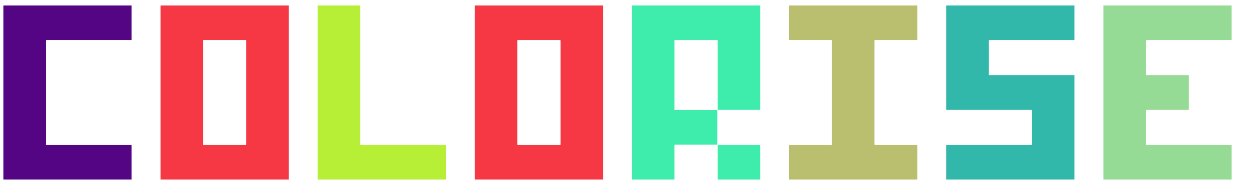
colorise

Release 1.0.1

Feb 22, 2021

Contents



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

Table of Contents:

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,

sets the current foreground color to red while

sets 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', 'lightmagenta', '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.

Note: It is not currently possible to mix color formats and Python's string formatting such as `colorise.fprint('{fg=red}{0}', 'test')`. See this issue if you want to know more or help.

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.

Disabling colors means that *no* ANSI escape sequences are emitted and no Windows Console API calls are made so if a color has been set previously, outputting disabled colors are still affected.

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.

HSV values are hue ([0; 360]), saturation ([0; 100]), and value ([0; 100]). HLS values are hue ([0; 360]), lightness ([0; 100]), and saturation ([0; 100]).

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 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 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 in the *logical color table*).

`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 and automatically restores the original color table.

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

CHAPTER 9

Screenshots

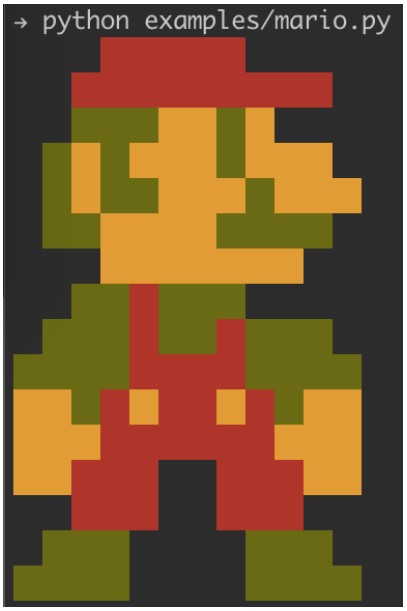
Using colorise.cprint on Ubuntu

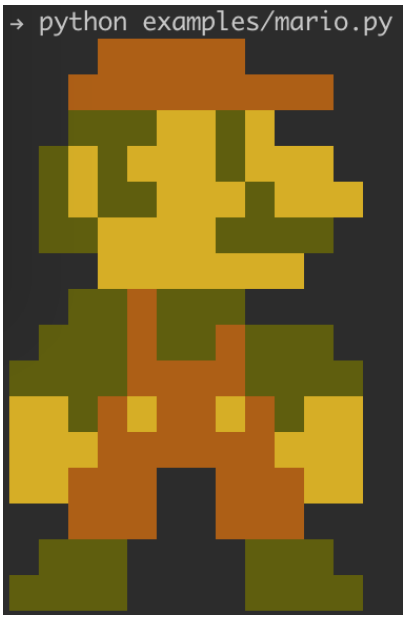
```
>>> colorise.cprint("Error: Expected a string, found int", fg='red', bg='green')
Error: Expected a string, found int
>>>
```

