Where in the World is CS

Problem Statement: This program that reads a sequence of lines from stdin and, where possible, converts them into GeoJSON features (with names if given) and writes those to a valid GeoJSON file that contains a *featurecollection*. If the given input that is impossible to interpret this way, e.g., Ha, tricked you! then writes 'Unable to process:' followed by the offending input line. The saved GeoJSON file finally visualized on map.

Technology: The task is solved using the following technologies:

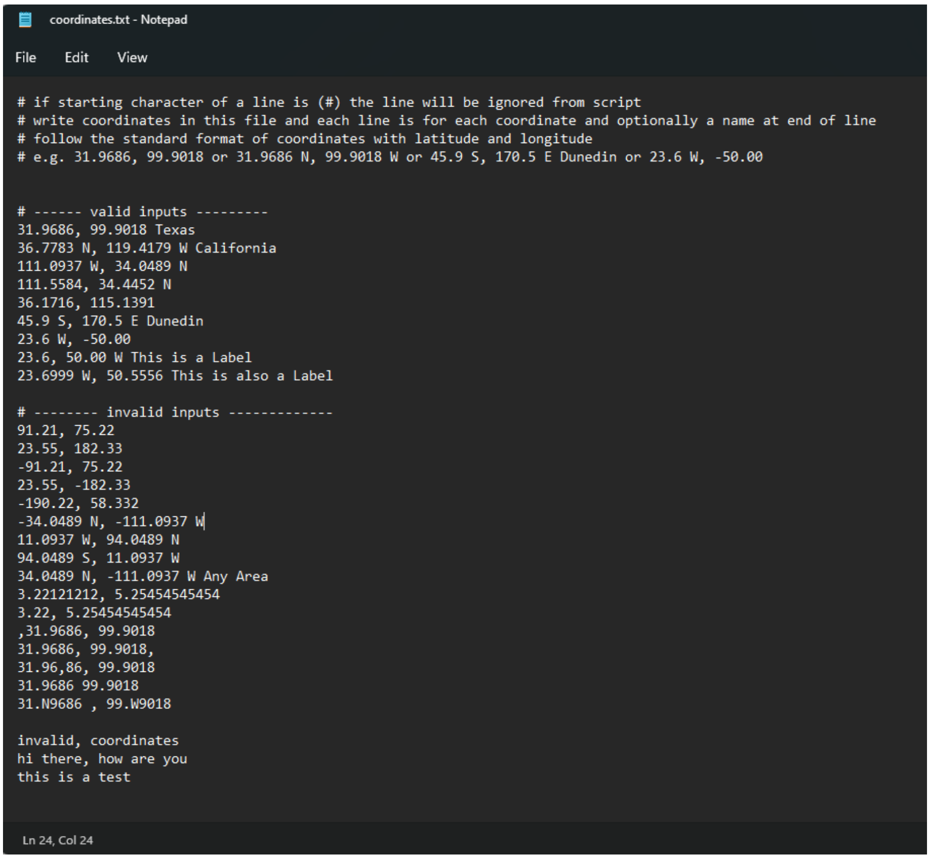
- Anaconda 3
- Python 3.8.8
- Jupyter Notebook
- Notebook Server 6.3.0

With the following python libraries:

- sys
- io
- pandas
- folium

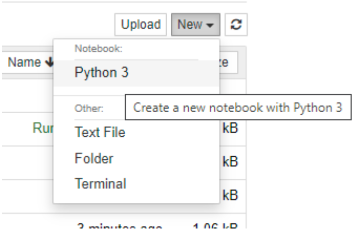
Methodology: A program is written in python to solve the problem with following steps:

Step 1: Create text file 'coordinates.txt' in a folder and provide coordinates in the text file. If the line starts with number sign (#), it is assumed that the line commented and skip the line from taking input. User can input every coordinate in each line.

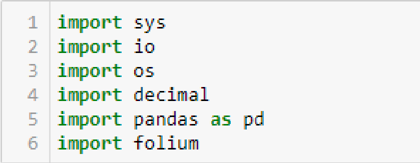


```
# if starting character of a line is (#) the line will be ignored from script
# write coordinates in this file and each line is for each coordinate and optionally a name at end of line
# follow the standard format of coordinates with latitude and longitude
# e.g. 31.9686, 99.9018 or 31.9686 N, 99.9018 W or 45.9 S, 170.5 E Dunedin or 23.6 W, -50.00


# ------ valid inputs ---------
31.9686, 99.9018 Texas
36.7783 N, 119.4179 W California
111.0937 W, 34.0489 N
111.5584, 34.4452 N
36.1716, 115.1391
45.9 S, 170.5 E Dunedin
23.6 W, -50.00
23.6, 50.00 W This is a Label
23.6999 W, 50.5556 This is also a Label

# -------- invalid inputs ------------
91.21, 75.22
23.55, 182.33
-91.21, 75.22
23.55, -182.33
-190.22, 58.332
-34.0489 N, -111.0937 W
11.0937 W, 94.0489 N
94.0489 S, 11.0937 W
34.0489 N, -111.0937 W Any Area
3.22121212, 5.25454545454
3.22, 5.25454545454
,31.9686, 99.9018
31.9686, 99.9018,
31.96,86, 99.9018
31.9686 99.9018
31.N9686 , 99.W9018

invalid, coordinates
hi there, how are you
this is a test
```

Step 2: Create a new notebook with python 3 from jupyter notebook.

