Speeding Ticket

Brief:

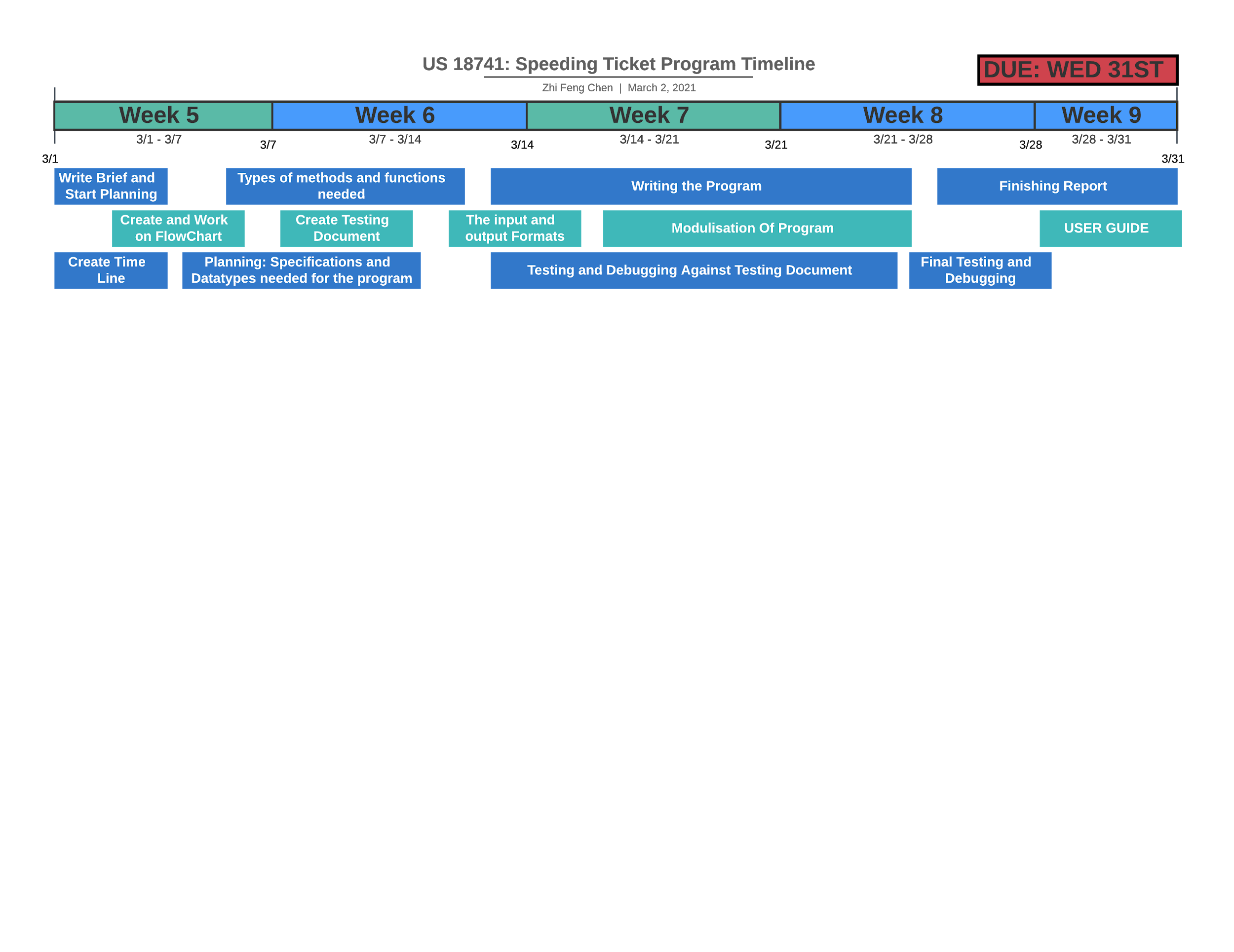
My target audience for this program is for the police’s and others alike. The problem to be solved by my program is to develop a program to help accurately assist the new method of image identifying speeding cars to help catch and fine those who are speeding. The purpose of my program will take data provided by the speeding cameras which provide the data in the form of a text file, the data in the file is then processed and used by the program to calculate and identify the average speed of which a car is travelling through the tunnel. After the speed has been calculated the program will decide whether the car is speeding or not. If the car is speeding the program will calculate how much the car is speeding above the zone limit and with that, the program will issue a speeding ticket associated with how much the car is speeding above the limit ie. if the car is speeding by 10km/h then a $30 fine is issued. If the car isn’t speeding then my program will give that car a “no fine” value or “$0” to represent no fines. The outcome of whether a driver is fined will be omitted to the screen so it can be read by an operator to see who is speeding so that the operator can send out speeding tickets accordingly. My program will also flag any invalid inputs such as a negative speed or invalid data from the data source to alert the operator whether there is an accident or an issue that the car is still in the tunnel or that the car has been missed by the system.

**Requirements / specifications :**

* My program will need to print out all the number plates of cars and their respective speed, and their fine (if any,) as well as flag any vehicles that may be stuck for more than 4 minutes or have been reported to not have exited the tunnel.
* The entry and exit times associated with a speed of car that is speeding will be printed along with the fine.
* The total cost and number of each fine needs to be printed out.

