

# Systems Development 1

# **Programming Project**

---

Lucy Craddock, Beth Harper, Jaime Viegas, and Felipe Warrener  
Due: 12th January, 2018

# Introduction

<b>Introduction</b>	<b>1</b>
<b>Collaboration</b>	<b>5</b>
Coding Standard	5
Tools	5
<b>Output</b>	<b>7</b>
<b>Unit testing</b>	<b>8</b>
Bounding_box.py	8
Centre_point.py	9
Crimes_in_box.py	9
Csv_save.py	9
Geodist.py	10
Get_files.py	11
Get_headings.py	11
Month_list.py	11
Plot_histo.py	12
Validate_date.py	13
Validate_postcode.py	13
Write_csv.py	14
<b>Research</b>	<b>15</b>
Template Reports	15
Web Scraping	15
<b>Limitations</b>	<b>16</b>
User Input	16
File Handling	16

This Hardcopy is used to show how unit testing was implemented into our programming project and how we collaborated on the work produced.

**Note:** The original crime data provided was only the months of 2016. To increase the complexity of our project, we included all the months from January 2016 to November of 2017. These files were found on data.police.uk and have the same format as the original dataset provided.

Date range: January 2017 ▼ to November 2017 ▼

Forces:

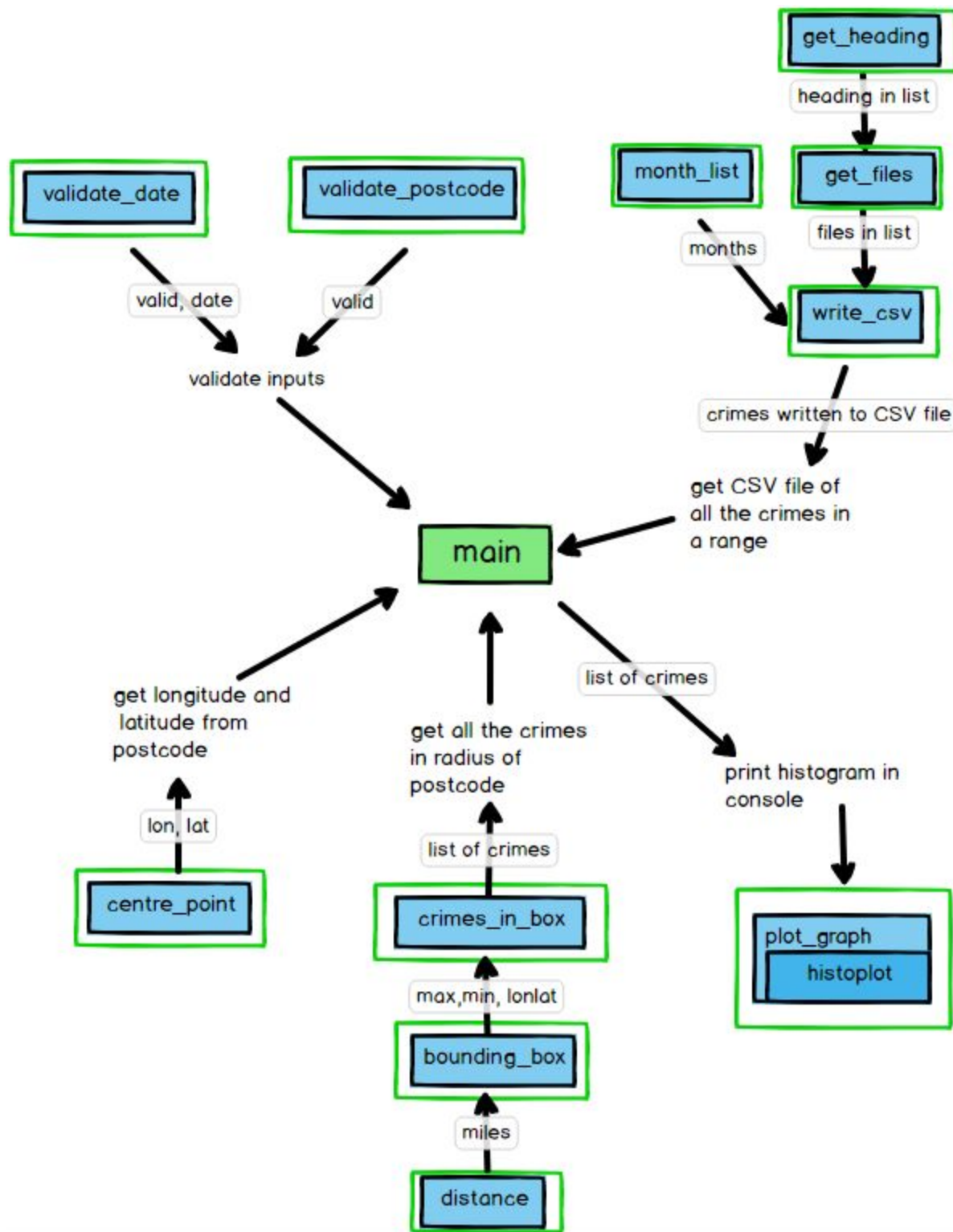
- ☐ All forces
- ☐ Avon and Somerset Constabulary
- ☐ Bedfordshire Police
- ☐ British Transport Police
- ☐ Cambridgeshire Constabulary
- ☐ Cheshire Constabulary
- ☐ City of London Police
- ☐ Cleveland Police
- ☐ Cumbria Constabulary
- ☐ Derbyshire Constabulary
- ☒ Devon & Cornwall Police
- ☐ Dorset Police
- ☐ Durham Constabulary