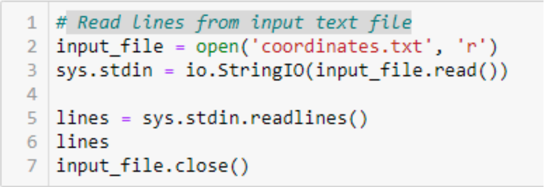


Step 3: Import python libraries in first cell.



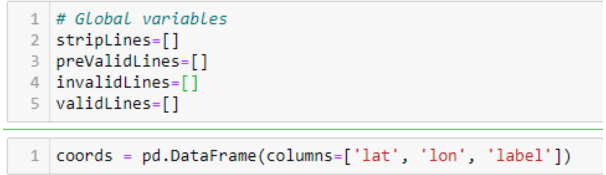
```python
import sys
import io
import os
import decimal
import pandas as pd
import folium
```

Step 4: Read lines from input text file



```python
# Read lines from input text file
input_file = open('coordinates.txt', 'r')
sys.stdin = io.StringIO(input_file.read())

lines = sys.stdin.readlines()
lines
input_file.close()
```

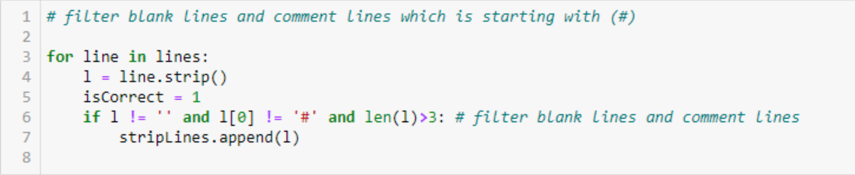
Step 5: Create global variables



```python
# Global variables
stripLines=[]
preValidLines=[]
invalidLines=[]
validLines=[]
```

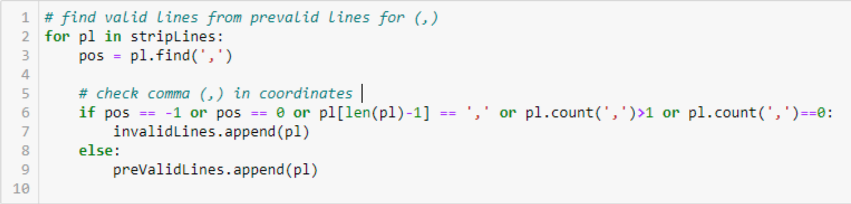
```python
coords = pd.DataFrame(columns=['lat', 'lon', 'label'])
```

Step 6: filter blank lines and comment lines which is starting with (#)



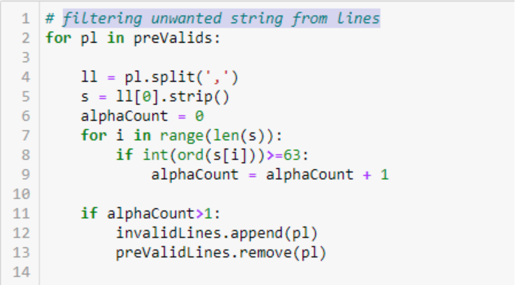
```python
# filter blank lines and comment lines which is starting with (#)

for line in lines:
    l = line.strip()
    isCorrect = 1
    if l != '' and l[0] != '#' and len(l)>3: # filter blank lines and comment lines
        stripLines.append(l)

```

Step 7: Filter invalid comma positions



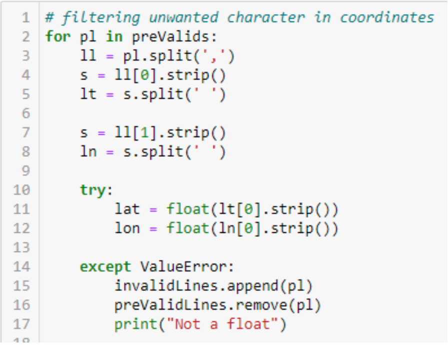
```python
# find valid lines from prevalid lines for (,)
for pl in stripLines:
    pos = pl.find(',')

    # check comma (,) in coordinates
    if pos == -1 or pos == 0 or pl[len(pl)-1] == ',' or pl.count(',')>1 or pl.count(',')==0:
        invalidLines.append(pl)
    else:
        preValidLines.append(pl)

```

Step 8: filter unwanted string from lines



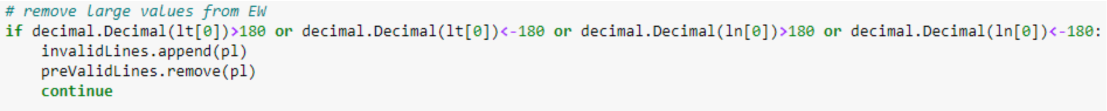
```
# filtering unwanted string from lines
for pl in preValids:

    ll = pl.split(',')
    s = ll[0].strip()
    alphaCount = 0
    for i in range(len(s)):
        if int(ord(s[i]))>=63:
            alphaCount = alphaCount + 1

    if alphaCount>1:
        invalidLines.append(pl)
        preValidLines.remove(pl)