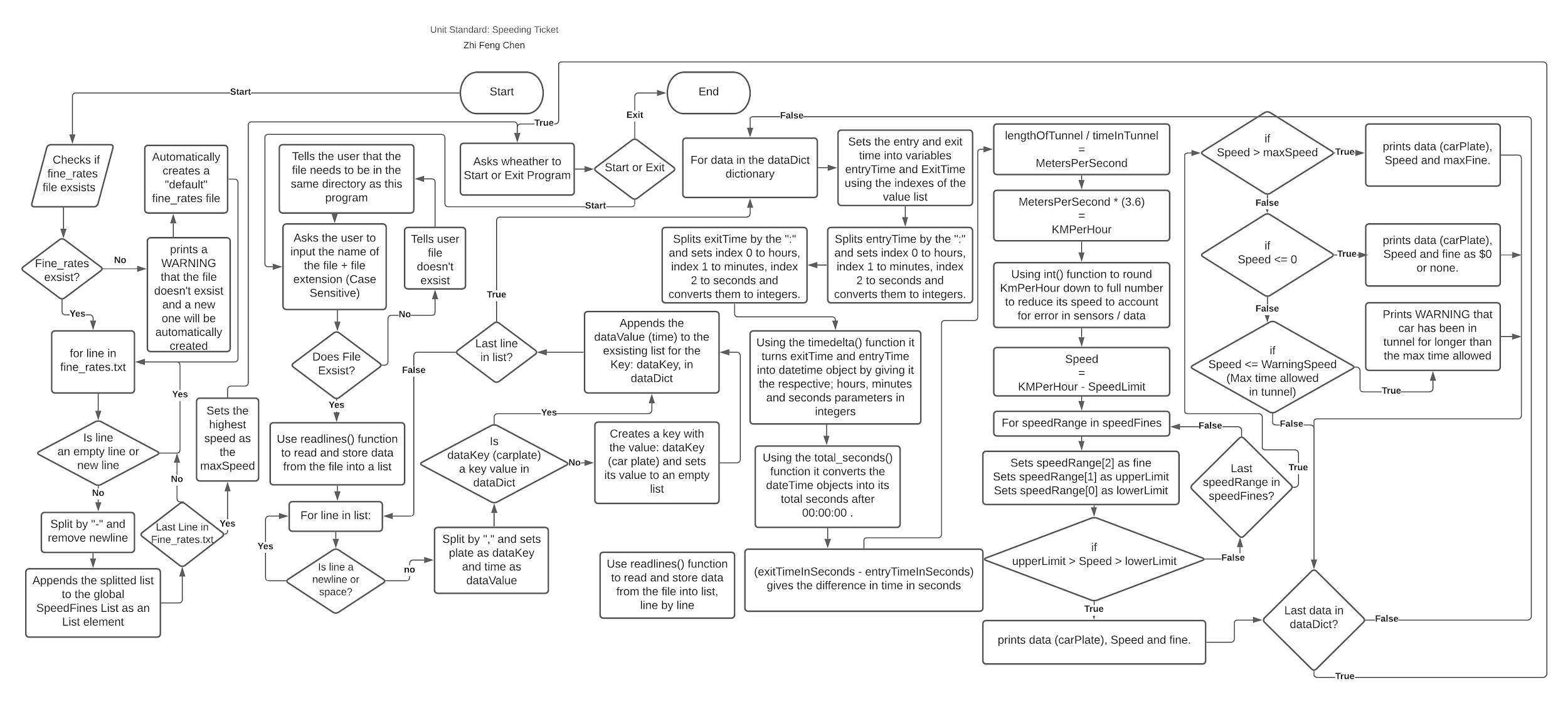
Timeline:

Using a timeline I can plan and ensure that goals are set and done by the intended deadline to ensure that I am on track with my set schedule. For this I will be using [Lucid Charts](https://www.lucidchart.com/) to create a plan of what should be done by each week.



**Flowchart / Decomposition of my program:**

I will create a flowchart of my program to help me determine what each part of my program needs to do to achieve my desired outcome and what parts of my program needs to / can be modulated so that it can be repeated / called when needed. A flowchart will allow me to see how my program will flow and what types of datatypes and variables will be needed as well as what type of user input will be present which will help me to prevent the user from inputting invalid data/inputs to prevent the program from crashing or omitting invalid data. This will be done by having error checking methods and boundary conditions that will raise the error to the user.



The above flowchart shows a logical structure of how my program will flow and meets the design requirements of printing / showing / omitting : all number plates of cars which have received a fine and their speed. As well a warning for if a car has not exited within 4 minutes.

**Specifications - Input information and constraints / format of data (if any):**

Using my flowchart above I can identify the input and output information:

**Storage of speeding Data fine data:**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Variable Name:** | **Data Type IN Program** | **Input Data Type:** | **How is it inputted (on a per line basis / format)** | **Purpose** |
| speedFines | List | .txt file | LowerLimit-UpperLimit-Fine | To store the fines and speeds in a local variable to easily accessed |
| dataDict | Dictionary | \*any file format as long as it is in the correct data format\* | Carplate,time\* | To store all times associated with the car plate |
| carSpeeds | Dictionary | From dataDict, input as string for car Plate and integer for speed. | CarPlate **:** Speed | To store the speed associated with the car to calculate fine later. |

*\*time entered / exited*

**Types of I/O needed:**

|  |  |  |  |
| --- | --- | --- | --- |
| **Variable Name:** | **Data Type:** | **How is it asked:** | **Purpose** |
| speedFileName | String | Please input the name of the data file: | To check and validate that the file exists. |
| activeState | string | Press ENTER to start program or type EXIT to end program: | For the user to choose to start or exit the program. |
| continueState | Boolean | s or stop to abort, c or continue to continue. | To continue or exit the program after an error has been spotted by the program. |

**Specifications - Output Information:**

|  |  |
| --- | --- |
| **Type of Output:** | **How it is outputted** |
| Fines | %CarPlate travelling at %Speed KM/HR so fine is %Fine |
| Warnings that a car hasn’t left within 4 minutes | WARNING!!! : %CarPlate reported to not have left the tunnel for more than %Time |
| Invalid input | Invalid Input ! ! ! |
| Warning Due to format error | WARNING!!! : Error occurred for %CarPlate on line, %LineNumber |
| Aborting due to format Error | Operation Canceled . . . |
| Continuing even after format Error | WARNING!!! : Continuing program may not work as intended i.e miscalculations , missing data. |
| Ticket of a speeding car | %carPlate, %entryTime, %exitTime, %speed, %fine |
| Tickets generated and amount | %fine, %No. Of tickets, %total for this ticket amount |
| Total number of tickets issued | Total Number of tickets Issued: %totalTickets |
| Total fine amount | Total Fine Amount: %totalFineAmount |
| When exiting program | Exiting Program . . . |

**Calculations needed:**

From the requirements, specifications above and my flowcharts the following calculations are needed.