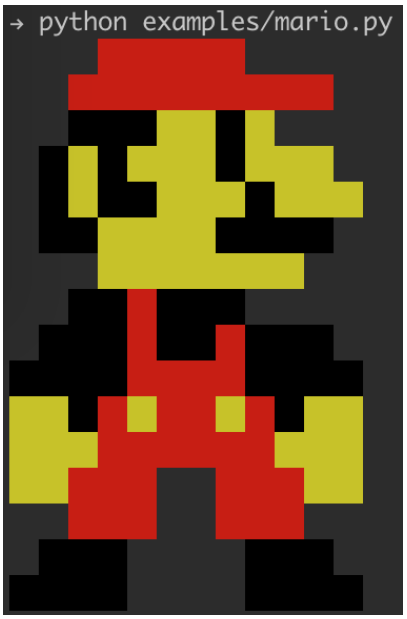
Using colorise.highlight on Windows

```
>>> 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 its best to provide uniform results although platform differences makes this hard to do 100% correct.

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 available colortable for 256 colors.

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

On Windows Vista or Windows Server 2008, colorise will read the current colortable and use that to lookup and approximate colors so your custom console colors should be reflected. If you are working on a Windows version before that, your custom colors will not be properly reflected as colorise will have to assume a default colortable.

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. This might be the case if you defined your own colors. 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()*.

The same is true for other terminals on other platforms. For example, iTerm.app allows you to redefine the basic system 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 could potentially interfere causing colors sequences to be intermingled, although it should be possible to manually achieve 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? Q: Why does colorise not do colored output on Windows?

Redirecting output or running colorise in a subprocess call means that the Windows Console API will return a 'The handle is invalid.' error (errno 6) since the output is no longer a valid console.

tox and pytest capture stdout and stderr which causes the above mentioned error.

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

Modules

12.1 colorise package

Python module for easy, cross-platform colored output to the terminal.

colorise.**can_redefine_colors**(*file*)
Return True if the terminal supports redefinition of colors.

Only returns True for Windows 7/Vista and beyond as of now.

colorise.**redefine_colors**(*color_map*, *file=<_io.TextIOWrapper name='<stdout>' mode='w' encoding='UTF-8'>*)
Redefine colors using a color map of indices and RGB tuples.

Note: It is not currently possible to redefine colors on Mac and Linux systems via colorise.

colorise.**color_names**()
Return a list of supported color names.

colorise.**num_colors**()
Return the number of colors supported by the terminal.

colorise.**set_color**(*fg=None*, *bg=None*, *attributes=None*, *file=<_io.TextIOWrapper name='<stdout>' mode='w' encoding='UTF-8'>*)
Set the current colors.

If no arguments are given, sets default colors.

colorise.**reset_color**(*file=<_io.TextIOWrapper name='<stdout>' mode='w' encoding='UTF-8'>*)
Reset all colors and attributes.

colorise.**cprint**(*string*, *fg=None*, *bg=None*, *attributes=None*, *end='\n'*, *file=<_io.TextIOWrapper name='<stdout>' mode='w' encoding='UTF-8'>*, *enabled=True*)
Print a string to a target stream with colors and attributes.

The fg and bg keywords specify foreground- and background colors while attributes is a list of desired attributes. The remaining two keyword arguments are the same as Python's built-in print function.

Colors and attributes are reset before the function returns.

colorise.**fprint**(*fmt*, *autoreset=True*, *end='\n'*, *file=<_io.TextIOWrapper name='<stdout>' mode='w' encoding='UTF-8'>*, *enabled=True*)

Print a string with color formatting.

The autoreset keyword controls if colors and attributes are reset before each new color format. For example:

```
>>> colorise.fprint('{fg=blue}Hi {bg=red}world', autoreset=False)
```

would print 'Hi' in blue foreground colors and 'world' in blue foreground colors AND a red background color, whereas:

