**Class** KeyboardHandler(object)

Class Variables

* self.abc

-Set containing ASCII values of A-Z letters.

* self.WC

-WordCompleter object.

* self.buffer

-Stores string of the letters entered so far.

* self.hm

-pyHook hookmanager

* self.suggest\_word

-The word suggested by WC to complete the buffer.

* self.spaces

-Boolean describing whether or not to add a space after sending a word.

* self.finished

-Boolean describing whether the message pump loop should exit.

* self.GUI\_q

-Queue object used to communicate with the GUI.

Methods

* \_\_init\_\_(self, WordCompleter, gui\_Q, SPACES)

-Sets class variables to parameters passed in, sets up defaults for others.

* begin(self)

- Hooks the keyboard KeyDown event with callback, and starts message pump.

* OnKeyboardEvent(self, event)

-Called by pyHook on KeyDown.

* set\_to\_end(self)

-Sets self.finished to True.

* end(self)

-Unhooks the keyboard, tells the GUI and WC objects to end, and ends own message pump loop.

**Class** WordCompleter(object)

Class Variables

* self.pkl\_filename

-What to name the pickle file used to store word frequency data.

* self.current\_words

-A set used to keep track of the dictionary words available, i.e. ones that haven't been already checked and discarded as possibilities by find\_mfw().

* self.word\_freq

-A dictionary with words for keys and frequencies for values.

Methods

* \_\_init\_\_(self, pkl\_filename)

-Loads the word\_freq dictionary, or creates a default one with all frequencies set to 1.

* find\_mfw(self, string)

-Returns the word which the given string could become, with the highest frequency.

* add\_word(self, word):

-Increases the frequency value for the [word] key by 1. If the key doesn't exist, it will create one.

* clear\_current\_words(self):

-Resets the self.current\_words set to an empty set.

* end(self):

-Saves self.word\_freq to a pickle file, with name self.pkl\_filename.

**Class** WSGui(threading.Thread)

Class Variables

* self.root

-Tkinter Tk() root instance.

* self.dcm

-Boolean value for Data Collection Mode.

* self.q

-Queue object to get() from.

* self.fg

-Foreground color, i.e. what color the text displayed will be.

* self.textvar

-Tkinter StringVar used to store the text to display.

* self.label

-Tkinter Label used to display the text.

Methods

* \_\_init\_\_(self, bg, corner, dcm, fontsize, txtcolor, Q, height=2, width=40)

-Sets the class variables up, sets the root window to always be on-top of other windows, but be non-interactable, creates and packs the Label into the root window, and sets the root window position to one of the four corners of the screen.

* run(self)

-Sets up the Queue polling, and then runs the root mainloop.

* poll(self)

-Hangs getting a value from the self.q, then reacts to that word by updating its text, configuring its options, ending, or flashing the text a different color momentarily. Then it sets up to call itself after a delay.

* update\_word(self, word)

-Changes the text displayed to (word).

* update\_config(self, options\_l)

-Updates the label configurations based on the values in the options list.

* def flash\_color(self, color="blue", color\_opt="black")

-Sets the text to the color passed in, or blue as default, then calls \_reset\_color after 100 milliseconds.

* def end(self)

-Destroys the root, closing the mainloop.

**Class** SysTrayIcon(threading.Thread)

Class Variables

* self.icon

-The icon to display in the task tray.

* self.hover\_text

-The text to display when cursor is held over the icon.

* self.on\_quit

-Function to run when quit button is selected.

* self.KbH

-KeyboardHandler object.

* self.\_window\_class\_name

-Name of window class.

* self.hwnd

-Windows handle of tasktray icon window.

Methods

* \_\_init\_\_(self, keyboard\_handler, icon, hover\_text, menu\_options, on\_quit=None,

default\_menu\_index=None, window\_class\_name=None)

-Creates and registers the Window class, and parses menu\_options to create selectable options.

* run(self)

-Create the window, and set the infinite message pump loop.

* refresh\_icon(self)

-Try to find a custom icon in the same folder as the script or .exe, and load it. Otherwise, use a Windows default icon.

* restart(self, hwnd, msg, wparam, lparam)

-Calls refresh\_icon

* notify(self, hwnd, msg, wparam, lparam)

-Callback to properly react to button clicks.

* show\_menu(self)

-Shows the popup menu when icon is right clicked.

* create\_menu(self, menu, menu\_options)

-Setup the popup menu behind the scenes.

* prep\_menu\_icon(self, icon)

-Load the icon, fill in the background color, and draw the object.

* command(self, hwnd, msg, wparam, lparam)

-Callback which identifies the menu option selected and then calls execute\_menu\_option.

* execute\_menu\_option(self, id)

-Call the function passed in corresponding to the appropriate menu option.

**Class** OptionsWindow(QtGui.QWidget)

Class Variables

* self.q

-Queue variable to communicate with the WSGui object.

* self.reg\_data

-The string value to place for the value of the Windows startup registry key.

* self.systray

-SysTrayIcon object.

Methods

* \_\_init\_\_(self, Q, sysTrayIcon)

-Sets up class variables, and calls initUI().

* initUI(self)

-Create all of the comboboxes, checkboxes, text descriptions, and tooltips used. Construct the options window according to the user's screen dimensions, and then lay all buttons out properly in that window. Then show the window.