

[Help](#) [Docs](#) [Sponsors](#) [Log in](#) [Register](#)

colorama 0.4.6

✓

[Latest version](#)

```
pip install colorama
```

Released: Oct 25, 2022

Cross-platform colored terminal text.

Navigation

[Project description](#)[Release history](#)[Download files](#)

Verified details

These details have been [verified by PyPI](#)

Maintainers

[tartley](#)[wigin15](#)

Unverified details

*These details have **not** been verified by PyPI*

Project links

[Homepage](#)

Meta

- **License:** BSD License
- **Author:** [Jonathan Hartley](#)
- ansi, color, colour, crossplatform, terminal, text, windows, xplatform
- **Requires:** Python !=3.0.*, !=3.1.*, !=3.2.*, !=3.3.*, !=3.4.*, !=3.5.*, !=3.6.*, >=2.7

Classifiers

Development Status

- [5 - Production/Stable](#)

Environment

- [Console](#)

Intended Audience

- [Developers](#)

License

- [OSI Approved :: BSD License](#)

Operating System

- [OS Independent](#)

Programming Language

- [Python](#)
- [Python :: 2](#)
- [Python :: 2.7](#)
- [Python :: 3](#)
- [Python :: 3.7](#)
- [Python :: 3.8](#)
- [Python :: 3.9](#)
- [Python :: 3.10](#)
- [Python :: Implementation :: CPython](#)
- [Python :: Implementation :: PyPy](#)

Topic

- [Terminals](#)




Indeed is a Contributing sponsor of the Python Software Foundation.

PSF Sponsor · Served ethically

[Report project as malware](#)

Project description

PyPI **v0.4.6** | python 2.7 | 3.7 | 3.8 | 3.9 | 3.10 | 

Colorama

Makes ANSI escape character sequences (for producing colored terminal text and cursor positioning) work under MS Windows.

[PyPI for releases](#) | [Github for source](#) | [Colorama for enterprise on Tidelift](#)

If you find Colorama useful, please [Donate](#) to the authors. Thank you!

Installation

Tested on CPython 2.7, 3.7, 3.8, 3.9 and 3.10 and Pypy 2.7 and 3.8.

No requirements other than the standard library.

```
pip install colorama
# or
conda install -c anaconda colorama
```

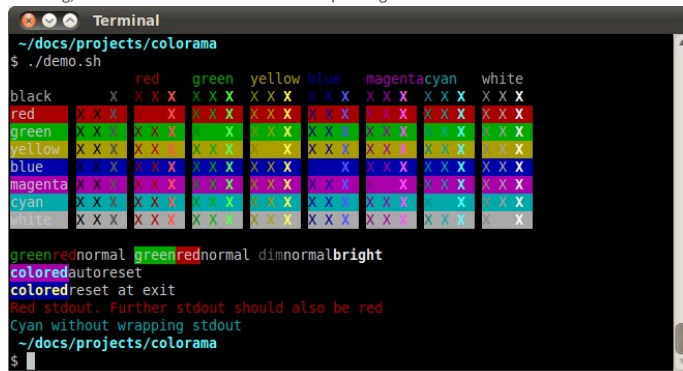
Description

ANSI escape character sequences have long been used to produce colored terminal text and cursor positioning on Unix and Macs. Colorama makes this work on Windows, too, by wrapping `stdout`, stripping ANSI sequences it finds (which would appear as gobbledygook in the output), and converting them into the appropriate win32 calls to modify the state of the terminal. On other platforms, Colorama does nothing.

This has the upshot of providing a simple cross-platform API for printing colored terminal text from Python, and has the happy side-effect that existing applications or libraries which use ANSI sequences to produce colored output on Linux or Macs can now also work on Windows, simply by calling `colorama.just_fix_windows_console()` (since v0.4.6) or `colorama.init()` (all versions, but may have other side-effects – see below).

An alternative approach is to install `ansi.sys` on Windows machines, which provides the same behaviour for all applications running in terminals. Colorama is intended for situations where that isn't easy (e.g., maybe your app doesn't have an installer.)