```
>>> colorise.fprint('{fg=blue}Hi {bg=red}world', autoreset=True)
```

would print 'Hi' in blue foreground colors but 'world' only with a red background color since colors are reset when '{bg=red}' is encountered.

The remaining two keyword arguments are the same as Python's built-in print function.

Colors and attribtues are reset before the function returns.

colorise.**highlight**(*string*, *indices*, *fg=None*, *bg=None*, *attributes=None*, *end='\n'*, *file=<_io.TextIOWrapper name='<stdout>' mode='w' encoding='UTF-8'>*, *enabled=True*)

Highlight characters using indices and print to a target stream.

The indices argument is a list of indices (not necessarily sorted) for which to apply the colors and attributes. Indices that are out of bounds are ignored.

fg and bg specify foreground- and background colors while attributes is a list of desired attributes. The remaining two keyword arguments are the same as Python's built-in print function.

Colors and attribtues are reset before the function returns.

12.1.1 Subpackages

colorise.nix package

Color functionality for Linux and Mac systems.

Submodules

colorise.nix.cluts module

Nix color look-up tables (CLUTs) and functions.

colorise.nix.cluts.**can_redefine_colors**(*file*)

Return whether the terminal allows redefinition of colors.

colorise.nix.cluts.**color_from_index**(*idx*, *color_count*, *bg*, *attributes*, *file*)

Return the color prefix and color value for a given color index.

colorise.nix.cluts.**color_from_name**(*name*, *color_count*, *bg*, *attributes*)

Return the color value and color count for a given color name.

colorise.nix.cluts.**get_clut**(*color_count*)
Return the appropriate color look-up table.

colorise.nix.cluts.**get_prefix**(*color_count*, *bg*)
Get the color code prefix corresponding to the supported color count.

colorise.nix.cluts.**get_rgb_color**(*color_count*, *bg*, *rgb*, *attributes*, *file*)
Get the color for an RGB triple or approximate it if necessary.

colorise.nix.color_functions module

Linux/Mac color functions.

colorise.nix.color_functions.**attributes_to_codes**(*attributes*)
Convert a set of attributes to ANSI escape codes.

colorise.nix.color_functions.**num_colors**()
Attempt to get the number of colors supported by the terminal.

colorise.nix.color_functions.**redefine_colors**(*color_map*, *file=<_io.TextIOWrapper name='<stdout>' mode='w' encoding='UTF-8'>*)
Redefine the base console colors with a new mapping.

colorise.nix.color_functions.**reset_color**(*file=<_io.TextIOWrapper name='<stdout>' mode='w' encoding='UTF-8'>*)
Reset all colors and attributes.

colorise.nix.color_functions.**set_color**(*fg=None*, *bg=None*, *attributes=None*, *file=<_io.TextIOWrapper name='<stdout>' mode='w' encoding='UTF-8'>*, *num_colors_func=<function num_colors>*)
Set color and attributes of the terminal.

'fg' and 'bg' specify foreground- and background colors while 'attributes' is a list of desired attributes. The 'file' keyword specifies the target output stream.

colorise.nix.color_functions.**to_ansi**(**codes*)
Convert a set of ANSI codes into a valid ANSI escape sequence.

colorise.win package

Submodules

colorise.win.cluts module

colorise.win.cluts.**get_clut**(*color_count*)
Return the appropriate color look-up table.

colorise.win.cluts.**to_codes**(*bg*, *color*, *attributes*)
Convert a set of attributes to Windows character attributes.

colorise.win.cluts.**color_from_name**(*name*, *color_count*, *bg*, *attributes*)
Return the color value and color count for a given color name.

colorise.win.cluts.**color_from_index**(*idx*, *color_count*, *bg*, *attributes*)
Return the color value and color count for a given color index.

colorise.win.cluts.**get_rgb_color**(*color_count*, *bg*, *rgb*, *attributes*)
Get the color for an RGB triple or approximate it if necessary.

colorise.win.color_functions module

colorise.win.color_functions.**num_colors**()
Get the number of colors supported by the terminal.

colorise.win.color_functions.**reset_color**(*file=sys.stdout*)
Reset all colors and attributes.

colorise.win.color_functions.**or_bit_flags**(**bit_flags*)
Bitwise OR together a list of bitflags into a single flag.

colorise.win.color_functions.**set_color**(*fg=None*, *bg=None*, *attributes=[]*, *file=sys.stdout*)