**Time calculations:**

To calculate the difference between the entry and exit times a function called timedelta() from datetime is used to convert the time format: HH:MM:SS into an datetime object which is then put into the function: total\_seconds() to convert the date time object into total seconds from 00:00:00. Once both the exit and entry times are converted into seconds, the enytryTimeInSeconds is taken away from the exitTimeInSeconds which gives the difference between the two times which is the amount of time which the car was in the tunnel for:

(enytryTimeInSeconds - exitTimeInSeconds = differenceInTime)

**Speed Calculations:**

The total distance between the two speed camera is then divided by the differenceInTime which gives the car's speed in meters per second as v = d / t (where v is velocity, distance, and t is time): (totalTunnelLength / differenceInTime = metersPerSecond)

MetersPerSecond then needs to be converted into KM / HR. To do this, metersPerSecond needs to be multiplied by 60\*60 to get meters per hour, and divided by 1000 to get KM per hour:

(metersPerSecond \* 60\*60) / 1000 = kmPerHour

Or can be simplified to:

metersPerSecond \*3.6 = kmPerHour

This is then rounded down to the nearest WHOLE number to reduce the speed to account for minor calibration errors or faulty equipment or readings.

**Potential Problems and issues during recording of data and explanation:**

* The accuracy of the times and data recorded and saved in the file is dependent on software used on the cameras and how often the cameras are calibrated. For this reason, how accurate the data is recorded is out of my control.
* Other issues that can occur during the recording of the data are the recording of number plates. Some motor vehicles such as motorbikes only have their registration plate on the back of their vehicle. So depending on how the speeding cameras are set up, the cameras may be able to catch the vehicle entering / exiting or the cameras may not even pick them up at all.
* Duplicate numbers plates as cars can enter and exit the tunnel multiple times during the day.
* Daylight savings can cause the times in the data to be different, eg entry time is later than the exit time or the exit time is significantly later than the entry time i.e 1 hour. Since the time when daylight savings happens at 3am in the morning, which also happens to be the time where the least people are active, awake and driving. I have talked to my client and as during this time as there are very few people active, this can be ignored, and instead an error will be printed to the screen to alert the user when the entry time is later than the exit time or when the driver is in the tunnel for a very long period of time.

**Potential Issues during processing of data and explanation:**

* The accuracy of the times and data recorded is out of my control, but to give the driver a margin of error due to certain calibration errors and external factors, the final calculated speed of the vehicle (in KM / HR) will be rounded down to the nearest whole number. This is done so that the speed being evaluated by the program will be lower to account for calibration errors and other external factors that are out of my control.
* Checking whether an file exists or not as if the program tries to open an file that doesn’t exist in read mode then it will throw an “file not found” error and the whole program may crash if there are no error checking methods.
* Daylight savings, will cause the time / clock to roll forwards 1 hour causing the time being recorded at the time of the day to be 1 hours behind the previously recorded times.
* Another issue is that data across multiple different files, (can be seen as one day per file) may not merge together, i.e if the entry time is recorded on one file and the exit time is recorded on another file.
* The input times will either need to be in a 24 hour time format and not with the 12 hour time format and cannot be a mix of the two.
* The format of which data is being inputted into this speed program MUST be in the following formats:

Speeding Fines : LowerLimit-UpperLimit-Fine

SpeedingData : CarPlate,Time

If the data isn’t in the above formats and there are no error checking / format checking functions then if the data is processed it may be processed incorrectly or an error may be thrown and the program will crash if the format of the data is incorrect.

To account for **formatting errors** and **data entry errors** I will create a set of testing tables of data and implement these boundaries into my program over the course of the development of my program andtest them against my final program.

**Testing Table for formatting of Fines and Speed Ranges (per line / entry basis):**

|  |  |  |
| --- | --- | --- |
| **Input:** | **Expected Output:** | **Actual Output:** |
| 1-10-30 | Accepts |  |
| 41-45-510 | Accepts |  |
| 11min-15max-80fine | Prints Warning that data format is incorrect and asks whether to continue to abort. Also warns If were to continue, errors may occur. |  |
| 16,20,120 | Prints Warning that data format is incorrect and asks whether to continue to abort. Also warns If were to continue, errors may occur. |  |
| 31-35-$300 | Prints Warning that data format is incorrect and asks whether to continue to abort. Also warns If were to continue, errors may occur. |  |
| one-ten-thirty | Prints Warning that data format is incorrect and asks whether to continue to abort. Also warns If were to continue, errors may occur. |  |
| “\n” or “ ” or “” | Ignores |  |

**Testing Table for Formating of Speeding Data (per line / entry basis):**

|  |  |  |
| --- | --- | --- |
| **Input:** | **Expected Output (speed in KM / HR):** | **Actual Output:** |
| AKL40,11:04:00  AKL40,11:04:00 | Prints Warning that the entry and exit times are the same |  |
| AKL50,11:04:00  AKL50,11:06:07 | Prints AKL50 as car plate travelling at a speed of 76 and no fine |  |
| DB356,11:04:00  DB356,11:31:12 | Prints Warning that car hasn’t left tunnel within 4 minutes and also prints speed |  |
| D139,11:04:00  D139,11:05:58 | Prints D139 as car plate traveling at a speed of 82 with a fine of $30 |  |
| AKL50,11:08:10  AKL50,11:06:00 | Prints warning that entryTime > exitTime |  |
| FN806,11:04:07  FN806,11:04:34 | Prints \_\_ as car plate with a speed of 358 with a fine of $630 |  |
| AKL60,11:04:00 | Prints warning that the vehicle has been reported to not have exited the tunnel or that data is incorrect! |  |
| AKL70,11-04-00  AKL70,11:06:07 | Prints Warning that data format is incorrect and asks whether to continue to abort. Also warns If were to continue, errors may occur. |  |
| AKL80-11:05:50  AKL80-11:08:07 | Prints Warning that data format is incorrect and asks whether to continue to abort. Also warns If were to continue, errors may occur. |  |
| AKL90 11:01:10  AKL90 11:03:37 | Prints Warning that data format is incorrect and asks whether to continue to abort. Also warns If were to continue, errors may occur. |  |
| AKL10011:01:10  AKL10011:03:37 | Prints Warning that data format is incorrect and asks whether to continue to abort. Also warns If were to continue, errors may occur. |  |
| AKL110,11:01:100  AKL110,011:03:37 | Prints Warning that data format is incorrect and asks whether to continue to abort. Also warns If were to continue, errors may occur. |  |
| AKL120,11hr:02min:10sec  AKL120,11hr:05min:37sec | Prints Warning that data format is incorrect and asks whether to continue to abort. Also warns If were to continue, errors may occur. |  |

