A screen shot of a computer

Description automatically generated with low confidence

A screen shot of a computer

Description automatically generated with low confidence

Forensic tool project

Advanced programming for digital forensics

Table of contents

Contents

[Section 1: Overview 3](#_Toc89718990)

[Section 2: Code explanations 4](#_Toc89718991)

[Section3: GUI design 14](#_Toc89718992)

[Section 4: GUI implementation & explanation 15](#_Toc89718993)

[Section 5: Supporting files 16](#_Toc89718994)

[Section 6: Flowcharts 17](#_Toc89718995)

[Section 7: Testing 18](#_Toc89718996)

# Section 1: Overview

The aim of this project is to make a digital forensics tool, similar to EnCase or Autopsy. The program needed to have the ability to do things like:

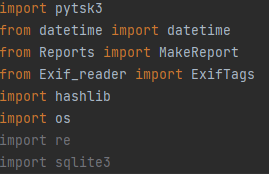
* Open different types of forensic file formats
* Discern whether the image uses GPT or MBR
* Retrieve information about the partitions
* Retrieve information about file systems
* Retrieve Information about files and directories

It is written entirely in the Python programming language and utilises a variety of both inbuilt and external modules, these include:

* PyTSK3
* PyEWF
* Re
* Sqlite3
* Hashlib
* Exifreader

The program is fully user input driven to allow the user to find the information they need, without crowding the screen with excess. The program uses a terminal interface. Currently, the program uses 10 files to function correctly: 6 Python files, 3 CSV files and 1 database.

# Section 2: Code explanations

Imports

This code is at the very top of the program, it imports all f the libraries necessary for the program to run. This includes built ins like datetime, Hashlib and os, alongside externals like pytsk3 and sqlite3. There are also some libraries (Exif\_reader and Reports) that are inherited from programs I have written

Hashlib is another inbuilt python library that allows the program to determine the hash values of files using algorithms like MD5 and sha.

OS is a library that allows python to make use of certain operating system functionalities, it is only used once in this program and is not essential to the overall functionality.

Re is an inbuilt library that allows the program to use regular expressions, they will be used for file signature analysis

Sqlite3 is an external library used to make transactions with a database, used in this program for file signature analysis. The database it talks to contains a collection of hex signatures and their corresponding extension, courtesy of Gary Kessler (<https://www.garykessler.net/library/file_sigs.html>)

Datetime is a built-in library that allows the program to get the current date and time, used in report making.

Reports is another inherited program that I made that will be discussed below. Its main function is to take some data and turn it into a readable, csv file

Pytsk3 is a python binding of The sleuth kit, which is what the forensic tool Autopsy uses, its function is to retrieve data of forensic significance from images. It has functions like “pytsk3.Img\_Info()” and “pytsk3.Volume\_Info()”, which will be elaborated on further down.

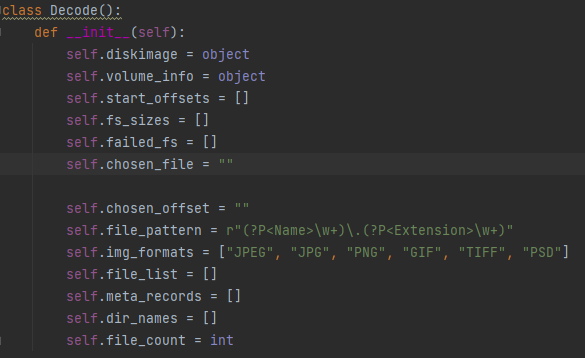
Exif\_reader is an inherited program I made that will be discussed in greater detail below. Its primary function is to accept the name of a file and return EXIF tags that I deemed forensically significant.

Class set up and variables:

“Self.diskimage” and “self.volume\_info” are both variables used for pytsk3 decoding. They are both assigned to the “Object” type here because if they are immediately assigned in the init method, the program won’t run.

The init function is important to set up the variables used throughout the program.

Defining the class name



These 3 variables are all lists, they are used to store the start offsets of all the file systems, the size of the file systems and the offsets that pytsk3 could decode

Self.chosen\_file stores the name of the file the user is decoding. Self.chosen\_offset does the same but for offset. Both set to string type

This is a regular expression to separate the name and extension of a file, used in signature analysis

These 3 lists are for storing the names of files, directories, and metadata record numbers.

Last variable is for storing the number of files found.