```

Step 9: filter lines for unwanted character in coordinates



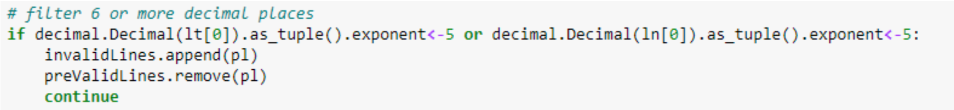
```
# filtering unwanted character in coordinates
for pl in preValids:
    ll = pl.split(',')
    s = ll[0].strip()
    lt = s.split(' ')

    s = ll[1].strip()
    ln = s.split(' ')

    try:
        lat = float(lt[0].strip())
        lon = float(ln[0].strip())

    except ValueError:
        invalidLines.append(pl)
        preValidLines.remove(pl)
        print("Not a float")
```

Step 10: filter large values which is larger than 180 and smaller than -180



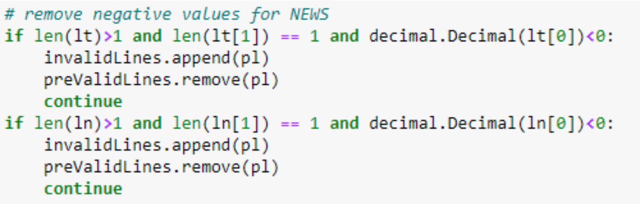
```
# remove large values from EW
if decimal.Decimal(lt[0])>180 or decimal.Decimal(lt[0])<-180 or decimal.Decimal(ln[0])>180 or decimal.Decimal(ln[0])<-180:
    invalidLines.append(pl)
    preValidLines.remove(pl)
    continue
```

Step 11: filter 6 or more decimal places in latitude and longitude



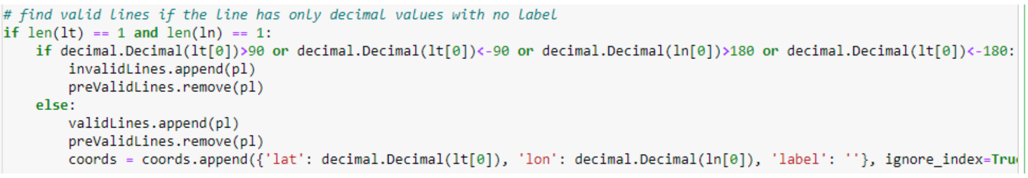
```
# filter 6 or more decimal places
if decimal.Decimal(lt[0]).as_tuple().exponent<-5 or decimal.Decimal(ln[0]).as_tuple().exponent<-5:
    invalidLines.append(pl)
    preValidLines.remove(pl)
    continue
```

Step 12: remove negative values for if N/E/W/S is added with latitude or longitude value



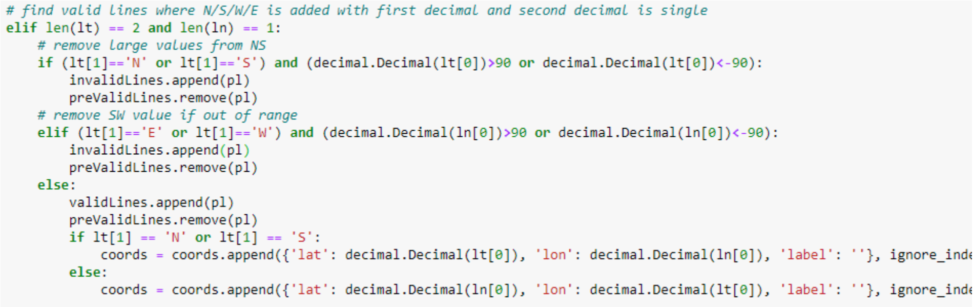
```
# remove negative values for NEWS
if len(lt)>1 and len(lt[1]) == 1 and decimal.Decimal(lt[0])<0:
    invalidLines.append(pl)
    preValidLines.remove(pl)
    continue
if len(ln)>1 and len(ln[1]) == 1 and decimal.Decimal(ln[0])<0:
    invalidLines.append(pl)
    preValidLines.remove(pl)
    continue
```

Step 13: find valid lines if the line has only decimal values with no label



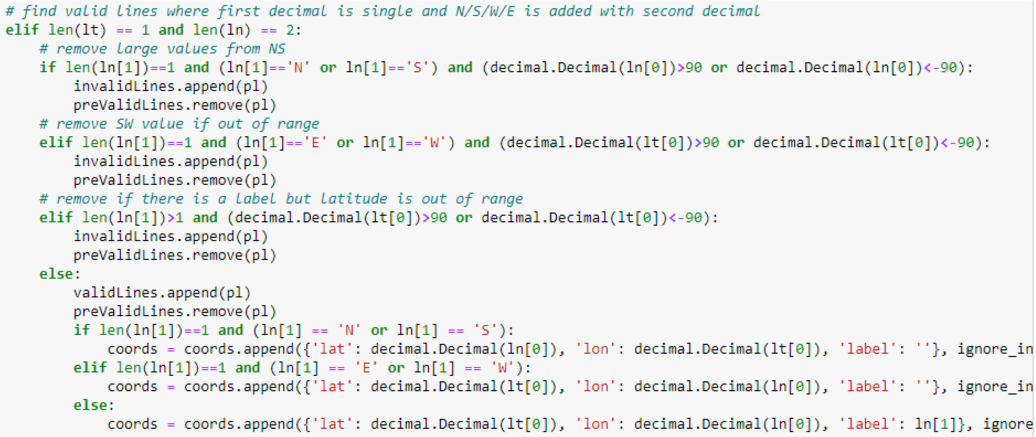
```
# find valid lines if the line has only decimal values with no label
if len(lt) == 1 and len(ln) == 1:
    if decimal.Decimal(lt[0])>90 or decimal.Decimal(lt[0])<-90 or decimal.Decimal(ln[0])>180 or decimal.Decimal(lt[0])<-180:
        invalidLines.append(pl)
        preValidLines.remove(pl)
    else:
        validLines.append(pl)
        preValidLines.remove(pl)
        coords = coords.append({'lat': decimal.Decimal(lt[0]), 'lon': decimal.Decimal(ln[0]), 'label': ''}, ignore_index=Tru
```