**Testing of each fine ranges:**

|  |  |  |  |
| --- | --- | --- | --- |
| **Range:** | **Input** | **Expected output** | **Actual Output** |
| Not speeding | AKL50,11:04:00  AKL50,11:06:00 | 80 KM/HR  \*NO FINE\* |  |
| 1 - 10 km/hr | BAK136,11:04:00  BAK136,11:05:55 | 84 KM/HR $30 |  |
| 11 - 15 km/hr | BAL560,11:04:00  BAL560,11:05:45 | 92 KM/HR $80 |  |
| 16 - 20 km/hr | BST1,11:04:00  BST1,11:05:40 | 96 KM/HR $120 |  |
| 21 - 25 km/hr | CKZ52,11:04:00  CKZ52,11:05:35 | 101 KM/HR $170 |  |
| 26 - 30 km/hr | D139,11:04:00  D139,11:05:30 | 107 KM/HR $230 |  |
| 31 - 35 km/hr | DB356,11:04:00  DB356,11:05:25 | 113 KM/HR $300 |  |
| 36 - 40 km/hr | DS323,11:04:00  DS323,11:05:23 | 116 KM/HR $400 |  |
| 41 - 45 km/hr | DCZ584,11:04:00  DCZ584,11:05:20 | 121 KM/HR $510 |  |
| 46 - 50 km/hr | DDA970,11:04:00  DDA970,11:05:15 | 129 KM/HR $630 |  |
| > 50 km/hr | SP33D,11:04:00  SP33D,11:04:50 | 193 KM/HR $630  \*Excessive speed\* |  |

All the testing tables are designed with the purpose to test this program to ensure that it works and can catch certain errors such as times or incorrect format or invalid data.

**Variables needed:**

Why use variables and why not just use the raw value right into the program?

Variables are used as certain values will / can be used throughout the entirety of the program so it say a variable of the tunnel length, or the distance between the two cameras were to changed (moved) then where all in the places in the code where a hard coded value is used, all of those values needs to be changed. On the other hand if the value is assigned to a variable, that variable can too, be used in place of where the value is needed i.e for calculations. Using variables opposed to hard coding in values offers a piece of mind and allows for easier maintenance and better control of writing the program. For instance, if say the distance between the speeding camera is changed or the speed limit is changed then the value of the variable could easily be changed and it will be applied to the whole program. On the other hand using hard coded values means that if the values needed to be changed then everywhere that value is used it needs to be found and replaced with the new one. On top of that typing out a value over and over again can have human error such as an incorrectly typed digit or an incorrect decimal place which can cause unnecessary debugging and troubleshooting.

**Examples of potential static variables which I may used are:**

*tunnelLength = 2690*

*speedLimit = 80*

*warningTimeLimit = 4*

**Development and justification of changes + testing of modules (to ensure functionality):**

I’ll be using github to store and save all versions of my speeding program so the development of my program can easily be tracked and looked at. It also allows me to revert to older versions if I were to want to revert to a previous version.

The link to the github repository is here: <https://github.com/ItzCino/US-Speeding-Ticket>

(older file is under “sideo” branch named “Speeding Program.py” )

I will also have internal comments in my program to document what each part of my program does.

**Modularisation of Speeding Program:**

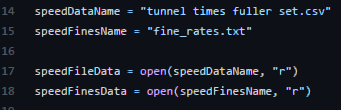
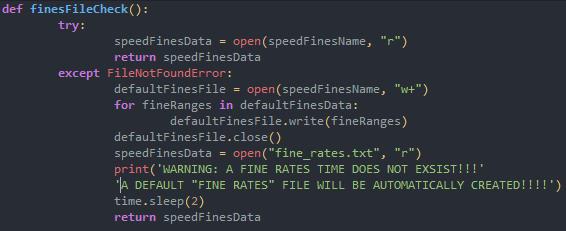
This speeding program will be modulated into many different definitions (definitions being a smaller program within the big program) which can be called and run repetitively to execute a particular action. An upside to why definitions are used is that, they can be called indefinitely with a single line whereas if definitions aren’t used then if a particular action was needed to be done again then that piece of code would need to be rewritten again, which would take up more space, make the size of the end file larger and make the code harder to read and maintain as if there is an problem with that chunk of code then where all the parts of the program where the code is used, the code needs to be changed. Definitions don’t have this problem as it's essentially a mini program that is being called so if there is an issue it can be fixed within the definition itself and it will apply globally to where the function is being called.

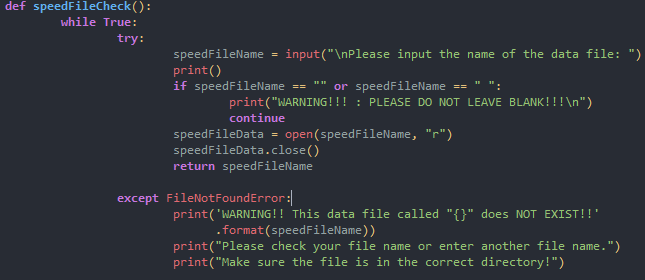
An example of use of definitions is for error checking for user inputs as they can be called and run until a required condition is satisfied.

**Functions that check whether a file exists:**

Some Error checking methods that are needed are to check for an fine\_rates.txt file and to check whether the name of the user inputted speed data file exists.

In my early versions of my program, there were no error checking methods to check whether a file existed or not, and as described earlier as one of the possible issues as if my program were to try to open a file that does not exist it will throw an “file not found” error.



The screenshots on the left shows that without error checking, if an invalid file name is entered then it will throw an error message and crash whereas with error checking it gives the user an warning message that the file does not exist and tells them check spelling or the directory of the file, then it asks for the file name again. As for the fines\_rates.txt file the program checks whether the fine\_rates.txt exists or not and if it doesn’t it’ll automatically create one. I’ve also chosen to allow the user to input the name of the file through the terminal as it can be repetitive and annoying to open and edit the speedFileName everytime that the user wants to check a different file. And as it is the user that is typing the file name in, I have to account for human error.