This is a list of image format extensions. Not currently in use

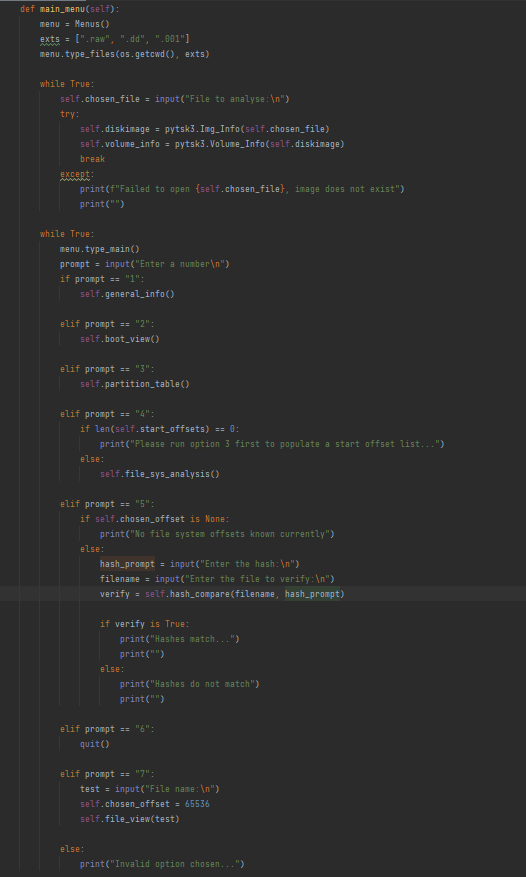
To summarise, the class definition and set up is the first part of code in the program. The program is put into a class to because classes are integral to OOP. They ensure the code is clean, reusable, and encapsulated. The “self” variables are helpful in large programs because without out them, functions cannot use variables created by other functions.

Main menu:

Sets up while loop which asks for file name input, then a try and except clause. Tries to open the selected file, if it succeeds then it sets up the disk image and volume info instance variables for pytsk3 later

Defining the function

Instantiating menu class and setting up a list of extensions and calling the menu maker program to print a menu displaying all files in the current directory that match those extensions



Main options menu set up by calling to Menu maker programs “type\_main” function. Same loop as file name loop; it asks for a menu option number and checks to see if it’s a valid option. If it is, it executes a call to the respective function, else it prints an error and continues to loop

Option 4 is the option that starts the analysis of the individual file systems and so a check must be done to see if the offset list is populated, if it isn’t then the program tells the user to run option 3 which is what populates that variable

Option 5 is for file hash verification of files and so in the same way as option 4, a check is needed to see if the file system offset list has been populated.

Stops the program

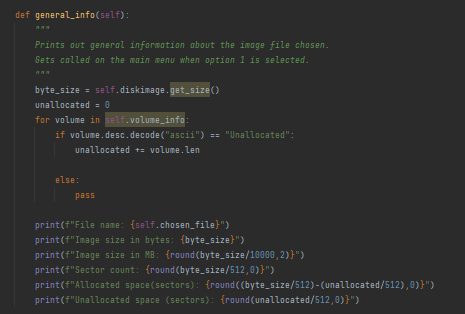
This is for if an incorrect option is chosen, it executes no function calls and just serves to go back to the loops beginning

This was a test for a hex view of a file that pytsk3 found. It has since been moved to the file system menu, where it belongs.

To summarise, the main menu is the first thing displayed when the program starts. It serves as the way the user navigates through the program. It has 6 options, 1 of which will lead to other menus for more in depth file system analysis. Originally, each menu option was printed out within the function, but this was changed in favour for a program I wrote that creates the menus and prints all options in 1 line as it cleans up the “main” file. This program is discussed below.

General information function:

Defining the function



Setting up a variable that stores the size of the image. This information is gathered via pytsk3’s “get\_size” function

Setting up a variable to store the unallocated space

These 2 lines print out the name of the file that the user selected by printing the associated instance variable and then the size of the image in bytes.

Looping through the partitions found and deciphering whether its unallocated or not by reading its name. If it is, the size of the partition is added to the previously set up “Unallocated” variable

The image size is then calculated in MB by taking the “byte\_size” variable and dividing it by 10,000 and then rounding that number to 2 decimal places using pythons built in “round” function.