Data sets: ☒ Include crime data

☐ Include outcomes data

The crime CSV files contain the latest outcome category, but this option will generate a separate set of files with the outcome history for all crimes. Outcomes data is not available for BTP or PSNI.

☐ Include stop and search data



## Collaboration

We identified abilities and strengths in the group to base tasks for the project:

**Lucy Craddock:** Previous experience with Python and with a project similar to this assigned. This was creating an alarm when a boat moved too far away from it's anchor - involving GPS and calculating great-circle distance. Created the initial bulk of code such as reading/writing to files, and overall basic functionality of the project. I also helped with the overall collaboration of the project. This involved passing files to other team members to review and test.

**Beth Harper:** Previous experience with Python, Trello and GitHub. Wrote some of initial code, and worked to improve initial program code written by Lucy, including code review and some unit testing. Managed Trello board and GitHub repository.

**Jaime Viegas:** Reviewing modules as per spec and unit testing.

**Felipe Warrenner:** Previous experience in coding but not with Python. Reviewing modules as per spec, handling exceptions and unit testing.

We will also each be submitting a document individually, outlining the work we have done and our contributions towards the project.

## Coding Standard

We used a linter (pylint) to keep the code clean and consistent to ease collaboration in the group. This automatically checks for bad indentation, docstrings were added, appropriate whitespace in expressions and statements, and consistent naming conventions.

## Tools

We have used Trello (<https://trello.com/b/b9no0Oxc>) and Github (<https://github.com/clucy10/Python-Assessment>) to collaborate and manage our project work. We used Trello to assign tasks to team members, comment on work and share the assignment resources such as this Hardcopy, the Specification, etc. We also used Github to work on code, using two branches '*master*' and '*Develop-(basic)*'. When working on code that was yet to be tested/reviewed, we pushed to '*Develop-(basic)*' and once finished, we moved these files to '*master*'. This, in conjunction with our Trello board, allowed us to keep on top of what bits of work had been completed, and what was yet to be done.

## Output

The program produces a table and histogram. The user also has a choice of saving a CSV file of data used for the charts.

An example of the histogram chart output for our program is shown below.

EX4 4PT has a total of 1349 crimes

Anti-social behaviour	:	*****	(41.7%)
Violence and sexual offences	:	*****	(16.5%)
Shoplifting	:	*****	(10.3%)
Criminal damage and arson	:	*****	(7.0%)
Other theft	:	*****	(6.8%)
Public order	:	****	(4.3%)
Burglary	:	***	(3.6%)
Drugs	:	***	(3.3%)
Vehicle crime	:	**	(2.0%)
Theft from the person	:	*	(1.6%)
Bicycle theft	:	*	(1.5%)
Possession of weapons	:	*	(0.7%)
Other crime	:	*	(0.6%)
Robbery	:		(0.2%)
Quit? (Y/N)	:		