Testing of finesFileCheck() function;

|  |  |  |
| --- | --- | --- |
| **Input:** | **Expected Output:** | **Actual Output:** |
| \*No fine\_rates.txt file exists\* | Warning and creates a new file |  |
| fine\_rates.txt | accepts | \*accepts\* |

Testing of the speedFileCheck() function:

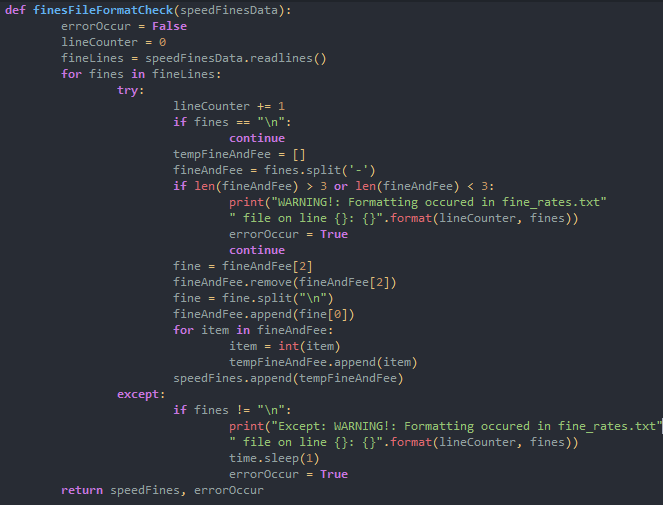
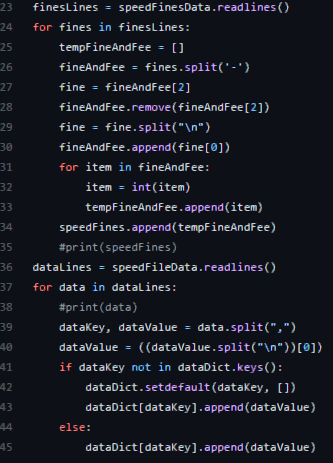
|  |  |  |
| --- | --- | --- |
| **Input file name:** | **Expected Output:** | **Actual Output:** |
| test\_tunnel.txt \*file exists \* | accepts |  |
| \*Blank\* | Prints “DO NOT LEAVE BLANK” |  |
| dummy\_tunnel.txt | Prints warning and asks for input again |  |
| 7UNN3L.txt | Prints warning and asks for input again |  |

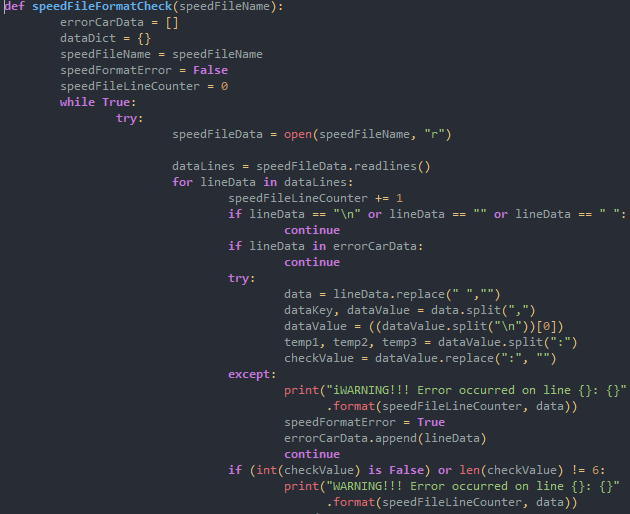
**Purpose of speedFileCheck():**

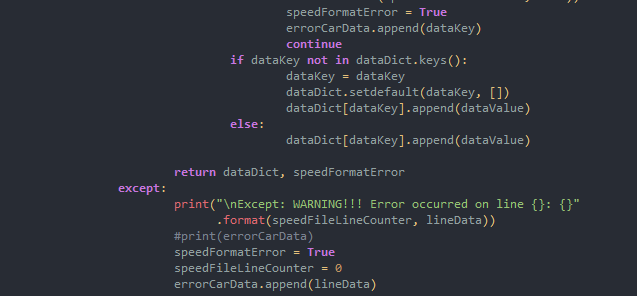
The checking whether a file exists is important as is it possible for an entered file name to not exist or to be entered incorrectly i.e mistyped through human error. The purpose of this function is to detect that the file does not exist and to NOT crash so that the program can continue to run without interruption.

**Functions that check for formatting errors from files:**

There needs to be error checking methods for checking the format of the input data from the files, such as the fine\_rates.txt file and the file containing the speed data. This is important as explained above as one of the potential problems; If there are no format checking functions and if there happens to be data that is formatted incorrectly then the data may be processed incorrectly which would give invalid or incorrect data, or it can potentially crash the entire program. Because of this, there needs to be error checking methods to remove the invalid data and warn the user IF there are formatting errors in the data so that the user is alerted that there is an issue without the program crashing so that they can go fix it. I have also integrated a line counter to help the user easily figure out which lines the error is occurring in the file so that it can be debugged faster and easier. This is useful for large chunks of data as it pinpoints exactly where the error occurred from. Once all of the format checking has been done and that the program doesn’t spot any errors then that piece of data is then automatically added to its respective dictionary / list.





The screenshots on the left shows the program before error checking and modulation , and which I’ve explained before is quite vulnerable to crashing if the format of the data is not right and the one screenshot on the right turns the does the same thing as the program on the left but it is modulated and has additional format and error checking to alert the user of where the error occurred so it can easily be debugged and fixed

**Purpose of these following functions:**

The finesFileFormatCheck() function works together with the continueOrExitForFines() function and the speedFileFormatCheck() function works together with the speedDataFormatError() function respectively so that is there is format / data error detected by the program it alerts the user and then runs their respective continue or abort function which allows the user to decide whether they would like to continue on with the calculations or abort the operation so that they can run another file through the program or fix up the error in the current data file before running the data file through the program again. And if the user does end up aborting the operation or continuing the calculation through one of their respective continue or exit functions it takes them to the exitDueToFormat() function which tells the user that the operation is cancel if they selected abort or a warning saying that the program may not work as intended will be printed if the user chose to continue.

There is a need for the finesFileFormatCheck() function as it allows the user to change the speed ranges and fines within a text file without editing the main program. This function alerts the user if there are any formatting errors with the fines file. The same applies for the speedFileFormatCheck() function.

