Some notes for the "migrator" function (in "class startMigrateScreen(Screen)

1. Working Principles
   1. In theory, a normal "migrator" will consist of 3 for-loops:
      1. 1st for loop: helps seek (iterate) through each directory in the source directory list (sourceDirList)
      2. 2nd for loop: picks each file type variations of 1 file type (e.g. in **executables** file type will be **\*.exe, \*.msi... type variations,** and get ready for the next loop.
      3. 3rd for loop: moves all files corresponding with each file type variations mentioned above (\*.exe,\*.msi) to the destination folders.
   2. However, there are a few problems:
      1. Since we are dealing with GUI programming, using normal loops (for,while) will not be possible, as this will make the application freeze **(only when the loop finishes can the program continue running.) => Need of recursion + Kivy's Clock.schedule\_once.**
   3. A recursive function of the above scheme (with the **image migrator** as an example):
      1. **directorySeekerForImages (1st function):**
         1. Function: Picks 1 directory for each in the source directory list (self.dirs), and after all files (with all type variations (\*.img, \*.png..) of "images") are migrated, it will move onto the next directory and repeat the process. It also initiates some index values for inner loops.
         2. Components/Variables:
            1. **self.dirs = configData["sourceDirList"]:** Gets a list of all source directories stored in the configurations.json file.

**Self.dirsIndex = 0**: An **index** of the **self.dirs List,** allowing the function to move onto the next directory in the **self.dirs** list, after each successful migration of all files in the previous directory.

**self.currentDir = self.dirs[self.dirsIndex]: Picks out each source directory** (in string-directory format) based on the index.

**Self.dirsLength = len(self.dirs):** Returns the length of the self.dirs list. This acts as a limit to inform self.dirsIndex when to stop, or else an infinite recursion will occur.

* + - * 1. **Self.imageFormats = self.fileTypes[4]:** A list contains all variations (user defined) of the image file type (e.g. ["\*.png","\*.jpeg","\*.jpg"]

**self.imgTailIndex = 0:** An index of the **self.imageFormats** list, allowing the 2nd recursive function (imageMigratorPerImgType) to move onto the next file type, after each success migration of all files of the previous file type. It is not supposed to be put in the 2nd recursive function (imageMigratorPerImgType), as it will be reset back to 0 after each recursion -> Can not move on to next file type.

**Self.imageFormatsLength = len(self.imageFormats):** Length of the self.imageFormats list, acting as a limiter for **self.imgTailIndex,** to prevent Python's "listIndexOutOfRange" error.

* + - * 1. **Self.imgFilesPerTailIndex = 0 *(1):*** An index for the **self.imgFilesPerTailList.** It is put outside of the second loop as if not, it will be reset back to zero after each recursion of the 2nd function.
    1. **imageMigratorPerImgType (2nd function)**
       1. **Function**: **Selects 1 image file type variation** (e.g. \*.jpg in \*.jpg,\*.jpeg...) per recursion, and also **generates a full list of that image file type variation (\*.jpg)**, which helps the 3rd function to move files of **that image file type variation.**
       2. **Components and variables:**
          1. **Os.chdir(self.currentDir):** change to the current source directory. The reason of having 2 os.chdir in this and 3rd function, is that, **this function's os.chdir prevents the skipping of the first file type variation** (e.g. in [\*.png,\*.jpg,\*.jpeg], it skips \*.png files.)
          2. **Self.imgTailType = str(self.imageFormats[self.imgTailIndex]):** picks the image file type variation based on index (e.g. index 0 in [\*.png,\*.jpg,\*.jpeg] -> \*.png)
          3. **Self.imgFilesPerTailList = glob.glob (self.imgTailType):** generates a list of files based on the current image file type variation **(e.g. generates a list of all image files ending with \*.png)**

**Self.imgFilesPerTailIndex (1):** helps the function to move onto next files in the list. The reason for putting this outside this function is explained above (check similar numbers with similar colors)

**Self.imgFilesPerTailListLength = len(self.imgFilesPerTailList)**: acts as an index limter for **self.imgFilesPerTailIndex.** Also prevents Python's "listIndexOutOfRange".

