colorise

Release 1.0.1

Jan 09, 2020

Contents

1 Basic Usage 3

2 `colorise.cprint()` 5

3 `colorise.fprint()` 7

4 `colorise.highlight()` 9

5 Attributes 11

6 Disabling Colors 13

7 More Colors! 15

8 Redefining Colors 17

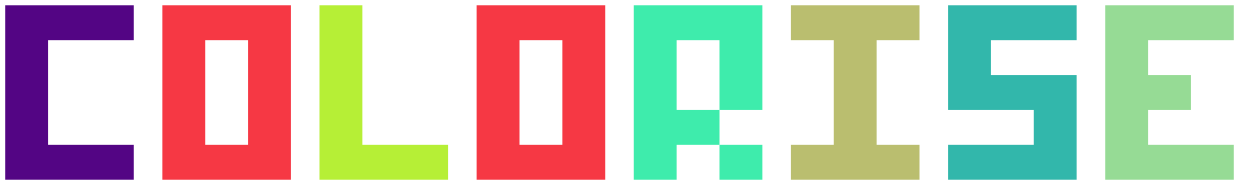
9 Screenshots 19

10 FAQ 21

11 Changelog 23
- 11.1 1.0.1 . . . 23
- 11.2 1.0.0 . . . 23
- 11.3 0.1.4 (pre-release) . . . 24
- 11.4 0.1.3 (pre-release) . . . 24
- 11.5 0.1.2 (pre-release) . . . 24
- 11.6 0.1.1 (pre-release) . . . 24
- 11.7 0.1.0 (pre-release) . . . 24

12 Tested Systems 25
- 12.1 Mac . . . 25
- 12.2 Windows . . . 25
- 12.3 Linux . . . 25

13 Features 27
- 13.1 Indices and tables . . . 27



In addition to this tutorial, you can find examples in the examples folder.

Table of Contents:

1. *Basic Usage*
2. *colorise.cprint*
3. *colorise.fprint*
4. *colorise.highlight*
5. *Attributes*
6. *Disabling Colors*
7. *More Colors!*
8. *Redefining Colors*

CHAPTER 1

Basic Usage

You may be interested to know how many colors your terminal can represent, which you can check using `colorise.num_colors()` which tries its best to guess the number of colors supported.

```
>>> import colorise
>>> colorise.num_colors()
256
```

Returned values may be 8, 16, 88, 256 and 16,777,216 (or 256^3 i.e. 24-bit true-color). You can also run the color_test.py script which prints all the capabilities of your console. To set the current color, you can use the `colorise.set_color()` function. For example,

would set the current foreground color to red while

would set the current foreground color to red and the background color to green. Supported color names can be queried via `colorise.color_names()`.

```
>>> colorise.color_names()
['black', 'red', 'green', 'yellow', 'blue', 'purple', 'magenta', 'cyan', 'gray', 'grey
↪', 'lightgrey', 'lightgray', 'lightred', 'lightgreen', 'lightyellow', 'lightblue',
↪'lightpurple', 'lightcyan', 'white']
```

Use `colorise.reset_color()` to reset colors to their defaults.

CHAPTER 2

`colorise.cprint()`

To print colored text, you can use the `colorise.cprint()` function.

CHAPTER 3

colorise.fprint()

The `colorise.fprint()` function provides more control than `colorise.cprint()` by letting you specify colors akin to Python 3's string formatting.

The `colorise.fprint()` function provides the *autoreset* keyword argument to control if colors should be reset when a new color format is encountered. It is `True` by default.

Notice in the second example that both fore- and background colors are reset. It would correspond to the following example where we explicitly reset all colors and attributes with `{reset}` before setting the foreground color to red.

CHAPTER 4

`colorise.highlight()`

The `colorise.highlight()` function can be used to highlight ranges of characters within a string.

CHAPTER 5

Attributes

Text attributes are supported via the `colorise.attributes.Attr` class.

As for `colorise.fprint()`, you can specify the attributes directly in the format string.

CHAPTER 6

Disabling Colors

It is sometimes useful to disable colors, for example in an application where colored output is controlled by a configuration file. The `colorise.cprint()`, `colorise.fprint()` and `colorise.highlight()` functions all support the `enabled` keyword argument for this purpose. Colors are enabled by default.

CHAPTER 7

More Colors!

Besides named colors, you can also specify colors via color table index, RGB, hex, HLS and HSV. Color indices index into color tables commonly supported by different platforms.

Note: Even if your terminal does not support 88/256 index color tables or true-color, colorise will attempt to approximate the color by finding the closest one (via linear distance) and use that. For example, Windows usually supports only 16 colors but using `colorise.cprint('Hello', fg='rgb(240;240;0)')` on such a system will still give you a yellow color (assuming standard Windows console colors). Also see the mario sprites in the Screenshots section.

CHAPTER 8

Redefining Colors

Some platforms allow you to redefine the standard colors but currently you can only redefine colors on Windows. As an example, let us redefine 'green' (color index 2).

`colorise.redefine_colors()` takes a dictionary of colortable indices as keys and RGB tuples as values. Here, we redefine the entry in the colortable at the color index for green (2) to be magenta instead. This change persists until the color is redefined again or colorise is quit.