Testing of finesFileFormatCheck() and continueOrExitForFines() functions:

|  |  |  |
| --- | --- | --- |
| **Input:** | **Expected Output:** | **Actual Output:** |
| 1-10-30 | Accepts | \*Accepts\* |
| 21-25-270 | Accepts | \*Accepts\* |
| 1min-10max-30fine | Prints error occurred on line 1 and Warns that the formatting is incorrect and is removed. Asks whether to continue or abort calculations and also warns that missing data and miscalculations may occur if the user were to continue. |  |
| one-ten-thirty | Prints error occurred on line 1 and Warns that the formatting is incorrect and is removed. Asks whether to continue or abort calculations and also warns that missing data and miscalculations may occur if the user were to continue. |  |
| min=1 max=10 fine=30 | Prints error occurred on line 1 and Warns that the formatting is incorrect and is removed. Asks whether to continue or abort calculations and also warns that missing data and miscalculations may occur if the user were to continue. |  |

Testing of speedFileFormatCheck() and speedDataFormatError() functions:

|  |  |  |
| --- | --- | --- |
| **Input:** | **Expected Output:** | **Actual Output:** |
| AKLONE,17:05:50  AKLONE,17:07:59 | Accepts | \*Accepts\* |
| AKL50,11:04:00  AKL50,11:06:07 | Accepts | \*Accepts\* |
| AKL60,11-03-10  AKL60,11-05-50 | Prints error occurred on line 1 & 2  Warns that the formatting is incorrect and errors are removed Asks whether to continue or abort calculations and also warns that missing data and miscalculations may occur if the user were to continue. |  |
| AKLONE13:05:50  AKLONE13:08:10 | Prints error occurred on line 1 & 2  Warns that the formatting is incorrect and errors have been removed . Asks whether to continue or abort calculations and also warns that missing data and miscalculations may occur if the user were to continue. |  |
| AKLTEST 17hr 05min 50sec  AKLTEST 17hr 07min 59sec | Prints error occurred on line 1 & 2  Warns that the formatting is incorrect and errors have been removed. Asks whether to continue or abort calculations and also warns that missing data and miscalculations may occur if the user were to continue. |  |

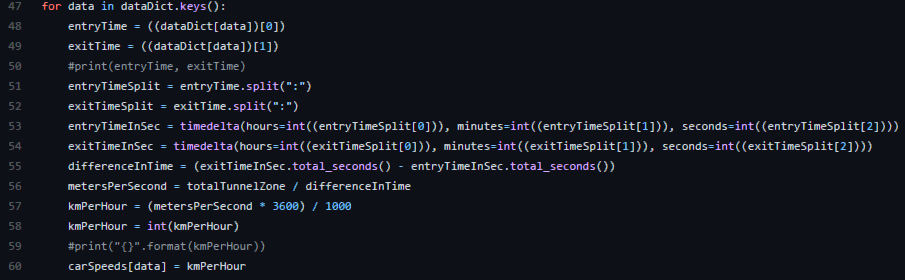
Testing of function respective function that asks whether to abort or continue calculations;

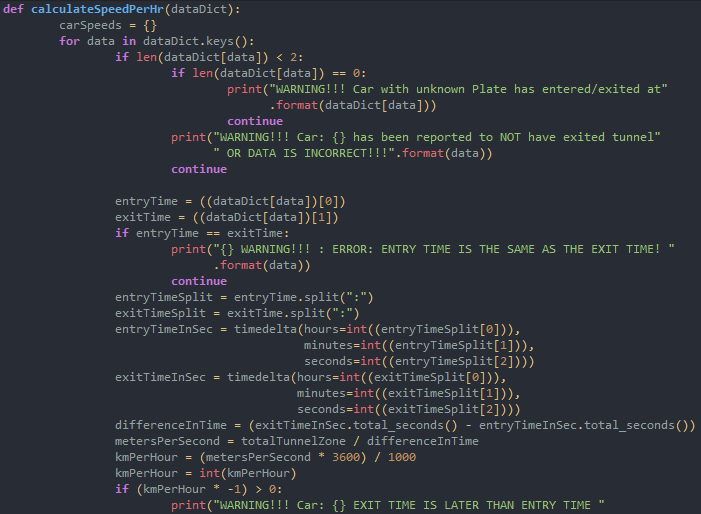
(testing of both speedDataFormatError() and continueOrExitForFines() functions as they essentially the same function along with the exitDueToFormat() function.)

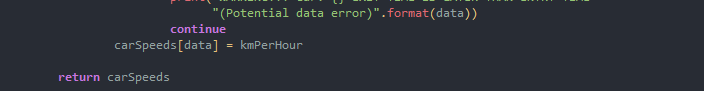
|  |  |  |
| --- | --- | --- |
| **Input:** | **Expected Output:** | **Actual Output:** |
| C or c | Accepted, and prints a warning that there maybe miscalculations or missing data |  |
| S or s | Accepted, and prints operation canceled . . . |  |
| CONTINUE or continue or CoNtinUE | Accepted, and prints a warning that there maybe miscalculations or missing data |  |
| STOP or STOP or StOP | Accepted, and prints operation canceled . . . |  |
| St0p or 5t0p | Prints invalid input and asks for input again |  |
| 29e9me | Prints invalid input and asks for input again |  |
| @#!$@ | Prints invalid input and asks for input again |  |

**Function of Calculating Car speed:**

This function is used to calculate the speed of which a car is travelling at by getting the difference in time between the time which the car exitted the tunnel and the time which the car entered the tunnel. A definition is used for this because it can easily be called over and over again rather than having the same piece of code being repeated when this action needs to be run.







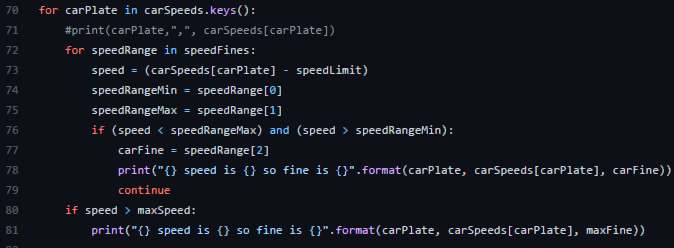
The first screenshot shows the first version and the second screenshot shows the final version. Improvements I have made are warnings which alert the user for when a car has been reported to NOT have exited the tunnel and for errors where the entry and exit times are the same and for when the exit time is later than the entry time.