Step 14: find valid lines where N/S/W/E is added with first decimal and second decimal is single



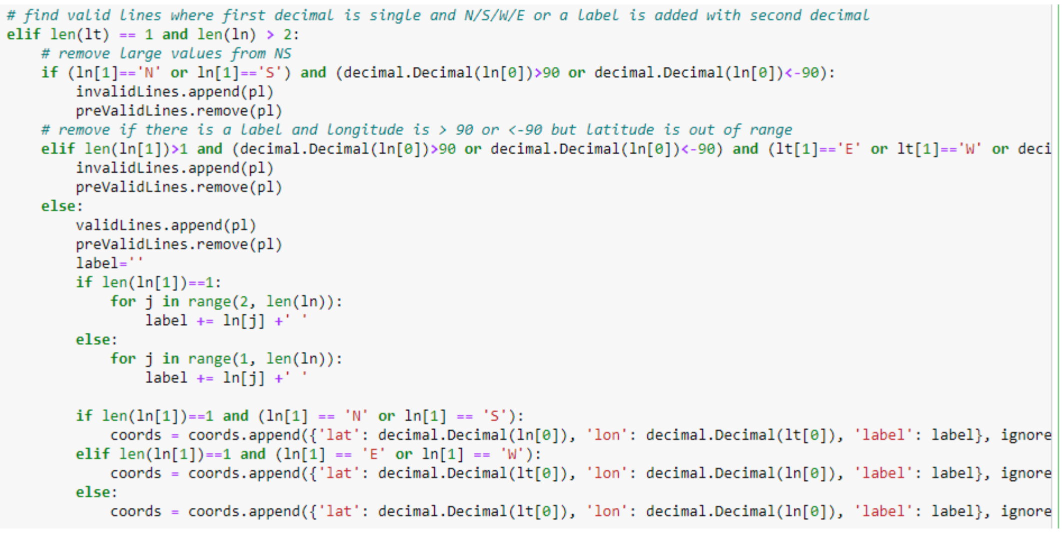
```
# find valid lines where N/S/W/E is added with first decimal and second decimal is single
elif len(lt) == 2 and len(ln) == 1:
    # remove large values from NS
    if (lt[1]=='N' or lt[1]=='S') and (decimal.Decimal(lt[0])>90 or decimal.Decimal(lt[0])<-90):
        invalidLines.append(pl)
        preValidLines.remove(pl)
    # remove SW value if out of range
    elif (lt[1]=='E' or lt[1]=='W') and (decimal.Decimal(ln[0])>90 or decimal.Decimal(ln[0])<-90):
        invalidLines.append(pl)
        preValidLines.remove(pl)
    else:
        validLines.append(pl)
        preValidLines.remove(pl)
        if lt[1] == 'N' or lt[1] == 'S':
            coords = coords.append({'lat': decimal.Decimal(lt[0]), 'lon': decimal.Decimal(ln[0]), 'label': ''}, ignore_inde
        else:
            coords = coords.append({'lat': decimal.Decimal(ln[0]), 'lon': decimal.Decimal(lt[0]), 'label': ''}, ignore_inde
```

Step 15: find valid lines where first decimal is single, and N/S/W/E is added with second decimal