## Unit testing

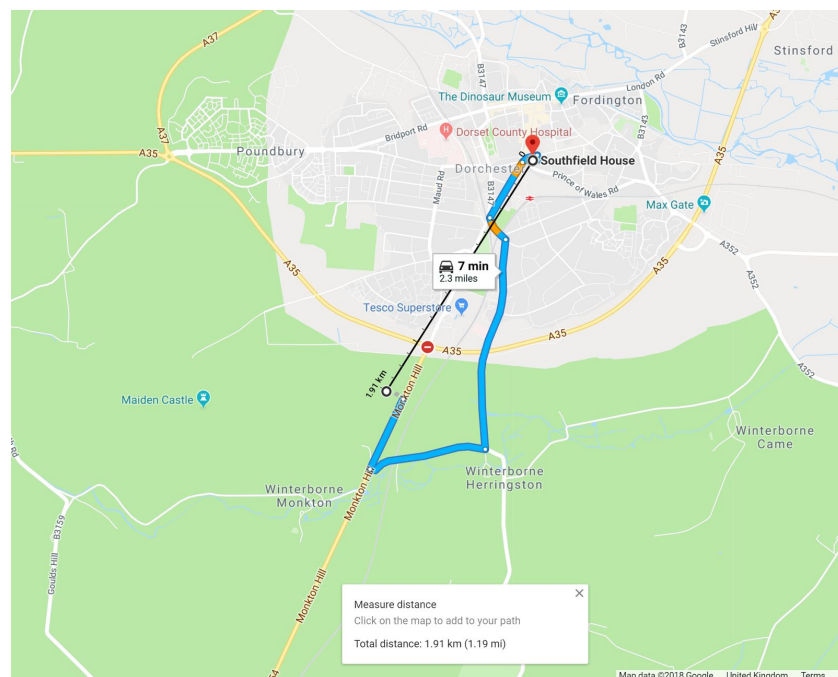
### Bounding\_box.py

#### Purpose

This function's purpose is to reduce the amount of times the geodist.py function is used. This was needed as once we added the rest of the months of the year, we felt that the program took too long to run. To reduce this, we created a bounding box that will take the centerpoint of a postcode and return the minimum and maximum of the latitude and longitude by adding the radius in degrees. This is not very accurate so some leeway is given but it means instead of calculating the great-circle distance for 20,000 crimes in the Southwest but 1000 for that area. This makes the program easily scalable.

#### Testing

The bounding box was tested by printed the results of the function and putting them into the assert. The parameters were chosen from random postcodes so it would only test in the South West. It was then tested on google maps and as you can see below, the minimum longitude and latitude is give or take 1 mile. Some leeway was given because the maths used was very simplified.



## Centre\_point.py

### Purpose

This module contains the function `centre_point` that returns the latitude and longitude for a given postcode. This is done by searching the csv file and returning the floats to calculate crimes in a given radius.

### Testing

To test this file, several assert statements were created to check that a given postcode returns the correct longitude and latitude from the `postcodes.csv` file. If the function does not do this correctly, an `assertionError` is raised and we are able to identify that there is a bug in the code.

```
34 # UNIT TESTING
35 if __name__ == "__main__":
36
37     assert centre_point("EX1 3PB", "postcodes.csv") == (50.72761138, -3.47565992)
38     assert centre_point("EX167BX", "postcodes.csv") == (50.92217857, -3.37649249)
39     assert centre_point("EX4 3SR", "postcodes.csv") == (50.72674566, -3.53502873)
40     assert (centre_point("EX2 5DW", "postcodes.csv")) == (50.7166919, -3.50669422)
41     assert (centre_point("EX4 6PX", "postcodes.csv")) == (50.73009844, -3.52075589)
42
43     print("All tests ran successfully!")
44
45
```

## Crimes\_in\_box.py

### Purpose

This function returns a list of crimes within the centrepoint's radius. This is done by filtering the crimes by using the `bounding_box.py` function and then using the `geodist.py` function to find the crimes more accurately.

### Testing

Since the function returns a list of around 800 crimes, we decided to check whether or not if the list was empty. This means that it will throw an error if the function fails to find any crimes. This was done using a function `empty_list` that returns false if the list is empty, causing the console to show an assertion error.

```
i = []
assert empty_list(i)
assert empty_list(i)
AssertionError
```



When tested with several postcode centre points and different radiuses, the file does not throw up an error.

```
<terminated> crimes_in_box.py [C:\Users\lucy.craddock\AppData\Local\Programs\Python\Python36\python.exe]  
testing crimes in radius
```

## Csv\_save.py

### Purpose

This function writes to a CSV file named by postcode and dates specified. It is the filtered list of crimes in the radius and is our 'tabular output'. It allows users to see the crimes in the area in more detail with the street and outcome of the crime. It is optional for the user.