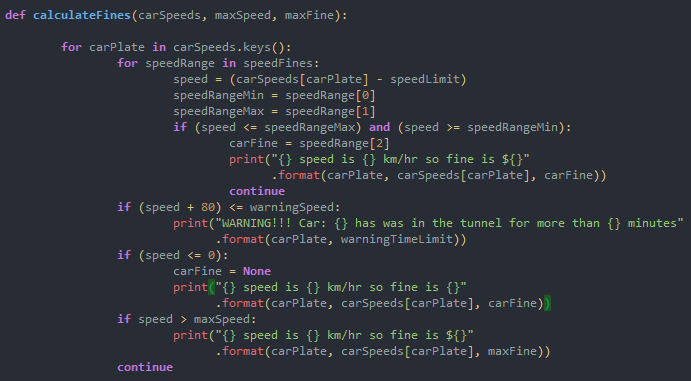
Testing for calculateSpeedPerHr() function:

|  |  |  |
| --- | --- | --- |
| **Input:** | **Expected output:** | **Actual output:** |
| AKL10,11:04:00  AKL10,11:04:00 | Prints Warning that the entry and exit times are the same |  |
| AKL20,11:04:00  AKL20,11:06:07 | Prints AKL20 as car plate travelling at a speed of 76 |  |
| DP348,11:04:00  DP348,11:31:12 | Prints DP348 as car plate traveling at a speed of 5 |  |
| DF199,11:04:00  DF199,11:05:58 | Prints DF199 as car plate traveling at a speed of 82 |  |
| DP128,11:04:00  DP128,11:05:24 | Prints DP128 as car plate traveling at a speed of 115 |  |
| DF111,11:03:23  DF111,11:05:09 | Prints DF111 as car plate traveling at a speed of 91 |  |
| SD232, 18:12:16  SD232, 18:13:15 | Prints SD232 as car plate traveling at a speed of 164 |  |

**Function for Calculating Fines:**

This function simply parses a dictionary containing the car plates and their speed (in KM/HR) and calculates its fine, if any. There is no need for any sort of error checking as formatting errors and alike would have been taken care of by the format and data checking functions so they should be error free. Reason that this is modulated into a definition is because it can easily be called i.e when a new set of speeding data is processed this definition can be called to calculate the fines which avoids the need of having repeated chunks of code when a particular action is required.

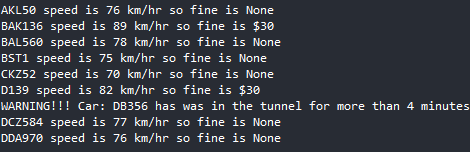




The first screenshot shows the initial program and the second screenshot shows the modulated version. Between the two different versions, some changes I’ve made is that I’ve also added edge cases for if speeds are greater or equal to the max speed as there was no coverage for speed values for above the max speed in the fine\_rates.txt file. I’ve modified it so that there are; greater than or equal to & lesser than or equal to comparative statements as in the previous version it wouldn’t be able to catch those edge values so many speeding cars weren’t able to be caught by my program previously. I;ve also added warnings for if the car has been in the tunnel for more than 4 minutes to meet the specifications of the program. And as there are no errors in the data being thrown into this program, as all the data has been format checked and error checked the data passed through should be processed correctly and shouldn’t crash. All these functions were done to meet the specific criterias and specifications, wanted by the client.

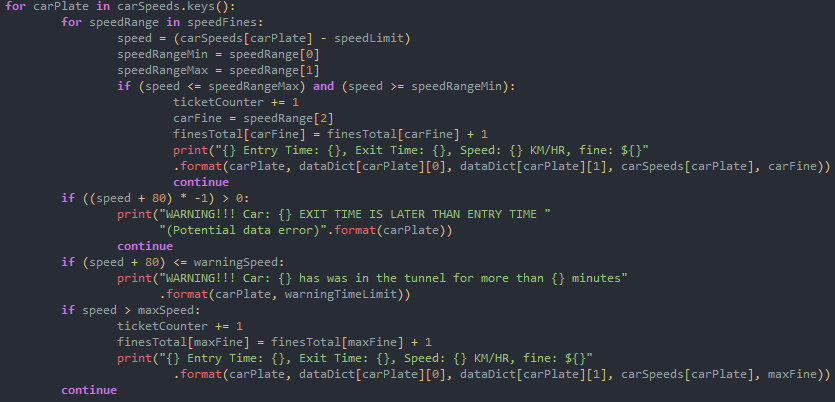
Testing of calculateFines() function:

|  |  |  |
| --- | --- | --- |
| **Input:** | **Expected output:** | **Actual output:** |
| AKL10,11:04:00  AKL10,11:04:00 | Prints Warning that the entry and exit times are the same |  |
| AKL20,11:04:00  AKL20,11:06:07 | Prints AKL20 as car plate travelling at a speed of 76 and no fine |  |
| DP348,11:04:00  DP348,11:31:12 | Prints Warning that car hasn’t left tunnel within 4 minutes and also prints speed |  |
| DF199,11:04:00  DF199,11:05:58 | Prints DF199 as car plate traveling at a speed of 82 with a fine of $30 |  |
| DP128,11:04:00  DP128,11:05:24 | Prints DP128 as car plate traveling at a speed of 115 with a fine of $300 |  |
| DF111,11:03:23  DF111,11:05:09 | Prints DF111 as car plate traveling at a speed of 91 with a fine of $80 |  |

**Meeting all the specifications:**

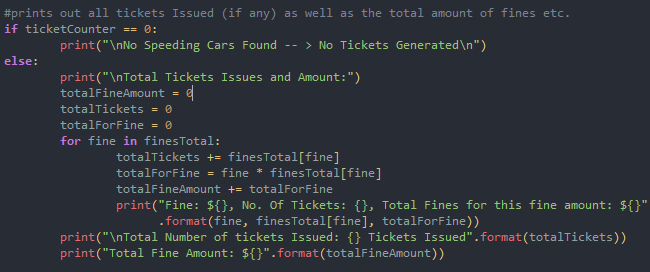
Although my program may seem like it does everything it needs to, it only prints out errors, the speed of the car with an associated fine (if any.)

To meet the specifications I need my program, on top of what it does right now, to print out all the cars which have been issued a ticket, with their number plate, entry and exit times, their speed and hence their fine. The program also needs to print out any warnings i.e cars that have been in the tunnel for more than 4 minutes and it will also need to print out the total number of tickets issued, the total amount in fines issued, the number of tickets for each fine and the total amount issued for each fine.To do this I will need to modify the calculateFines() function a bit in the calculations part to only print out those who are speed along with their tickets. I’ve modified the print statements to include their number plate, entry time, exit time, speed and their associated fine. This was done to meet the specifications of the program / client.