Note: Redefining colors does not currently work with ConEmu or on Mac and Linux systems.

CHAPTER 9

Screenshots

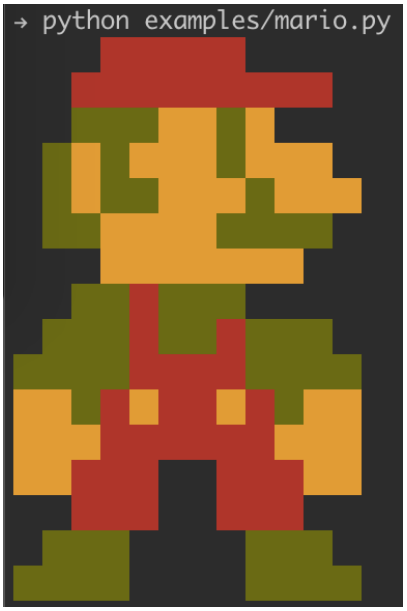
Using colorise.cprint on Ubuntu

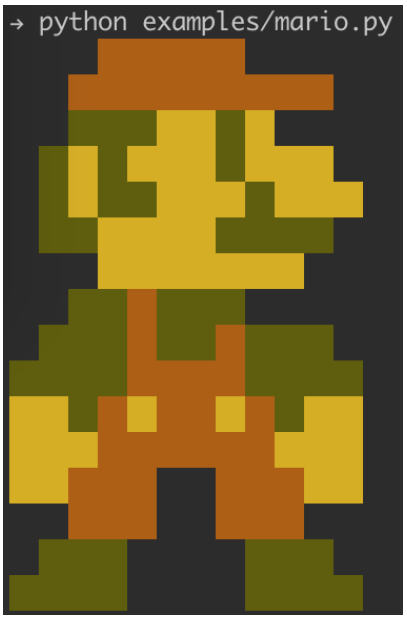
```
>>> colorise.cprint("Error: Expected a string, found int", fg='red', bg='green')
Error: Expected a string, found int
>>>
```

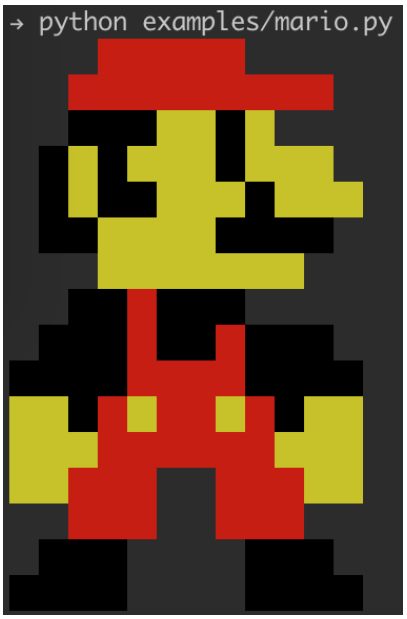
Using colorise.highlight on Windows

```
>>> colorise.highlight("Highlight this text!", fg='blue', indices=[4, 17, 3, 5, 10])
Highlight this text!
>>>
```

From left to right: True-color, 256 color and 16 color




→ python examples/mario.py


→ python examples/mario.py

CHAPTER 10

FAQ

Q: Why do I get different results on different platforms?

Different platforms and terminals have support for different numbers of colors, attributes and colortables, and colorise tries it best to provide uniform results although platform differences makes this hard to do is 100%.

For example, depending on your console/terminal color support, you may have anything from 8-, 16-, 88-, 256-color support or even full-blown 24-bit colors available. If you request a 24-bit color but only have 256 colors, colorise will try its best to approximate the requested color according to the colortable of 256 colors available.

Q: I have custom colors set up in Windows, why are they not reflected in colorise?

You can set up custom console colors in Windows but in order to detect them you need at least [Windows Vista or Windows Server 2008](https://docs.microsoft.com/en-us/windows/console/getconsolescreenbufferinfoex). If you are working on a Windows version before that, your custom colors will not be properly reflected.

Q: How come I can use more than 16 colors in Windows?

Some versions of Windows 10 have 24-bit color support and can interpret ANSI escape codes, the latter which is commonly how colors are emitted on Mac and Linux systems.

Q: Why are the named colors on Windows incorrect?

On Windows, named colors are actually indices into a color table and not actual colors. Typing

```
>>> colorise.cprint('This should be yellow', fg='yellow')
```

will give you the color in the table correspoding to the 'yellow' index, not necessarily the color yellow. You can see the current colors by right-clicking the top bar of the console and selecting 'Properties' then selecting the 'Colors' tab. You can also set these programatically using `colorise.redefine_colors()`.

Q: The blink and italic attributes do not appear to work in iTerm.app?

This has to be enabled manually in the settings in iTerm.app. Go into Preferences → Profiles → Text and check the boxes for "Blinking text" and "Italic text".

Q: Can I use colorise in different threads?

colorise is **not** thread-safe.