## Geodist.py

**Requirements:** This module is used to calculate the distance between the centrepoint of inputted postcode and the geoposition of street-level crime. It is called in the crime\_in\_radius.py module to check the crimes within a given radius of 1, 2 or 5 miles. In order to do this, the code was edited to return miles instead of kilometres.

### Testing

Tests were carried out in the South West since this is where the project would be used. This meant it was not necessary to test extraneous data.



Using google maps, I determined the distance of two points and rounded the function to get an

approximate result of 8 and this was repeated for more tests. It also tested the accuracy with a result of 0.5 miles which returned 1 mile when rounded. When run, the module does not throw an error, showing that it is working as it should be.

```
<terminated> geodist.py [C:\Users\lucy.craddock\AppData\Local\Programs\Python\Python36\python.exe]  
testing geodist
```

## Get\_files.py

### Purpose

This function returns a list of all the crimes for the months specified by user. This uses the headings function to get the first row and to avoid this line being repeated multiple times in the list. It then uses the month\_list function to find the csv file for each month and compile them together. This list is then returned.

### Testing

Since this function returns a list of more than 250,000, we decided to use the same function as for the crimes\_in\_box. It checks whether the list returned is empty.

```
C:\Users\Lucy\Documents\ass\compile_csv>python get_files.py  
testing get_files  
test has run successfully
```

## Get\_headings.py

### Purpose

This function takes the first file from the list of months and returns a list of the column headings in the file. This is done to avoid the list repeated the heading for every file used to compile the list of crimes.

### Testing

Since this function only returns one row of the file, we can set stricter checks for the format of the list. For example whether it returns an empty list, whether that list contains one item (a row is recognised as a string and therefore one element in the list).



## Month\_list.py

### Purpose

This function returns a list of months from the start month to the end month. For example if the user wants to find all the crimes between April and December of 2017, it will return a list of months in strings. This is then used to find the CSV files to compile and filter for those in the area.

### Testing

To test this function, assert statements are used to check that the correct list of months are returned for different date combinations.

## Plot\_histo.py

### Purpose

This function prints a histogram to the console. It shows the number of crimes in the area for the time given and the percentage of each crime by type. This will show the user how much crime is in the area and whether crimes they may be most interested in, such as drugs or violence, for people with children.

```

main.py [C:\Users\lucy.craddock\AppData\Local\Programs\Python\Python36\python.exe]
Enter a postcode: EX4 4QJ
Please choose a 1, 2 or 5 mile radius: 1
Enter a start date in the format YYYY-MM: 2016-01
Enter an end date in the format YYYY-MM: 2016-04
EX4 4QJ has a total of 4441 crimes

Anti-social behaviour      : ***** (39.9%)
Violence and sexual offences : ***** (17.2%)
Shoplifting                 : ***** (10.8%)
Criminal damage and arson  : ***** (6.9%)
Other theft                 : ***** (6.8%)
Public order                : **** (4.3%)
Drugs                      : *** (3.3%)
Burglary                   : *** (3.2%)
Vehicle crime               : ** (2.2%)
Bicycle theft               : ** (2.1%)
Theft from the person       : * (1.4%)
Other crime                  : * (1.0%)
Possession of weapons       : (0.6%)
Robbery                     : (0.2%)
Quit? (Y/N)

```

## Testing

The `plot_histo` is a void function. It returns no value therefore we cannot use any assert functions. Since this function prints to the console, we argue this itself is the 'test'. Anything that could be tested such as the data would be tested in other functions.

## Validate\_date.py

### Purpose

This function returns a boolean value, depending on whether a given date is in a valid format. For this program, the valid format required is YYYY-MM. However, this also has a degree of flexibility, as if the user were to enter a date such as "201601" or "20161" then the validation function will recognise it, and still consider it to be a valid date.

To do this, the function removes the hyphen (or space) in the middle of the date if applicable, determines whether the month part of the input is one or two characters in length (if one, then a '0' is added in front of the month) and then evaluates the validity. If the date is valid, it is returned in the correct format for use elsewhere in the program, i.e. 'YYYY-MM'.

This function is important, as the date entered by the user must be valid in order to find the correct set of crime data from the csv files.



## Testing

To test this function, there are two lists: valid format dates and invalid format dates. Iterating through each of these lists, the dates are fed in to the function to ensure that only the valid format ones return a boolean value of True. Once a date has been tested and its assertions are deemed true, the unit test will print “All tests ran successfully!”.