Demo scripts in the source code repository print some colored text using ANSI sequences. Compare their output under Gnome-terminal's built in ANSI handling, versus on Windows Command-Prompt using Colorama:



Same ANSI sequences on Windows, using Colorama.

These screenshots show that, on Windows, Colorama does not support ANSI 'dim text'; it looks the same as 'normal text'.

Usage

Initialisation

If the only thing you want from Colorama is to get ANSI escapes to work on Windows, then run:

```
from colorama import just_fix_windows_console
just_fix_windows_console()
```

If you're on a recent version of Windows 10 or better, and your stdout/stderr are pointing to a Windows console, then this will flip the magic configuration switch to enable Windows' built-in ANSI support.

If you're on an older version of Windows, and your stdout/stderr are pointing to a Windows console, then this will wrap `sys.stdout` and/or `sys.stderr` in a magic file object that intercepts ANSI escape sequences and issues the appropriate Win32 calls to emulate them.

In all other circumstances, it does nothing whatsoever. Basically the idea is that this makes Windows act like Unix with respect to ANSI escape handling.

It's safe to call this function multiple times. It's safe to call this function on non-Windows platforms, but it won't do anything. It's safe to call this function when one or both of your stdout/stderr are redirected to a file – it won't do anything to those streams.

Alternatively, you can use the older interface with more features (but also more potential footguns):

```
from colorama import init
init()
```

This does the same thing as `just_fix_windows_console`, except for the following differences:

- It's not safe to call `init` multiple times; you can end up with multiple layers of wrapping and broken ANSI support.
- Colorama will apply a heuristic to guess whether `stdout`/`stderr` support ANSI, and if it thinks they don't, then it will wrap `sys.stdout` and `sys.stderr` in a magic file object that strips out ANSI escape sequences before printing them. This happens on all platforms, and can be convenient if you want to write your code to emit ANSI escape sequences unconditionally, and let Colorama decide whether they should actually be output. But note that Colorama's heuristic is not particularly clever.
- `init` also accepts explicit keyword args to enable/disable various functionality – see below.

To stop using Colorama before your program exits, simply call `deinit()`. This will restore `stdout` and `stderr` to their original values, so that Colorama is disabled. To resume using Colorama again, call `reinit()`; it is cheaper than calling `init()` again (but does the same thing).

Most users should depend on `colorama >= 0.4.6`, and use `just_fix_windows_console`. The old `init` interface will be supported indefinitely for backwards compatibility, but we don't plan to fix any issues with it, also for backwards compatibility.

Colored Output

Cross-platform printing of colored text can then be done using Colorama's constant shorthand for ANSI escape sequences. These are deliberately rudimentary, see below.

```
from colorama import Fore, Back, Style
print(Fore.RED + 'some red text')
print(Back.GREEN + 'and with a green background')
print(Style.DIM + 'and in dim text')
print(Style.RESET_ALL)
print('back to normal now')
```

...or simply by manually printing ANSI sequences from your own code:

```
print('\033[31m' + 'some red text')
print('\033[39m') # and reset to default color
```

...or, Colorama can be used in conjunction with existing ANSI libraries such as the venerable [Termcolor](#) the fabulous [Blessings](#), or the incredible [Rich](#).

If you wish Colorama's `Fore`, `Back` and `Style` constants were more capable, then consider using one of the above highly capable libraries to generate colors, etc, and use Colorama just for its primary purpose: to convert those ANSI sequences to also work on Windows:

SIMILARLY, do not send PRs adding the generation of new ANSI types to Colorama. We are only interested in converting ANSI codes to win32 API calls, not shortcuts like the above to generate ANSI characters.

```
from colorama import just_fix_windows_console
from termcolor import colored

# use Colorama to make Termcolor work on Windows too
just_fix_windows_console()

# then use Termcolor for all colored text output
print(colored('Hello, World!', 'green', 'on_red'))
```

Available formatting constants are:

```
Fore: BLACK, RED, GREEN, YELLOW, BLUE, MAGENTA, CYAN, WHITE, RESET.
Back: BLACK, RED, GREEN, YELLOW, BLUE, MAGENTA, CYAN, WHITE, RESET.
Style: DIM, NORMAL, BRIGHT, RESET_ALL
```

`Style.RESET_ALL` resets foreground, background, and brightness. Colorama will perform this reset automatically on program exit.