Set color and attributes in the terminal.

colorise.win.color_functions.**redefine_colors**(*color_map*, *file=sys.stdout*)
Redefine the base console colors with a new mapping.

colorise.win.win32_functions module

Windows API functions.

colorise.win.win32_functions.**isatty**(*handle*)
Check if a handle is a valid console handle.

For example, if a handle is redirected to a file, it is not a valid console handle and all win32 console API calls will fail.

colorise.win.win32_functions.**can_redefine_colors**()
Return whether the terminal allows redefinition of colors.

colorise.win.win32_functions.**create_std_handle**(*handle_id*)
Create a Windows standard handle from an identifier.

colorise.win.win32_functions.**get_win_handle**(*target*)
Return the Windows handle corresponding to a Python handle.

colorise.win.win32_functions.**get_windows_clut**()
Query and return the internal Windows color look-up table.

colorise.win.win32_functions.**enable_virtual_terminal_processing**(*handle*)
Enable Windows processing of ANSI escape sequences.

colorise.win.win32_functions.**restore_console_mode**(*handle*, *restore_mode*)
Restore the console mode for a handle to its original mode.

colorise.win.win32_functions.**restore_console_modes**()
Restore console modes for stdout and stderr to their original mode.

colorise.win.win32_functions.**can_interpret_ansi**(*file*)
Return True if the Windows console can interpret ANSI escape codes.

colorise.win.win32_functions.**set_console_text_attribute**(*handle*, *flags*)
Set the console's text attributes.

colorise.win.win32_functions.**encode_rgb_tuple**(*rgb*)
Hexadecimally encode an rgb tuple as 0xbbggrr.

colorise.win.win32_functions.**redefine_colors**(*color_map*, *file=sys.stdout*)
Redefine the base console colors with a new mapping.

This only redefines the 8 colors in the console and changes all text in the console that already uses the logical names. E.g. if 'red' is mapped to the color red and this function changes it to another color, all text in 'red' will be rendered with this new color, even though it may already have been written to the console.

colorise.win.winhandle module

Wrapper around Windows output handles.

class colorise.win.winhandle.**WinHandle**
Represents a Windows stream handle.

colorise.win.winhandle.**__init__**(*handle*)
Initialise the Windows handle.

colorise.win.winhandle.**validate**(*handle*)

Classmethod

Check if a handle is valid to colorise.

colorise.win.winhandle.**from_sys_handle**(*syshandle*)

Classmethod

Return the handle identifier for a python handle.

colorise.win.winhandle.**get_nonconsole_handle**(*handle*)

Classmethod

Get a handle that works for non-console output streams.

colorise.win.winhandle.**valid**()

Property

Return True if the handle is valid, False otherwise.

colorise.win.winhandle.**value**()

Property

Return the internal Windows handle value.

colorise.win.winhandle.**is_console_handle**()

Property

Getter If the handle is a valid console handle.

Setter Set if a handle is a valid console handle or not.

colorise.win.winhandle.**fg**()

Property

Getter Return the current foreground color set for the handle.

Setter Set the current foreground color.

colorise.win.winhandle.**bg**()

Property

Getter Return the current background color set for the handle.

Setter Set the current background color.

colorise.win.winhandle.**default_fg**()

Property

Getter Return the default foreground color set for the handle.

Setter Set the default foreground color.

colorise.win.winhandle.**default_bg**()

Property

Getter Return the default background color set for the handle.

Setter Set the default background color.

colorise.win.winhandle.**console_mode**()

Property

Getter Return the current console mode for the handle.

Setter Set the current console mode for the handle.

colorise.win.winhandle.**__str__**()
Convert the handle to its string representation.

12.1.2 Submodules

12.1.3 colorise.attributes module

Attributes supported by colorise.

Depending on your terminal's capabilities, some of these attributes may have no effect.

class colorise.attributes.**Attr**
Bases: enum.Enum

Console attributes.

Each attribute's value is defined via its ANSI escape code.

Blink = 5