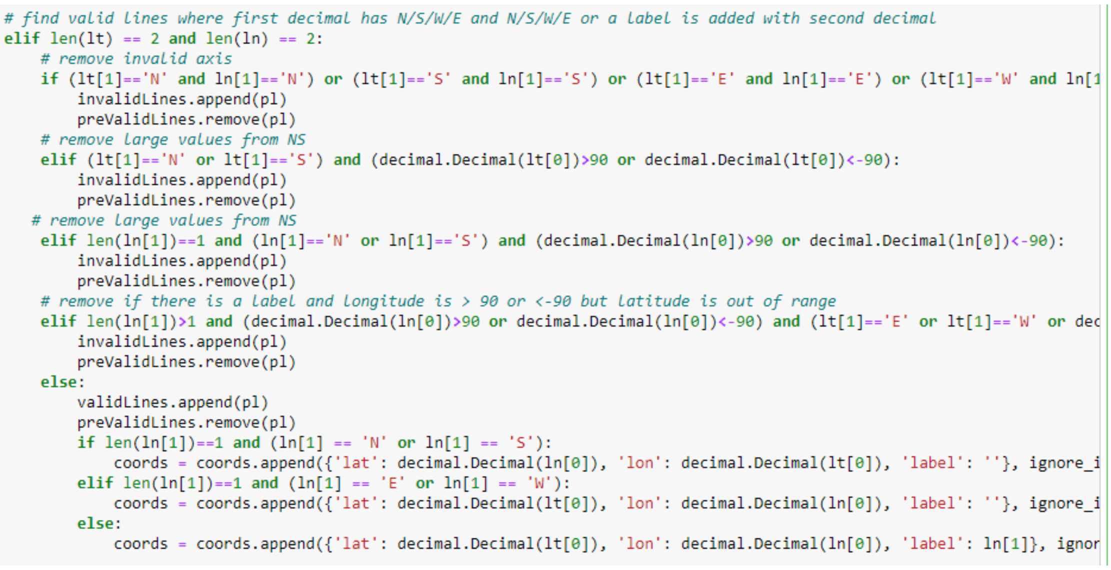
```
# find valid lines where first decimal is single and N/S/W/E is added with second decimal
elif len(lt) == 1 and len(ln) == 2:
    # remove large values from NS
    if len(ln[1])==1 and (ln[1]=='N' or ln[1]=='S') and (decimal.Decimal(ln[0])>90 or decimal.Decimal(ln[0])<-90):
        invalidLines.append(pl)
        preValidLines.remove(pl)
    # remove SW value if out of range
    elif len(ln[1])==1 and (ln[1]=='E' or ln[1]=='W') and (decimal.Decimal(lt[0])>90 or decimal.Decimal(lt[0])<-90):
        invalidLines.append(pl)
        preValidLines.remove(pl)
    # remove if there is a label but latitude is out of range
    elif len(ln[1])>1 and (decimal.Decimal(lt[0])>90 or decimal.Decimal(lt[0])<-90):
        invalidLines.append(pl)
        preValidLines.remove(pl)
    else:
        validLines.append(pl)
        preValidLines.remove(pl)
        if len(ln[1])==1 and (ln[1] == 'N' or ln[1] == 'S'):
            coords = coords.append({'lat': decimal.Decimal(ln[0]), 'lon': decimal.Decimal(lt[0]), 'label': ''}, ignore_in
        elif len(ln[1])==1 and (ln[1] == 'E' or ln[1] == 'W'):
            coords = coords.append({'lat': decimal.Decimal(lt[0]), 'lon': decimal.Decimal(ln[0]), 'label': ''}, ignore_in
        else:
            coords = coords.append({'lat': decimal.Decimal(lt[0]), 'lon': decimal.Decimal(ln[0]), 'label': ln[1]}, ignore
```

Step 16: find valid lines where first decimal is single and N/S/W/E or a label is added with second decimal



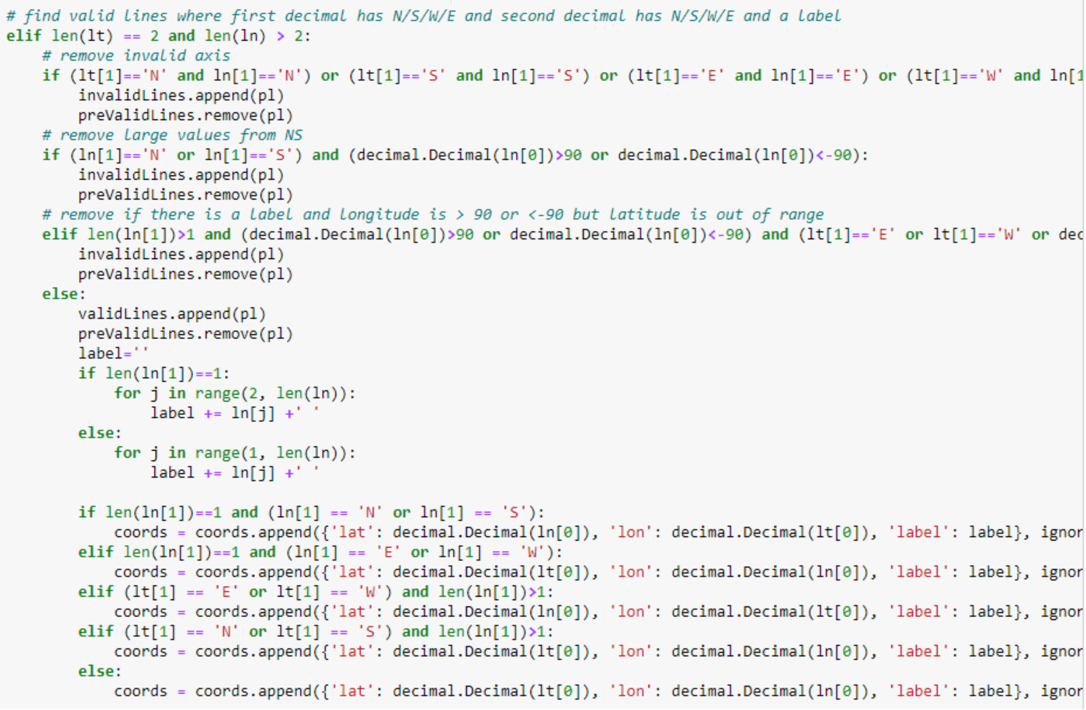
```
# find valid lines where first decimal is single and N/S/W/E or a label is added with second decimal
elif len(lt) == 1 and len(ln) > 2:
    # remove large values from NS
    if (ln[1]=='N' or ln[1]=='S') and (decimal.Decimal(ln[0])>90 or decimal.Decimal(ln[0])<-90):
        invalidLines.append(pl)
        preValidLines.remove(pl)
    # remove if there is a label and longitude is > 90 or <-90 but latitude is out of range
    elif len(ln[1])>1 and (decimal.Decimal(ln[0])>90 or decimal.Decimal(ln[0])<-90) and (lt[1]=='E' or lt[1]=='W' or deci
        invalidLines.append(pl)
        preValidLines.remove(pl)
    else:
        validLines.append(pl)
        preValidLines.remove(pl)
        label=''
        if len(ln[1])==1:
            for j in range(2, len(ln)):
                label += ln[j] +' '
        else:
            for j in range(1, len(ln)):
                label += ln[j] +' '

        if len(ln[1])==1 and (ln[1] == 'N' or ln[1] == 'S'):
            coords = coords.append({'lat': decimal.Decimal(ln[0]), 'lon': decimal.Decimal(lt[0]), 'label': label}, ignore
        elif len(ln[1])==1 and (ln[1] == 'E' or ln[1] == 'W'):
            coords = coords.append({'lat': decimal.Decimal(lt[0]), 'lon': decimal.Decimal(ln[0]), 'label': label}, ignore
        else:
            coords = coords.append({'lat': decimal.Decimal(lt[0]), 'lon': decimal.Decimal(ln[0]), 'label': label}, ignore
```