## Validate\_postcode.py

### Purpose

Similar to the `validate_date(...)` function, this function returns a boolean value depending on whether a given postcode is valid or not, and is important since the postcode must be valid to be able to find it, and its surrounding locations, within the crime data. As there are several different formats for postcodes, the function is designed so that it is capable of testing validity for each format.

### Testing

To test this function, there are two lists: valid format postcodes and invalid format postcodes. Iterating through each of these lists, the postcodes are fed in to the function to ensure that only the valid format postcodes return a boolean value of True.

## Write\_csv.py

### Purpose

The function takes the crimes for all the months specified for the user and writes it to a csv. This is so that the `crimes_in_box` can read from the csv and filter the crimes based on the radius from the centrepont.

### Testing

The unit testing has a function `check_csv()`. It returns the true or false depending on whether the length is more than 1 line to check whether there is data in the file.

This has limitations since it doesn't check whether the file has been updated. If it were to fail after a successful first try, I wouldn't throw an error as it would read the old data. Using the `os` module's function `os.path.getmtime(path)`, it could record the last update time and check whether it changed to see if it had been updated.

## Research

### Template Reports

We had the idea to create a template for the data our program outputs, we wanted to have templates where the template data is substituted for the program output, this would involve programmatically writing output to a word document. Through research we weren't able to find ways of doing this without using an externally produced library which meant we would have to make another version, a 'sophisticated version'. We decided the added work wasn't worth the implementation of a second version and we have not pursued this functionality.

The following is a link to the library we found which exports program output to PDF:

<https://pythonhosted.org/PyPDF2/>

The following is a link to the library we found which exports program output to a Word Document:

<https://python-docx.readthedocs.io/en/latest/>

### Web Scraping

#### Average Amount of Crimes

One idea we had for the output is to show the average amount of crimes for postcode in a neighbourhood. For example they could look up the average amount of crimes in EX or EX1 postcodes and show the difference between the postcode specified by user.

This could be done by running the program for every EX postcode and determining the average amount of crimes in those postcodes. However we decided this would take too long for the program to run, because of the use of CSV files and the heavy calculations used to determine the great-circle distance.

Web scraping would be an easier way of doing this, however we struggled to find a website that gave an average for the same criteria we had. It would have to be a similar year, in order to compare the two effectively, and we struggled to find averages for 2016. It was also hard to find an average for all crime, most would show a specific type. These crime sites also show their results on a map, or in other visual tools which would be too complicated to scrape in the amount of time we had, and we felt that the amount of work this would've taken would be better spent focused on the core aspects on the project.



## Updating Files

Since we were able to find the origin of the file provided for this project, web scraping could be used to update the files for 2017. This could be checking the files in the directory against the data available on the [data.police.uk](https://data.police.uk) website and adding the latest file if required. There is also a [page](#) on the website that explains how you would request data from the website.

## Limitations

### User Input

#### Postcode

Some postcodes have a space in the middle in the CSV file. The user must have to type in the same format as found in the CSV file for the program to find the longitude and latitude. This is a limitation as the user would have to try both formats, without which will work and why it may throw an error other than 'we cannot find your postcode'. For future development, the program should try both formats regardless of which was inputted and produce an output for the correct one.

#### Dates

The program is hardcoded to find the crimes for November 2017 if a time range is not specified. This means that the program will soon be obsolete in the next coming month.


### File Handling

The specification asked for file handling to be as followed:

**With file as f:**

**Do something**

This way of handling files is argued to be better than `open(file)` and `file.close` for it makes sure that the file is closed when not used. Closing files is a good practise as this avoids opening files and not closing them afterward, meaning that changes would be committed to the file. It also avoids the Too Many Open Files error. The reason why we haven't used this is because when gathering data from the CSV file, changing to this syntax was an incredibly laborious task. To explain why, the image shows the way we used to loops to read from the file and solving this



problem we decided wasn't worth our time. After all the problem with the committing files was with only writing to files. We however were very carefully to close files after using them.

```
[ '', 'D', 'T', '2', ' ', '7', 'B', 'E', ' ', ' ', ' ', '1', '0', ' ', ' ', ' ', ' ',  
"DT2 7BG", "10", "361683", "104727", "E92000001", "E19000002", "E18000010", "E10000009",
```