Note: the reason that images are not included in tkinter object initialization but rather added later is because otherwise the image that is supposed to be shown in the button or label gets removed by garbage collection
https://stackoverflow.com/questions/47138691/tkinter-image-is-blank

The GUI Class
Purpose: to link together the various classes that make up the GUI, and maintain a persistent state within the GUI - so all parts of the GUI know when the tranplanter is stopped or paused.
The GUI class is a composite of six other classes which contain various attributes of the GUI. It also contains the two booleans `is_paused`, and `is_stopped`.
The `is_paused` variable indicates that the transplanting is paused while the person monitoring the transplanter replaces the tray. The `is_stopped` variable indicates that the user has pressed the `Stop_Button` object and it is time for transplanting to end. The GUI receives these booleans via the ever-looping while loop in the main function. `is_paused` is returned by the `Tray` objects earlier in the main function, and `is_stopped` is returned by the `Stop_Button`. The GUI then returns the `is_paused()` and `is_stopped()` via the `get_is_paused()` and `get_is_stopped()` functions. They are both sent via the main loop into the `Start_Continue_Button` object (the effects of this are elaborated on more in the `Start_Continue_Button` class just below). The `is_paused` boolean is also passed to the `Window` class, which makes the window change from green to red, signaling to the person monitoring the device that one of the trays must be refilled.
Other than the two getter functions, there is one other function: `update_GUI()`. This is called at the very end of the main loop and comes originally from the `Window` object and instructs the tkinter window to display for another frame.

The Start_Continue_Button Class
Purpose: To provide a tkinter button object that switches between being a 'start' button which creates a separate transplanting thread, and a 'continue' button which makes the transplanter continue after being paused when a tray is full.
This button is a tkinter button instance which is set to a start button at the beginning of the program. When pressed, it creates a new thread. The new thread is necessary because the transplanting process must take place simultaneously with the GUI being displayed. This thread runs `start_transplant_function` which comes from the `Transplanter` object. The `start_transplant_functionn` is passed to the `Start_Continue_Button` class in the main file during the initialization process. It is only passed once. That function is the main function for all of the transplanting, which deals with the pauses (caused when the tray is full) and stops (caused when the user presses the stop button). After the button is pressed, the tkinter "disable button" feature is used to gray it out and ensure it cannot be pressed again. If the tray is ready to be reset, then the `set_to_paused_mode()` function is called in the while loop in main. Then the start button turns to a continue button. When the button is pressed as a

continue button, rather than making its own new thread it simply continues the old thread from where it left off. It also relabels the continue button so the message indicates that the tray needs to be replaced. There is also a `set_to_stopped_mode()` which resets everything and turns the button back to what it was originally, where it creates a new thread. And finally, there is a `__continue_tranplanting` which is a private function called in `set_to_stop_mode()`. A new function is needed because when the tkinter button is configured, a function needs to be input into the button to show what happens when it is pressed. And multiple things need to happen when the transplanting continues: first, the transplanter itself needs to be told to continue. Secondly, the button needs to change shape and structure. The button changing text and everything happens in the `__continue_tranplanting()` function.

The Stop_Button Class
Purpose: Provides a stop button so the user can shut the whole thing down whenever they need - this cannot shut it down in the middle of an arduino movement, it shuts it down AFTER the arduino movement is done happening.
The stop button takes in the stop function from the Transplant object, and it is sent into the stop button during the initialization process in main. It is always enabled, and always set to 'stop'.

The Toolhead_Illustrator Class
Purpose: The 2D image of how the toolhead is supposed to be moving.
The toolhead illustrator uses the turtle feature of TKinter. It contains a canvas which is placed in the `tkinter_instance` at the beginning, and when it is changed, that is displayed on the GUI because the GUI is given a reference. So even though updates aren't called for it, the turtle still moves in real time. The `turtle` object is a small 'turtle' object which is shaped like a small black circle (not a turtle - look I didn't design tkinter idk why they named it turtle). The location is updated in the loop in main via the `update_location`() function. You put the value into the function and it moves the turtle to that location. There are two rectangle images on the canvas which do not move, they just represent two trays to give the viewer a better sense of where the toolhead is in comparison to the tray. There are also grids which show which hole the toolhead is at.

WindowMaker
Purpose: Contain all the tkinter junk that the other parts of the GUI don't need to see
The purpose of the window class is mostly to bundle all the 'junk' that comes with using Tkinter. There needs to be a `title` which appears on the upper bar and says the name of the app, a `window_length` and `window_width` which set the default sizes of the GUI window, a `window` object which is just a gui window that needs to be updated every frame or it will disappear, an `update()` function which is called every frame by the GUI, an `instruction_label` that gives some VERY BASIC (1-2 sentence) instructions on how to use the GUI and gives credit to us and Lieth, and the `tkinter_object` which is necessary to do

anything with tkinter. It makes linter angry because it has zero public methods, but it is cleaner to make it it's own class imo.

Toolhead_Location_Label
Purpose: A more precise version of the toolhead illustrator - it provides exact coordinates for where the toolhead is.
This gives the exact position of the toolhead. It is more precise than the 2D image, but less intuitive. It gives the value sent to the arduino, rather than the input from the JSON file, because it is more useful for debugging. The `label_instace` is the tkinter Label object. The `update_location()` updates the text in the label. It is initialized in the main function and updated in the while loop.