Step 17: find valid lines where first decimal has N/S/W/E and N/S/W/E or a label is added with second decimal



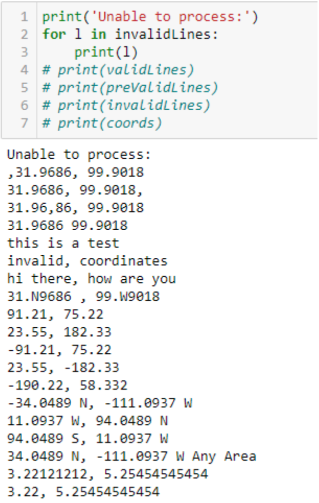
```
# find valid lines where first decimal has N/S/W/E and N/S/W/E or a label is added with second decimal
elif len(lt) == 2 and len(ln) == 2:
    # remove invalid axis
    if (lt[1]=='N' and ln[1]=='N') or (lt[1]=='S' and ln[1]=='S') or (lt[1]=='E' and ln[1]=='E') or (lt[1]=='W' and ln[1
        invalidLines.append(pl)
        preValidLines.remove(pl)
    # remove large values from NS
    elif (lt[1]=='N' or lt[1]=='S') and (decimal.Decimal(lt[0])>90 or decimal.Decimal(lt[0])<-90):
        invalidLines.append(pl)
        preValidLines.remove(pl)
   # remove large values from NS
    elif len(ln[1])==1 and (ln[1]=='N' or ln[1]=='S') and (decimal.Decimal(ln[0])>90 or decimal.Decimal(ln[0])<-90):
        invalidLines.append(pl)
        preValidLines.remove(pl)
    # remove if there is a label and longitude is > 90 or <-90 but latitude is out of range
    elif len(ln[1])>1 and (decimal.Decimal(ln[0])>90 or decimal.Decimal(ln[0])<-90) and (lt[1]=='E' or lt[1]=='W' or dec
        invalidLines.append(pl)
        preValidLines.remove(pl)
    else:
        validLines.append(pl)
        preValidLines.remove(pl)
        if len(ln[1])==1 and (ln[1] == 'N' or ln[1] == 'S'):
            coords = coords.append({'lat': decimal.Decimal(ln[0]), 'lon': decimal.Decimal(lt[0]), 'label': ''}, ignore_i
        elif len(ln[1])==1 and (ln[1] == 'E' or ln[1] == 'W'):
            coords = coords.append({'lat': decimal.Decimal(lt[0]), 'lon': decimal.Decimal(ln[0]), 'label': ''}, ignore_i
        else:
            coords = coords.append({'lat': decimal.Decimal(lt[0]), 'lon': decimal.Decimal(ln[0]), 'label': ln[1]}, ignor
```

Step 18: find valid lines where first decimal has N/S/W/E and second decimal has N/S/W/E and a label



```
# find valid lines where first decimal has N/S/W/E and second decimal has N/S/W/E and a label
elif len(lt) == 2 and len(ln) > 2:
    # remove invalid axis
    if (lt[1]=='N' and ln[1]=='N') or (lt[1]=='S' and ln[1]=='S') or (lt[1]=='E' and ln[1]=='E') or (lt[1]=='W' and ln[1
        invalidLines.append(pl)
        preValidLines.remove(pl)
    # remove large values from NS
    if (ln[1]=='N' or ln[1]=='S') and (decimal.Decimal(ln[0])>90 or decimal.Decimal(ln[0])<-90):
        invalidLines.append(pl)
        preValidLines.remove(pl)
    # remove if there is a label and longitude is > 90 or <-90 but latitude is out of range
    elif len(ln[1])>1 and (decimal.Decimal(ln[0])>90 or decimal.Decimal(ln[0])<-90) and (lt[1]=='E' or lt[1]=='W' or dec
        invalidLines.append(pl)
        preValidLines.remove(pl)
    else:
        validLines.append(pl)
        preValidLines.remove(pl)
        label=''
        if len(ln[1])==1:
            for j in range(2, len(ln)):
                label += ln[j] +' '
        else:
            for j in range(1, len(ln)):
                label += ln[j] +' '

        if len(ln[1])==1 and (ln[1] == 'N' or ln[1] == 'S'):
            coords = coords.append({'lat': decimal.Decimal(ln[0]), 'lon': decimal.Decimal(lt[0]), 'label': label}, ignor
        elif len(ln[1])==1 and (ln[1] == 'E' or ln[1] == 'W'):
            coords = coords.append({'lat': decimal.Decimal(lt[0]), 'lon': decimal.Decimal(ln[0]), 'label': label}, ignor
        elif (lt[1] == 'E' or lt[1] == 'W') and len(ln[1])>1:
            coords = coords.append({'lat': decimal.Decimal(ln[0]), 'lon': decimal.Decimal(lt[0]), 'label': label}, ignor
        elif (lt[1] == 'N' or lt[1] == 'S') and len(ln[1])>1:
            coords = coords.append({'lat': decimal.Decimal(lt[0]), 'lon': decimal.Decimal(ln[0]), 'label': label}, ignor
        else:
            coords = coords.append({'lat': decimal.Decimal(lt[0]), 'lon': decimal.Decimal(ln[0]), 'label': label}, ignor
```