* + 1. **imageMigratorReal (a.k.a The Real Deal - 3rd function)** 
       1. **Function:** Moves files from predetermined sources in 2nd function to the final destination (determined in another function.)
       2. **Components and variables:**
          1. **eachFileName = self.imgFilesPerTailList[self.imgFilesTailIndex]:** Gets each file's name in the self.imgFilesPerTailList.
          2. **shutil.move (str(eachFileName),self.imagePath):** Moves each file from original directory to destination, based on its name and the imagePath created earlier in another function.
          3. **Self.counter = self.counter + 1:** Counts the number of files moved, and also helps fill up the progress bar + percentage on the screen.
          4. **Self.progBarValue + self.percentage:** Responsible for changing the progress bar and percentage on the screen, based on **self.counter.**
          5. **From image to self.displayList...:** Basically adds a card of the file name on the screen
          6. **Try...except:** Handles (prevents program from crashing) these errors: NoFileFound and possibly other errors. (to be updated.)
          7. **If else blocks:**

From **"If self.imgFilesPerTailIndex < ..." to "...self.imgFilesPerTailIndex + 1"**

**1st check:** Checks if there are any files in the **self.imgFilesPerTailList.** If **Yes**, updates the index (moves onto the next file for migration). If **No,** then do **2.**

**From "else:..." to "self.directorySeekerForImages()"**

If **1.** Fails, then:

Stops the 3rd function (stop migrating)

Resets self.imgFilesPerTailIndex to 0 (or else it will run out of bound (larger than self.imgFilesPerTailList's length).

**2nd Check:** Checks if there are any **image file type variations** (e.g. \*jpeg, \*.jpg) remaining in **self.imageFormats list.** **IF YES,** then moves onto next type variation **(self.imgTailIndex = self.imgTailIndex + 1),** and return to the 2nd function to create a new list of files based on that next type variation (e.g. \*jpeg). **IF NO, then moves onto another directory (self.dirsIndex = self.dirsIndex + 1),** and return to the 1st function to assign the next source directory.

* 1. Function workflow
     1. **Step 1: In directorySeekerForImages function:**
        1. Get current source directory (self.currentDir)
        2. Initialize imgTailIndex = 0 (helps move onto other image file types later on.)
        3. **Initialize imgFilesPerTailIndex = 0 (**helps move onto other image files in the self.imgFilesPerTailList)
        4. **Initialize self.imageFormatsLength = 0 (**sets the index limit for imgTailIndex)
        5. **Heads for the 2nd function (2nd step)**
     2. **Step 2: In imageMigratorPerImgType function:**
        1. **Change to the current source directory**
        2. **Selects an image file type (**self.imgTailType =str(self.imageFormats[self.imgTailIndex])
        3. **Creates a file list based on that image file type.** (self.imgFilesPerTailList = glob.glob(self.imgTailType)
        4. **Gets the length of the "just" created list** (self.imgFilesPerTailListLength = len(self.imgFilesPerTailList)
        5. **Heads for the 3rd function (3rd step)**
     3. **Step 3: In imageMigratorReal function:**
        1. **Gets each file name in the self.imgFIlesPerTailList**
        2. **Move it to the destination.**
        3. **Updates the percentages + Adds file name's card to the list on the screen.** (from self.counter to self.displayList.add\_widget(self.slot)
        4. **Enters the conditional checks:**
           1. IF THERE ARE FILES LEFT in self.imgFilesPerTailList, **updates imgFilesPerTailIndex by 1 (move to next file),** and return to step 3. IF NONE **(no files left)**, **go to 4b.**
           2. **Stops imageMigratorReal function**, and **resets imgFilesPerTailIndex** (goes back to the list's beginning) IF THERE ARE IMAGE FILE TYPES LEFT in self.imageFormats, updates imgTailIndex by 1 (move to next image file type), and return to step 2. IF NONE (no image file types left), go to 4c.
           3. **Moves on to another directory, then return to step 1. If there are none of them, then it's over** (move onto other functions.)
  2. Some challenges -> Experiences
     1. At first, I decided to put in some **Clock.schedule\_once** within each recursive functions, then I realize that this will skip inner functions randomly.
        1. E.g. putting Clock.schedule\_once in 2nd function, which is right below Clock.schedule\_once of 3rd function, results in 3rd function being skipped.
        2. FIX: When I tried deleting Clock.schedule\_once in 2nd function, Clock.schedule\_once in 3rd function works normally. =>**There can only be 1 Clock.schedule\_once RUNNING in the function sequence, and they should be at the FINAL FUNCTION of the function sequence. This also means that all updates (indices of 3 functions should also be updated at the FINAL FUNCTION of the sequence).**