Port_Name_Label
Purpose: Two labels that list the ports that the client is connected to which allow it to communicate to the arduinos.
The port name label class contains two TKinter Labels that indicate which port the arduinos are connected to. This is useful for debugging because the port connection system is very hard to automate. Each OS does things very differently, and has different port names. It would be ill advised and probably somewhat dangerous to try to force the computer to connect if the port connection folders are private, or can't be found, etc. So it is better to simply display the port names on the corners, and the user can debug from there. If it can't connect to the port, it says "no connection". This can mean that the user is simply looking at the GUI without the arduinos in it to see how it functions. The `toolhead_arudino_port_label` object and the `frame_arudino_port_label` is created once during initialization and is fed the names of the ports that come out of the `Toolhead_Arduino` and `Frame_Arduino` classes and from there it is never changed. These variables are decorated with property tags so they can be accessed but not altered.

Relocate plant
Purpose: Takes plant from point a, puts it at point b
Transporting plants actually takes quite a few steps. You have to go behind the first cup, lower the toolhead, go to the cup, raise the toolhead, go to the cup, lower the toolhead, go behind the cup, and raise it. So there is a unique class which has the functions for all of the above motions passed into it during its initialization in the main function which is called `transport_plant()`, as well as a `reset_transplanter()` which raises the toolhead and goes to the origin.

`Transplanter`
Purpose: Handels the starting, stopping, and pausing part of the transplanting and gives the functions to the start and stop buttons. During initialization in main, it takes in the `next_hole()` function from the tray objects for both the source and the destination trays and sets them to the `next_source_hole()` function to find the next tray hole to take the plant pot out of, and the `next_destination_hole()` to find where to put the lettuce plant. It also takes

the `is_tray_full()` function to determine whether the function should continue to run, and if the `is_paused` boolean in the `GUI` class should be set to true. It takes the `reset` and `transport` functions from the `relocate_plant` class. It then handles the starting, stopping, and pausing. It does this by creating the `transplant` function which is run on a separate thread by the `start_continue_button` class, a `continue_transplant` function which is called by the `start_continue_button` when it is in continue mode, a `stop` function which is sent to the `StopButton` class, and a `restart` function which is called in the `stop_transplanter_handler` in the main function.

Arduino

Purpose: The arduino class is an abstract class which controls what is being sent to the two arduinos. It contains an `arduino_conection` object which is the serial connection between the client and the arduino. It is established during initialization using the serial number of the arduino (which is contained in the config file and fed in during the arduino's creation in the main function). The client's ports are scanned to find which port, if any, is connected to an arduino with that serial number. The connection is then stored in the `arduino_conection` object. It also contains a `port_name` string which is determined when the connection is being created - it contains the name of the port that the arduino is connected on. This name is later displayed on the gui for the purpose of debugging. It contains a `calibrate()` function called at the very beginning when it is initialized. For now, all that the `calibrate()` function does is send the string 'calibrate' to the arduino which will run the calibration code. However it is likely that it will become more complex in later iterations so I'm making it its own function. There is also a `signal_arduino()` function which sends the input value to the correct arduino. This function is called every time one of the child classes wants to communicate with the arduino. The child classes contain information about the specific string being sent to the arduino, but all the information about converting to bytes, encoding, etc, is stored here.

Frame Arduino

Purpose: This controls the arduino that moves on the xy plane and is attached to the frame. It handles the conversion from absolute coordinates to frame coordinates. The `distance_to_lift_cup` value which is set by the config file determines how far behind the cup the toolhead should be when it is lowered. The toolhead needs to be slightly behind the cup so that it does not just go down directly on top of it, but rather it goes behind it and scoops it up. This is also necessary when dropping the cup so that it can slide out from underneath it. It also has a `name` which is just for ease of debugging. When you want to see which arduino is causing the issue, it's easier if you can figure out its name rather than a serial number. There are three different functions for moving it to given coordinates. The first is `go_behind_cup()` which takes in a tuple of coordinates, and goes a few millimeters behind that location (behind being determined by the y axis). After that it will be lowered by the toolhead arduino and then the function `go_to_cup()` will be called, sending it directly to the location of the cup it's trying to lift. Then finally there is the go_to_origin() function which sends it back to the middle of the fist tray hole. This is called at the very end of the transplanting process.

Toolhead Arduino
Purpose: This controls the arduino that moves the toolhead up and down. It has 2 functions, one for sending the toolhead to go up, which is represented by 1, and going down represented by 0. It also has a `name`, for debugging purposes.

Tray
Purpose: Tell everything else which is the next hole, and if the tray is full
The tray function receives a json file on initialization which contains tuples representing the locations of the holes. These are the absolute coordinates so (0,0) is the absolute corner of the tray, NOT the original location of the frame arduino. The json file is represented by the `hole_location_map`, which is a map linking the nth hole to the nth hole's location. The `number_of_holes` variable indicates how many holes are in the tray, which is calculated from the json file. This is used in the `is_tray_full()` function which creates a boolean which is sent to both the `GUI` and the `Transplanter` objects so they can determine when to wait for the trays to be replaced. The `hole_itterator` attribute keeps the `get_next_hole()` function incrementing so that the hole location it returns is incremented every time. The `get_next_hole()` function is called every time the main file loop runs and the next hole is sent to the `Toolhead_Illustrator` object, and the `Toolhead_Location_Label` objects.