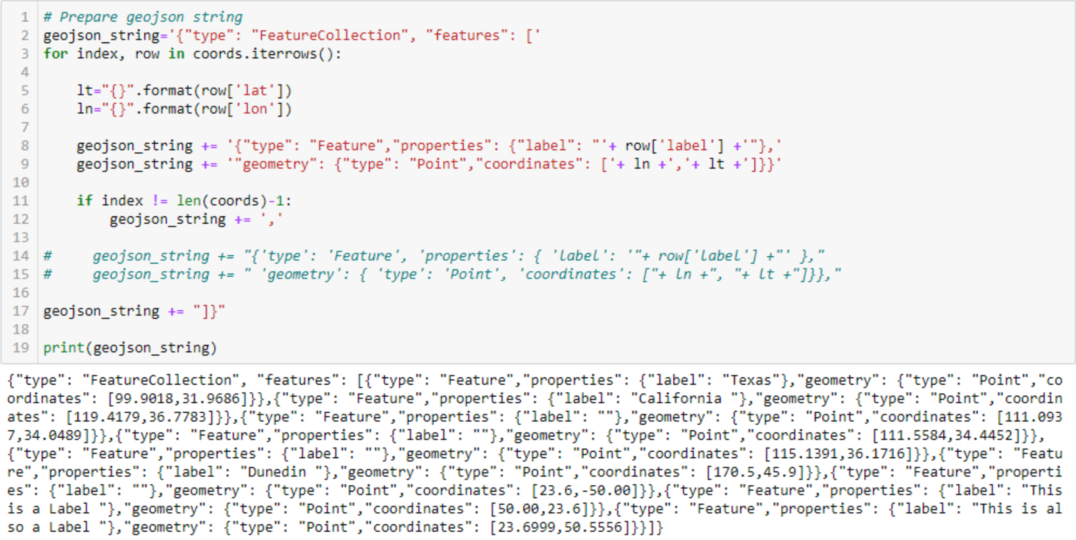
Step 19: Print where the lines are unable to process



```
print('Unable to process:')
for l in invalidLines:
    print(l)
# print(validLines)
# print(preValidLines)
# print(invalidLines)
# print(coords)
```

```
Unable to process:
,31.9686, 99.9018
31.9686, 99.9018,
31.96,86, 99.9018
31.9686 99.9018
this is a test
invalid, coordinates
hi there, how are you
31.N9686 , 99.W9018
91.21, 75.22
23.55, 182.33
-91.21, 75.22
23.55, -182.33
-190.22, 58.332
-34.0489 N, -111.0937 W
11.0937 W, 94.0489 N
94.0489 S, 11.0937 W
34.0489 N, -111.0937 W Any Area
3.22121212, 5.25454545454
3.22, 5.25454545454
```

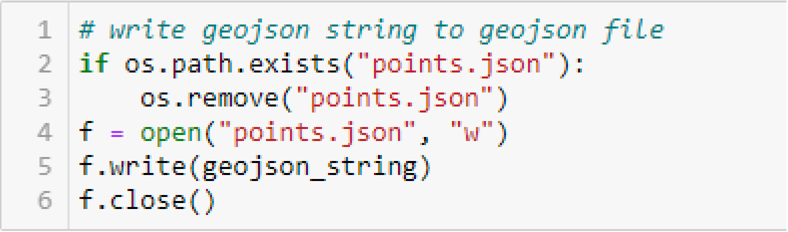
Step 20: Prepare GeoJSON file in string



```
# Prepare geojson string
geojson_string='{"type": "FeatureCollection", "features": ['
for index, row in coords.iterrows():

    lt="{}".format(row['lat'])
    ln="{}".format(row['lon'])

    geojson_string += '{"type": "Feature","properties": {"label": "'+ row['label'] +'"},'
    geojson_string += '"geometry": {"type": "Point","coordinates": ['+ ln +','+ lt +']}}'

    if index != len(coords)-1:
        geojson_string += ','

#     geojson_string += "{'type': 'Feature', 'properties': { 'label': '"+ row['label'] +"' },"
#     geojson_string += " 'geometry': { 'type': 'Point', 'coordinates': ["+ ln +", "+ lt +"]}},"

geojson_string += "]}"

print(geojson_string)
```

