**Genetic Algo**

**ImportExportHandler**

The code starts by importing the required libraries. It then creates a function called getCSVFile which takes in two parameters: type and fileName. the primary parameter is the name of the file that will be created, and therefore the second parameter is a string representing where to save it on your computer. If you pass in None because the value for fileName, this suggests that you want to create a new CSV file with no filename yet assigned to it. the subsequent line of code creates an instance of QFileDialog which will return an open File Name dialog box if there are any files selected from your computer's hard drive or other storage devices (such as USB drives). If not, this line returns False meaning that we cannot create our CSV file at this point because there are no files available on our computer's hard drive or other storage devices (such as USB drives). Next comes another function called getOpenFileName which accepts three arguments: None, '', and 'CSV File (\*.csv)'. This function simply exposes a window where users can select one or more files from their computers' hard drive or other storage devices (such as USB drives) so they can be imported into Python using csv library functions like reader().  
The code may be a function that returns True if the user has selected an existing file. The code may be a function that saves the content of the CSV file to disk.

**ResourceTracker**

The psutil library is imported first in the code. The percentage of CPU use on a single core is given via the getCPUUsage() method. The getMemoryUsage() function returns the amount of memory used in megabytes and the percentage of total memory that is still available. Bytes are converted to megabytes using the byteToMegabyte() function, rounding up as appropriate. The code then used these routines to determine how much RAM is being used and what proportion of the entire RAM pool it represents (the MemoryPercentage). The entire amount of memory being used by the system will be returned by the function. The code will output the machine's memory utilisation rate.

**Share**

The The code starts by creating a cursor object. The cursor is then wont to execute the SQL statement "SELECT id, name, subjects FROM sections WHERE active = 1 AND id != ?". this may return all of the rows from the sections table where the active column has a value of 1 and that row's ID does not equal self.section\_id. Next, we use another SQL statement to urge all of the rows from our sharings table where subjectId equals self.id and final equals 1. The code ends with an execution that uses a get statement to get all of the rows in our sections table where subjectId equals self.id and final equals 0 (meaning it hasn't been deleted). The code attempts to list all the sections of a course with a lively status. If the section doesn't exist, then it'll create a new section and populate it with data from the database. If the section exists, then it'll execute a query to find out which courses share that particular subject.  
The code starts by creating an inventory of sections. It then creates a dictionary called sharings that maps the section ID to the JSON string representation of the sharing. The code then iterates through each section and adds it to its own row within the model, which is an object representing all data about this project. The finish() function checks if there are any selected indexes on the tree view and returns false if there are not any . If so, it gets a connection from db with conn and executes an INSERT statement into sharings for self's ID as subjectId and self's section\_id as value . Then it sets self's shareID adequate to lastrowid from cursor . Otherwise, it gets subjectID , text , and split values for every index on the tree view item (which represents one row) using getItemText() method on model .  
The code attempts to make a list of sections and the corresponding sharing information. the primary thing that happens in this code is that a list of all the sections are created. Then, for every section, an inventory of shares associated with it is created. Finally, for every share, an insert query is executed into the sharings table to feature the share's subject ID and section ID to the sharings table.

**Settings**

The code starts by declaring a category called Settings. The settings are stored within the file "settings.json". so as to read from this file, it's opened with open('settings.json') as json\_file: self.settings = json.load(json\_file) The code then uses the load() method of the JSON library to read altogether of the settings from this file and store them into an instance variable called "self." This instance variable is then used throughout the program for various purposes like setting values or retrieving values that have been set previously: def getSetting(self, key): return self.settings[key] The code will parse the settings.json file and return a dictionary of all settings. The code above will set the worth for each key in the Settings class to its corresponding value from the settings.json file.