The allocated space is found out by first finding out the total sectors and then subtracting the previously calculated unallocated size (divided by 512). Unallocated sectors are figured out by taking the unallocated bytes and dividing by 512 (standard sector size)

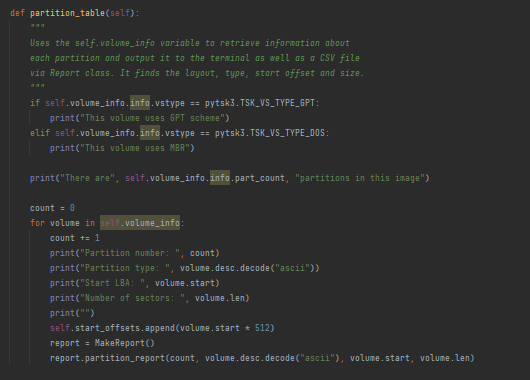
Sector count is found out by taking the byte size of the file and dividing by 512, the standard sector size

To summarise, the general information function is the function that calculates basic information about the image. This includes the size in bytes and mb, the allocated and unallocated sectors, and the total sector count. The “self.volume\_info” and “get\_size()” lines are highlighted by PyCharm because in the “init” method of the class, the pytsk3 “volume\_info” and diskimage” are set up as object types and are only assigned to pytsk3 objects in the main menu.

Partition table decode:

This If-Elif statement is used to determine what partitioning scheme the image uses.

Defining the function



Print statement to display the total amount of partitions that pytsk3 found

Partition type is the file system that it uses, NTFS for example

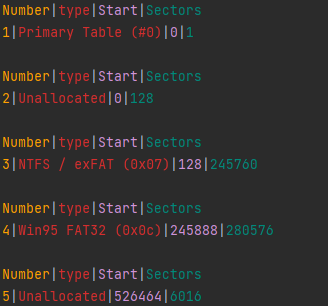
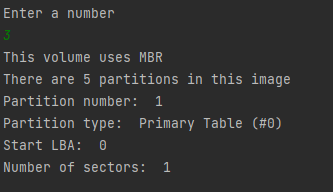
Setting up a count variable to give a number to each partition

Volume.start is the start offset of the file system

This line adds the start offset of the file system to a list of start offsets. The list is one that was created in the “init” method. These offsets are used later in another menu.

Volume.len shows the size of the partition in sectors

“report is a variable that gets assigned the instantiated “MakeReport” class. This is an inherited class that is used to make CSV reports on the things pytsk3 discovers. The info found earlier in the loop is passed into the function of ReportMaker. Below is an example CSV output and terminal output.



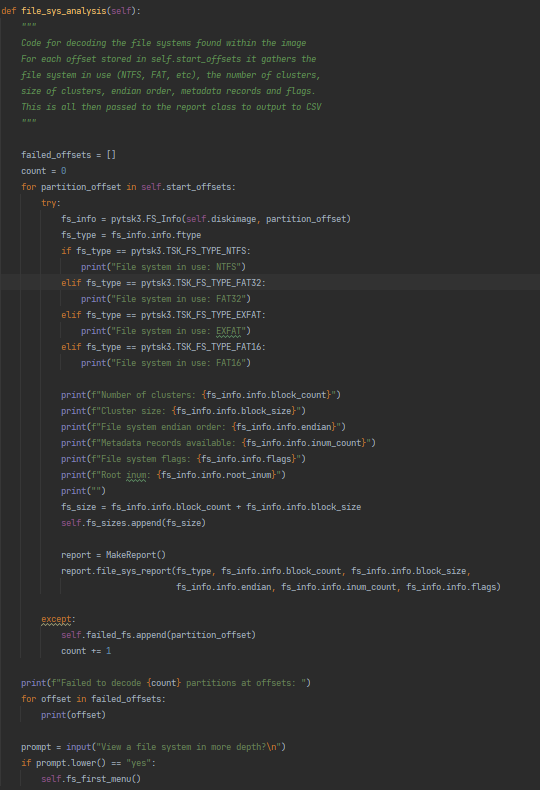
To summarise, the partition decode function is the function that uses pytsk3’s volume\_info to find out information about the partitions on the image. It displays the file system type, start offset and length. It also displays the type of partitioning scheme in use, MBR or GPT. The final part of the function is a call to an inherited, custom written, program that outputs the data to CSV.

File system general analysis:

This If statement group will determine the type of file system in use by comparing the “fs\_type” variable to pytsk3’s variables.

Defining the function

This print statement block prints out the information about the file system like cluster count, endianness, and root inum.



