The data is stored into 4 separate lists containing the 2 sets of x and y values which are then passed through the graphing and comparison functions producing the outputs.

```
fileHandling.getNewData() #updates the data in the csv file at the cost of a small delay (everytime you run the program, the csv file is updat

place1=fileHandling.readCountryData(country1,date1, date2) #retrieves vaccination data between the dates from csv files
place2=fileHandling.readCountryData(country2,date1, date2) #2 lists are created each cointaing the x and y values for the respective country

x1=[i[0] for i in place1] #
y1=[i[1] for i in place1] #x and y values are
x2=[i[0] for i in place2] #seperated into 4 lists
y2=[i[1] for i in place2] #


compare(y1,y2,country1,country2) #compares data between the countries
 reeGraphs(x1,y1,x2,y2, "Time", "Vaccinated People", country1,country2, save) #creates the plots
```

File Handling

Storing and reading the vaccination data was an important part of this project. First, it was necessary to choose a format for the stored data. Using the csv file format was an obvious choice, as it allowed dates and their associated data to be easily stored in the correct order. It was also desirable to have separate files for each county, as this would allow a user to easily create their own file, rather than having to modify any existing ones. This was tested using two testCountry files with a small amount of made-up data. To read the required data, the readCountryData function would look through the file line-by-line between the dates specified by the user. To standardise the naming of each country file, ISO alpha-3 codes were used.

Another important aspect of reading the files was dates. By formatting the dates YYYY-MM-DD, it was easy to sort them into the correct order. However, a function was also needed to check the user input dates. If these were incorrect, it could cause a crash or incorrect output. As dates are quite complex in format, with varying number of days depending on the month for example, the only practical way to check them was using the datetime module. This checkDate function was then included in readCountryData.

```
def readCountryData(country, startDate, endDate): # read data from country-specific csv and return it as a list
    for date in startDate, endDate: # make sure dates are valid before trying to read
        checkDate(date)
    if convertDate(startDate) >= convertDate(endDate):
        print("Error: end date is not later than start date")
        sys.exit(1)
    buildCountryData(country) # build country data as part of reading rather than a separate option
    with open("countriesData/"+country+".csv", "r") as file:
        fileReader = readFile(file)
        dataList = []
        for row in fileReader: # this is too long and messy to fit in a list comprehension
            if convertDate(startDate) <= convertDate(row[0]) <= convertDate(endDate): # with YYYYMMDD format, da
                if row[1] != "": # a lot of countries have gaps in data that have to be skipped
                    dataList.append([row[0], float(row[1])]) # data from the csv file is all floats
        return(dataList)
```

The method of data input was a more difficult problem. Initially, it was thought that the user could manually input data day-by-day to create a new file or add to an existing one. However, this would have been impractical and slow. Instead, the requests module was used to download freely available data collated by Our World in Data (https://covid.ourworldindata.org/data/owid-covid-data.csv) using the getNewData function. This ensured that the latest data was automatically available to the user.

```
def getNewData(): # download the latest data from Our World In Data and save it in the data folder
    newData = requests.get("https://covid.ourworldindata.org/data/owid-covid-data.csv")
    open("countriesData/owid-covid-data.csv", "wb").write(newData.content)
```

As this file is very large, containing a large amount of unnecessary data, it takes a long time to read whenever specific data was needed. Therefore, a separate buildCountryData function was created to gather the specific information and write it to a separate file with writeCountryData, which could then be referenced more quickly in the future. Unfortunately, it was not possible to implement this function separately within the time limit for this project, so buildCountryData was simply included as part of readCountryData.

```
def buildCountryData(country): # take the relevant data from the OWID csv and put it in a country file
    with open("countriesData/owid-covid-data.csv") as file:
        fileReader = readFile(file)
        dataList = [[row[3], row[39]] for row in fileReader if row[0] == country] # 3 is the column for dates,
    writeCountryData(country, dataList)
```

In addition to functions necessary for the programme, some tools were created to help manage the very large csv data file. listColumns printed a list of column titles and their associated numbers, making it easy to find the required data. getCountryCodes was used to create a set of all country ISO codes used in the file, so the user input could be checked against it.

```
def getCountryCodes(): # create a set of country codes for reference
    with open("countriesData/owid-covid-data.csv") as file:
        fileReader = readFile(file)
        countries = set([row[0] for row in fileReader if row[0] != "iso_code"])
        print(countries)
```