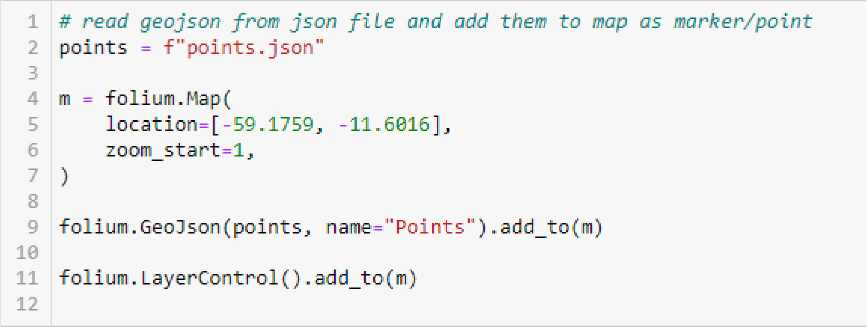
```
{"type": "FeatureCollection", "features": [{"type": "Feature","properties": {"label": "Texas"},"geometry": {"type": "Point","co
ordinates": [99.9018,31.9686]}},{"type": "Feature","properties": {"label": "California "},"geometry": {"type": "Point","coordin
ates": [119.4179,36.7783]}},{"type": "Feature","properties": {"label": ""},"geometry": {"type": "Point","coordinates": [111.093
7,34.0489]}},{"type": "Feature","properties": {"label": ""},"geometry": {"type": "Point","coordinates": [111.5584,34.4452]}},
{"type": "Feature","properties": {"label": ""},"geometry": {"type": "Point","coordinates": [115.1391,36.1716]}},{"type": "Featu
re","properties": {"label": "Dunedin "},"geometry": {"type": "Point","coordinates": [170.5,45.9]}},{"type": "Feature","properti
es": {"label": ""},"geometry": {"type": "Point","coordinates": [23.6,-50.00]}},{"type": "Feature","properties": {"label": "This
is a Label "},"geometry": {"type": "Point","coordinates": [50.00,23.6]}},{"type": "Feature","properties": {"label": "This is al
so a Label "},"geometry": {"type": "Point","coordinates": [23.6999,50.5556]}}]}
```

Step 21: Write GeoJSON string to a json file



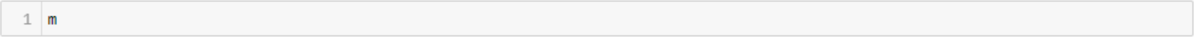
```
# write geojson string to geojson file
if os.path.exists("points.json"):
    os.remove("points.json")
f = open("points.json", "w")
f.write(geojson_string)
f.close()
```

Step 22: read geojson from json file and add them to map as marker/point

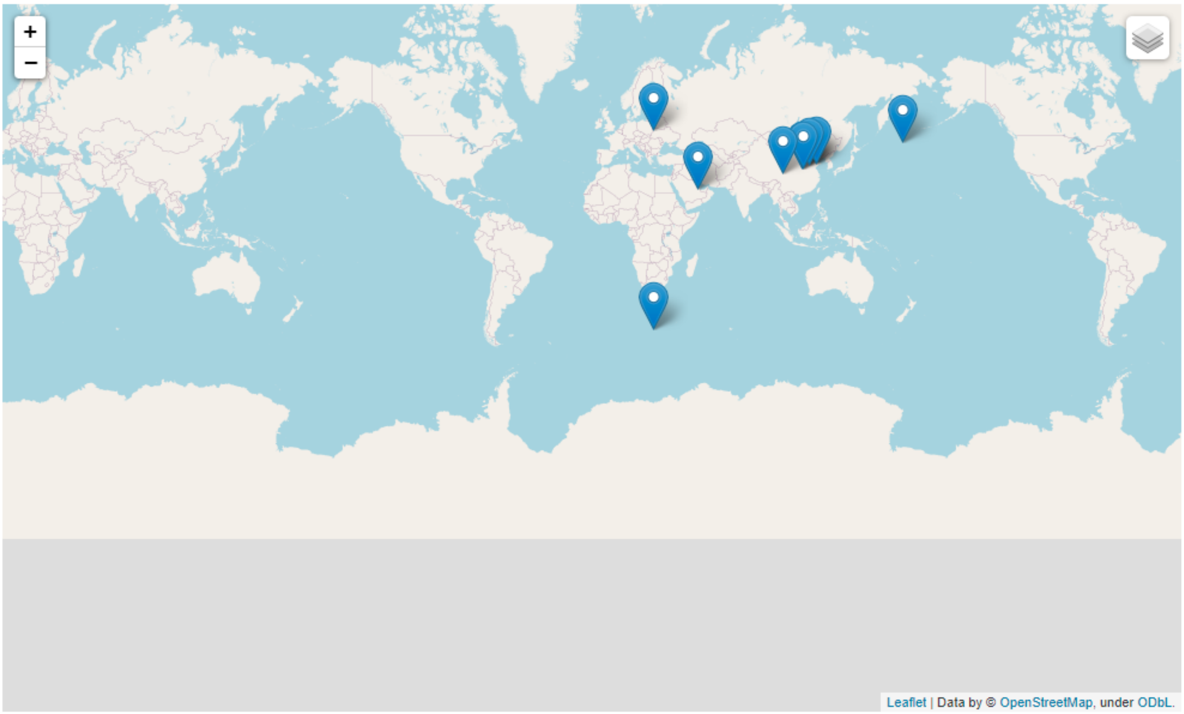


```
# read geojson from json file and add them to map as marker/point
points = f"points.json"

m = folium.Map(
    location=[-59.1759, -11.6016],
    zoom_start=1,
)

folium.GeoJson(points, name="Points").add_to(m)

folium.LayerControl().add_to(m)

```

Step 23: finally print the map



```
m
```



Validation: Finally compared valid lines from text file, script after filtering the invalid lines and from the map and found that the lines from input file are translated to valid coordinates, invalid lines are filtered, geojson file has created and finally points on map are in their correct position.