On nix systems, colorise emits ANSI escape codes to print colored output. Internally, this happens in a way where multiple threads would interfere although it should be possible to perform this in a thread-safe manner.

On Windows systems that do *not* support ANSI escape codes, multiple threads would also interfere with each other. On Windows systems that *do* support ANSI escape codes, it should still be possible to output colored text in a thread-safe manner.

Q: Why do the tests fail with 'The handle is invalid.' on Windows?

tox and pytest capture stdout and stderr which does not play well with Windows handle creation hence the error. You can tell pytest not to capture stdout and stderr and the tests should run but also show the output of all tests.

```
$ pytest -s tests
```

or

```
$ pytest --capture=no tests
```

Q: Was the colorise logo generated using colorise?

Yes :)

CHAPTER 11

Changelog

Version numbers follow Semantic Versioning (i.e. <major>.<minor>.<patch>).

11.1 1.0.1

2020-01-09

- [fix] Check for duplicate color format specifications. E.g.

```
>>> colorise.fprint('{fg=red,fg=blue}Hello')
```

  This is now an error and raises a `ValueError`.

- [docs] Update the github branch for generating links to source code, that was pointing at an old deleted branch. Update links and code links in changelog.
- [fix] Fix a bug where the `colorise.attributes.Attr.Intense` attribute (alias for `colorise.attributes.Attr.Bold`) would not be recognised by `colorise.fprint()`.
- [refactor] Updated the code for the `colorise.formatter.ColorFormatter` class.

11.2 1.0.0

2019-12-17

Warning: Major update with breaking changes.

- [new] Support for 88/256 colortable indices, and RGB, HSV/HLS and hexadecimal color formats.
- [new] Support for virtual terminal processing on Windows.

- [new] Changed parser to use Python 3's str.format syntax, e.g. `<fg=red>` becomes `{fg=red}`. Removed ColorManager classes since no state needs to be stored. The `colorise.formatter.ColorFormatter` class parses formats.
- [new] Better detection of terminal color capabilities.
- [new] If an unsupported color format is specified which the terminal does not support it (e.g. an RGB color in a 16 color terminal), colorise will automatically find color on your system that matches the desired color (via linear distance).
- [new] More thorough testing.
- [refactor] Reworked entire library.
- [refactor] Removed formatcolor and formatbyindex functions.
- [docs] Online documentation and updated comments.
- Changed license from MIT to BSD 3-clause.

11.3 0.1.4 (pre-release)

2014-06-11

- [Fix] Fixed a bug on nix platforms that caused background colors to break.

11.4 0.1.3 (pre-release)

2014-06-02

- [Fix] Fixed a bug where passing a string without any color formatting would print the empty string.

11.5 0.1.2 (pre-release)

2014-05-31

- [Fix] Fixed a bug in `nix/ColorManager.py` which caused `set_color` to malfunction.

11.6 0.1.1 (pre-release)

2014-05-24

- [Fix] Fixed a bug where putting a `:` or escaped > or < just before or after some color formatted text would raise a `ColorSyntaxError`.

11.7 0.1.0 (pre-release)

2014-05-14

- Initial version.

CHAPTER 12

Tested Systems

This is a maintained list of systems that colorise has been tested on.

Something not working as expected with your terminal? Please report an issue or submit a pull request using the contribution guidelines.

12.1 Mac

Terminal	OS	Python version
iTerm 3.2.9	macOS Catalina v10.15.1	Python 3.7.4
Terminal.app 2.9.4 (421.1.1)	macOS Catalina v10.15.1	Python 3.7.4

12.2 Windows

Terminal	OS	Python version
Default Windows console	Windows 8.1 Pro, version 6.3, build 9600	Python 3.7.3
ConEmu	Windows 8.1 Pro, version 6.3, build 9600	Python 3.7.3

12.3 Linux

None yet.

colorise is a Python module for printing colored text in terminals.

You can install it via pip.

```
$ pip install colorise
```

CHAPTER 13

Features

- Supports 8, 16, 88, 256 colors and true-color.
- Colors can be specified by name, index, hexadecimal, HLS, HSV or RGB formats.
- Custom color format akin to Python 3.0 string formatting.
- Automatically find the closest color based on the terminal's capabilities.

```
>>> import colorise
>>> colorise.set_color(fg='red')
>>> print('This is red')
This is red
>>> colorise.reset_color()
>>> colorise.cprint('This is blue with a yellow background', fg='blue', bg='yellow')
This is blue with a yellow background
>>> colorise.fprint('{fg=blue,bg=yellow}This is also blue with a yellow background')
This is also blue with a yellow background
>>> colorise.highlight('Highlighted', fg='red', indices=[0, 3, 5, 10])
Highlighted
>>> from colorise import Attr
>>> colorise.cprint('Hello', fg='yellow', bg='purple', attributes=[Attr.Italic])
Hello
>>> colorise.fprint('{underline,fg=yellow,bg=purple}Hello')
Hello
>>> colorise.cprint('Hello', bg='rgb(100;95;194)')
Hello
>>> colorise.fprint('{fg=#77acde}Hello')
Hello
```

13.1 Indices and tables

- genindex

- modindex
- search