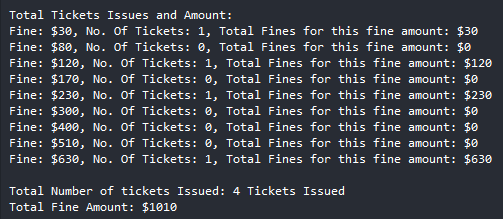
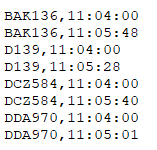


As stated above the program also needs to be able to print the number plate, no. of tickets issued for a fine and the total fines amounts as well as the total tickets issued and the sum total fine amount. So the code in the screenshot above and below shows this as well as a comparison.

To do this a new dictionary called also had to be created to store the number of each ticket which are issued when the speed limit has been passed. The dictionary is then looped through at the end to calculate the total fine amounts issued as well as the total number of tickets issued.



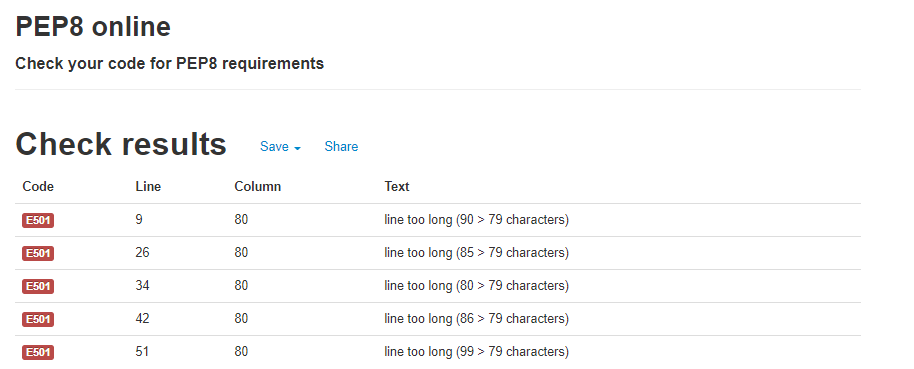
Using this as test input:



My testing methodology, to test whether certain parts of my program works is that, I’ll test it with a small sample. So that I can test different parts of my program and control my testing as a large sample will be very difficult to track and analyse compared to a smaller sample. Also a smaller sample means less time to analyse which means less time is needed in total to fix an issue. And in the end I’ll test my completed program against a slightly larger sample to show whether or not my program has met the requirements and specifications.

**PEP8:**

I have also used a PEP8 checker to ensure that my code is readable and is formatted accordingly to the specifications for easy readability thus debugging.



**Limitations of my Program (and possible solutions if any):**

* Although I plan to have this program catch and report / alert all possible errors to the user, it may not be 100% bug free and there could potentially be more edge cases and formatting errors which can be caught / blocked, but as for now the current program will catch most of these errors.
* Input data must be in a specified format or else it will print a warning message to the user.
* The user must have python installed to use it.
* This program treats each file independently, meaning that speeding data from different files don’t carry over if the entry and exit times are in different data files.
* My program only works with the 24 hour time format and not with the 12 hour time format.
* This program doesn’t automatically correct times due to the sudden change in time caused by daylight savings.
* This program doesn't carry over times and data from other files if there is missing data. And solution to this is to have the program that stores all the values of cars which have entered the tunnel but not have exited the tunnel in a separate global dictionary i.e noExitTime = {} and then move that piece of data out of that dictionary once a file that has the exit speed is run through it. This can be easily fixed by having the time format to include the date for example: dd/mm/yy,hh:mm:ss.
* It also does not validate the number plate i.e does NOT check for symbols etc.

**Sample of the final program in action (with no errors in data):**

|  |  |  |
| --- | --- | --- |
| Input | Expected output | Actual output |
| AKL50,11:04:00  AKL50,11:06:07  BAK136,11:04:00  BAK136,11:05:48  BAL560,11:04:00  BAL560,11:06:03  BST1,11:04:00  BST1,11:06:08  CKZ52,11:04:00  CKZ52,11:06:17  D139,11:04:00  D139,11:05:58  DB356,11:04:00  DB356,11:31:12  DCZ584,11:04:00  DCZ584,11:06:05  DDA970,11:04:00  DDA970,11:06:07 | AKL50, 76KM/HR, Fine: $0  BAK136, 89KM/HR, Fine: $30  DDA970, 76KM/HR, Fine: $0  BST1, 75KM/HR, Fine: $0  CKZ52, 70KM/HR, Fine: $0  D139, 82KM/HR, Fine: $30  DB356, 5KM/HR, Fine: $0  DCZ584, 77KM/HR, Fine: $0  BAL560, 78KM/HR, Fine: $0  Total fines**:** $60  Total tickets issued: 2 | **Raw output:**  AKL50 speed is 76 km/hr so fine is None  BAK136 speed is 89 km/hr so fine is $30  BAL560 speed is 78 km/hr so fine is None  BST1 speed is 75 km/hr so fine is None  CKZ52 speed is 70 km/hr so fine is None  D139 speed is 82 km/hr so fine is $30  WARNING!!! Car: DB356 has was in the tunnel for more than 4 minutes  DCZ584 speed is 77 km/hr so fine is None  DDA970 speed is 76 km/hr so fine is None  **Tickets Issued + WARNINGS :**  BAK136 Entry Time: 11:04:00, Exit Time: 11:05:48, Speed: 89 KM/HR, fine: $30  D139 Entry Time: 11:04:00, Exit Time: 11:05:58, Speed: 82 KM/HR, fine: $30  WARNING!!! Car: DB356 has was in the tunnel for more than 4 minutes  **Total Tickets Issues and Amount:**  Fine: $30, No. Of Tickets: 2, Total Fines for this fine amount: $60  Fine: $80, No. Of Tickets: 0, Total Fines for this fine amount: $0  Fine: $120, No. Of Tickets: 0, Total Fines for this fine amount: $0  Fine: $170, No. Of Tickets: 0, Total Fines for this fine amount: $0  Fine: $230, No. Of Tickets: 0, Total Fines for this fine amount: $0  Fine: $300, No. Of Tickets: 0, Total Fines for this fine amount: $0  Fine: $400, No. Of Tickets: 0, Total Fines for this fine amount: $0  Fine: $510, No. Of Tickets: 0, Total Fines for this fine amount: $0  Fine: $630, No. Of Tickets: 0, Total Fines for this fine amount: $0  Total Number of tickets Issued: 2 Tickets Issued  Total Fine Amount: $60 |