**Cassandra Pizarro**

**Issue 16: Support ANSI escape codes**

This issue calls for supporting ANSI color coding. However, the ncurses library itself does not support ANSI coloring. Curses does, however, have its own built in colors.

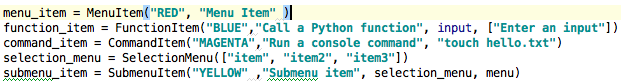
Curses has 8 standard colors:

|  |  |
| --- | --- |
| Color | Curses naming |
| Black | curses.COLOR\_BLACK |
| Red | curses.COLOR\_RED |
| Green | curses.COLOR\_GREEN |
| Yellow | curses.COLOR\_YELLOW |
| Blue | curses.COLOR\_BLUE |
| Magenta | curses.COLOR\_MAGENTA |
| Cyan | curses.COLOR\_CYAN |

These are initialized when you call curses.initscr() and curses.start\_colors(), or you can call curses.wrapper() which will initialize both those functions.

So that a user does not have to dig through to find the function within the curses\_menu.py file where the colors are actually stored, there are 4 global variables at the top of the curses\_menu.py file in which they can just set the background font color, highlighting foreground font color, the highlighting background font color there, and the exit menu item color. To set the font foreground color for each menu item itself, this must be done in the testing file used to specify your menu build.

**Take for example:**

*From testing\_curse.py*

*Global variables in curses\_menu.py*

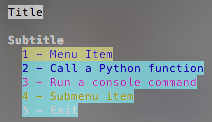
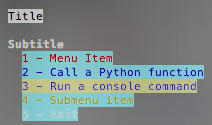
font\_background\_color = "cyan"

highlight\_foreground\_color = "blue"

highlight\_background\_color = "yellow"

exit\_color = "white"

*Your output should look like this:*

*Here you have the text colors varying by item on the menu.*

As you can see, the text colors are set in the testing file as the first input for each menu item (excluding the selection menu items and exit item) where you build the menu.

The background font color for each menu item is set globally in the curses\_menu.py file, in most cases a user would generally set it to black or the color of your terminal in which it would appear that only the text color would change. This way it doesn’t interfere with the color scheme of the highlighting colors, but I wasn’t going to assume that a user would never want to change the font background color so user’s choice!

The colors set for highlighting are also globally set in the curses\_menu.py file. In this scenario, the highlighting color for text is blue and the background is yellow. As for the colors of each item set in the testing file, there is a function in the Menu Item class that will correspond a user’s input string color to the appropriate curses color.

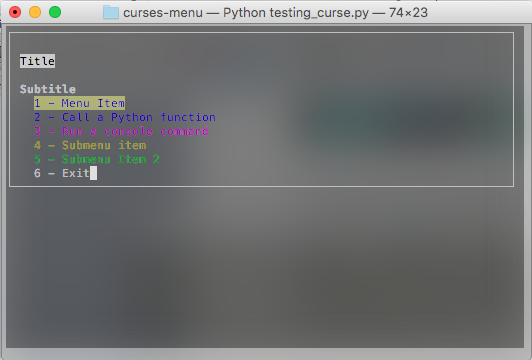
The color input can be written with either uppercase or lowercase. The naming convention for the global variables in the curses\_menu.py file is the same as in the testing file. You can simply write “Green” or “BLUE” or “red” and it will be converted to the appropriate color.

**Issue #25: Filter menu by user input**

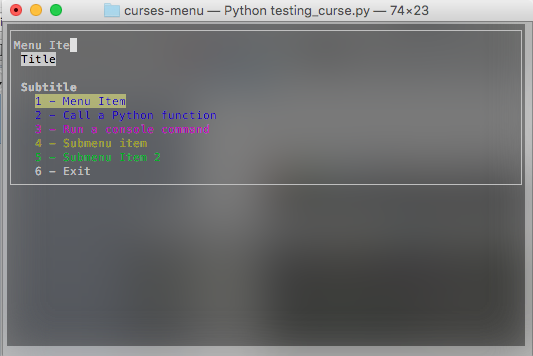
This issue calls for being able to filter the curses menu built from your testing file according to user input. Filtering is automatically enabled when you are in the menu.