Setting up a list to contain the offsets pytsk3 couldn’t decode and a count variable like seen in other functions.

This line appends the file system sizes to a list created in the init method for later use. This is similar to how start offsets were appended to a list in the partition decode function.

Sets up a for loop to loop through the file systems found in the previous function (partition decode).

All of the following code relies on a Try & Except clause, which checks first to see if pytsk3 can decode the file system found at the offset. If it can’t, it appends the file system offset to the failed offset list to print later and carries on the loop.

This print statement displays the amount of file systems pytsk3 failed to decode and then prints the offsets of those file systems by looping through the list.

The last lines of the function give the user the option to continue on with file system decoding or return to the main menu. The first file system menu is called if the user responds with a “yes”. Pythons “.lower)(“ is used here to avoid errors.

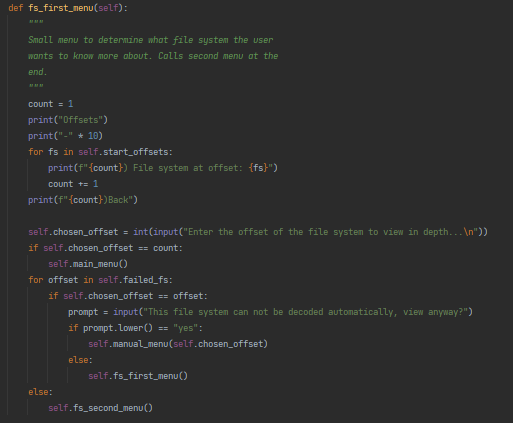
These line call to the inherited Report class in the same way Partition decode did, to make a CSV report of the found file systems.

To summarise, the file system analysis function takes the start offsets list from the previous function (partition decode) and loops through them. If pytsk3 can decode them, their information is printed to the screen and saved to a CSV report. If pytsk3 can’t decode the file system, its offset is added to the failed offsets list and printed out at the end, the loop continues until the whole list has been iterated over.

File system first menu:

Sets the chosen offset instance variable to the user input. This variable is used in other functions.

Defining the function



Set up count variable and print a menu header and underline it.

If the chosen offset input is the same number assigned to the “back” option on the menu, the program goes back to the main menu.

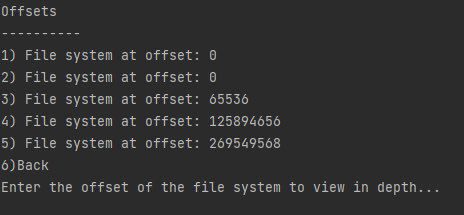
Start a loop to loop through the previously discovered file system offsets and print them out. It also prints a “back” option to display on the menu

Loops through the failed offsets to see if the user selected one, if they did then the program asks if they want to manually decode it.

If the user chooses to manually decode then the function that deals with that is called, if they choose not to then the loop for this menu restarts.

Finally, the second menu is called if the file system offset chosen is a valid, decodable one.

Example terminal output.

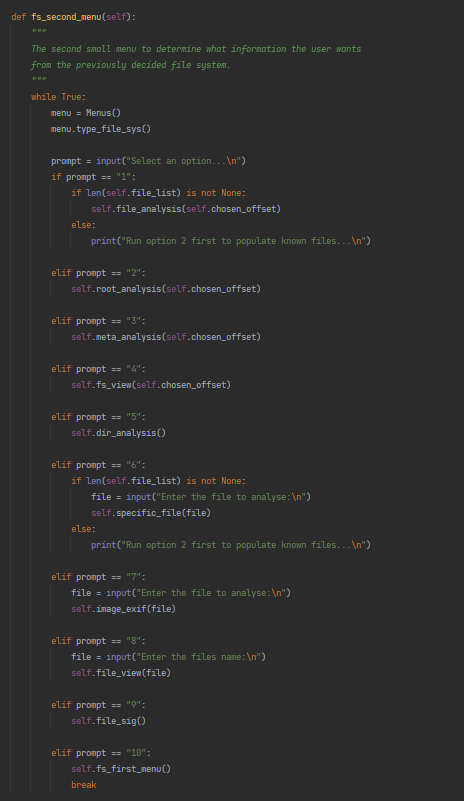


To summarise, this is the first file system analysis menu, it simply prints out a list of the known file systems and then asks the user to input one. The bulk of the code here is error handling to make sure the program doesn’t just crash if something goes wrong.