These are fairly well supported, but not part of the standard:

```
Fore: LIGHTBLACK_EX, LIGHTRED_EX, LIGHTGREEN_EX, LIGHTYELLOW_EX, LIGHTBLUE_EX, LIGHTMAGENTA_EX, LIGHTCYAN_EX, LIGHTWHITE_EX
Back: LIGHTBLACK_EX, LIGHTRED_EX, LIGHTGREEN_EX, LIGHTYELLOW_EX, LIGHTBLUE_EX, LIGHTMAGENTA_EX, LIGHTCYAN_EX, LIGHTWHITE_EX
```

Cursor Positioning

ANSI codes to reposition the cursor are supported. See `demos/demo06.py` for an example of how to generate them.

Init Keyword Args

`init()` accepts some `**kwargs` to override default behaviour.

`init(autoreset=False):`

If you find yourself repeatedly sending reset sequences to turn off color changes at the end of every print, then `init(autoreset=True)` will automate that:

```
from colorama import init
init(autoreset=True)
print(Fore.RED + 'some red text')
print('automatically back to default color again')
```

init(strip=None):

Pass `True` or `False` to override whether ANSI codes should be stripped from the output. The default behaviour is to strip if on Windows or if output is redirected (not a tty).

init(convert=None):

Pass `True` or `False` to override whether to convert ANSI codes in the output into win32 calls. The default behaviour is to convert if on Windows and output is to a tty (terminal).

init(wrap=True):

On Windows, Colorama works by replacing `sys.stdout` and `sys.stderr` with proxy objects, which override the `.write()` method to do their work. If this wrapping causes you problems, then this can be disabled by passing `init(wrap=False)`. The default behaviour is to wrap if `autoreset` or `strip` or `convert` are `True`.

When wrapping is disabled, colored printing on non-Windows platforms will continue to work as normal. To do cross-platform colored output, you can use Colorama's `AnsiToWin32` proxy directly:

```
import sys
from colorama import init, AnsiToWin32
init(wrap=False)
stream = AnsiToWin32(sys.stderr).stream

# Python 2
print >>stream, Fore.BLUE + 'blue text on stderr'

# Python 3
print(Fore.BLUE + 'blue text on stderr', file=stream)
```

Recognised ANSI Sequences

ANSI sequences generally take the form:

```
ESC [ <param> ; <param> ... <command>
```

Where `<param>` is an integer, and `<command>` is a single letter. Zero or more params are passed to a `<command>`. If no params are passed, it is generally synonymous with passing a single zero. No spaces exist in the sequence; they have been inserted here simply to read more easily.

The only ANSI sequences that Colorama converts into win32 calls are:

```
ESC [ 0 m      # reset all (colors and brightness)
ESC [ 1 m      # bright
ESC [ 2 m      # dim (looks same as normal brightness)
ESC [ 22 m     # normal brightness

# FOREGROUND:
ESC [ 30 m     # black
ESC [ 31 m     # red
ESC [ 32 m     # green
ESC [ 33 m     # yellow
ESC [ 34 m     # blue
ESC [ 35 m     # magenta
ESC [ 36 m     # cyan
ESC [ 37 m     # white
ESC [ 39 m     # reset

# BACKGROUND
ESC [ 40 m     # black
ESC [ 41 m     # red
ESC [ 42 m     # green
ESC [ 43 m     # yellow
ESC [ 44 m     # blue
ESC [ 45 m     # magenta
ESC [ 46 m     # cyan
ESC [ 47 m     # white
ESC [ 49 m     # reset

# cursor positioning
ESC [ y;x H    # position cursor at x across, y down
ESC [ y;x f    # position cursor at x across, y down
ESC [ n A      # move cursor n lines up
ESC [ n B      # move cursor n lines down
ESC [ n C      # move cursor n characters forward
ESC [ n D      # move cursor n characters backward

# clear the screen
ESC [ mode J   # clear the screen

# clear the line
ESC [ mode K   # clear the line
```

Multiple numeric params to the `'m'` command can be combined into a single sequence:

```
ESC [ 36 ; 45 ; 1 m      # bright cyan text on magenta background
```

All other ANSI sequences of the form `ESC [<param> ; <param> ... <command>` are silently stripped from the output on Windows.