**Take for example:**

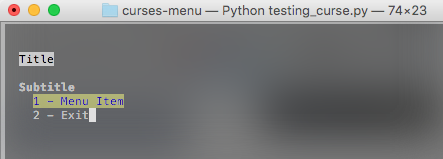
*Here is a menu newly generated, no characters have been input to filter yet.*



*Now we are currently trying to write “Menu Item”, the first item in our menu.*



*Once we complete that string the menu will filter to look like this:*



There are some things to take note of in regards to the filter:

* The filter does allow backspaces. The cursor will simply move back one and hover over the last character indicating it can be overwritten but the character will not disappear until you enter another character. When backspacing, when you reach the first character in the string the cursor will not hover over it but it can still be overwritten.
* Once a menu is filtered, there is no way to return to the original menu. If you filter in a submenu you can return to the main curses menu but if you go back into the submenu you will only get the filtered item, not the entire submenu.
* Once you begin entering characters to filter, they should appear at the top of the menu directly above the title string. However, once you begin to type in a filter string, navigating the menu by numbers is no longer available until the filter string is empty.
* With regards to the multi-select menu, if you wanted to filter there you must include the [ ] with the name of the item to filter properly.
* In order for the filter to retrieve the item, you must match the case that the menu item is written in. For example, if you create a menu item called “Call a Function” then in order to filter to find that particular item you must type in “Call” and “Function”, not “call” and “function”.

**Testing:**

**Issue 16: Coloring**

There are two scenarios for this particular issue. They both follow the same basic outline as far as steps, the only difference is that one tests change in color for individual menu item versus the entire menu changing color. In writing and executing this this test, I have done a hybrid of things:

* Step: Colors enabled on terminal
  + I have utilized the curses library itself to check if the terminal has colors enabled.
* Step: Creating item(s)
  + These are essentially excerpts from the testing\_curse.py file that shows the line(s) of code that are setting the menu item(s). As I can’t physically show you the creation of these menus (this really happens in the next step), I have just asserted true that these are built.
* Step: Menu appears
  + I have utilized, in a separate file, a function to compare images.\*
  + I have compared what the terminal looks like before I run the command “python3.4 testing\_curse.py” and after in which the file is ran and the menu appears.
* Step: Seeing that the color has changed in the menu
  + I have utilized, in a separate file, a function to compare images.\*
  + I have compared a black and white item/menu to that of an item/menu utilizing other colors.
* \*The comparison of these images is done in the compare\_images.py in the equal function (courtesy of Nllesh Sharma on Stackoverflow).
  + This function performs a quick way to determine if two images have exactly the same contents. To do this, the function gets the difference between the two images, and then calculates the bounding box of the non-zero regions in this image. If the images are identical, all pixels in the difference image are zero, and the bounding box function returns None. If the function returns True, the images are the same and False if they are not.

**Issue 25: Filter menu by user input**

This test really relies on assumptions. In a general sense, you’ll be assuming that the images contain what they say they do. If I’m comparing an image called “unfiltered\_menu” and “filtered\_menu”, then you will make the assumption that I am actually comparing a menu that hasn’t been filtered against one that is a new filtered menu. The images do actually contain screenshots of the appropriate content according to each step, as you can click and see for yourself, but, also note, for the tests we’re comparing screenshots of the menu versus an actual live menu currently running on bash. For these reasons, I changed from previously having a scenario outline to one simple scenario as I could not pull active data in the moment of running this test and from a running curses menu simultaneously. Again, the screenshots do contain the appropriate image according to its respective step.

Again, like in the previous issue, I am utilizing the equal function in the compare\_images.py file that compares and determines if there are any differences between the two images, signifying if they are the same image.