File system second menu:

Defining the function

Setting up a while loop for the menu and then calling the MenuMaker “Menu” class to print a menu containing the relevant choices.



Setting up the variable “prompt” to store the user’s menu choice.

This large If statement block is for determining which function needs running based on the user’s input.

This is error handling because option 1 needs to first have a populated list of files to be able to execute its own code.

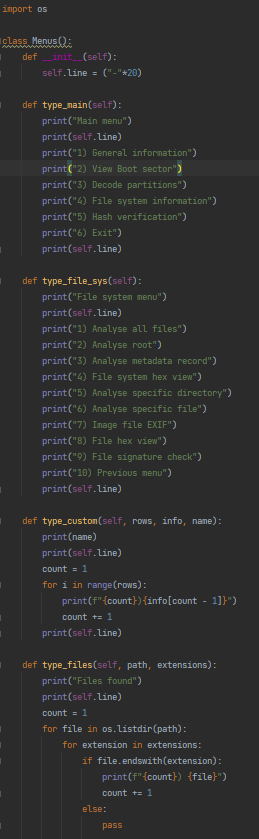
The “\n” character found on these print statements is a newline character in python and just serves to make input/output cleaner

This option also has error handling because it needs a file list to work.

Exits to the first file system menu.

To summarise, the second file system menu is a gateway to the more complex functions of the program like file signature analysis and ExifTag viewing. The bulk of the code is a large If statement collection which will run the various functions based on the user’s input. The menu options are again displayed via the Menus() class to clean up the code and make it easier to read.

Menu maker:



Class definition and init method, which sets up one variable.

Importing the OS module to add operating system functionality to the program.

First function is for printing out the main menu options. Uses the instance variable twice to print out lines.

Second function is for the file system second menu. Same as the previous function, uses the instance variable twice.

Third function is for creating a custom menu. It accepts a row count, info and name argument and uses these to set up a custom menu.

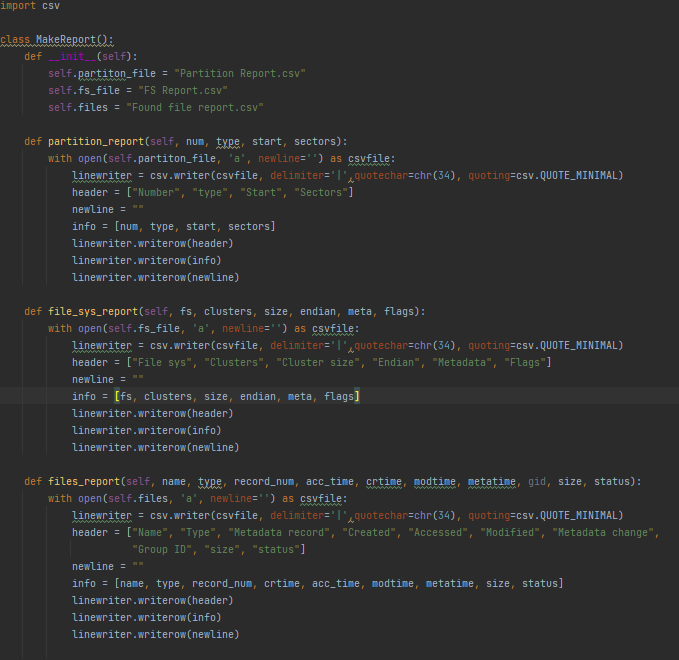
Fourth function is for the very first “menu” shown when the main program is running. It uses OS to search the current directory (cwd) and look for files that end in appropriate extensions. It the prints these out so the user can see what files the program can work with.

To summarise, this program was made to help clean up the code in the main program. I felt that including so many print statements made the main program cluttered and hard to read through. Now the main program only needs 2 lines of code to generate the appropriate menu.

CSV reports:

Defining the class and importing the csv module

Setting up the instance variables. All of them are the names of the files that are written to.



The bulk of the program is 3 functions that all do the same thing, just with different data. They open their respective files using the instance variable as a csvfile. They then set up a “linewriter” object that holds the “csv.writer” function and all its arguments. It then sets up a header variable which just stores column headings, so the report is readable. It then sets up a list of the information and writes all of this to the csv file.