Any other form of ANSI sequence, such as single-character codes or alternative initial characters, are not recognised or stripped. It would be cool to add them though. Let me know if it would be useful for you, via the Issues on GitHub.

Status & Known Problems

I've personally only tested it on Windows XP (CMD, Console2), Ubuntu (gnome-terminal, xterm), and OS X.

Some valid ANSI sequences aren't recognised.

If you're hacking on the code, see [README-hacking.md](#). ESPECIALLY, see the explanation there of why we do not want PRs that allow Colorama to generate new types of ANSI codes.


See outstanding issues and wish-list: <https://github.com/tartley/colorama/issues>

If anything doesn't work for you, or doesn't do what you expected or hoped for, I'd love to hear about it on that issues list, would be delighted by patches, and would be happy to grant commit access to anyone who submits a working patch or two.

License

Copyright Jonathan Hartley & Arnon Yaari, 2013-2020. BSD 3-Clause license; see LICENSE file.

Professional support



Professional support for colorama is available as part of the [Tidelift Subscription](#). Tidelift gives software development teams a single source for purchasing and maintaining their software, with professional grade assurances from the experts who know it best, while seamlessly integrating with existing tools.

Thanks

See the CHANGELOG for more thanks!

- Marc Schlaich (schlamar) for a `setup.py` fix for Python2.5.
- Marc Abramowitz, reported & fixed a crash on exit with closed `stdout`, providing a solution to issue #7's `setuptools/distutils` debate, and other fixes.
- User 'eryksun', for guidance on correctly instantiating `ctypes.windll`.
- Matthew McCormick for politely pointing out a longstanding crash on non-Win.
- Ben Hoyt, for a magnificent fix under 64-bit Windows.
- Jesse at Empty Square for submitting a fix for examples in the README.
- User 'jamessp', an observant documentation fix for cursor positioning.
- User 'vaal1239', Dave Mckee & Lackner Kristof for a tiny but much-needed Win7 fix.
- Julien Stuyck, for wisely suggesting Python3 compatible updates to README.
- Daniel Griffith for multiple fabulous patches.
- Oscar Lesta for a valuable fix to stop ANSI chars being sent to non-tty output.
- Roger Binns, for many suggestions, valuable feedback, & bug reports.
- Tim Golden for thought and much appreciated feedback on the initial idea.
- User 'Zearin' for updates to the README file.
- John Szakmeister for adding support for light colors
- Charles Merriam for adding documentation to demos
- Jurko for a fix on 64-bit Windows CPython2.5 w/o ctypes
- Florian Bruhin for a fix when `stdout` or `stderr` are `None`
- Thomas Weininger for fixing `ValueError` on Windows
- Remi Rampin for better Github integration and fixes to the README file
- Simeon Visser for closing a file handle using 'with' and updating classifiers to include Python 3.3 and 3.4
- Andy Neff for fixing RESET of LIGHT_EX colors.
- Jonathan Hartley for the initial idea and implementation.



Help

Installing packages
Uploading packages
User guide
Project name retention
FAQs

About PyPI

PyPI Blog
Infrastructure dashboard
Statistics
Logos & trademarks
Our sponsors

Contributing to PyPI

Bugs and feedback
Contribute on GitHub
Translate PyPI
Sponsor PyPI
Development credits

Using PyPI

Terms of Service
Report security issue
Code of conduct
Privacy Notice
Acceptable Use Policy

Status: [All Systems Operational](#)

Developed and maintained by the Python community, for the Python community.
[Donate today!](#)

"PyPI", "Python Package Index", and the blocks logos are registered [trademarks](#) of the [Python Software Foundation](#).

© 2025 [Python Software Foundation](#)
[Site map](#)