Graphing

The Graphing.py file takes 9 inputs; 4 of them are the lists of data to be displaced, 2 of them are the labels for the axis, 2 of them are the abbreviations for the 2 countries chosen by the user and the final input tells the code if the user wants to save the figure as a png file or not.

```
def threeGraphs(A, B, C, D, xAxis, yAxis, country1, country2, save):
```

The number of inputs could be reduced however we wanted to make the file work for different data, this way any data could be used when the function is called, and the code would work smoothly assuming the timescales are the same.

```
difference = []
for i in range(len(B)):
    difference.append(B[i]-D[i])
gs = gridspec.GridSpec(2, 2)
```

Here the code compares the 2 given lists and creates a new list of data.

```
gs = gridspec.GridSpec(2, 2)            #creat


plt.figure()
plt.rcParams['font.size'] = '6'
ax = plt.subplot(gs[0, 0])
plt.plot(A,B,'tab:blue')
plt.grid(b=True, which='major', axis='both')
plt.ylabel(yAxis)
plt.title(country1, loc='center')
plt.xticks(rotation = 90)
```

Here the top line creates a 2 by 2 grid space to make plots in and the chunk of code beneath it shows the code for 1 plot.

```
if save == True:
    fileName = country1+" vs "+country2+".png"
    plt.savefig(fileName, dpi=300)
```

This small if statement decides whether to save the plot or not depending on the input.

Source: https://matplotlib.org/stable/gallery/index.html

Comparison

To begin with, the purpose of the comparison code is to compare the different metrics of data from the two selected countries (country 1 and country2). The data being compared in this code are the cumulative vaccination numbers between country 1 and country 2. It is essential that the numbers in the lists be compared in cumulative form, since this is a logically easier metric to go by. To do the comparison a function called compare (country1, country2, name1, name2) is defined. As indicated the function contains four arguments each with their respective tasks that will be highlighted below.

```
def compare(country1,country2,name1,name2):
    highest1 = country1[0]
    lowest1 = country1[0]
    for item in country1:
        if item > highest1:
            highest1 = item
        if item < lowest1:
            lowest_1 = item
```

Figure 1: code snippet

In the figure above, a for loop is used to iterate through all the vaccination numbers in both lists and hence the highest and lowest numbers are determined and assigned to the variables highest and lowest. After this is done for country1 the same is done for country 2 as indicated by figure 2 below.

```
for item in country2:
    if item > highest2:
        highest2 = item
    if item < lowest2:
        lowest2 = item
```

Figure 2: code snippet

After this was completed, the next task performed by the code is to compare the highest and lowest values in both country 1 and country 2 to establish which of the two had performed better in rolling out the vaccinations as compared to the other. The code to indicate this comparison is indicated below in figure 3.

```
highest = 0
if highest1 > highest2:
    highest = highest1
else:
    highest = highest2
if lowest1 < lowest2:
    lowest = lowest1
else:
    lowest = lowest2

difference=highest-lowest
```

Figure 3: code snippet

In the figure above, the highest and lowest value of both countries are both indicated and assigned to their respective variables. Moreover, the difference between the highest and lowest values is also calculated. This is done to show the contrast between the highest and lowest value and hence portray a vivid number to show the difference. After the metrics are determined, the last and final step is to output the data in a legible format, indicated in figure 4.

```
difference=highest-lowest
print("\n"+name1+" Highest value: ",highest1,
      "Lowest value: ", lowest1,'\n'+name2+" Highest value: ",
      highest2,"Lowest value: ", lowest2,)
print(" \n \nThe total highest value is: ",
      highest," \nThe total lowest value is:"
      ,lowest,
      " \nThe total difference in vaccinations between the dates: ",difference)
```

Figure 4: code snippet

To display the output, the print function is used. In the above figure, the name1 and name2 variables are used to bypass a bug that would cause the result to print just the values instead of indicating the names of the individual countries which would be confusing to the reader. The above output shows the highest and lowest vaccination numbers and indicates the country's iso-code and at the very end it shows the difference between the highest and lowest vaccination numbers.

Outputs with pics