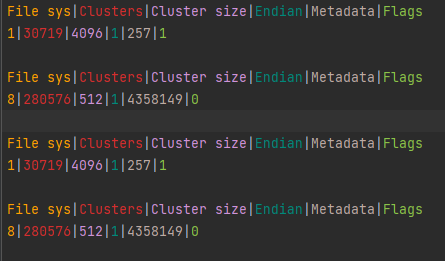
The arguments in csv.writer:

“delimiter” – sets up what character is printed in between the information.

“quotechar” – Similar to delimiter set to its default here.

“quoting” - csv.QUOTE\_MINIMAL means add quote only when required, for example, when a field contains either the quotechar or the delimiter. This is the default.

This is an example of a report generated by the program, for found file systems.



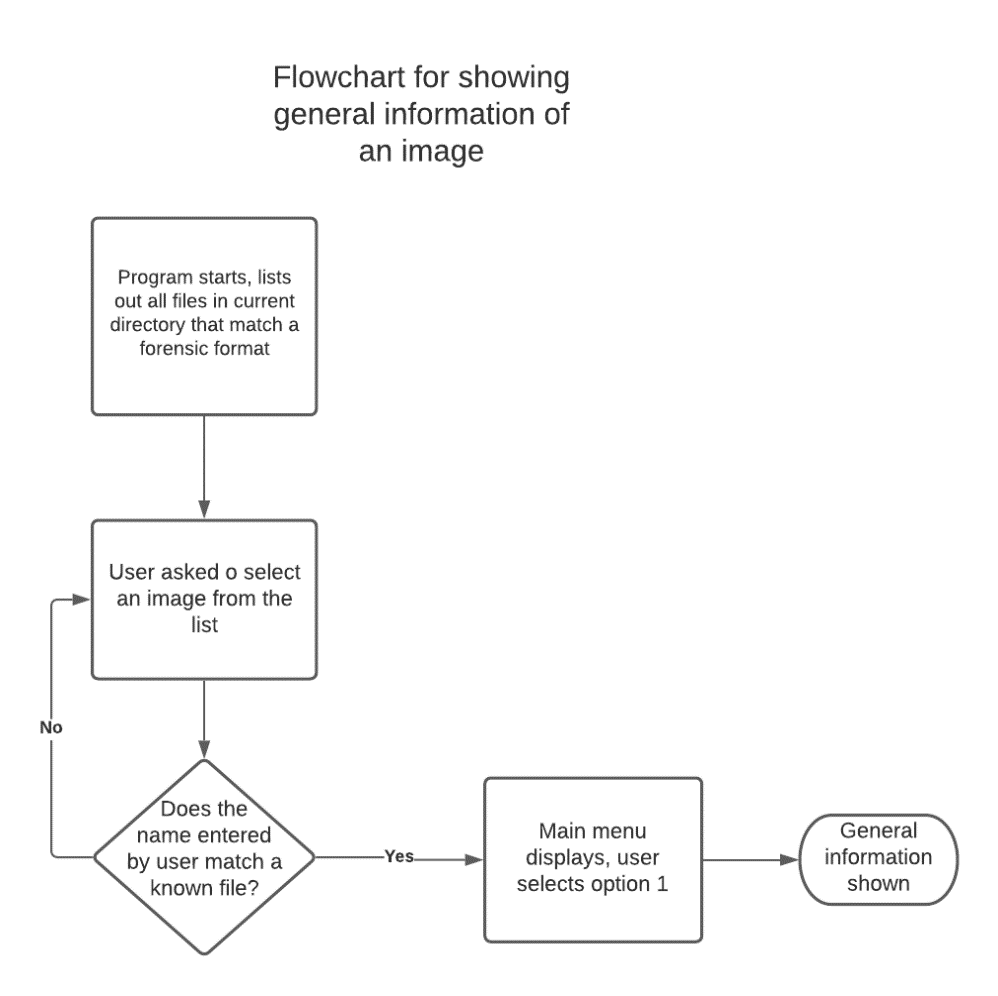
To summarise, the report program uses Pythons CSV module to take in data as an argument and turn that into a csv file report. CSV was chosen because it is simple and easy to read.

# Section3: GUI design

# Section 4: GUI implementation & explanation

# Section 5: Supporting files

# Section 6: Flowcharts



This flowchart shows the process of showing general information about the image like its name, size, cluster count and unallocated space. It comprises of 5 steps.

This is a larger flowchart; it shows the process of going from the main menu to viewing file metadata from a user selected file.

Diagram

Description automatically generated

# Section 7: Testing