Bold = 1

Faint = 2

Intense = 1

Italic = 3

Reset = 0

Reverse = 7

Underline = 4

from_name = <bound method Attr.from_name of <enum 'Attr'>>

names = <bound method Attr.names of <enum 'Attr'>>

names_with_aliases = <bound method Attr.names_with_aliases of <enum 'Attr'>>

12.1.4 colorise.cluts module

Main color look-up table (CLUT) functions.

All look-up tables presented here are entirely based on research and the author's own knowledge as it is impossible to ensure complete cross-platform support across all kinds of terminals. Therefore, there is no guarantee that any of these tables will be accurate.

If you have corrections or suggestions, please submit an issue.

colorise.cluts.**get_color**(*value*, *color_count*, *cluts*, *bg=False*, *attributes=None*, *file=<_io.TextIOWrapper name='<stdout>' mode='w' encoding='UTF-8'>*)
Return the color given by a color format.

colorise.cluts.**match_color_formats**(*value*)
Return the color format of the first format to match the given value.

12.1.5 colorise.color_tools module

Functions for converting and comparing colors.

colorise.color_tools.**closest_color**(*rgb*, *clut*)
Return the CLUT index of the closest RGB color to a given RGB tuple.

colorise.color_tools.**color_difference**(*rgb1*, *rgb2*)
Return the sums of component differences between two colors.

colorise.color_tools.**hls_to_rgb**(*hue*, *lightness*, *saturation*)
Convert HLS (hue, lightness, saturation) values to RGB.

colorise.color_tools.**hsv_to_rgb**(*hue*, *saturation*, *value*)
Convert HSV (hue, saturation, value) values to RGB.

12.1.6 colorise.error module

Custom colorise exceptions.

exception colorise.error.**NotSupportedError**
Bases: Exception

Raised when functionality is not supported.

12.1.7 colorise.formatter module

ColorFormatter class for colorise.fprint.

This class extends the string.Formatter class.

class colorise.formatter.**ColorFormatter**(*set_color_func*, *reset_func*)
Bases: string.Formatter

Class for formatting strings containing color syntax.

As opposed to an ordinary derived string.Formatter, this one does not construct and return a formatted string but immediately outputs the formatted string. This is necessary to support Windows consoles that cannot interpret ANSI escape sequences since they have no way of embedding colors into strings but instead set colors through an API call.

autoreset
If True, automatically reset before each color format field.

enabled
Whether colors are enabled or not.

file
Target output handle of the formatter.

parse(*format_string*)
Parse a format string and generate tokens.

vformat(*format_string*, *args*, *kwargs*)
Hijack the internals of string.Formatter.vformat.

Copied (almost) verbatim from the Python 3.7 source code but does not return a formatted string.

12.1.8 colorise.terminal module

Terminal functions.

colorise.terminal.**terminal_name**()
Return the name of the terminal.

CHAPTER 13

Tested Systems

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

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

13.1 v2.0.0

Mac

Terminal	OS	Python versions
iTerm 3.4.3	macOS Big Sur v11.1	3.9.1
Terminal.app 2.11 (440)	macOS Big Sur v11.1	3.8.2
bash	Mac OS X 10.15.7 (GitHub Actions)	3.5, 3.6, 3.7, 3.8, 3.9

Windows

Terminal	OS	Python versions
Default Windows console	Windows 8.1 Pro, version 6.3, build 9600	3.7.3
ConEmu	Windows 8.1 Pro, version 6.3, build 9600	3.7.3
cmd.exe	Windows Server 2019 (GitHub Actions)	3.5, 3.6, 3.7, 3.8, 3.9

Linux

Terminal	OS	Python versions
Most likely sh or bash	Ubuntu 18.04.5 LTS (GitHub Actions)	3.5, 3.6, 3.7, 3.8, 3.9

13.2 v1.0.1

Mac

Terminal	OS	Python versions
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

Windows

Terminal	OS	Python versions
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

Linux

None yet.

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

You can install it via pip.

```
$ pip install colorise
```

CHAPTER 14

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.
- Cross-platform: Works on Mac, Linux and Windows systems.

```
>>> 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
```

14.1 Indices and tables

- genindex
- modindex
- search

Python Module Index

C

Index

Symbols

A

B

C

D

E

F

G

H

I

M

N

O

P

R

S

T

U

V

W